Chapter 15: Numbers and Equations

 Contents of *Writing with Inform*

 Chapter 14: Adaptive Text and Responses

 Chapter 16: Tables

 Indexes of the examples

§15.1. How do we measure things?

In a poem, or in a novel, exact scientific measurements are not the point. So a writer who wants to set up ways to describe the sky at different times might go for something like this:

```
The sky can be cadmium, mackerel, overcast or cornflower.
```

And nobody is interested in the sun angle, the percentage of cloud cover, or any of the other numbers behind all of this. Similarly, if we walk into a familiar office which has been disturbed, we might well say "Look! The filing cabinet is in the middle of the floor." We are not likely to exclaim "Look! The filing cabinet is 1.2m from the east wall and 2.1m from the north wall."

But some writers of interactive fiction do like to make use of physical realism. For instance, it's easier to forbid a bulky object being taken through a narrow doorway if there is a way to measure and compare sizes.

Most computer programs write numbers in the same way, whatever they're used for. But human beings don't. If someone says "How far is Duluth?", we're more likely to say "100 miles" than just "100". This is a useful feature of natural language, because it means we always know how to translate that number into reality - it's 100 miles, not 100 km, or 100 inches; and it's definitely a distance, not 100 apples or 100 kilograms.

Inform lets us use plain numbers if we want to, but it also allows us to create numerical kinds of value:

```
A distance is a kind of value. 5 miles specifies a distance.
```

That kind of definition, and the consequences, will be the subject of this chapter. But we will first look a little harder at the two numerical kinds of value we get for free: "number" and "real number".

Start of Chapter 15: Numbers and Equations
Back to Chapter 14: Adaptive Text and Responses: §14.12. The RESPONSES testing command
Onward to §15.2. Numbers and real numbers

---

§15.2. Numbers and real numbers

Inform uses two different kinds of numerical quantity: "number" and "real number". Neither is better than the other: they're different approaches, each good for a different purpose.

What Inform calls a "number" is a whole number, positive, negative or zero. The range of numbers we can hold is not unlimited - if the format Setting for a project is the Z-machine, then we have:

```
-32768, -32767, ..., -3, -2, -1, 0, 1, 2, 3, ..., 32767
```

and if it is set to Glulx, then we have:

```
-2147483648, -2147483647, ..., -3, -2, -1, 0, 1, 2, 3, ..., 2147483647
```

Numbers from zero to twelve may be written out, but larger ones must be written as numerals. So "twelve" or "12", but "13" only.

If we're using Glulx, Inform also has "real numbers" such as

```
2.1718, 4.0, -1633.9
```

which are not restricted to whole numbers, but which are stored only approximately: only about six to nine decimal digits can be relied on. For example,

```
showme 1.2345654321;
showme 1.2345667890;
```

produces

```
real number: 1.23457
real number: 1.23457
```

because these two numbers are so close together that Inform can't tell them apart. But we do also get the ability to represent enormously large or small quantities, and to help with that, Inform can read and write "scientific notation". For example,

```
let Avogadro's number be 6.022141 x 10^23;
```

is equivalent to typing

```
let Avogadro's number be 602214100000000000000000.0;
```

The "x 10^23" part tells Inform that the decimal point belongs 23 places to the left of where it's written. (In scientific papers, the 23 would be printed as a superscript -- it's 10 to the power 23 -- but that's not convenient to type in to the source text, so we use the "^" symbol to indicate superscript.) The range we can hold is roughly:

```
1.18 x 10^−38 to 3.4 x 10^38
```

It's hard to convey just how enormously different these two numbers are: if we used them to measure widths in meters, one would be a hundred trillion trillion times smaller than an atom, the other a billion times larger than the entire visible universe. Scientific notation is the ultimate adjustable spanner.

Inform also allows the two most famous real numbers in mathematics to be given by their names:

```
pi
e
```

which are close to 3.14159265 and 2.7182818 respectively. (Lower case letters must be used: these can't be written "Pi" or "E". Euler's constant gamma, always in the bronze medal position, will have to be written out longhand as 0.5772156649.)

Most computer programming languages traditionally write floating-point numbers using the E notation, like so:

```
6.022141E+23;
```

Inform will follow suit if the use option "Use engineering notation." is set, but by default it isn't.

Start of Chapter 15: Numbers and Equations

Back to §15.1. How do we measure things?

Onward to §15.3. Real number conversions

Example 253: **Alias** A telephone with phone numbers of the standard American seven-digit length.

---

§15.3. Real number conversions

This section notes down some technicalities about real numbers which need to be put down in writing somewhere, but won't affect most people most of the time.

Inform allows us to use numbers whenever real numbers are expected, and converts them automatically. For example,

```
cosine of 2
```

is read as if it were

```
cosine of 2.0
```

and produces -0.41615 either way. This conversion goes from exactness to approximation, so we may lose a little accuracy: real numbers measure to an accuracy of about 1 part in 16000000, so they'll have trouble telling the difference between 16000000 and 16000001. But this is unlikely to matter, since real numbers are used only for approximate calculations anyway.

The ordinary arithmetic operations work on both numbers and real numbers, so the meaning of "N plus M" depends on the kinds of N and M. In general the rule is that if either is a real number then the other one is automatically converted, and real arithmetic is used. So:

```
3 divided by 2 = 1
3 divided by 2.0 = 1.5
3.0 divided by 2 = 1.5
3.0 divided by 2.0 = 1.5
```

In general we can't do the reverse, that is, we can't silently use a real number where a number is expected. For example,

```
word number 1.6 in "The Great Wall of China"
```

makes no sense. But we can explicitly convert them:

(real number) **to the nearest whole number ... number**

This phrase performs signed addition on the given values, whose kinds must agree, and produces the result. Examples:

```
1.4 to the nearest whole number = 1
1.6 to the nearest whole number = 2
-1.6 to the nearest whole number = -2
```

We probably ought to bear in mind that the limited range of "number" means that the nearest whole number might not be all that near. For example:

```
6 x 10^23 to the nearest whole number = 2147483647
```

because 2147483647 is the highest value a "number" can have.

Finally, real number can also store two interesting not-really-number sorts of value. First, we have

```
plus infinity, minus infinity
```

which are used to keep track of what happens when we divide by really small quantities. It's mathematically impossible to divide by 0, but this can be hard to avoid when we're using real

numbers, because they're only approximately stored - so it's not always possible to say whether they're exactly 0 or not. So in real number arithmetic,

```
showme 1.0 divided by 0.0;
```

doesn't throw a run-time problem the way that

```
showme 1 divided by 0;
```

does. Instead, it produces "plus infinity". Infinity behaves roughly the way we might expect - for example, "2 divided by plus infinity" produces 0 - but once it comes into a calculation the result probably lies on some extreme and won't be very useful. Amusingly, the following is correct Inform syntax:

```
plus infinity to the nearest whole number
```

and evaluates of course to 2147483647. We can use the adjectives "infinite" and "finite" to talk about these numbers: plus infinity and minus infinity are infinite, everything else is finite.

The same problem occurs for calculations like square roots. It's impossible to take the square root of a negative number, but we don't want to throw a run-time problem, because approximation means we can't always guarantee to stay the right side of 0. So for a few calculations like this, Inform generates what's called a "nonexistent" real number. We can use the adjectives nonexistent or existent to talk about this. Every number mentioned on this page so far is "existent", including the infinities. The only way to get a nonexistent number is to carry out an impossible mathematical operation such as

```
logarithm of -10
```

(The design of "real number" here follows well established trade-offs for scientific computing. Inform follows the IEEE-754 binary32 standard for floating-point arithmetic, so Inform's "real number" behaves very like the "float" type in C, C++, Java and similar programming languages. A "nonexistent" number is what's often called a NaN - a Not-a-Number.)


Start of Chapter 15: Numbers and Equations


Back to §15.2. Numbers and real numbers


Onward to §15.4. Printing real numbers

---

§15.4. Printing real numbers

say "[(real number) **to** (number) **decimal places]"**

This text substitution writes out the number to the given number of decimal places. Examples:

"The semicircle is roughly [pi to 3 decimal places] paces around."

produces "The semicircle is roughly 3.142 paces around." The number of places can only usefully be from 1 to 8. Note that, for example, "[1.235 x 10^-7 to 3 decimal places]" produces 0.0; "[1.235678 x 10^8 to 3 decimal places]" produces "1.236 × 10^8".

say "[(real number) **in decimal notation]"**

This text substitution writes out the number in decimal form, that is, avoiding "x 10^n" even for very large or very small quantities. For example,

"[1.23457 x 10^8 in decimal notation]"

produces 123457000.0 rather than 1.23457 x 10^8. This can look pretty extreme: for example, "[1.8983 x 10^27 in decimal notation]", the mass of the planet Jupiter in kilograms, produces 1898296960000000000000000000.0.

say "[(real number) **to** (number) **decimal places in decimal notation]"**

This text substitution writes out the number in decimal form, but rounding to the accuracy given.

say "[(real number) **in scientific notation]"**

This text substitution writes out the number in scientific form, that is, using "x 10^n" even for easy-to-judge quantities. For example,

"[the reciprocal of 137 in scientific notation]"

produces 7.29927 x 10^-3 rather than 0.0073. This can look odd: for example, "[pi in scientific notation]" comes out as 3.14159 × 10^0 rather than 3.14159.

say "[(real number) **to** (number) **decimal places in scientific notation]"**

This text substitution writes out the number in scientific form, but rounding to the accuracy given.

Start of Chapter 15: Numbers and Equations
Back to §15.3. Real number conversions
Onward to §15.5. Arithmetic

---

§15.5. Arithmetic

We are allowed to perform about the same operations on numbers as are provided by a simple office calculator, starting with addition, subtraction, multiplication and division. We can use the traditional typewriter symbols for these, +, -, * and /, or can spell them out in words as "plus", "minus", "times" (or "multiplied by"), and "divided by". Definitively:

(arithmetic value) + (arithmetic value) **... value**

or:
(arithmetic value) **plus** (arithmetic value) **... value**

This phrase performs signed addition on the given values, whose kinds must agree, and produces the result. Examples:

```
200 + 1 = 201
10:04 AM + two minutes = 10:06 AM
```

(arithmetic value) **-** (arithmetic value) **... value**

or:
(arithmetic value) **minus** (arithmetic value) **... value**

This phrase performs signed subtraction on the given values, whose kinds must agree, and produces the result. Examples:

```
200 - 1 = 199
10:04 AM - two minutes = 10:02 AM
```

(arithmetic value) ***** (arithmetic value) **... value**

or:
(arithmetic value) **times** (arithmetic value) **... value**

or:
(arithmetic value) **multiplied by** (arithmetic value) **... value**

This phrase performs signed multiplication on the given values, whose kinds must be dimensionally compatible, and produces the result. Examples:

```
201 times 3 = 603
two minutes times 4 = eight minutes
```

(arithmetic value) / (arithmetic value) **... value**

or:
(arithmetic value) **divided by** (arithmetic value) **... value**

This phrase performs signed division on the given values, whose kinds must be dimensionally compatible, and produces the result. Examples:

```
201 divided by 3 = 67
202 divided by 3 = 67
202.0 divided by 3 = 67.33334
twenty minutes divided by 4 = five minutes
twenty minutes divided by five minutes = 4
```

Division rounds whole-number values down to the nearest whole number. An attempt to divide a number by 0 will cause a run-time problem message; but an attempt to divide a real number by 0 will instead produce plus infinity or minus infinity.

remainder after dividing (arithmetic value) **by** (arithmetic value) **... value**

This phrase performs signed division on the given values, whose kinds must be dimensionally compatible, and then produces the remainder. Examples:

```
remainder after dividing 201 by 5 = 1
remainder after dividing twenty minutes by 7 = six minutes
```

It is mathematically impossible to divide by 0, so any attempt to find the remainder after dividing a number by 0 will cause a run-time problem message. For a real number this won't arise and the remainder will usually be 0.0.

The verbal and symbolic forms of these phrases are equivalent:

```
the score + 10
the score plus 10
```

It's probably better style to spell them out in full when writing text, and keep the symbols for writing equations, as we'll see later on in the chapter. (If we do use the symbols, then spaces

around them are obligatory: to Inform, they are words which just happen to be spelt with symbols instead of letters.)

Arithmetic often produces fussily exact answers which seem inappropriate in a conversation. Nobody says "Steeple Barton is 7.655 miles down the road", but "Steeple Barton is eight miles down the road" sounds perfectly normal. In order to make that sort of report easier to make, Inform provides another arithmetic operation, one that's not found in most computer programming languages:

(arithmetic value) **to the nearest** (arithmetic value) **... value**

This phrase rounds the given value off, rounding upward in boundary cases. Examples:

```
201 to the nearest 5 = 200
205 to the nearest 10 = 210
10:27 AM to the nearest five minutes = 10:25 AM
```

Inform has very few mathematical functions built in as phrases, because these aren't very often needed in story-telling. But it does provide these:

square root of (arithmetic value) **... value**

This phrase produces an approximate square root, to the nearest integer, of the given value, which must be of a kind which has square roots. Example:

```
square root of 16 = 4
```

Trying to take the square root of a negative number will cause a run-time problem, because then we can't even nearly solve it.

(Warning: this is slow to compute if the Z-machine setting is used. For

best performance, use Glulx.)

real square root of (arithmetic value) **... value**

This phrase produces a square root, as accurately as a real number can hold it, of the given value, which must be of a kind which has square roots. Example:

```
real square root of 2 = 1.41421
```

The real square root of a negative number is nonexistent.

cube root of (arithmetic value) **... value**

This phrase produces an approximate cube root, to the nearest integer, of the given value, which must be of a kind which has cube roots. Example:

```
cube root of 27 = 3
cube root of -27 = -3
```

(Warning: this is not very accurate if the Z-machine setting is used. For

best performance, use Glulx.)

We can compare numbers using either the traditional computer-programming symbols, or using words:

```
if the score is less than 10
if the score < 10
```

and similarly for "greater than", "at least" and "at most", with the symbols ">", ">=" and "<=". But we are not allowed the equals sign: for that we need only use "is" -

```
if the score is 10
```

Start of Chapter 15: Numbers and Equations

Back to §15.4. Printing real numbers

Onward to §15.6. Powers and logarithms

---

§15.6. Powers and logarithms

If the last section provided a basic office calculator, this section and the next provide the more exotic rows of buttons found on a scientific calculator. All of these are done using real number arithmetic. To start with some dull ones, here are two ways to round off numbers:

ceiling of (real number) **... real number**

Produces the smallest integer value greater than or equal to the one given. Examples:

```
ceiling of pi = 4.0
ceiling of -16.315 = -16.0
```

(Note that the result is still a real number; it simply has no fractional part any more.)

floor of (real number) **... real number**

Produces the largest integer value less than or equal to the one given. Examples:

```
floor of pi = 3.0
floor of -16.315 = -17.0
```

(Note that the result is still a real number; it simply has no fractional part any more.)

Two more easy functions:

absolute value of (real number) **... real number**

Removes the sign from a value, leaving positive numbers alone but making negative ones positive. Examples:

```
absolute value of 62.1 = 62.1
absolute value of 0 = 0.0
absolute value of -62.1 = 62.1
absolute value of minus infinity = plus infinity
```

reciprocal of (real number) **... real number**

Calculates 1/x, that is, divides up 1 into this many pieces. Examples:

```
reciprocal of -2 = -0.5
reciprocal of 0.1 = 10.0
reciprocal of 7 = 0.14286
reciprocal of plus infinity = 0.0
```

Now for taking powers. In general we have:

(real number) **to the power** (real number) **... real number**

Computes x to the power y. Examples:

```
2 to the power 4 = 16.0
100 to the power 0.5 = 10.0
7 to the power -1 = 0.14286
pi to the power 0 = 1.0
```

In the words of the Glulx specification document (section 2.12), "the special cases are breathtaking": if you need to know exactly what, say, "minus infinity to the

power Y" will do for different cases of Y, refer to the details of the "pow" opcode.

To compute square roots, it's more efficient to use "real square root of X" function than "X to the power 0.5", though both work. To obtain the Nth root of X, we might use:

```
X to the power (reciprocal of N)
```

being careful to use "reciprocal of N" rather than "1 divided by N" to make sure we're using real and not integer arithmetic.

Similarly, the following is more efficient than "e to the power ...", but equivalent to it:

exponential of (real number) **... real number**

Computes e to the given power, where e is the base of natural logarithms. Examples:

```
exponential of 0 = 1.0
exponential of 1 = e = 2.7182818
exponential of -10 = 4.53999 x 10^-5
exponential of 10 = 22026.46484
exponential of logarithm of 7.12 = 7.12
```

The reverse of taking powers is taking logarithms.

logarithm to base (number) **of** (real number) **... real number**

Finds what power the base would have to be raised to in order to get this value. Examples:

```
logarithm to base 10 of 1000000 = 6.0
logarithm to base 10 of 350 = 2.54407
logarithm to base 2 of 256 = 8.0
```

Logarithms of zero or negative numbers are nonexistent. Note that "logarithm to base 10 of ..." is what most calculators call simply "log", but Inform doesn't: it uses "log" for natural logarithms.

natural/-- **logarithm of** (real number) **... real number**

Finds what power e would have to be raised to in order to get this value. Examples:

```
logarithm of e = 1.0
logarithm of 1 = 0.0
```

```
logarithm of 1000 = 6.90776
logarithm of exponential of 7.12 = 7.12
```

Logarithms of zero or negative numbers are nonexistent. This is the function which most calculators label as "ln", for "log natural", but in mathematical and scientific papers it's more often written "log", and Inform follows that convention.


Start of Chapter 15: Numbers and Equations


Back to §15.5. Arithmetic


Onward to §15.7. Trigonometry

---

§15.7. Trigonometry

We have twelve functions left to cover, though they are all closely related.

(real number) **degrees ... real number**

Inform measures angles in radians, a convention in which the angle for a half circle is pi, and a right angle is pi divided by 2. This is better from a mathematical point of view, but in practice most people think about angles using degrees, where 180 degrees is a half-circle and a right angle is 90 degrees. This phrase helps with that by converting from degrees to radians: in other words, it multiplies by 0.0174532925, since that's roughly 1/180th of pi. Examples:

```
sine of 90 degrees = 0.0
cosine of 60 degrees = 0.5
```

sine of (real number) **... real number**

The length of the upright of a right-angled triangle with this angle and a hypotenuse of length 1, where angle is measured in radians. Examples:

```
sine of 0 = 0
sine of 45 degrees = 0.70711
sine of (pi divided by 4) = 0.70711
sine of (pi divided by 2) = 1.0
sine of pi = 0
```

cosine of (real number) **... real number**

The length of the base of a right-angled triangle with this angle and a hypotenuse of length 1, where angle is measured in radians. Examples:

```
cosine of 0 = 1.0
cosine of 45 degrees = 0.70711
cosine of (pi divided by 4) = 0.70711
cosine of (pi divided by 2) = 0.0
cosine of pi = -1.0
```

tangent of (real number) **... real number**

The ratio of the upright length to the base length in a right-angled triangle with this angle and a hypotenuse of length 1, where angle is measured in radians. Examples:

```
tangent of 0 = 0.0
tangent of 45 degrees = 1.0
tangent of (pi divided by 4) = 1.0
tangent of (pi divided by 2) = plus infinity
```

arcsine of (real number) **... real number**

The inverse of the sine function.

arccosine of (real number) **... real number**

The inverse of the cosine function.

arctangent of (real number) **... real number**

The inverse of the tangent function.

hyperbolic sine of (real number) **... real number**

The hyperbolic sine function, often written "sinh" but pronounced "shine".

hyperbolic cosine of (real number) **... real number**

The hyperbolic cosine function, often written "cosh".

hyperbolic tangent of (real number) **... real number**

The hyperbolic tangent function, often written "tanh".

hyperbolic arcsine of (real number) **... real number**

The inverse of the hyperbolic sine function.

hyperbolic arccosine of (real number) **... real number**

The inverse of the hyperbolic cosine function.

hyperbolic arctangent of (real number) **... real number**

The inverse of the hyperbolic tangent function.

Start of Chapter 15: Numbers and Equations
Back to §15.6. Powers and logarithms
Onward to §15.8. Units

---

§15.8. Units

Suppose we want to talk about how tall people are. We could just create a "number" property, like this:

```
A person has a number called height.
```

But then we would have to write lines like "Isabella has height 68", which nobody would naturally say. What we want is to be able to write "Isabella is 5 foot 8." Perhaps the computer will need to store that measurement as the number 68 in some register or other, but we don't want to know about that.

"5 foot 8" is a complicated notation in a way - it involves both feet and inches - so let's start with a simpler example:

```
A weight is a kind of value. 10kg specifies a weight.
```

This is a little different to the kinds of value seen so far, which were all created like so:

```
A colour is a kind of value. The colours are red, green and blue.
```

We can't mix the two styles: a new kind of value will either be numerical at heart ("10kg") or verbal at heart ("blue").

The effect of "10kg specifies a weight" is to tell Inform that this is the notation for writing a constant "weight". So, for instance,

```
The maximum load is a weight that varies. The maximum load is 8000kg.

if the maximum load is greater than 8000kg, ...
```

Inform is then careful not to allow weights to be mixed up with other numerical values. For instance, it won't allow "if the maximum load is 400", because 400 is a number, not a weight.

More or less anything we can do with numbers, we can now do with weights. For instance, we can write:

```
The Weighbridge is a room. "A sign declares that the maximum load is [maximum load]."
```

...which will produce the text "A sign declares that the maximum load is 8000kg."

Numerical kinds of value are sometimes called "units", because one of their main uses is to allow us to write quantities using scientific units such as kilograms. But they have other uses too. We have a great deal of freedom in creating notations like "10kg", or "4 foot 10" - the main thing is that new notations must not already mean a value. So "10 specifies a weight" will not be allowed, because 10 specifies a number already.

By default we can only write whole-number values. As we've seen, Inform can handle both integer (whole-number) and real arithmetic, and they each have their advantages. The default here is to use whole numbers, so

```
10 kg specifies a weight.
```

will store only whole numbers of kilograms (unless clever scaling tricks are used: see the next section). That may be fine, but if we need to handle a wider range of weights, or do scientific calculations that need to be more accurate, this is better:

```
1.0 kg specifies a weight.
```

Here Inform can see from the decimal point in the prototype number that real numbers will be involved. We can still write "8000kg", but we can now also write "1.9885 x 10^30 kg" (the mass of the Sun) or "9.109383 x 10^−31 kg" (the mass of an electron). On the other hand, any calculations we do will be limited in accuracy to about 6 to 9 decimal places, exactly as for real numbers.

By default we can only write positive values when whole numbers are used. Sometimes it is unnatural to write negative values, and so Inform will issue a Problem message if this is tried - for instance, Inform would not allow us to write a weight of -4 kg. (This doesn't mean that arithmetic on units is forbidden to get a negative result: we may want to work out the difference between two weights. Inform's Problem message is simply to try to prevent the accidental writing of incorrect values.) If we do want the ability to write negative values in the source text, we signal that in the notation itself:

```
-10 kg specifies a weight.
```

That alerts Inform that both positive and negative values for this unit make sense.

If we set up a spread of multiple notations (see the next section) then this is automatically enabled, because then we're clearly dealing with proper physics, where negative values are common; and similarly if we use real numbers (as above).

Start of Chapter 15: Numbers and Equations

Back to §15.7. Trigonometry

Onward to §15.9. Multiple notations

Example 254: **rBGH** The player character's height is selected randomly at the start of play.

Example 255: **Lethal Concentration 1** A poisonous gas that spreads from room to room, incapacitating or killing the player when it reaches sufficient levels.

Example 256: **Wonderland** Hiking Mount Rainier, with attention to which locations are higher and which lower than the present location.

Example 257: **Lethal Concentration 2** Poisonous gas again, only this time it sinks.

---

§15.9. Multiple notations

Going back to our weight example:

```
A weight is a kind of value. 10kg specifies a weight.
```

The notation here is a single word, even if it contains digits as well as letters - "10kg". But it doesn't have to be one word. These would have worked, too:

```
10kg net specifies a weight.
10 kg specifies a weight.
```

In fact, we are allowed to have all three at once, as alternatives:

```
A weight is a kind of value. 10kg specifies a weight. 10kg net specifies a weight. 10 kg
specifies a weight.
```

If we often have to deal with large weights, it becomes a little cumbersome to keep on writing something like "80000kg". An engineer would write "80 tonnes" for this. Similarly, we wouldn't like road maps to use light years, or speed limit signs to use furlongs per

fortnight. So it's sometimes useful to provide a spread of different notations, at different scale factors, for the same kind of value. Here's one way of setting up the tonne, that is, the metric ton:

```
1 tonne specifies a weight scaled up by 1000.
```

This really is an alternative way to write the same thing: for instance, Inform will allow "25kg plus 3 tonne", the result being "3.025 tonne".

That's all very well, but a value like "3 tonne" reads a little oddly, even if it's correct in theory. Outside of scientific journals with old-school copy editing, most people would write "3 tonnes", not "3 tonne". Here's a better try:

```
1 tonne (singular) specifies a weight scaled up by 1000.
2 tonnes (plural) specifies a weight scaled up by 1000.
```

Now Inform will not only recognise both forms, but also use the right one when printing back.


Start of Chapter 15: Numbers and Equations


Back to §15.8. Units


Onward to §15.10. Scaling and equivalents

---

§15.10. Scaling and equivalents

As we've seen, there are two ways to store values like lengths or weights: as whole numbers, or as real numbers. If we prefer to use whole numbers, or if real numbers aren't available (for example if we're using the Z-machine setting), then we might run into an awkward problem: when we write

```
1 kg specifies a weight.
```

we make this correspond to the whole number "1", and that means Inform can never handle weights smaller than 1 kg.

But as we've seen, we can provide differently scaled notations for the same unit:

```
A length is a kind of value. 1m specifies a length.
1km specifies a length scaled up by 1000.
```

And this allows us to write "0.45km" instead of "450m", if we want to, both having the same effect. "0.45km" doesn't make a real number, despite the decimal point - it's simply another way to write "450m", stored internally as the whole number 450.

Just as we can scale up, so we can also scale down:

```
1cm specifies a length scaled down by 100.
```

Now we have a spread of three notations, so "3cm", "0.03m" and "0.00003km" all mean the same thing. But something quite interesting happened at the same time: Inform realised that we want to know lengths to a greater accuracy than just a whole number of meters.

If we're using whole numbers, and we want to resolve down to very small values, that reduces the size of the largest value we can have. For instance, with the Glulx format setting, writing just

```
A length is a kind of value. 1m specifies a length.
```

gives us a range of 1m up to 2147483647m, which is plenty - it's about six times the distance from the Earth to the Moon. Going down to centimeters:

```
A length is a kind of value. 1m specifies a length. 1cm specifies a length scaled down by
100.
```

gives us instead 1cm up to 21474836.47m, which is still enough to represent any possible distance on the Earth's surface. For instance, London to Sydney is about 17000000m.

Left to itself, Inform chooses the scaling for a unit so that it can represent exactly 1 of the smallest notation - so in our example Inform resolves down to 0.01m, not 1m, in order that it can represent 1cm accurately. But we can also fix the scaling ourselves:

```
A length is a kind of value. 1m specifies a length scaled at 10000.
```

Notice "scaled at", not "scaled down" or "scaled up" - this is now the first notation for length, so there's no existing notation which it could scale up or down. Anyway, now the range is 0.0001m, the width of a human hair, up to 214748.3647m, which is about 130 miles. (The Kinds index automatically keeps track of the range of values represented exactly.) The "scaled at" feature is meaningless if we're using real numbers, so it throws a Problem message.

Finally, for a really deluxe kind of value, we can also provide "equivalent" notations. The idea here is that we might want both miles and kilometers to work, even though they aren't direct scalings of each other. We can only do this approximately, but:

```
1 mile specifies a length equivalent to 1609m.
```

Equivalent notations are never normally used in printing values back (but see the next section) - we wouldn't want Inform to print a sequence of values such as "1.6km", "1.65km", "1.056 miles", ... in an effort to be helpful.


Start of Chapter 15: Numbers and Equations


Back to §15.9. Multiple notations


Onward to §15.11. Named notations

---

§15.11. Named notations

When it has a variety of notations to choose from, Inform will normally use the neatest one given the size of the value it is printing. Suppose we've set up "weight", with three notations:

```
A weight is a kind of value. 10kg specifies a weight.
1 tonne (singular) specifies a weight scaled up by 1000.
2 tonnes (plural) specifies a weight scaled up by 1000.
```

Inform will then print back values like so:

```
45kg -> "45kg"
1000kg -> "1 tonne"
2500kg -> "2.5 tonnes"
80000kg -> "80 tonnes"
```

Note the way Inform goes into decimal places in order to talk about 2500kg in terms of tonnes rather than kilograms - it is minimising the integer part of the unit, but trying to keep it non-zero. So Inform prefers "45kg" to "0.045 tonnes".

Although Inform's habit of choosing the best notation available is usually just what we want, we sometimes want to make the choice ourselves. For instance, if we were printing out a table of different weights, we might want to give all of them in kilograms, whatever their size. In that case we can, if we want, give names to our different notations:

```
1 tonne (singular, in tonnes) specifies a weight scaled up by 1000.
2 tonnes (plural, in tonnes) specifies a weight scaled up by 1000.
```

Now we could write, for instance:

```
"The weighbridge warns you not to exceed [the maximum load in tonnes]."
```

And the figure will always use tonnes now, even if Inform would normally think it odd: "The weighbridge warns you not to exceed 0.001 tonnes." But it will still correctly use "tonne" or "tonnes" as appropriate - what has changed is that instead of choosing from all of the weight notations, Inform now chooses from the notations labelled as "in tonnes".


Start of Chapter 15: Numbers and Equations


Back to §15.10. Scaling and equivalents


Onward to §15.12. Making the verb "to weigh"

---

§15.12. Making the verb "to weigh"

So now we can invent notations for weight. We could, for instance, write:

```
Weight is a kind of value. 1kg specifies a weight. Every thing has a weight.
```

And that allows us to write:

```
The lead pig is in the Salt Mine. The weight of the lead pig is 45kg.
```

But nobody would say it that way: they'd say "The lead pig weighs 45kg." So what we really need to complete our setup is a verb "to weigh".

We have already created new verbs, but none of those methods are quite convenient for this. We want to relate something tangible (the lead pig) to something intangible (45kg), and there's no convenient relation to express this; if we set it up as a condition, we'd get something we couldn't assert, only test. Instead, we'll do something different this time:

```
The verb to weigh means the weight property.
```

Previous definitions like this ended "means the ... relation", rather than "means the ... property", but the idea is the same. The meaning of "X weighs Y" is that the weight property of X is equal to Y. So we can now write:

```
A thing usually weighs 1kg. The lead pig weighs 45kg.
something weighing 20kg
if three things weigh 5kg, ...
```

And as we saw in the chapter on Descriptions, we can also set up adjectives, comparatives and superlatives:

```
Definition: A thing is heavy if its weight is 20kg or more.
```

which creates "heavy", "heavier" and "heaviest".


Start of Chapter 15: Numbers and Equations


Back to §15.11. Named notations


Onward to §15.13. The Metric Units extension


Example 258:  **Dimensions** This example draws together the previous snippets into a working implementation of the weighbridge.


Example 259:  **Lead Cuts Paper** To give every container a breaking strain, that is, a maximum weight of contents which it can bear - so that to put the lead pig into a paper bag invites disaster.

---

§15.13. The Metric Units extension

To sum all of this up, what started out as a simple business of setting a notation for lengths becomes something quite elaborate when we try to match the actual notations used by scientists and engineers. It's all optional, of course, but as we want more and more of this, we might find ourselves with a spread of notations like this:

```
1mm ... 1cm ... 1m ... 1km
```

In addition we might want equivalents for the inch, the yard and the mile; and verbal forms like the meter and the millimeter, and then alternate spellings like the kilometre; and then both singular and plural forms. And that's just length - what about density, area, pressure,

velocity and a dozen other physical quantities? After a while these declarations start to look as vastly fussy as a box of presentation cutlery.

Fortunately the whole set is indeed available in a presentation box, and at no extra charge.

(a) The built-in extension "Metric Units by Graham Nelson" sets up a whole range of scientific units, with all the notations we are likely to want. Real numbers are used throughout, so large and small-scale calculations can be carried out quite accurately. Like the other built-in extensions, it has its own documentation and examples.

(b) The built-in extension "Approximate Metric Units by Graham Nelson" does the same but using whole numbers, scaled about right for human situations. This won't be much use for extensive calculations, and won't be as accurate, but it will work reasonably well if real arithmetic isn't available.


Start of Chapter 15: Numbers and Equations


Back to §15.12. Making the verb "to weigh"


Onward to §15.14. Notations including more than one number

---

§15.14. Notations including more than one number

We've seen quite enough scientific notation for the time being. There are plenty of other notations used in natural language, for everyday concepts, where people don't use a tidy spread of powers of 10. Instead they use mixtures, with some sort of punctuation or text to divide them. For instance, the running time of a piece of music is easier to follow in minutes and seconds than in seconds alone: old-fashioned LP sleeves used to quote running times in the form 4'33.

```
A running time is a kind of value. 3'59 specifies a running time.
```

The choice of "3" here makes no difference, much as the choice of "10" in the weight examples was arbitrary. But the "59" is significant. Numbers after the first one are expected to range from 0 up to the value we quote - so in this case, the number of seconds can be anything from 0 to 59. Or, for instance:

```
A height is a kind of value. 5 foot 11 specifies a height.
```

A specification can contain up to eight numbers like this, but once again we might need to worry about the maximum value which can be stored. For instance, using the 3'59 notation, we can only go up to 546'07 (if we're using the Z-machine format setting) - a little over 9 hours, so the new Tori Amos album will not be a problem, but some of the more punishing German operas might break the bank.

In notations like this, only the first-appearing number part is allowed to be negative, and then only when declared with a minus sign:

```
A secret sign is a kind of value. -2x17 specifies a secret sign with parts mystery and enigma.
```

Here, the mystery can be negative, but not the enigma.

Notations must not contain double-quotation marks because, even though people did once use these to denote minutes of arc, they would simply confuse programs like Inform's user interface which have to keep track of what is quoted text and what is not. But other punctuation marks are fine *provided they occur between two digits*. For instance, in

```
A monetary value is a kind of value. $1.99 specifies a monetary value.
```

the full stop between the 1 and the 99 is not interpreted as a division of two sentences; and similarly for colons in examples such as

```
An aspect ratio is a kind of value. 16:9 specifies an aspect ratio.
```


Start of Chapter 15: Numbers and Equations


Back to §15.13. The Metric Units extension


Onward to §15.15. The parts of a number specification

---

§15.15. The parts of a number specification

We often need to break up a number specification into its pieces. For instance, suppose we want to know the dollars part of $1.99? We can do this by naming the parts:

```
A monetary value is a kind of value. $1.99 specifies a monetary value with parts dollars
and cents.
```

We can now find the relevant parts like so. Suppose that "sum" is a monetary value. Then:

```
dollars part of sum
cents part of sum
```

are both numbers, so for instance we can

```
say "Looks like around [dollars part of sum in words] dollar[s]."
```

We can also go the other way:

```
monetary value with dollars part 4 cents part 72
```

produces the monetary value $4.72. (Note the lack of commas or "and"s, and that the parts have to be given in the right order.) This is really intended to be useful when we manipulate such values in unusual ways:

```
An aspect ratio is a kind of value. 16:20 specifies an aspect ratio with parts width and
height.

To decide which aspect ratio is the wider version of (AR - an aspect ratio):
    let W be the width part of AR multiplied by 2;
```

```
let H be the height part of AR;
let the wider ratio be the aspect ratio with width part W height part H;
decide on the wider ratio.
```

Declaring the parts of a number specification individually also enables us to tack one or more options onto any of the parts:

```
A monetary value is a kind of value. $1.99 specifies a monetary value with parts dollars and cents (optional, preamble optional).
```

This declares that the "cents" part is optional - it will be 0 if not specified - and that if omitted, the non-numeric "preamble" before it should also be omitted. Thus "$3" is now valid and equivalent to "$3.00": indeed it will be the preferred form when Inform prints out a monetary value which is an exact number of dollars. If we had said that "cents" was optional, but not said that the preamble was optional, then "$3." would have been the form - which is less satisfactory.

There is only one other option: "without leading zeros", as in the following.

```
An aspect ratio is a kind of value. 16:20 specifies an aspect ratio with parts width and height (without leading zeros).
```

This ensures that when the ratio 4:3 is printed, it will be printed as "4:3" and not "4:03" as would otherwise happen.

 Start of Chapter 15: Numbers and Equations

 Back to §15.14. Notations including more than one number

 Onward to §15.16. Understanding specified numbers

 Example 260:  **Zqlran Era 8** Creating an alternative system of time for our game, using new units.

---

§15.16. Understanding specified numbers

It may be worth noting in passing that number specifications, like all other kinds of value, can be understood in typed commands. (See the chapter on Understanding for more on what can go in such square brackets.) For instance:

```
"America Stands Tall"

The Oval Office is a room. Josh and Toby are men in the Oval. A height is a kind of value. 5 foot 11 specifies a height. A person has a height. Josh is 5 foot 8. Toby is 5 foot 10.

Height guessing is an action applying to one thing and one height. Understand "guess [someone] is [height]" as height guessing.

Check height guessing: if the noun is not a person, say "You can only guess the height of people." instead. Carry out height guessing: if the height of the noun is the height understood, say "Spot on!"; if the height of the noun is greater than the height
```

```
understood, say "No, [the noun] is taller than that."; if the height of the noun is less than
the height understood, say "No, [the noun] is shorter than that."

Test me with "guess josh is 6 foot 3 / guess josh is 5 foot 9 / guess josh is 5 foot 3 /
guess josh is 5 foot 8".
```

Start of Chapter 15: Numbers and Equations

Back to §15.15. The parts of a number specification

Onward to §15.17. Totals

Example 261: ★★★ **Snip** A string which can be cut into arbitrary lengths, and then tied back together.

---

§15.17. Totals

This chapter began by mentioning arithmetic, and then went on a long diversion to create scientific units, everyday weights and measures, and other notational conveniences. Putting all of that together, it's time now to calculate something with all of these numerical quantities.

Suppose we invent the idea of weight, and give everything a weight of its own. Most items will have a nominal weight of 1kg, but people will be heavier. Going on actuarial tables, we might say:

```
A weight is a kind of value. 10kg specifies a weight. Everything has a weight. A thing
usually has weight 1kg. A man usually has weight 80kg. A woman usually has weight
67kg.

Definition: A thing is light if its weight is 3kg or less.
Definition: A thing is heavy if its weight is 10kg or more.
```

and this provides us with "lighter", "lightest", "heavier" and "heaviest" as before. Now we could say "if Peter is heavier than Paul", or even "if Peter is heavier than 75kg", and so forth. We need one more tool:

total (arithmetic values valued property) **of** (description of values) **... value**

This phrase produces the total of some property held by all of the values matching the description. A problem message is produced if the values in question can't have that property ("the total carrying capacity of scenes"), or if it holds a kind of value which can't meaningfully be added up ("the total description of open doors"). Example:

```
total carrying capacity of people in the Deep Pool
```

That gives us everything we need for a working balance platform:

```
The balance platform is a supporter in the Weighbridge. "The balance platform is
currently weighing [the list of things on the platform]. The scale alongside reads: [total
weight of things on the platform]."
```

Note that this only works because we said that "everything has a weight": otherwise it would make no sense to add up the weights of things.

This enables us to get the average weight of a group of things, too:

```
the total weight of things on the platform divided by the number of things on the platform
```

But we should be careful that this does not accidentally divide by zero, which it will if the platform has nothing on it! As well as the average, we could find the maximum and minimum weights:

```
the weight of the heaviest thing on the platform
the weight of the lightest thing on the platform
```

We should remember that "the heaviest thing on the platform" may be ambiguous, because there may be several equally heavy things there. That means

```
if the lead pig is the heaviest thing on the platform
```

will only reliably work if there is no possibility of a tie. A safer bet is:

```
if the lead pig is the weight of the heaviest thing on the platform
```


Start of Chapter 15: Numbers and Equations


Back to §15.16. Understanding specified numbers


Onward to §15.18. Equations


Example 262:  **Nickel and Dimed** A more intricate system of money, this time keeping track of the individual denominations of coins and bills, specifying what gets spent at each transaction, and calculating appropriate change.

---

§15.18. Equations

Forming totals is all very interesting in its way, but it's book-keeping rather than physics. As a glance at any school science textbook shows, the way to apply physics is to work out an unknown quantity - say, the time taken for a dropped ball to hit the ground - by combining known quantities into an equation - the height it is dropped from, and the strength of gravity.

It's a convention centuries old now that textbooks and research papers never describe these equations in running text. Even for simple formulae, we like to write "F=ma", not "let the force be the mass times the acceleration". And the standard way to print this is to break off and display an equation, not to squeeze it into the text as if it were ordinary verbiage. Just as Inform's Tables imitate those in printed books (see the next chapter), so its Equations do.

In this section, we'll use a combination of three equations to work out how soon and how hard an object pushed off a table will hit the floor. First, we'll include Metric Units, to define all of the kinds of value and notations we need.

```
Include Metric Units by Graham Nelson.
```

Now we'll give everything a mass (Metric Units likes to talk about mass instead of weight, but on Earth it's the same thing) and also set up a typical strength for gravity - it's a little less at the poles, a little more at the equator, but this is the conventional approximate value to use.

```
The acceleration due to gravity is an acceleration that varies. The acceleration due to gravity is usually 9.807 m/ss. A thing has a mass. The mass of a thing is usually 10g.
```

To a Renaissance scientist, typically living in a walled European town, a cannon ball was a familiar thing, and it often featured in imaginary experiments:

```
Laboratory is a room. The cannon ball is in the Laboratory. "A cannon ball perches delicately on a lab bench." The mass of the cannon ball is 2kg.
```

And now we're ready for the three equations. These will all have names, but we could just as easily have numbered them, calling them (say) "Equation 1", "Equation 2" and "Equation 3".

```
Equation - Newton's Second Law
   F=ma
where F is a force, m is a mass, a is an acceleration.

Equation - Principle of Conservation of Energy
   mgh = mv^2/2
where m is a mass, h is a length, v is a velocity, and g is the acceleration due to gravity.

Equation - Galilean Equation for a Falling Body
   v = gt
where g is the acceleration due to gravity, v is a velocity, and t is an elapsed time.
```

An equation has to take the form of one formula equals another, where each formula is made up from symbols defined afterwards. The symbols can be defined as definite values (as "g" is defined in the Galilean Equation), or just by telling Inform their kinds of value (as "v" and "t" are defined).

Equations are read using standard mathematical conventions. So "x + yz" means that we multiply y and z, then add that to x; "ab/cd" divides the product of a and b by the product of c and d. Multiplication signs can be omitted, just as science books normally do (though we can always write them if we want to, using the asterisk *, as usual in computing). The need for brackets is minimised, with any luck, but we can use them if we need to: "x(y+ab)" is legal, for instance.

One difference between Inform's conventions and mathematical ones, though, is that Inform generally ignores upper-versus-lower-case when reading variable names, so it wouldn't be a good idea to write "F = gMm/r^2" and expect "M" and "m" to be different from each other.

Here is the calculation:

```
Instead of pushing the cannon ball:
    let the falling body be the cannon ball;
    let m be the mass of the falling body;
    let h be 1.2m;
    let F be given by Newton's Second Law where a is the acceleration due to gravity;
    let v be given by the Principle of Conservation of Energy;
    let t be given by the Galilean Equation for a Falling Body;
    say "You push [the falling body] off the bench, at a height of [h], and, subject to a
downward force of [F], it falls. [t to the nearest 0.01s] later, this mass of [m] hits the floor
at [v].";
    now the falling body is in the location.
```

And the result is:

```
You push the cannon ball off the bench, at a height of 1.2m, and, subject to a downward
force of 19.614N, it falls. 0.49s later, this mass of 2.0kg hits the floor at 4.85147 m/s.
```

Not all that fast-moving - it's only about 10 mph, ten times slower than one fired by a Renaissance cannon - but half a second wouldn't give you long to get your foot out of the way.

How was that done? The crucial lines are the ones in the form "let X be given by E...", which is a new form of "let".

let (a name not so far used) **be given by** (equation name)

or:
let (a temporary named value) **be given by** (equation name)

This phrase creates a new temporary variable, starting it with the value found by solving the given equation. The variable lasts only for the present block of phrases, which certainly means that it lasts only for the current rule. Example:

```
    let F be given by Newton's Second Law where a is the acceleration due to
    gravity;
```

There is also a more compact syntax, giving the equation explicitly:

```
    let KE be given by KE = mv^2/2 where KE is an energy;
```

When we solve with "let", then, all of the other symbols should either already have values (because they exist as "let" values already made) or else be specified in the line. For instance,

```
let F be given by Newton's Second Law where a is the acceleration due to gravity;
```

is allowed because "F" is one of the symbols in "F = ma"; of the other two symbols, we have a "let" variable called "m" already - it's the mass of the cannon ball - and we declare exactly what "a" is.

The next calculation is more interesting:

```
let v be given by the Principle of Conservation of Energy;
```

Since the equation here is "mgh = mv^2/2", Inform has to do some algebra to work out "v" in terms of the other unknowns - it's the square root of 2gh, but we don't need to work that out. Inform can't always solve implicit equations - for instance, it can't deduce "m" from this equation - but it's correct on all the easy cases which occur in basic physics, and that enables us to write equations in their most natural form, which is easier to read and understand.

The advantage of setting out an equation formally is that it can be used in many places - we could use Newton's Second Law again for something quite different, for example. But it's a little cumbersome for something simple which we only need once, so this is neater:

```
let KE be given by KE = mv^2/2 where KE is an energy;
```

Here the equation is written out explicitly instead of being named, but otherwise everything works in the same way.

Equations can also contain many of our standard functions, which are written for this purpose with their standard mathematical abbreviations. For example:

```
let x be given by sin x = 1 where x is a real number;
```

works out x as pi divided by 4, which is to say, 90 degrees. The Phrasebook entries on the mathematical functions give their abbreviations, but here they all are as a list:

```
abs, root, ceiling, floor, int, log, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh,
tanh, arcsinh, arccosh, arctanh
```

As an example, here's the definition of arcsinh given in the Standard Rules:

```
To decide which real number is the hyperbolic arcsine of (R - a real number):
    let x be given by x = log(R + root(R^2 + 1)) where x is a real number;
    decide on x.
```

Start of Chapter 15: Numbers and Equations

Back to §15.17. Totals

Onward to §15.19. Arithmetic with units

Example 263: **Widget Enterprises** Allowing the player to set a price for a widget on sale, then determining the resulting sales based on consumer demand, and the resulting profit and loss.

---

§15.19. Arithmetic with units

The example equations in the previous section carried out quite a lot of arithmetic, but they may have given the impression that Inform always allows arithmetic - which is not true.

This is actually a good thing, because it keeps us from error. For instance, Inform will not allow:

```
Equation - Newton's Totally Bogus Law
    F = m^2
where F is a force, m is a mass.
```

because whatever you get when you square a mass, you don't get a force - in the same way that a length times another length makes an area, not another length. Physicists call this "dimensional analysis", and it often provides clues about which equations are right. Just after the Second World War, someone correctly worked out the explosive power of an atomic bomb without any classified information simply by guessing what values would appear in the formula, and then finding the simplest equation they could appear in.

In general, Inform will not allow numerical kinds of value to be multiplied or divided by each other (or square or cube rooted) unless we give it instructions that this would make sense.

Of course, there's plenty we can still do without any need for such instructions. For instance, going back to weight,

```
The Weighbridge is a room. "A sign declares that the maximum load is [100kg multiplied
by 3]."
```

...will produce the text "A sign declares that the maximum load is 300kg." Here Inform knows that it makes sense to multiply a weight by 3, and that the result will be a weight. Similarly, Inform allows us to add and subtract weights, and several different forms of division are allowed:

```
The blackboard is in the Weighbridge. "A blackboard propped against one wall reads:
'122 / 10 is [122 divided by 10] remainder [remainder after dividing 122 by 10]; but 122kg
/ 10kg is [122kg divided by 10kg] remainder [remainder after dividing 122kg by 10kg];
and 122kg / 10 is [122kg divided by 10] remainder [remainder after dividing 122kg by
10].'"
```

When we visit the Weighbridge, we find:

```
A blackboard propped against one wall reads: "122 / 10 is 12 remainder 2; but 122kg /
10kg is 12 remainder 2kg; and 122kg / 10 is 12kg remainder 2kg."
```

Whereas we are not allowed to divide 122 by 10kg: that would make no sense, since 122 is a number and not made up of kilograms. Inform will produce a problem message if we try. Similarly, Inform won't normally allow us to multiply two weights together - but see the next section.

Start of Chapter 15: Numbers and Equations

Back to §15.18. Equations

Onward to §15.20. Multiplication of units

Example 264: ★ **Frozen Assets** A treatment of money which keeps track of how much the player has on him, and a BUY command which lets him go shopping.

Example 265: ★★ **Money for Nothing** An OFFER price FOR command, allowing the player to bargain with a flexible seller.

Example 266: ★★★ **Lemonade** Containers for liquid which keep track of how much liquid they are holding and of what kind, and allow quantities to be moved from one container to another.

Example 267: ★★★ **Savannah** Using the liquid implementation demonstrated in Lemonade for putting out fires.

---

§15.20. Multiplication of units

To recap, then, it is forbidden to multiply 122kg and 10kg, not because it could never make sense (a scientist might occasionally multiply two weights) but because the result is - what? Not a number, and not a weight any more. But we are allowed to tell Inform what the result ought to be, and once we have done so, the multiplication will be allowed:

```
A length is a kind of value. 10m specifies a length. An area is a kind of value. 10 sq m specifies an area.

A length times a length specifies an area.

The balance platform is in the Weighbridge. "The balance platform is 10m by 8m, giving it an area of [10m multiplied by 8m]."
```

which will turn up as:

```
The balance platform is 10m by 8m, giving it an area of 80 sq m.
```

And having told Inform that lengths multiply to area, we could also divide an area by a length to get a length: no further instructions would be needed.

The built-in "Metric Units" extension includes all of the standard ways that physical quantities are multiplied, and a good way to see these is to try out one of the Metric Units examples and look at the Kinds index, which includes a table showing how all of this works.

Start of Chapter 15: Numbers and Equations

Back to §15.19. Arithmetic with units

Onward to Chapter 16: Tables: §16.1. Laying out tables

Example 268: ★ **Depth** Receptacles that calculate internal volume and the amount of room available, and cannot be overfilled.

Example 269: ★★ **Fabrication** A system of assembling clothing from a pattern and materials; both the pattern and the different fabrics have associated prices.

Example 270: ★★ **The Speed of Thought** Describing scientifically-measured objects in units more familiar to the casual audience.

Examples from Chapter 15: Numbers and Equations

Start of this chapter

Chapter 16: Tables

Indexes of the examples

253

★★★ **Example** **Alias**

RB

A telephone with phone numbers of the standard American seven-digit length.

Seven-digit telephone numbers are too long for Inform to handle when compiling to the Z-Machine, but they will work under Glulx. To have this example succeed, make sure that you have selected the Glulx option in your settings menu.

```
"Alias"

A telephone is a kind of thing. Understand "phone" or "telephone" as a
telephone.

A phone number is a kind of value. 999-9999 specifies a phone number.
```

Now we borrow some techniques from the Understanding chapter to set up dialing actions:

```
Understand "dial [phone number] on [telephone]" as dialing it on. Understand
"dial [phone number] on [something]" as dialing it on.

Understand the commands "phone" or "telephone" or "call" as "dial".

Understand "call [text]" or "phone [text]" or "dial [text]" or "telephone [text]" as a
mistake ("That's not a number you know.").

Dialing it on is an action applying to one phone number and one thing.

Report dialing it on:
    say "You dial [the phone number understood]."
```

This much is enough to let us dial telephone numbers and have Inform report that we've done so; it doesn't actually provide a telephone system such that we could reach and converse with other characters (but see the other telephone examples in the recipe book for more on how one might do that).

We'll set up a little political espionage scenario from which our player can make calls:

```
The Senator's Junior Suite is a room. "The Senator appears, unfortunately, to have very precise habits: little in the room has been moved from its usual place; the trash can is empty; the bed has been remade[if the blue paper is unexamined]. There may in fact be nothing to find here[end if]."

The bed is an enterable scenery supporter in the Junior Suite.

The player is wearing a housekeeping uniform and a brunette wig. The player carries a telephone called a Nokia.
```

Borrowing again from the chapter on Understanding, we might arrange things so that the player knows and can call a few standard numbers with such syntax as CALL HOME:

```
Understand "home" as 555-9200.
```

And what if we'd like to have the player learn some phone numbers during the game?

```
A thing can be examined or unexamined. Carry out examining something: now the noun is examined.

Understand "Stephen" as 555-2513 when the blue paper is examined.
```

This will understand CALL STEPHEN once the paper is examined; before that, the player will just get the "That's not a number you know" response that Inform uses for all attempts to call unknown names.

We'd better plant this paper for the player to find:

```
The blue paper is in the drawer. The description of the blue paper is "It reads: 'Call Stephen - 555-2513'."

The drawer is part of the dresser. It is closed and openable. The dresser is in The Senator's Junior Suite. The lamp is on the dresser. The description of the dresser is "The single drawer is [if the drawer is open]open[otherwise]shut[end if]."

Test me with "dial 555-9999 / call home on the telephone / phone the president / call stephen / open drawer / read paper / call stephen / put phone in drawer / close drawer / call stephen".
```

The player character's height is selected randomly at the start of play.

As with ordinary numbers, we can choose random units when this is useful:

```
"rBGH"

The Pharmaceutical Testing Facility is a room. "A [if the player is short]large [end
if][if the player is tall]cramped [end if]white space with sterile counters and a[if
the player is tall]n uncomfortable little[end if] stool. There is also a mirror, behind
which someone must be watching you. But you can't see through to that."

A counter, a one-way mirror, and a stool are scenery in the Facility. The stool is
an enterable supporter. The counter supports a plate.

Height is a kind of value. 5 feet 11 inches specifies a height. 5'11 specifies a
height. A person has a height.

Definition: a person is tall if its height is 6 feet 0 inches or more.

Definition: a person is short if its height is 5 feet 4 inches or less.

When play begins:
    now the height of the player is a random height between 5 feet 2 inches and 6
feet 4 inches;
    now the right hand status line is "[height of player]".

Instead of examining the player:
    say "You, Test Subject, are [height of the player] tall."

The growth pill is a kind of thing. A growth pill is always edible. The description is
usually "It is leaf-green and has a reassuring logo of a curling vine on the side.
Nothing to worry about, nothing at all." Two growth pills are on the plate.

After eating the growth pill:
    increase the height of the player by 0 feet 6 inches;
    say "Your spine does something frightening and painful, and you find yourself
looking down on the room from a wholly new angle.";
    try looking.

Test me with "examine me / eat pill / examine me / eat pill / examine me".
```

255

Example **Lethal Concentration 1**

RB

A poisonous gas that spreads from room to room, incapacitating or killing the player when it reaches sufficient levels.

```
"Lethal Concentration"

A concentration is a kind of value. 200.9ppm specifies concentration. 200.9 ppm
specifies concentration.
```

```
A room has a concentration called current concentration. A room has a
concentration called former concentration.

Probability inverse is a number that varies. [This is expressed as an inverse of
the actual probability of diffusion from one room to another, to avoid error.]
Probability inverse is 20. [That is, any given molecule of gas has a 5% chance of
leaving by a given doorway at any given minute. Probability inverse should never
drop below 10, the maximum number of exits from the room.]

Every turn:
    follow the diffusion rules.

The diffusion rules are a rulebook.

A diffusion rule (this is the gas movement rule):
    repeat with space running through rooms:
        let sum be 0.0 ppm;
        repeat with way running through directions:
            let second space be the room way from the space;
            if second space is a room:
                let difference be the former concentration of the second space minus
the former concentration of the space;
                increase sum by the difference;
        let sum be sum divided by probability inverse;
        now current concentration of the space is the former concentration of the
space plus the sum.
```

A technical note: it would be possible to write "repeat with space running through rooms... repeat with second space running through rooms adjacent to the space" instead, but in practice this loops through all the rooms * all the rooms again * all the directions (to determine adjacency). Phrasing the loop this way omits the second multiplier. For a map of 25 rooms, this means that the loop runs 25 times faster than it would otherwise, and of course for a larger map the effect would be even more dramatic.

```
A diffusion rule (this is the resetting concentration rule):
    repeat with space running through rooms:
        now the former concentration of the space is the current concentration of
the space.

The last diffusion rule (this is the lethal dosage rule):
    if the current concentration of the location is greater than LC50:
        say "The concentration in the air overpowers you...";
        end the story;
    otherwise:
        if the current concentration of the location is greater than TLV-STEL:
            say "You feel extremely uncomfortable in this environment."

Instead of doing something when the current concentration of the location is
greater than TLV-STEL:
    if going, continue the action;
    say "You can't work in this environment: your eyes and nose sting and it hurts
to breathe."
```

And, for testing purposes, a square grid of rooms:

```
Room 1A is west of Room 1B. Room 1B is west of Room 1C. Room 1C is west of Room 1D. Room 1D is west of Room 1E.

Room 2A is south of room 1A and west of Room 2B. Room 2B is west of Room 2C and south of Room 1B. Room 2C is west of Room 2D and south of Room 1C. Room 2D is west of Room 2E and south of Room 1D. Room 2E is south of Room 1E.

Room 3A is south of room 2A and west of Room 3B. Room 3B is west of Room 3C and south of Room 2B. Room 3C is west of Room 3D and south of Room 2C. Room 3D is west of Room 3E and south of Room 2D. Room 3E is south of Room 2E.

Room 4A is south of room 3A and west of Room 4B. Room 4B is west of Room 4C and south of Room 3B. Room 4C is west of Room 4D and south of Room 3C. Room 4D is west of Room 4E and south of Room 3D. Room 4E is south of Room 3E.

Room 5A is south of room 4A and west of Room 5B. Room 5B is west of Room 5C and south of Room 4B. Room 5C is west of Room 5D and south of Room 4C. Room 5D is west of Room 5E and south of Room 4D. Room 5E is south of Room 4E.

The former concentration of room 3C is 800.0 ppm.
```

For variety of testing, here is another room set-up, this time with some corridors and walls within; uncommenting it, and commenting out the connected grid, will let us explore what would happen in alternative cases, to prove to ourselves that the model works consistently.

```
[Room 1A is west of Room 1B. Room 1B is west of Room 1C. Room 1C is west of Room 1D. Room 1D is west of Room 1E.

Room 2A is west of Room 2B. Room 2B is west of Room 2C. Room 2C is west of Room 2D. Room 2D is west of Room 2E. Room 2E is south of Room 1E.

Room 3A is south of room 2A and west of Room 3B. Room 3B is west of Room 3C. Room 3C is west of Room 3D. Room 3D is west of Room 3E.

Room 4A is west of Room 4B. Room 4B is west of Room 4C. Room 4C is west of Room 4D. Room 4D is west of Room 4E. Room 4E is south of Room 3E.

Room 5A is south of room 4A and west of Room 5B. Room 5B is west of Room 5C and south of Room 4B. Room 5C is west of Room 5D and south of Room 4C. Room 5D is west of Room 5E and south of Room 4D. Room 5E is south of Room 4E.]
```

For the sake at least of seeing what's going on in the example, let's also provide the player with the means to view the gas diffusion graphically:

```
The status grid is a device carried by the player. The status grid is switched on.
```

```
Every turn:
    try examining the grid.

Instead of examining the status grid:
    say "[fixed letter spacing][bar][line break]";
    say "|[state of room 1A]|[state of room 1B]|[state of room 1C]|[state of room
1D]|[state of room 1E]|[line break]";
    say "[bar][line break]";
    say "|[state of room 2A]|[state of room 2B]|[state of room 2C]|[state of room
2D]|[state of room 2E]|[line break]";
    say "[bar][line break]";
    say "|[state of room 3A]|[state of room 3B]|[state of room 3C]|[state of room
3D]|[state of room 3E]|[line break]";
    say "[bar][line break]";
    say "|[state of room 4A]|[state of room 4B]|[state of room 4C]|[state of room
4D]|[state of room 4E]|[line break]";
    say "[bar][line break]";
    say "|[state of room 5A]|[state of room 5B]|[state of room 5C]|[state of room
5D]|[state of room 5E]|[line break]";
    say "[bar][variable letter spacing][line break]".

To say bar:
    say "-----------------------------------------------".

TLV is a concentration that varies. TLV is 30.0ppm. [Long-term exposure
maximum, safe for 8 hours a day.]

TLV-STEL is a concentration that varies. TLV-STEL is 50.0ppm. [Short-term
exposure maximum, safe for fifteen minutes max.]

TLV-C is a concentration that varies. TLV-C is 150.0ppm. [Absolute exposure
ceiling.]

LC50 is a concentration that varies. LC50 is 300.0ppm. [Concentration at which
50 percent of test subjects die of exposure, usually expressed in terms of time
and body weight; in our LC50 these are factored in for the player's weight for
one minute.]
```

The values set for these would depend on the type of poisonous gas in question; we'd want to adjust appropriately.

```
To say state of (space - a room):
    if the space is the location, say bold type;
    if current concentration of space is less than 10.0ppm, say " ";
    if current concentration of space is less than 100.0ppm, say " ";
    say current concentration of space;
    say roman type.
```

Now, in theory we might also want to account for sources and sinks, items that either inject poisonous gas into the environment or remove it again. For simplicity, we will assume that these contributions can also be calculated in ppm and that the total number of inert and poisonous gas molecules in a room never changes (so if poison gas molecules are added, an equal number of inert molecules are removed). If room pressure were able to change, our model would have to be improved, so let us

assume for now that that never happens. We want this sink/source business to calculate before any other portion of the diffusion rulebook, so set it as a first diffusion rule.

```
A gas source is a kind of thing. A gas source has a concentration called the
contribution. The contribution of a gas source is usually 30.0ppm.

Room 2B contains a gas source called a spigot. The contribution of the spigot is
50.0ppm. Room 5A contains a gas source.

A gas sink is a kind of thing. A gas sink has a concentration called the
contribution. The contribution of a gas sink is usually 30.0ppm.

Room 5E contains a gas sink called a fan. The contribution of the fan is
80.0ppm.

The first diffusion rule (this is the sources and sinks rule):
   follow the sources rule;
   follow the sinks rule.

This is the sinks rule:
   repeat with item running through gas sinks:
      let space be the location of the item;
      decrease the former concentration of the space by the contribution of the
item;
      if the former concentration of the space is less than 0.0ppm, now the former
concentration of the space is 0.0ppm.

This is the sources rule:
   repeat with second item running through gas sources:
      let space be the location of the second item;
      increase the former concentration of the space by the contribution of the
second item;
      if the former concentration of the space is less than 0.0ppm, now the former
concentration of the space is 0.0ppm.

Test me with "z / z / z / z / z / z / z / z".
```

---

256  **Example Wonderland** RB

Hiking Mount Rainier, with attention to which locations are higher and which lower than the present location.

Suppose we have a landscape with a great deal of up and down variation, where GO UP and GO DOWN will be significant almost everywhere, and specifying them all individually a tremendous pain:

```
"Wonderland"

An altitude is a kind of value. 1000 feet specifies an altitude. A room has an
altitude.
```

```
Definition: a room is low if its altitude is 3000 feet or less. Definition: a room is
high if its altitude is 5000 feet or more.

Instead of going down:
    if an adjacent room is lower than the location:
        let the valley be the lowest adjacent room;
        let the way be the best route from the location to the valley;
        say "(that is, [way])[paragraph break]";
        try going the way;
    otherwise:
        say "You're in a local valley: there's no down from here."

Instead of going up:
    if an adjacent room is higher than the location:
        let the peak be the highest adjacent room;
        let the way be the best route from the location to the peak;
        say "(that is, [way])[paragraph break]";
        try going the way;
    otherwise:
        say "You're on a local peak."

Paradise is a room. Paradise has altitude 5400 feet. "A handsome parking lot, a
picnic ground, and the Henry M. Jackson Memorial Visitor Center. The latter
offers, for serious climbers, a hot shower; for nature enthusiasts, an interpretive
museum; and for car-trippers, a gift shop selling canned slugs. All of which is a
largely unsuccessful distraction from the peak of Mt. Rainier beyond."

Cougar Rock is southwest of Paradise. The altitude of Cougar Rock is 3180 feet.
"Numerous individual campsites and (on the road inventively labeled 'F') a
handful of larger campgrounds suitable for church groups and family reunions."

Longmire is southwest of Cougar Rock. It has altitude 2760 feet. "A tiny town: it
has to offer a few groceries, a post office, and a lodge for people who do not
care to camp, all built in a rustic Park Service way."

Panorama Point is north of Paradise. It has altitude 6800 feet. Camp Muir is
north of Panorama Point. It has altitude 10188 feet. Columbia Crest is northwest
of Camp Muir. It has altitude 14410 feet. St Andrews Rock is west of Columbia
Crest. It has altitude 10992 feet. Camp Schuman is northeast of Columbia Crest.
It has altitude 9510 feet.
```

Since Mount Rainier National Park runs to over 235,000 acres, we will omit the rest of the locations, but it does seem fair to give a little more credit to anyone who makes the summit:

```
Instead of going up in the highest room:
    say "You're standing at the summit of Mt. Rainier, the highest point in the state
of Washington. There is no up."

Test me with "up / up / up / down / down / up / up".
```

Poisonous gas again, only this time it sinks.

This is a slight variation on the previous gas diffusion example: the main difference is that gas preferentially moves towards lower rooms, and will gradually settle in the bottom floor. We do this by calculating the probability of movement separately for each pair of rooms.

```
"Lethal Concentration"

A concentration is a kind of value. 200.9ppm specifies concentration. 200.9 ppm
specifies concentration.

A room has a concentration called current concentration. A room has a
concentration called former concentration.

To decide what number is the probability inverse between (space - a room) and
(second space - a room):
    let guess be 20;
    let way be the best route from space to second space;
    if way is up, let guess be 50;
    if way is down, let guess be 10;
    if the guess is less than 10, decide on 10;
    decide on guess.
```

If we wanted, we could introduce other concerns into the calculation here: open and closed doors, windows between rooms, rooms that are outdoors vs. those that are indoors, and so on. The possibilities are numerous, so we will stick with the simple principle that our poison gas sinks.

```
Every turn:
    follow the diffusion rules.

The diffusion rules are a rulebook.

A diffusion rule (this is the gas movement rule):
    repeat with space running through rooms:
        let sum be 0.0 ppm;
        repeat with way running through directions:
            let second space be the room way from the space;
            if second space is a room:
                let incoming be the former concentration of the second space divided
by the probability inverse between second space and space;
                let outgoing be the former concentration of the space divided by the
probability inverse between space and second space;
                let difference be incoming minus outgoing;
                increase sum by the difference;
        now current concentration of the space is the former concentration of the
space plus the sum.

A diffusion rule (this is the resetting concentration rule):
    repeat with space running through rooms:
        now the former concentration of the space is the current concentration of
the space.
```

```
The last diffusion rule (this is the lethal dosage rule):
    if the current concentration of the location is greater than LC50:
        say "The concentration in the air overpowers you...";
        end the story;
    otherwise:
        if the current concentration of the location is greater than TLV-STEL:
            say "You feel extremely uncomfortable in this environment."

Instead of doing something when the current concentration of the location is
greater than TLV-STEL:
    if going, continue the action;
    say "You can't work in this environment: your eyes and nose sting and it hurts
to breathe."

Room 1A is west of Room 1B. Room 1B is west of Room 1C. Room 1C is west
of Room 1D. Room 1D is west of Room 1E.

Room 2A is west of Room 2B and below room 1A. Room 2B is west of Room 2C
and below Room 1B. Room 2C is west of Room 2D and below Room 1C. Room
2D is west of Room 2E and below Room 1D. Room 2E is south of Room 1E and
below Room 1E.

The former concentration of Room 1C is 800.0 ppm.

The status grid is a device carried by the player. The status grid is switched on.
```

And just for fun, this time we'll make the grid prettier, too; but this will work only on the Z-machine setting, not Glulx.

```
Every turn:
    try examining the grid.

    Instead of examining the status grid:
    say "[unicode box drawings light down and right][top bar][unicode box
drawings light down and left][line break]";
    say "[unicode box drawings light vertical]";
    say "[state of room 1A][state of room 1B][state of room 1C][state of room 1D]
[state of room 1E] upstairs[line break]";
    say "[unicode box drawings light vertical and right][middle bar][unicode box
drawings light vertical and left][line break]";
    say "[unicode box drawings light vertical]";
    say "[state of room 2A][state of room 2B][state of room 2C][state of room 2D]
[state of room 2E] downstairs[line break]";
    say "[unicode box drawings light up and right][bottom bar][unicode box
drawings light up and left][variable letter spacing][line break]"

Include Unicode Character Names by Graham Nelson.

To say top bar:
    repeat with N running from 1 to 9:
        if the remainder after dividing N by 2 is 0, say "[unicode box drawings light
down and horizontal]";
        otherwise say "[unicode box drawings light horizontal]".
```

```
To say middle bar:
    repeat with N running from 1 to 9:
        if the remainder after dividing N by 2 is 0, say "[unicode box drawings light
vertical and horizontal]";
        otherwise say "[unicode box drawings light triple dash horizontal]".

To say bottom bar:
    repeat with N running from 1 to 9:
        if the remainder after dividing N by 2 is 0, say "[unicode box drawings light
up and horizontal]";
        otherwise say "[unicode box drawings light horizontal]".

TLV is a concentration that varies. TLV is 30.0ppm. [Long-term exposure
maximum, safe for 8 hours a day.]

TLV-STEL is a concentration that varies. TLV-STEL is 50.0ppm. [Short-term
exposure maximum, safe for fifteen minutes max.]

TLV-C is a concentration that varies. TLV-C is 150.0ppm. [Absolute exposure
ceiling.]

LC50 is a concentration that varies. LC50 is 300.0ppm. [Concentration at which
50 percent of test subjects die of exposure, usually expressed in terms of time
and body weight; in our LC50 these are factored in for the player's weight for
one minute.]

Include Basic Screen Effects by Emily Short.

To say state of (space - a room):
    if the current concentration of space is less than TLV, say blue letters;
    if the current concentration of space is TLV, say blue letters;
    if the current concentration of space is greater than TLV, say green letters;
    if the current concentration of space is greater than TLV-STEL, say yellow
letters;
    if the current concentration of space is greater than TLV-C, say red letters;
    say "[unicode square with diagonal crosshatch fill]";
    say default letters;
    say "[unicode box drawings light vertical]".

Test me with "z / z / z / z / z / z / z / z".
```

---

258  **Example Dimensions** RB

This example draws together the previous snippets into a working implementation of the weighbridge.

The following is not a very sophisticated approach, because it does not allow for weight to accumulate: if we put a gold ingot into a paper bag, then put the bag on the balance platform, only the bag's weight will register. But it will do for a first try.

```
"Dimensions"
```

A length is a kind of value. 10m specifies a length. An area is a kind of value. 10 sq m specifies an area. A length times a length specifies an area.

A weight is a kind of value. 10kg specifies a weight. Everything has a weight.

The verb to weigh means the weight property. A thing usually weighs 1kg.

Definition: A thing is light if its weight is 3kg or less.

Definition: A thing is heavy if its weight is 10kg or more.

The Weighbridge is a room.

A blackboard is in the Weighbridge. "A blackboard propped against one wall reads: '122/10 is [122 divided by 10] remainder [remainder after dividing 122 by 10]; 122kg/10kg is [122kg divided by 10kg] remainder [remainder after dividing 122kg by 10kg]; 122kg/10 is [122kg divided by 10] remainder [remainder after dividing 122kg by 10].'" The blackboard weighs 10kg.

A feather and a lead pig are in the Weighbridge. The lead pig weighs 45kg.

The balance platform is a supporter in the Weighbridge. "The balance platform is 10m by 8m, giving it an area of [10m multiplied by 8m], currently weighing [the list of things on the platform]. The scale alongside reads: [total weight of things on the platform]. [if two things are on the platform]Average weight is: [the total weight of things on the platform divided by the number of things on the platform]. Heaviest item is [the heaviest thing on the platform], at [weight of the heaviest thing on the platform]. Lightest item is [the lightest thing on the platform], at [weight of the lightest thing on the platform].[otherwise]It seems to be able to weigh several things at once."

Test me with "get feather / put it on platform / look / get pig / put it on platform / look".

259

Example **Lead Cuts Paper**

RB

To give every container a breaking strain, that is, a maximum weight of contents which it can bear - so that to put the lead pig into a paper bag invites disaster.

The following shows the kind of "realism" rules which could be introduced using weights. Not entirely realistic: we do not bother to rupture containers out of the player's sight.

"Lead Cuts Paper"

A weight is a kind of value. 10kg specifies a weight. Everything has a weight. A thing usually has weight 1kg.

A container has a weight called breaking strain. The breaking strain of a container is usually 50kg. Definition: A container is bursting if the total weight of

things in it is greater than its breaking strain.

A lead pig, a feather, a silver coin and a paper bag are in a room called the Metallurgy Workshop. The paper bag is a container with breaking strain 2kg. The lead pig has weight 50kg.

Every turn when a container (called the sack) held by someone visible (called the chump) is bursting:
    say "[The sack] splits and breaks under the weight! [if the player is the chump]You discard[otherwise][The chump] discards[end if] its ruined remains, looking miserably down at [the list of things in the sack] on the floor.";
    now all of the things in the sack are in the location;
    now the sack is nowhere.

Test me with "get bag / get feather / put feather in bag / get pig / put pig in bag / look".

---

260  **Example Zqlran Era 8** RB

Creating an alternative system of time for our game, using new units.

Suppose that our game takes place on an alien planet that does not follow Earth time. On this planet, we want to track time with different units. We also want time to advance in those units, and we want to be able to set a schedule of timed events.

"Zqlran Era 8"

The Barren Lavender Surface of Zql is a room. "It is late twilight on Zql. Overhead, two crescent moons, both green, mark the sluggish passage of time. A cold wind is blowing over the pale purplish ground cover, but it does not penetrate your airtight suit."

A Zqlran date is a kind of value. 14-88 specifies a Zqlran date with parts zqls and frbs. Current zqlran date is a zqlran date that varies. The current zqlran date is 8-22. Previous zqlran date is a zqlran date that varies. The previous zqlran date is 8-20.

When play begins:
    now left hand status line is "[current zqlran date], or [current zqlran date in words]".

To say (Zqlra - a Zqlran date) in words:
    say "[zqls part of Zqlra] Z, [frbs part of Zqlra] f."

Inform automatically supplies a way to say a new unit, which will look similar to the format in which we defined that unit in the first place. But we can (as shown here) create our own alternative say phrases to express the units in other ways as well.

Next, we need to meddle with time advancement so that time is tracked in Zqlran date rather than in minutes. This requires borrowing a trick from a later chapter, to

replace Inform's built-in time handling with an alternative time handling rule of our own:

```
The Zqlran time rule is listed instead of the advance time rule in the turn
sequence rules.

This is the Zqlran time rule:
    increment turn count;
    now the previous zqlran date is current zqlran date;
    increase the current zqlran date by 0-02;
    repeat through the Table of Zql Schedule:
        if era entry is greater than previous zqlran date and era entry is not greater
than current zqlran date:
            say event entry;
            say paragraph break;
            blank out the whole row.

Table of Zql Schedule

    era   event
    8-24 "A wisp-thin cloud blows rapidly across the face of Nepenthe, the lesser of the two green moons."
    8-28 "The cloud across Nepenthe clears."
```

Note that we could if we wished use a different device for scheduling events: this one simply prints text at scheduled eras, but we might also (for instance) make the event entry be a rule for Inform to follow, and tell Inform to carry out that rule at the scheduled time.

---

261  **Example Snip** *RB*

A string which can be cut into arbitrary lengths, and then tied back together.

```
"Snip!"

Length is a kind of value. 30 inch specifies a length. 20 in specifies a length. 50
inches specifies a length.

A string is a kind of thing. A string has a length. The length of a string is usually
36 inches.

Before printing the name of a string, say "[length] piece of ". Rule for printing the
plural name of a string: say "[length] pieces of string".

Understand the command "cut" as something new. Understand "cut [length]
from/off [something]" as trimming it by (with nouns reversed). Understand "cut
[something] by [length]" as trimming it by. Understand the command "trim" as
"cut".

Trimming it by is an action applying to one thing and one length.
```

```
Check trimming it by:
    if the length understood is 0 inches, say "You're approaching Zeno's string at
this point." instead;
    if the length understood is greater than the length of the noun, say "[The noun]
is only [length of the noun] long to start with." instead;
    if the length understood is the length of the noun, say "[The noun] is already
exactly [length of the noun] long." instead.

Carry out trimming it by:
    now the length of the noun is the length of the noun minus the length
understood;
    let the other half be a random string in the string repository;
    now the length of the other half is the length understood;
    move the other half to the player.

Report trimming it by:
    reset string lengths; [we will define this in a moment; it helps guarantee that
our descriptions of the strings are correct when we write the output list]
    say "You now have [a list of strings carried by the player]."

Understand "cut [something] in half" as halving. Halving is an action applying to
one thing.

Carry out halving:
    let half measure be the length of the noun divided by 2;
    now the length understood is half measure;
    try trimming the noun by half measure.
```

This fudges slightly, since an odd-length string will be divided into uneven halves. Keeping track of fractional inches would complicate matters, though, so let's assume for now that this doesn't matter.

```
The player carries a string.

The Scissors Room is a room.

The string repository contains 35 strings.
```

Since our initial string is 36 inches long and it is impossible for the player to divide it into pieces smaller than an inch each, we need a total of 36 items to represent all the string-bits: one that the player carries at the outset, and 35 others. We should bear in mind that it is usually a good idea to use the smallest number of spare objects we can get away with: writing a game that required 1000 strings in the string repository would place silly demands on the resources of the system, so it's best to avoid that sort of thing if possible.

Now with a bit of fiddling we can also teach Inform to recognize descriptors such as "the shortest string":

```
Ordinariness is a kind of value. The ordinarinesses are longest, medium,
shortest. A string has an ordinariness. Understand the ordinariness property as
referring to a string.
```

```
Definition: a string is small if its length is 2 in or less. Definition: a string is large if
its length is 20 in or more.

Before reading a command:
    reset string lengths.

To reset string lengths:
    let upper measure be the length of the largest visible string;
    let lower measure be the length of the smallest visible string;
    repeat with item running through strings:
        now the ordinariness of the item is medium;
        if the length of the item is the upper measure, now the item is longest;
        if the length of the item is the lower measure, now the item is shortest.

After reading a command:
    if the player's command includes "shorter", replace the matched text with
"shortest";
    if the player's command includes "longer", replace the matched text with
"longest".

Instead of tying a string to a string:
    move the second noun to the string repository;
    now the length of the noun is the length of the noun plus the length of the
second noun;
    decrease the length of the noun by 1 inch;
    say "You end up with [a noun], as some is taken up by the knot."
```

This is still a little incomplete because we cannot refer to strings by their lengths, as in "the 2 inch string" and so on. To do this, we borrow a line from the chapter on Understanding:

```
Understand the length property as referring to a string.

Test me with "trim string by 4 in / cut longer string in half / cut longest string in
half / cut shortest string in half / g / g / tie longest string to shortest string / tie
longest string to medium string / i / x 16 inch string / drop 8 inch string / i".
```

---

262 ★★★ **Example** **Nickel and Dimed** *RB*

A more intricate system of money, this time keeping track of the individual denominations of coins and bills, specifying what gets spent at each transaction, and calculating appropriate change.

Typically games which keep track of the player's wealth need only do so as an abstract number, but occasionally it becomes useful to represent money as physical coins and bills. Here is an example that does exactly that:

```
"Nickel and Dimed"

Section 1 - Currency
```

```
Price is a kind of value. $10.99 specifies a price with parts dollars and cents. A
thing has a price. The price of a thing is usually $0.00.

Money is a kind of thing. Coin is a kind of money.

A dollar bill is a kind of money. The price of a dollar bill is $1.00. The printed
name of a dollar bill is "dollar bill". Rule for printing the plural name of a dollar
bill: say "dollar bills". The description of a dollar bill is "It has George
Washington's head on the front, which with a bit of creative folding can be
scrunched to look like a mushroom. All important things really are learned in
kindergarten."

A five-dollar bill is a kind of money. The price of a five-dollar bill is $5.00.
Understand "five" or "five dollar" as the five-dollar bill. The description of a five-
dollar bill is "Abraham Lincoln. He looks slightly less dignified here than he does
on the penny."

A hundred-dollar bill is a kind of money. The price of the hundred-dollar bill is
$100.00. Understand "hundred" or "hundred dollar" as the hundred-dollar bill.
Understand "dollar" as the dollar bill. The description of a hundred-dollar bill is
"It's got Benjamin Franklin, who always gets shafted: a denomination too large
for anyone to carry conveniently, and a lot of local fame in Philadelphia."
```

Our choice of understand rules guarantees that "five dollar" will be understood as the five, but "dollar" alone as the single. We will learn more about "understand" in later chapters, but here is a test to check the functionality:

```
Test bills with "x hundred dollar bill / x five dollar bill / x hundred / x five / x dollar /
x dollar bill".

A quarter is a kind of coin. The price of a quarter is $0.25. The description of a
quarter is "One of the old-fashioned variety, not a state quarter."

A dime is a kind of coin. The price of a dime is $0.10. The description of a dime
is "Franklin Roosevelt, trying not to look too annoyed that his coin is so small
and thin."

A nickel is a kind of coin. The price of a nickel is $0.05. The description of a
nickel is "A chubby coin, but you've always liked Thomas Jefferson, and the
Monticello on the back is a nice touch."

A penny is a kind of coin. The price of a penny is $0.01. The description of the
penny is "A profile of Abe Lincoln. Sometime soon they'll stop minting these,
you're sure of it."

Section 2 - Ownership

Ownership relates one person (called the owner) to various things. The verb to
own means the ownership relation.

Definition: a thing is owned if the player owns it.

Instead of buying something which is owned by the player:
    say "You already own [the noun]."
```

```
Instead of going somewhere when the player encloses something (called the
stolen goods) which is not owned by the player:
    if the owner of the stolen goods is not a person:
        now the player owns the stolen goods;
        continue the action;
    if the owner of the stolen goods can see the player,
        say "'Hey there buddy, not so fast,' says [the owner of the stolen goods].
'You going to buy [the stolen goods] first, or am I gonna call the cops?'";
    otherwise continue the action.

After taking inventory:
    say "Altogether, you've got [the player's cash] on your person."

To decide what price is the player's cash:
    let sum be the total price of money enclosed by the player;
    decide on sum.

To decide what price is the sum in (item - a container):
    let sum be the total price of the money in the item;
    decide on sum.

When play begins: now every thing carried by the player is owned by the player.

Section 3 - Purchasing and Sales

Definition: a thing is worthless if the price of it is $0.00. Definition: a thing is
valuable if it is not worthless.

A thing can be for sale.

Rule for printing room description details of something (called target) which is for
sale (this is the disclose prices in room description rule): say " ([price of the
target])".

Before listing contents: group money together giving articles.

Instead of examining a for sale thing (this is the describe things by price rule):
    say "[The noun] costs [the price of the noun], payable to [the owner of the
noun]."
```

The cashbox is a theoretical construct, not something the player will ever encounter in the course of the game. It contains all the money that is available for non-player characters to use in making change. If we wanted, we could give each character his own stash of change, but this would increase the likelihood that any given person would run out of cash to make change with. (And in this example there is only one vendor anyway.)

```
The cashbox is a container. The cashbox contains 10 pennies. The cashbox
contains 10 nickels. The cashbox contains 10 dimes. The cashbox contains 10
quarters. The cashbox contains 10 dollar bills. The cashbox contains 10 five-
dollar bills.

The block buying rule is not listed in the check buying rules.
```

```
Check buying something:
    if the noun is not for sale, say "[The owner of the noun] does not want to sell
you [the noun]." instead;
    if the player's cash is less than the price of the noun, say "You can't afford the
asking price of [the price of the noun] for [the noun]." instead.

Carry out buying something:
    let sum paid be $0.00;
    while sum paid is less than the price of the noun:
        let current target be the price of the noun minus the sum paid;
        let bill offered be the best money from the player for the current target;
        if the bill offered is money:
            move the bill offered to the owner of the noun;
            now the bill offered is spent;
            increase the sum paid by the price of the bill offered;
            let current target be the price of the noun minus the sum paid;
    say "You hand [the owner of the noun] [a list of spent money]. [run paragraph
on]";
    let change be $0.00;
    if the sum paid is greater than the price of the noun:
        now the change is the sum paid minus the price of the noun;
    if change is greater than the sum in the cashbox:
        now the player carries every spent money;
        now every spent thing is fresh;
        say "'Whoa,' says [the owner of the noun], handing the cash back to you. 'I
can't make change for that, man, sorry.'" instead;
    now every spent thing is in the cashbox;
    now every spent thing is fresh;
    while change is greater than $0.00:
        let change bill be the best money from the cashbox for change;
        decrease change by the price of the change bill;
        now change bill is spent;
        move change bill to player;
    if money is spent, say "[The owner of the noun] makes change with [a list of
spent money]. [run paragraph on]";
    now every spent thing is fresh;
    if the noun is not enclosed by the player and the owner of the noun can touch
the noun:
        say "'Here ya go,' says [the owner of the noun], handing [the noun] to you.
[run paragraph on]";
        move the noun to the player;
    now the player owns the noun.

Money can be spent or fresh.

Report buying something:
    if the player owns the noun,
        say "Your transaction is now complete, leaving you with [the player's cash]."
```

We've skipped over defining what makes a denomination the best for a given transaction, so we'd better do that now. Our goal is to avoid ever having the player gratuitously overpay -- he should always offer the smallest amount of money that will meet the price of what he's buying.

We also assume that all money "enclosed by the buyer" -- that is, somewhere in the buyer's possession -- is available for use. This might not be true in a game where the

player could pick up, say, a sealed lucite container with a ten-dollar bill inside; in that case we would have to define our terms more rigorously, perhaps by requiring that the bills be both enclosed and touchable by the buyer. The touchability check adds an extra layer of calculation, however, and since it is not necessary in this example (and probably not in most other cases either), we'll leave it out:

```
Definition: money is costly if its price is $2.50 or more. Definition: money is
cheap if its price is $0.99 or less.

Functional relation is a kind of value. The functional relations are overpayment,
underpayment and irrelevant. Money has a functional relation.

To decide what money is the best money from (buyer - a thing) for (cost - a
price):
    repeat with bill offered running through money:
        if the bill offered is enclosed by the buyer:
            if the price of the bill offered is the cost, decide on the bill offered;
            if the price of the bill offered is greater than the cost, now the functional
relation of bill offered is overpayment;
            otherwise now the functional relation of the bill offered is underpayment;
        otherwise:
            now the functional relation of the bill offered is irrelevant;
    [say "underpayment: [a list of underpayment money]
    overpayment: [a list of overpayment money]";]
    if the total price of underpayment money is less than the cost:
        decide on the cheapest money which is overpayment;
    otherwise:
        decide on the costliest money which is underpayment.
```

Notice the "say underpayment/overpayment section..." noted out, above. This is for debugging purposes: when writing complex code, it is sometimes useful to put in lines that will say explicitly what is going on. We can enclose them in brackets and Inform will ignore them as though they were comments; if we run into any problems with the code later, we can erase the brackets and see the diagnostic printed to the screen as we play.

```
Instead of giving money to someone:
    say "Best to keep the transaction simple by buying whatever you want."

Section 4 - The Scenario

The player carries 2 dollar bills. The player carries a nickel. The player carries 2
pennies. The player carries a five-dollar bill. The player carries 1 hundred-dollar
bill.

The Subway Station is a room.

The Bitterly Cold Street is north of the Subway Station. "Even though there is no
actual snow or ice, the street is about as cold as you can stand, for which reason
walking the twenty blocks uptown is not an acceptable option." The Bitterly Cold
Street contains a dollar bill.

The newspaper man is a man in the Subway Station. "A newspaper man in a
knit cap and fingerless gloves is hopping up and down behind his stand[if the
```

```
turn count is 1]. Cold weather, caffeine overdose, or mental illness? You may
never know. Welcome to New York[end if]." The description is "Eye contact with
strangers is something to avoid around here."

The stand is a supporter in the Station. The stand is scenery.

A copy of the New York Times is on the stand. The price of the New York Times
is $1.25.

A pack of gum is on the stand. The price of the gum is $0.40.

A paperback novel is on the stand. The price of the paperback novel is $7.99.

A packet of trading cards is on the stand. The price of the packet is $0.99.

When play begins:
    now every thing on the stand is owned by the newspaper man;
    now every thing on the stand is for sale.
```

We could have done all that by hand, but the initialization requires a little less work.

```
The ticket machine is a container in the Subway Station. It is fixed in place. The
description of the ticket machine is "An LED screen on the front instructs you to
insert [remaining ticket total] to complete your purchase. You also notice that the
NO CHANGE light is lit up." The light is part of the ticket machine. The printed
name of the light is "no change light". Understand "no change" or "no change
light" as the light.

The description of the light is "In the whole of your recollection, the ticket
machine has actually had change a total of twice. Usually, as now, the no-
change light gleams angrily, daring you to put in more than you owe." A cash
return button is part of the ticket machine. Instead of pushing the cash return
button: say "The ticket machine regurgitates [the list of things in the ticket
machine]."; now every thing in the ticket machine is carried by the player.
Instead of taking something which is in the ticket machine: say "The ticket
machine has swallowed your money, but it can be retrieved (you hope) with the
cash return button."

Instead of inserting a hundred-dollar bill into the ticket machine:
    say "What, are you nuts?"

To decide what price is the remaining ticket total:
    let absolute cost be $2.25;
    let remaining cost be absolute cost minus the total price of things in the ticket
machine;
    if remaining cost is less than $0.00, decide on $0.00;
    decide on remaining cost.

Instead of inserting something which is not money into the ticket machine: say
"The ticket machine only accepts money, not other tokens of your esteem and
regard."

Instead of inserting a penny into the ticket machine:
    say "The penny rattles out again mockingly: not even the ticket machine
```

```
thinks these are worth anything."

A subway pass is a kind of thing. 15 subway passes are in the cashbox. The
description of a subway pass is usually "A rectangle of thick lavender paper with
a black magnetic stripe running up the back side. It is good for one trip on the
subway."

After inserting something into the ticket machine:
    if the remaining ticket total is $0.00:
        let purchased ticket be a random subway pass in the cashbox;
        if purchased ticket is not a subway pass, say "The ticket machine grunts
disobligingly and then the unwelcome word MALFUNCTION parades across the
LED screen, three letters at a time." instead;
        repeat with item running through things in the machine:
            now the item is nowhere;
        move purchased ticket to player;
        say "The ticket machine beeps obligingly and disgorges a single subway
pass.";
    otherwise:
        say "The ticket machine beeps obligingly and adjusts its price down to
[remaining ticket total]."
```

And because even though the ticket machine is a container, we don't want to say (empty) after it in the room description:

```
Rule for printing room description details of the ticket machine:
    do nothing instead.

Test me with "buy novel / n / get dollar / s / buy novel".
```

After all that, we should probably give the player a chance to win, as well:

```
The turnstile is south of the Subway Station. "A turnstile is all that separates you
from the subway platform stairs." The turnstile is north of the Platform. The
turnstile is a door. Before going down in the presence of the turnstile, try going
south instead. The turnstile is openable. The turnstile is open.

Instead of going through the turnstile when the player carries a subway pass:
say "You enter the turnstile and begin your journey uptown..."; end the story
finally saying "At last". Instead of going through the turnstile: say "You can't go
through the turnstile without a subway pass. They're very strict about this."

Instead of inserting money into the turnstile: say "The turnstile takes passes, not
money." Instead of inserting a subway pass into the turnstile, try entering the
turnstile.

Test more with "buy times / put all but five-dollar bill in machine / press button /
buy gum / buy cards / i / put dollar in machine / g / put quarter in machine / i / d".
```

In fairness to the Metropolitan Transit Authority, we should admit that most of the ticketing machines in the real New York subway are better than this, and will accept, say, a debit card. But that would be so much less exciting to implement.

263

Example Widget Enterprises

RB

Allowing the player to set a price for a widget on sale, then determining the resulting sales based on consumer demand, and the resulting profit and loss.

Suppose the player is responsible for pricing at Widget Enterprises. Widget production entails a certain fixed cost as well as a cost per unit; and somewhere out in the world there are a number of customers interested in purchasing widgets, but the player starts without knowing what this distribution looks like.

We can express the profits as an equation: the total made by selling widgets, minus the cost thereof.

The Table of Customers holds the data about customer preferences, and whenever the player selects a widget price, we consult it to determine how many customers in total would be willing to buy at that price.

```
"Widget Enterprises"

Widget Stand is a room.

A monetary value is a kind of value. $1.99 specifies a monetary value with parts
dollars and cents.

Equation - Profit Equation
   P = nV - (F + nC)
where P is a monetary value, F is the fixed cost, C is the unit cost, V is a
monetary value, and n is a number.

The fixed cost is a monetary value that varies. The fixed cost is $5.00.
The unit cost is a monetary value that varies. The unit cost is $10.66.

Table of Customers

    base maximum value
    2    $26.00
    5    $20.00
    8    $15.00
    2    $13.50
    1    $9.00

To decide what number is the units sold at (V - a monetary value):
   let total units be 0;
   repeat through the Table of Customers:
      if V is less than the maximum value entry:
         increase total units by the base entry;
   decide on total units.

Understand "set price to [monetary value]" as setting price to. Setting price to is
an action applying to one monetary value.

Carry out setting price to:
   let V be the monetary value understood;
```

```
    let n be the units sold at the monetary value understood;
    let P be given by the Profit Equation;
    say "You set the price of your widgets to [V], resulting in sales of [n] unit[s] and
";
    if P is less than $0.00:
        let L be $0.00 - P;
        say "a loss of [L].";
    otherwise if P is $0.00:
        say "break even.";
    otherwise:
        say "a profit of [P].".

Test me with "set price to $0.00 / set price to $100.00 / set price to $15.00 / set
price to $8.00 / set price to $25.00 / set price to $14.99".
```

As written this will be a rather dull guessing game for the player; more interesting would be to enhance it into a fuller economic simulator with more control over fixed costs and customer price points.

264

Example Frozen Assets

RB

A treatment of money which keeps track of how much the player has on him, and a BUY command which lets him go shopping.

In our brave new world, everything will have a price, so we had better spell this out.

```
"Frozen Assets"

Price is a kind of value. $10.99 specifies a price. A thing has a price. The price of
a thing is usually $0.00. After examining something for sale, say "It can be yours
for [the price of the noun]."
```

Now we assume a simple shopping model in which the player can't take anything without paying for it.

```
Definition: a thing is free if the price of it is $0.00.
Definition: a thing is for sale if it is not free.

Instead of taking something for sale:
    say "You'll have to pay for that."

Before buying something for sale when the money is not in the wallet:
    say "You're broke." instead.

Before buying something for sale when the money is free:
    say "You're broke." instead.

Before buying something for sale when the price of the money is less than the
price of the noun:
    say "Your funds do not run to the price of [the noun]." instead.
```

```
Instead of buying something:
    decrease the price of the money by the price of the noun;
    say "You fork over [the price of the noun] for [the noun], leaving yourself with
[the price of the money].";
    if the money is free:
        now the money is nowhere;
    now the price of the noun is $0.00;
    now the player is carrying the noun.
```

The player's money object is going to be a bit unusual, because it has value but cannot itself be bought.

```
The player carries a wallet. The wallet contains money. The price of the money
is $4.50. The printed name of the money is "[price of the money] in cash".
Understand "cash" as the money.

Instead of taking the money:
    say "Best to leave it alone until you need to buy something."

Instead of buying something free:
    say "[The noun] is yours already."

Instead of buying the money:
    say "The money belongs to you; you buy things with it."
```

Now we just need something to buy.

```
The Dessert Parlor is a room. "An underlit, over-crowded room campily furnished
with a lot of gilt-frame mirrors and twinkle lights: it is essentially a brothel of food.
The service is slow at best, and on Saturday nights glacial. However. The wares
on display more than make up for these trivial inconveniences."

The vanilla ice cream is an edible thing in the Parlor. The price of the ice cream
is $2.45. The description is "In the scale of ice creams, you recognize this as a
very inferior vanilla because it has no adjectives in the title."

The raspberry tart is an edible thing in the Parlor. The price of the tart is $4.50.
The description is "An almond-laced shell packed with raspberries-under-glaze."

The syllabub is an edible thing in the Parlor. The price of the syllabub is $4.25.
The description is "Whipped cream, alcohol, and lime juice, a substance without
any redeeming food value whatever."

The espresso cake is an edible thing in the Parlor. The price of the espresso
cake is $5.50. The description is "A lethal wedge of purest blackness."

Test me with "inventory / examine syllabub / get syllabub / buy syllabub / drop it /
get it / buy raspberry tart".
```

Implementing caloric units for this scenario is left as an exercise for the reader.

An OFFER price FOR command, allowing the player to bargain with a flexible seller.

"Money for Nothing"

Section 1 - Prices and Bargaining

Price is a kind of value. $10.99 specifies a price with parts dollars and cents (optional, preamble optional).

A person has a price called wealth. The wealth of the player is $15.

A thing has a price called minimum value. The minimum value of a thing is usually $0.50.

A thing has a price called desired value. The desired value of a thing is usually $5.00.

Offering it for is an action applying to one price and one visible thing.

Understand "offer [price] for [something]" as offering it for.

After taking inventory, say "You have [the wealth of the player]."

```
Check offering it for:
    if the price understood is greater than the wealth of the player, say "You don't
have that kind of cash." instead;
    if the second noun is not carried by someone, say "There's no one in a
position to sell you [the second noun]." instead;
    if the second noun is carried by the player, say "[The second noun] is already
yours." instead;
    if the minimum value of the second noun is greater than the price understood,
say "[The holder of the second noun] cackles disdainfully. 'If yer just here to
insult me you can take your business elsewhere!' he says." instead;
    if the desired value of the second noun is greater than the price understood:
        let difference be the desired value of the second noun minus the price
understood;
        let difference be difference divided by two;
        decrease the desired value of the second noun by difference;
        now the last object offered is the second noun;
        say "'How about [desired value of the second noun]?' suggests [the holder
of the second noun]." instead;
    otherwise:
        unless the desired value of the second noun is the price understood:
            say "From the avaricious gleam in the eye of [the holder of the second
noun], you guess you could've gotten this purchase for less..."

Carry out offering it for:
    increase the wealth of the holder of the second noun by the price understood;
    decrease the wealth of the player by the price understood;
    move the second noun to the player.
```

```
Report offering it for:
    say "You spend [the price understood], and now you possess [the second
noun]."

When play begins: now right hand status line is "Your funds: [wealth of the
player]".
```

Now, since the man does make counter-offers, it would be reasonable to let the player accept or reject those, as well:

```
The last object offered is a thing that varies.

Instead of saying yes when the last object offered is carried by a person (called
seller) who is not the player:
    if the seller is not visible:
        continue the action;
    otherwise:
        now the price understood is the desired value of the last object offered;
        try offering the desired value of the last object offered for the last object
offered.

Instead of saying no when the last object offered is carried by a person (called
seller) who is not the player:
    if the seller is not visible:
        continue the action;
    otherwise:
        now the last object offered is the player;
        say "You reject the offer firmly."
```

And we borrow just a line or two from a later chapter to take care of some alternate syntax the player might try:

```
Understand "offer [price] to [someone]" as a mistake ("You'll need to specify
what you want to buy -- try OFFER $1000.00 FOR BROOKLYN BRIDGE.").
Understand "offer [someone] [price]" as a mistake ("You'll need to specify what
you want to buy -- try OFFER $1000.00 FOR BROOKLYN BRIDGE.").

Understand "buy [something]" as a mistake ("You'll have to name your price: try
OFFER $1000.00 FOR BROOKLYN BRIDGE.").

Section 2 - The Scenario

The Flea Market is a room. The crotchety man is a man in the Market. "A
crotchety man here is selling [the list of things carried by the crotchety man]."
The crotchety man carries a broken television set, a Victorian rhinestone brooch,
and a cracked shaving mug.

The minimum value of the brooch is $2.50.

Test me with "offer $0.50 for mug / offer $0.50 to man / offer $6.00 for mug / offer
$50.00 for brooch / offer $1.50 for brooch / offer $4.50 for brooch / no / offer
$4.50 for brooch / yes".
```

266

Example Lemonade

RB

Containers for liquid which keep track of how much liquid they are holding and of what kind, and allow quantities to be moved from one container to another.

Liquids, and all substances that can be mixed or broken off in partial amounts, pose a challenge to model in interactive fiction. The following example is a simple one, but adequate for many scenarios.

We start by assuming that all liquids in the game will always appear in containers. The player can pour liquids from one container to another, and the containers keep track of how full they are and describe themselves appropriately. The player can also refer to containers by content.

Mixture, however, is not allowed, nor is it possible to put liquids on other objects, pour them out on the ground, etc. These ideas would require a more complicated set-up.

```
"Lemonade"

A volume is a kind of value. 15.9 fl oz specifies a volume with parts ounces and
tenths (optional, preamble optional).

A fluid container is a kind of container. A fluid container has a volume called a
fluid capacity. A fluid container has a volume called current volume.

The fluid capacity of a fluid container is usually 12.0 fl oz. The current volume of
a fluid container is usually 0.0 fl oz.

Liquid is a kind of value. The liquids are water, milk, lemonade, and iced tea. A
fluid container has a liquid.

Instead of examining a fluid container:
    if the noun is empty,
        say "You catch just a hint of [the liquid of the noun] at the bottom.";
    otherwise
        say "[The noun] contains [current volume of the noun in rough terms] of
[liquid of the noun]."

To say (amount - a volume) in rough terms:
    if the amount is less than 0.5 fl oz:
        say "a swallow or two";
    otherwise if tenths part of amount is greater than 3 and tenths part of amount
is less than 7:
        let estimate be ounces part of amount;
        say "[estimate in words] or [estimate plus 1 in words] fluid ounces";
    otherwise:
        if tenths part of amount is greater than 6, increase amount by 1.0 fl oz;
        say "about [ounces part of amount in words] fluid ounce[s]".

Before printing the name of a fluid container (called the target) while not drinking:
    if the target is empty:
```

```
        say "empty ";
    otherwise:
        do nothing.

After printing the name of a fluid container (called the target) while not
examining:
    unless the target is empty:
        say " of [liquid of the target]";
        omit contents in listing.

Instead of inserting something into a fluid container:
    say "[The second noun] has too narrow a mouth to accept anything but
liquids."

Definition: a fluid container is empty if the current volume of it is 0.0 fl oz.
Definition: a fluid container is full if the current volume of it is the fluid capacity of
it.

Understand "drink from [fluid container]" as drinking.

Instead of drinking a fluid container:
    if the noun is empty:
        say "There is no more [liquid of the noun] within." instead;
    otherwise:
        decrease the current volume of the noun by 0.2 fl oz;
        if the current volume of the noun is less than 0.0 fl oz, now the current
volume of the noun is 0.0 fl oz;
        say "You take a sip of [the liquid of the noun][if the noun is empty], leaving
[the noun] empty[end if]."
```

We have allowed all liquids to be drunk, but it would be possible also to add checking, if we had a game where some liquids were beverages and others were, say, motor oil.

```
Understand the command "fill" as something new.

Understand "pour [fluid container] in/into/on/onto [fluid container]" as pouring it
into. Understand "empty [fluid container] into [fluid container]" as pouring it into.
Understand "fill [fluid container] with/from [fluid container]" as pouring it into (with
nouns reversed).

Understand "pour [something] in/into/on/onto [something]" as pouring it into.
Understand "empty [something] into [something]" as pouring it into. Understand
"fill [something] with/from [something]" as pouring it into (with nouns reversed).

Pouring it into is an action applying to two things.

Check pouring it into:
    if the noun is not a fluid container, say "You can't pour [the noun]." instead;
    if the second noun is not a fluid container, say "You can't pour liquids into [the
second noun]." instead;
    if the noun is the second noun, say "You can hardly pour [the noun] into itself."
instead;
    if the liquid of the noun is not the liquid of the second noun:
        if the second noun is empty, now the liquid of the second noun is the liquid
```

```
of the noun;
        otherwise say "Mixing [the liquid of the noun] with [the liquid of the second
noun] would give unsavory results." instead;
    if the noun is empty, say "No more [liquid of the noun] remains in [the noun]."
instead;
    if the second noun is full, say "[The second noun] cannot contain any more
than it already holds." instead.

Carry out pouring it into:
    let available capacity be the fluid capacity of the second noun minus the
current volume of the second noun;
    if the available capacity is greater than the current volume of the noun, now
the available capacity is the current volume of the noun;
    increase the current volume of the second noun by available capacity;
    decrease the current volume of the noun by available capacity.

Report pouring it into:
    say "[if the noun is empty][The noun] is now empty;[otherwise][The noun] now
contains [current volume of the noun in rough terms] of [liquid of the noun]; [end
if]";
    say "[the second noun] contains [current volume of the second noun in rough
terms] of [liquid of the second noun][if the second noun is full], and is now
full[end if]."
```

This is probably a drier description than we would actually want in our story, but it does allow us to see that the mechanics of the system are working, so we'll stick with this for the example.

Now we need a trick from a later chapter, which allows something to be described in terms of a property it has. This way, the story will understand not only "pitcher" and "glass" but also "pitcher of lemonade" and "glass of milk" -- and, indeed, "glass of lemonade", if we empty the glass and refill it with another substance:

```
Understand the liquid property as describing a fluid container. Understand "of" as
a fluid container.
```

And now the scenario itself:

```
The Porch is a room. The porch swing is an enterable supporter in the Porch.
"An inviting swing hangs here at the end of the porch, allowing you to enjoy the
summer with a cool beverage, and watch your neighbor Ted mowing his lawn
with the very last manual powerless lawnmower on the block."

The glass is a fluid container carried by the player. The liquid of the glass is milk.
The current volume of the glass is 0.8 fl oz.

The pitcher is a fluid container in the Porch. The fluid capacity of the pitcher is
32.0 fl oz. The current volume of the pitcher is 20.0 fl oz. The liquid of the pitcher
is lemonade.

Ted's Lawn is outside from the Porch. Ted is a man in Ted's Lawn. "Ted has
taken off his shirt, but still seems a bit oppressed by the sun." The description of
Ted is "He looks hot. In all senses."
```

```
After deciding the scope of the player: place Ted in scope.

Instead of doing something to Ted when the player is in the Porch: say "You
can't really interact with Ted from this distance, except in the sense of eyeing
him surreptitiously."

Instead of giving an empty fluid container to Ted: say "Yes, taunt the poor man,
why don't you?"

Instead of giving a fluid container to Ted when the liquid of the noun is milk: say
"Ted looks ruefully at the milk. 'Thanks, but I'm lactose-intolerant,' he says."

The block giving rule is not listed in the check giving it to rules.

Every turn:
    if Ted is in the location:
        if Ted carries a fluid container (called refreshment):
            try Ted drinking the refreshment;
        otherwise if a random chance of 1 in 3 succeeds:
            say "Ted pushes the ineffective mower over some dandelions."

Instead of someone drinking a fluid container:
    if the noun is empty:
        try the person asked giving the noun to the player;
    otherwise:
        decrease the current volume of the noun by 2.0 fl oz;
        if the current volume of the noun is less than 0.0 fl oz, now the current
volume of the noun is 0.0 fl oz;
        say "[The person asked] gulps down some [liquid of the noun]."

After someone giving something to the player:
    say "'Here,' says [the person asked], handing [the noun] back to you. 'Thanks,
I owe you one.'";
    end the story finally.

Test me with "x milk / x lemonade / drink lemonade / drink milk / pour lemonade
into glass / drink milk / x milk / drink milk / g / i / fill glass with lemonade / drink
lemonade / drop glass / drink lemonade / pitcher".

Test Ted with "out / give milk to ted / drink milk / g / g / g / give glass to ted / in /
fill glass with lemonade / out / give lemonade to ted / wait / z / z / z ".
```

267

Example **Savannah**

RB

Using the liquid implementation demonstrated in Lemonade for putting out fires.

Here we build very slightly on the existing liquid implementation to add a puzzle where the player puts out a fire with a bucket of water. Most of the liquid implementation remains the same as before, but now we understand the names of containers according to the liquids they contain.

The new material, pertaining to extinguishing fires, is at the bottom in section 2.

```
"Savannah"

Section 1 - Essentials of Liquid

A volume is a kind of value. 15.9 fl oz specifies a volume with parts ounces and
tenths (optional, preamble optional).

A fluid container is a kind of container. A fluid container has a volume called a
fluid capacity. A fluid container has a volume called current volume.

The fluid capacity of a fluid container is usually 12.0 fl oz. The current volume of
a fluid container is usually 0.0 fl oz.

Liquid is a kind of value. A fluid container has a liquid.

Instead of examining a fluid container:
    if the noun is empty,
        say "You catch just a hint of [the liquid of the noun] at the bottom.";
    otherwise
        say "[The noun] contains [current volume of the noun in rough terms] of
[liquid of the noun]."

To say (amount - a volume) in rough terms:
    if the amount is less than 0.5 fl oz:
        say "a swallow or two";
    otherwise if tenths part of amount is greater than 3 and tenths part of amount
is less than 7:
        let estimate be ounces part of amount;
        say "[estimate in words] or [estimate plus 1 in words] fluid ounces";
    otherwise:
        if tenths part of amount is greater than 6, increase amount by 1.0 fl oz;
        say "about [ounces part of amount in words] fluid ounce[s]".

Before printing the name of a fluid container (called the target) while not drinking:
    if the target is empty:
        say "empty ";
    otherwise:
        do nothing.

After printing the name of a fluid container (called the target) while not
examining:
    unless the target is empty:
        say " of [liquid of the target]";
        omit contents in listing.

Instead of inserting something into a fluid container:
    say "[The second noun] has too narrow a mouth to accept anything but
liquids."

Definition: a fluid container is empty if the current volume of it is 0.0 fl oz.
Definition: a fluid container is full if the current volume of it is the fluid capacity of
it.
```

```
Understand "drink from [fluid container]" as drinking.

Instead of drinking a fluid container:
    if the noun is empty:
        say "There is no more [liquid of the noun] within." instead;
    otherwise:
        decrease the current volume of the noun by 0.2 fl oz;
        if the current volume of the noun is less than 0.0 fl oz, now the current
volume of the noun is 0.0 fl oz;
        say "You take a sip of [the liquid of the noun][if the noun is empty], leaving
[the noun] empty[end if]."

Understand the command "fill" as something new.

Understand "pour [fluid container] in/into/on/onto [fluid container]" as pouring it
into. Understand "empty [fluid container] into [fluid container]" as pouring it into.
Understand "fill [fluid container] with/from [fluid container]" as pouring it into (with
nouns reversed).

Understand "pour [something] in/into/on/onto [something]" as pouring it into.
Understand "empty [something] into [something]" as pouring it into. Understand
"fill [something] with/from [something]" as pouring it into (with nouns reversed).

Pouring it into is an action applying to two things.

Check pouring it into:
    if the noun is not a fluid container, say "You can't pour [the noun]." instead;
    if the second noun is not a fluid container, say "You can't pour liquids into [the
second noun]." instead;
    if the noun is the second noun, say "You can hardly pour [the noun] into itself."
instead;
    if the liquid of the noun is not the liquid of the second noun:
        if the second noun is empty, now the liquid of the second noun is the liquid
of the noun;
        otherwise say "Mixing [the liquid of the noun] with [the liquid of the second
noun] would give unsavory results." instead;
    if the noun is empty, say "No more [liquid of the noun] remains in [the noun]."
instead;
    if the second noun is full, say "[The second noun] cannot contain any more
than it already holds." instead.

Carry out pouring it into:
    let available capacity be the fluid capacity of the second noun minus the
current volume of the second noun;
    if the available capacity is greater than the current volume of the noun, now
the available capacity is the current volume of the noun;
    increase the current volume of the second noun by available capacity;
    decrease the current volume of the noun by available capacity.

Report pouring it into:
    say "[if the noun is empty][The noun] is now empty;[otherwise][The noun] now
contains [current volume of the noun in rough terms] of [liquid of the noun]; [end
if]";
    say "[the second noun] contains [current volume of the second noun in rough
terms] of [liquid of the second noun][if the second noun is full], and is now
full[end if]."
```

```
Understand the liquid property as describing a fluid container. Understand "of" as
a fluid container.

Section 2 - Putting Out Fires

The Beach is a room. "The Atlantic stretches east to the horizon, though it is at
low tide at the moment. It is dawn: time to pack up and go home."
```

We will skip implementing the Pacific ocean itself, though the example Lakeside Living shows how to incorporate large bodies of water into our liquid simulation.

```
The liquids are seawater. [We could include others, but for the moment...]

Instead of drinking a fluid container when the liquid of the noun is seawater:
    say "Blech!"

The bucket is a fluid container carried by the player. The liquid of the bucket is
seawater. The current volume of the bucket is 64.0 fl oz.

The fire is a fixed in place thing in the beach. "A low fire crackles here, left over
from an attempt at s'mores much earlier in the evening."

Instead of touching or rubbing or taking the fire, say "You're not such a glutton
for punishment."

Instead of pouring something into the fire:
    now the fire is nowhere;
    now the current volume of the noun is 0.0 fl oz;
    say "[The second noun] goes out in a great hiss."

Test me with "drink seawater / pour seawater on fire / x bucket / i".
```

This is still a specific implementation: if we wanted to weave liquids together with a full-scale burning model (as in "In Fire or in Flood"), where pretty much any object in the game can be flaming (currently on fire) or damp (extinguished), we might generalize our rule to

```
Instead of pouring something into a flaming thing:
    now the second noun is damp;
    now the current volume of the noun is 0.0 fl oz;
    say "[The second noun] goes out in a great hiss."
```

Of course, the merging of fire and liquids also raises the possibility of gasoline and explosives, of heating and boiling liquids, etc.: as always, it's wise to incorporate a simulation that is only as detailed as the game's interactions really justify.

---

268 **Example Depth** RB

Receptacles that calculate internal volume and the amount of room available, and cannot be overfilled.

In the following, we pretend that every item has a cuboidal shape. Every thing has a length, width and depth, while a "measured container" also has interior dimensions. (Thus a 10x10x10 container with 1cm-thick sides might have interior dimensions 9x9x9.)

```
"Depth"

A length is a kind of value. 10 cm specifies a length. An area is a kind of value. 10 sq cm specifies an area. A length times a length specifies an area. A volume is a kind of value. 10 cu cm specifies a volume. A length times an area specifies a volume.

A thing has a length called height. A thing has a length called width. A thing has a length called depth. The height of a thing is usually 10 cm. The width of a thing is usually 10 cm. The depth of a thing is usually 10 cm.

To decide what volume is the exterior volume of (item - a thing):
    let base area be the height of the item multiplied by the width of the item;
    let base volume be the base area multiplied by the depth of the item;
    decide on the base volume.
```

In order to see how these shapes might fit together spatially, we need to work out the three dimensions in order of size. (If we were only dealing with portable objects, we could simply insist that the length always be greater than the width which in turn must be greater than the depth, because we could always turn them over in our hands until this was so: but some of the things we deal with may be fixed in place.) A clever way to do this might be to put them in a table of three rows and sort it, but we will write the calculation out longhand:

```
To decide what length is the largest dimension of (item - a thing):
    let long side be the height of item;
    if the width of the item is greater than the long side, now the long side is the width of the item;
    if the depth of the item is greater than the long side, now the long side is the depth of the item;
    decide on the long side.

To decide what length is the middling dimension of (item - a thing):
    let longer side be the height of item;
    let shorter side be the width of item;
    if the width of the item is greater than the height of the item:
        let shorter side be the height of item;
        let longer side be the width of item;
    if the depth of the item is greater than the longer side, decide on the longer side;
    if the depth of the item is less than the shorter side, decide on the shorter side;
    decide on the depth of the item.

To decide what length is the shortest dimension of (item - a thing):
    let short side be the height of item;
    if the width of the item is less than the short side, now the short side is the width of the item;
    if the depth of the item is less than the short side, now the short side is the
```

```
depth of the item;
    decide on the short side.
```

When testing this example, the author made use of the following: it's no longer needed, but may be useful to anyone else planning elaborations.

```
To test the dimensions of (item - a thing):
    say "[the item] - height [height of the item], width [width of the item], depth
[depth of the item].";
    say "largest side [largest dimension of the item], middling [middling dimension
of the item], smallest [shortest dimension of the item]."
```

We now introduce a new kind: a measured container, which not only has exterior dimensions - the height, width and depth which every thing now has - but also interior measurements. A convenient way to do calculations with the hollow interior is to regard it as if it were a solid shape in its own right, and we do this with the aid of something out of world, which the player never sees: the "imaginary cuboid", which is made into the shape of whatever measured container's interior is being thought about.

```
A measured container is a kind of container. A measured container has a length
called interior height. A measured container has a length called interior width. A
measured container has a length called interior depth.

There is an imaginary cuboid.

To imagine the interior of (receptacle - a measured container) as a cuboid:
    now the height of the imaginary cuboid is the interior height of the receptacle;
    now the width of the imaginary cuboid is the interior width of the receptacle;
    now the depth of the imaginary cuboid is the interior depth of the receptacle.

To decide what volume is the interior volume of (receptacle - a measured
container):
    imagine the interior of the receptacle as a cuboid;
    decide on the exterior volume of the imaginary cuboid.
```

If we assume that we could always pack items into a measured container with perfect ease, never wasting any space, then the only volume constraint will be that the total volume of the contents must not exceed the volume of the inside of the container. So we need to calculate the available volume.

```
To decide what volume is the available volume of (receptacle - a measured
container):
    let the remaining space be the interior volume of the receptacle;
    repeat with item running through things in the receptacle:
        decrease the remaining space by the exterior volume of the item;
    if the remaining space is less than 0 cu cm, decide on 0 cu cm;
    decide on the remaining space.
```

If we only constrained volume, a 140 cm-long fishing rod could fit into a 12 cm by 12 cm compact disc box. So we also insist the basic shape must fit, in some orientation perpendicular to one of the sides (i.e.: we can turn the item over in any of its three sides, but not turn it diagonally or wedge it in at a tilt). This requires the

longest side of the item to be less than the longest side of the receptacle, and the middle-length side, and also the shortest side. The number of these conditions to fail gives us a clue as to how we can best describe the reason why the shape won't squeeze in.

```
Check inserting something (called the item) into a measured container (called
the receptacle):
    if the exterior volume of the item is greater than the interior volume of the
receptacle, say "[The item] will never fit inside [the receptacle]." instead;
    if the exterior volume of the item is greater than the available volume of the
receptacle, say "[The item] will not fit into [the receptacle] with [the list of things
in the receptacle]." instead;
    imagine the interior of the receptacle as a cuboid;
    if the largest dimension of the item is greater than the largest dimension of the
imaginary cuboid, say "[The item] is too long to fit into [the receptacle]." instead;
    if the middling dimension of the item is greater than the middling dimension of
the imaginary cuboid, say "[The item] is too wide to fit into [the receptacle]."
instead;
    if the shortest dimension of the item is greater than the shortest dimension of
the imaginary cuboid, say "[The item] is too bulky to fit into [the receptacle]."
instead.
```

And finally a situation to try out these rules.

```
The Cubist Lab is a room. "A laboratory which, as the art critic Louis Vauxcelles
said about Braque's paintings in 1908, is full of little cubes: everyday objects
rendered as if cuboidal."

The box is a measured container. The interior height is 10 cm. The interior depth
is 5 cm. The interior width is 6 cm. The player carries the box.

A pebble is a kind of thing. The height is usually 2 cm. The depth is usually 2 cm.
The width is usually 2 cm. The player carries 25 pebbles.

A red rubber ball is carried by the player. The depth is 5 cm. The width is 5 cm.
The height is 5 cm.
An arrow is carried by the player. The height is 40 cm. The width is 1 cm. The
depth is 1 cm.
A crusty baguette is carried by the player. The height is 80 cm. The width is 4
cm. The depth is 5 cm.
A child's book is carried by the player. The height is 1 cm. The width is 9 cm.
The depth is 9 cm.
A featureless white cube is carried by the player. The height is 6 cm. The width
is 6 cm. The depth is 6 cm.

Test me with "put arrow in box / put book in box / put cube in box / put ball in box
/ put baguette in box / put pebbles in box".
```

Several warnings about this. First, the numbers can't go very high (if the Settings for the project set the story file format to the Z-machine): while the volume can in theory go to 32,767, in practice this equates to an object 32 cm on a side, which is not very large. One way to avoid this is to use the Glulx format, allowing for sizes in excess of 10 m on a side: or we could simply scale the dimensions to suit our purposes, using a decimeter (10 cm) as the basic unit of measurement, for instance.

Second, the system will require a height, width, and depth for every portable object in the game, which is a large commitment to data entry; it may become tiresome. So it is probably not worth bothering with this kind of simulation unless it is going to be genuinely significant.

---

269  **Example Fabrication** *RB*

A system of assembling clothing from a pattern and materials; both the pattern and the different fabrics have associated prices.

When we make a new kind of value, the new named values can themselves have properties. That is convenient because, for instance, we might want to associate a material (itself the property of an object) with certain features, such as price.

```
"Fabrication"

Section 1 - Procedure

A material is a kind of value. The materials are silk, velvet, cotton, and wool.

Price is a kind of value. $1.99 specifies a price.

Area is a kind of value. 5 sq yards specifies an area.

Cost is a kind of value.. $1.99 per sq yard specifies a cost. A cost times an area
specifies a price.

A material has a cost.

The cost of silk is usually $5.75 per sq yard. The cost of velvet is usually $9.50
per sq yard. The cost of cotton is usually $2.29 per sq yard. The cost of wool is
usually $4.75 per sq yard.

A pattern is a kind of thing. A pattern has a material. A pattern has an area. A
pattern has a price. The price of a pattern is usually $9.99. Understand "pattern"
as a pattern. Understand "patterns" as the plural of a pattern.

After printing the name of a pattern:
    if planning:
        do nothing;
    otherwise:
        say " pattern".

To decide what price is the material price of (chosen item - pattern):
    let C be the cost of the material of the chosen item;
    let A be the area of the chosen item;
    decide on C * A.

To decide what price is the overall price of (chosen item - pattern):
    let P be the price of the chosen item;
```

```
    let M be the material price of the chosen item;
    decide on P + M.

Understand "plan [material] [pattern]" as planning it for.

Planning it for is an action applying to one material and one thing.

Carry out planning it for:
    now the material of the second noun is the material understood.

Report planning it for:
    say "You lay plans for a [material understood] [second noun], running [material
price of the second noun] for materials and [price of the second noun] for the
pattern itself, for a total of [overall price of the second noun]."

Section 2 - Scenario

Joanne's Fabrics is a room. Joanne's Fabrics contains a pattern bin.

The cape is a pattern. The material of the cape is velvet. The area of the cape is
9 sq yards.

The bodice is a pattern. The material of the bodice is silk. The area of the bodice
is 2 sq yards. The price of the bodice is $11.99.

The cape and the bodice are in the pattern bin.

Test me with "plan silk bodice / plan velvet bodice / plan velvet cape / plan wool
cape".
```

---

270 **Example The Speed of Thought** *RB*

Describing scientifically-measured objects in units more familiar to the casual audience.

Suppose that we have a number of objects in the game that are sized in some conventional unit (such as meters), but which we would like to describe in slightly less formal terms. To do this, we will start with measurements as defined in the built-in extension Metric Units, so we don't have to recreate all these.

We'll add our own set of "conceptual units" -- things we're familiar with in real life. As we'll see below, Inform will automatically choose a unit of the right order to express a given distance if we tell it to print a length "in conceptual units".

Note: the following will compile only if you have settings set for Glulx. (To change this, go to the Settings panel and click on the Glulx option.) The Glulx virtual machine is capable of handling larger numbers than the Z-machine.

```
"The Speed of Thought"
```

Section 1 - Procedure

Include Metric Units by Graham Nelson.

1 quarter (in conceptual units, in quarters, singular) or 2 quarters (in conceptual units, in quarters, plural) specifies a length equivalent to 24mm.
1 pencil (in conceptual units, in pencils, singular) or 2 pencils (in conceptual units, in pencils, plural) specifies a length equivalent to 18cm.
1 bathtub (in conceptual units, in bathtubs, singular) or 2 bathtubs (in conceptual units, in bathtubs, plural) specifies a length equivalent to 152cm.
1 Olympic swimming pool (in conceptual units, in Olympic swimming pools, singular) or 2 Olympic swimming pools (in conceptual units, in Olympic swimming pools, plural) specifies a length equivalent to 50 meters.
1 Empire state building (in conceptual units, in Empire State buildings, singular) or 2 Empire State buildings (in conceptual units, in Empire State buildings, plural) specifies a length equivalent to 443m.

1 credit card (in conceptual units, in credit cards, singular) or 2 credit cards (in conceptual units, in credit cards, plural) specifies an area equivalent to 46 sq cm.
1 letter sheet (in conceptual units, in letter sheets, singular) or 2 letter sheets (in conceptual units, in letter sheets, plural) specifies an area equivalent to 603 sq cm.
1 queen-sized mattress (in conceptual units, in queen-sized mattresses, singular) or 2 queen-sized mattresses (in conceptual units, in queen-sized mattresses, plural) specifies an area equivalent to 3 square meters.
1 football field (in conceptual units, in football fields, singular) or 2 football fields (in conceptual units, in football fields, plural) specifies an area equivalent to 5351 square meters.

Understand "report [something]" as reporting. Reporting is an action applying to one thing.

Check reporting:
    if the noun is not a fact:
        say "The public doesn't want to hear about [the noun]." instead.

Carry out reporting:
    now the noun is nowhere.

Report reporting:
    if the extent of the noun is greater than 0mm and the surface of the noun is greater than 0 sq cm:
        contextualize "'[The noun] has a length of [about] [extent of the noun in conceptual units] and an area of [about] [surface of the noun in conceptual units].'";
    otherwise if the extent of the noun is greater than 0mm:
        contextualize "'[The noun] has a length of [about] [extent of the noun in conceptual units].'";
    otherwise if the surface of the noun is greater than 0 sq cm:
        contextualize "'[The noun] has an area of [about] [surface of the noun in conceptual units].'";
    otherwise:
        say "'[The noun] is... pretty hard to imagine,' you say weakly. That's not going to go over well."

```
To say about:
    say "[one of]roughly[or]about[or]around[or]approximately[at random]";

To contextualize (chosen information - text):
    say "[one of]You turn to the camera and speak:[or][or]Turning to another
camera angle, you add:[or][stopping] ";
    say "[chosen information] ";
    say "[one of][line break][or]Right now the station will be cutting over to a visual
of that.[or][line break][or]Pity the kids in audiovisual who have to scare that
image together in a hurry.[or]You smile brightly.[stopping]";

Section 2 - Scenario

The Science Journalism Desk is a room. "From here you, the Science Anchor,
have the privilege of reporting the latest and most fascinating stories to an eager
public."

After looking:
    try thinking.

Instead of thinking:
    say "Currently you have to report on the International Space Station. Your
story could include [the list of facts carried by the player]."

Instead of taking inventory:
    say "It looks foolish to be fiddling with your possessions on camera."

Instead of dropping a fact:
    say "You decide to omit [the noun] from your lineup.";
    now the noun is nowhere.

A fact is a kind of thing. Every fact is carried by the player. A fact has a length
called the extent. A fact has an area called the surface.

The experiment module is a fact. The extent is 1116cm.
The logistics module is a fact. The extent is 421cm.
The solar array is a fact. The surface is 375 sq m. The extent is 58m.
An individual solar cell is a fact. The surface is 8 sq cm.
The orbit height is a fact.

Report reporting the orbit height:
    contextualize "'The station orbits at heights between [about] [278km in
conceptual units] and [460km in conceptual units] above the earth.'" instead.

Every turn:
    if the player carries no facts:
        say "And that's all! The channel cuts to weather.";
        end the story saying "Time for lunch".

Test me with "report experiment module / report logistics / report height / report
array / report solar cell".
```

Chapter 16: Tables


Contents of *Writing with Inform*


Chapter 15: Numbers and Equations


Chapter 17: Understanding


Indexes of the examples

§16.1. Laying out tables

When printed books need to display detailed information in a systematic way, they break off from running text and print a table instead. Inform does the same. Here is a typical example:

```
Table 2.1 - Selected Elements

Element     Symbol Atomic number Atomic weight
"Hydrogen"  "H"    1             1
"Iron"      "Fe"   26            56
"Zinc"      "Zn"   30            65
"Uranium"   "U"    92            238
```

After the two titling lines, each line represents one row in the table, and entries on a line must be separated by at least one tab character. A table must occupy a single whole paragraph, with no skipped lines or missing entries.

The top line is a title, the first word of which must be the word 'Table'. We can then either give a table number (this need not actually be a number: Table C2, or some such, would be fine), or give a name, or both - as in this case. The possible titling formats are:

```
Table 2.3
Table of Population Statistics
Table 2.3 - Population Statistics
```

In the last example we could call the table either "Table 2.3" or "Table of Population Statistics".

Each column then has a name, and the contents must all be the same kind of value. In the elements table the "Symbol" column contains only text, for instance, and the "Atomic weight" column contains only numbers. Any kinds of value will do, so long as all the entries in the column are mutually compatible. (For instance, mixing rooms and things in a single

column would be fine, as these can be reconciled, but mixing numbers and rooms would not.)

Start of Chapter 16: Tables
Back to Chapter 15: Numbers and Equations: §15.20. Multiplication of units
Onward to §16.2. Looking up entries

---

§16.2. Looking up entries

The simplest way to access the information inside tables is to ask explicitly for it, specifying the row number, the column name and what table is to be consulted. So, given our example table

Table 2.1 - Selected Elements

Element	Symbol	Atomic number	Atomic weight
"Hydrogen"	"H"	1	1
"Iron"	"Fe"	26	56
"Zinc"	"Zn"	30	65
"Uranium"	"U"	92	238

we can write the following description:

```
symbol in row 3 of the Table of Selected Elements
```

to produce the value "Zn". Or the following will run off some chemical data:

```
repeat with N running from 1 to the number of rows in the Table of Selected Elements:
    say "The atomic weight of [element in row N of the Table of Selected Elements] is
[atomic weight in row N of the Table of Selected Elements]."
```

The result of which will be:

```
The atomic weight of Hydrogen is 1.
The atomic weight of Iron is 56.
The atomic weight of Zinc is 65.
The atomic weight of Uranium is 238.
```

Note that the first row in a table is row number 1, and that the last can be found with the phrase:

number of rows in/from (table name) **... number**

This phrase produces the number of rows (including any blank rows) in the given table. Example:

```
number of rows in the Table of Selected Elements
```

 Start of Chapter 16: Tables

 Back to §16.1. Laying out tables

 Onward to §16.3. Corresponding entries

§16.3. Corresponding entries

Continuing our example of the elements:

Table 2.1 - Selected Elements

Element	Symbol	Atomic number	Atomic weight
"Hydrogen"	"H"	1	1
"Iron"	"Fe"	26	56
"Zinc"	"Zn"	30	65
"Uranium"	"U"	92	238

If we want to know the atomic number of Uranium, say, it seems artificial to have to talk about the particular row number where the information happens to be. So we are also allowed to cross-reference, like so:

```
the atomic number corresponding to a symbol of "Fe" in the Table of Selected Elements
```

This results in 26, and similarly

```
the symbol corresponding to an atomic number of 26 in the Table of Selected Elements
```

results in "Fe". But we have to be careful:

```
the element corresponding to an atomic number of 27 in the Table of Selected Elements
```

This is not allowed (it produces an error at run-time), because there is no row with atomic number 27 in this rather limited table. We can check this in advance with the condition:

```
if there is an element corresponding to an atomic number of 27 in the Table of Standard Elements ...
```

Or more simply:

```
if there is an atomic number of 27 in the Table of Standard Elements ...
```

The condition "if there is..." can be used with any reference to a table entry: for instance, "if there is a symbol in row 5 of the Table of Standard Elements" would be false, because there are only four rows.

Start of Chapter 16: Tables
Back to §16.2. Looking up entries
Onward to §16.4. Changing entries
Example 271:  **Dubai** An elevator which connects any of 27 floors in a luxury hotel.

---

§16.4. Changing entries

Here is another rather definitive, immutable-looking table:

Table 4 - Recent Monarchs

Name	Accession	Family
"Anne"	1702	Stuart
"George I"	1714	Hanover
"George II"	1720	Hanover
"George III"	1760	Hanover
"George IV"	1820	Hanover
"William IV"	1830	Hanover
"Victoria"	1837	Hanover
"Edward VII"	1901	Saxe-Coburg-Gotha
"George V"	1910	Windsor
"Edward VIII"	1936	Windsor
"George VI"	1936	Windsor
"Elizabeth II"	1952	Windsor

But table entries can be changed as freely as variables: that is, any value can be entered so long as it has the right kind. We cannot put a dynasty into the "Name" column, or text in the "Accession" column. The phrase needed is "now ... is ...", just as it is for properties or variables:

```
Dynasty is a kind of value. The dynasties are Stuart, Hanover, Saxe-Coburg-Gotha and
Windsor.

The Table Office is a room. The Succession is in the Table Office. "The Succession, a
ponderous list of English monarchs, takes pride of place."

Instead of examining the Succession:
    say "The Succession List runs as follows...";
    repeat with N running from 1 to the number of rows in the Table of Recent Monarchs:
        say "[accession in row N of Table 4]: [name in row N of Table 4] ([family in row N of
Table 4])."

Instead of attacking the Succession:
    now the family corresponding to an accession of 1720 in the Table of Recent
Monarchs is Stuart;
    now the name in row 4 of the Table of Recent Monarchs is "Graham I";
    now the name in row 5 of the Table of Recent Monarchs is "Trixibelle IV";
    say "You deface the English succession, making suitable amendments with a quill
pen. Considering it is supposed to be mightier than the sword the effect is a little
disappointing."

Test me with "examine succession / attack it / examine it".
```

Once we start changing tables, it sometimes becomes useful to check what they contain.

showme the contents of (table name)

This phrase prints a crude but sometimes useful display on screen of the current contents of the named table. It's intended for authors to see when testing, not for players of the finished version to see.

say "[current table row]"

This text substitution produces a crude but sometimes useful listing of the entries in the currently chosen table row.

say "[row (number) **in**/from **table** (table name)**]"**

This text substitution produces a crude but sometimes useful listing of the entries in the specified row.

say "[(column name) **in**/from **table** (table name)**]"**

This text substitution produces a crude but sometimes useful listing of the entries in the specified column.


Start of Chapter 16: Tables


Back to §16.3. Corresponding entries


Onward to §16.5. Choosing rows

---

§16.5. Choosing rows

The following would be one way to print out a list of recent Kings and Queens:

```
To list the succession:
say "The Succession List runs as follows...";
repeat with N running from 1 to the number of rows in the Table of Recent Monarchs:
    say "[accession in row N of the Table of Recent Monarchs]: [name in row N of the
Table of Recent Monarchs] ([family in row N of the Table of Recent Monarchs])."
```

This works, but is repetitive. We often want to work on a single row for a while, either to change things or think about the contents, and it is tiresome to keep specifying the row over

and over again. The following shorthand provides some relief:

choose a/the/-- **row** (number) **in**/from (table name)

This phrase selects the row with the given number. Row numbers in a table start from 1, so

```
choose row 1 from the Table of Recent Monarchs
```

selects the top row.

That allows us to improve the loop:

```
To list the succession:
    say "The Succession List runs as follows...";
    repeat with N running from 1 to the number of rows in the Table of Recent Monarchs:
        choose row N in the Table of Recent Monarchs;
        say "[accession entry]: [name entry] ([family entry]).";
```

Actually, as we'll see in the next section, this kind of loop is needed so often that there's a shorthand wording for it.

Note that since "accession" is a column name, "accession entry" means the entry in that column of the currently chosen row. This notation can only be used if a "choose" has certainly already happened, and it is a good idea to make that choice somewhere close by in the source code (and certainly in the same rule or phrase definition) for the sake of avoiding errors. We can also choose rows by specifying something about them, like so:

choose a/the/-- **row with** (table column) **of** (value) **in**/from (table name)

This phrase selects the first row, working down from the top of the given table, in which the given column has the given value. Example:

```
choose row with a name of "Victoria" in the Table of Recent Monarchs;
```

A run-time problem message is produced if the value isn't found anywhere in that column.

Sometimes it will happen that a column's name clashes with the name of something else: for instance, if we call a column "apples" but we also have a kind called "apple", so that the word "apples" could mean either some fruit or the column. Inform will generally prefer the former meaning as more likely. In case of such trouble, we can simply refer to "the apples column" rather than just "the apples": for instance, "choose row with an apples column of..." rather than "choose row with an apples of..."

We can also choose a row quite at random:

choose a/the/-- **random row in**/from (table name)

This phrase makes a uniformly random choice of non-blank rows in the given table. Note that although a table always has at least one row, it can't be guaranteed that it always has a non-blank row, so it's possible for this to fail: if it does, a real-time problem message is thrown.


Start of Chapter 16: Tables


Back to §16.4. Changing entries


Onward to §16.6. Repeating through tables

---

§16.6. Repeating through tables

We very often want to run through a table doing something to, or with, each row in turn, so a special loop is provided for this. Rather than having to write all this out:

```
To list the succession:
    say "The Succession List runs as follows...";
    repeat with N running from 1 to the number of rows in the Table of Recent Monarchs:
        choose row N in the Table of Recent Monarchs;
        say "[accession entry]: [name entry] ([family entry])."
```

We can simply use this instead:

repeat through (table name)**:**

This phrase causes the block of phrases following it to be repeated once for each row in the given table, choosing each row in turn, from top to bottom. Blank rows are skipped. Example:

```
To list the succession:
    say "The Succession List runs as follows...";
    repeat through the Table of Recent Monarchs:
        say "[accession entry]: [name entry] ([family entry])."
```

Note that there is no loop variable here, unlike in other forms of "repeat", because it's the choice of row which keeps track of how far we have got.

We can alternatively go backwards:

repeat through (table name) **in reverse order:**

This phrase causes the block of phrases following it to be repeated once for each row in the given table, choosing each row in turn, from bottom to top. Blank rows

are skipped.

More often we want a sequence which is neither forwards nor backwards, but which depends on the actual values in the table.

repeat through (table name) **in** (table column) **order:**

This phrase causes the block of phrases following it to be repeated once for each row in the given table, choosing each row in turn, in order of the values in the given column. Blank rows are skipped. Example:

```
repeat through the Table of Recent Monarchs in name order: ...
repeat through the Table of Recent Monarchs in accession order: ...
```

work through the same table in rather different orders. The sequence is lower to higher (small numbers to high numbers, A to Z, and so on); insert "reverse" after "in" to reverse this.

repeat through (table name) **in reverse** (table column) **order:**

This phrase causes the block of phrases following it to be repeated once for each row in the given table, choosing each row in turn, in order of the values in the given column. Blank rows are skipped. Example:

```
repeat through the Table of Recent Monarchs in reverse name order: ...
repeat through the Table of Recent Monarchs in reverse accession order: ...
```

work through the same table in rather different orders. The sequence is higher to lower (high numbers to small numbers, Z to A, and so on); delete the "reverse" after "in" to reverse this.

In a loop like this, the data is not searched very efficiently, which is fine for modest-sized tables like the examples in this chapter, but might be a problem for much larger tables: see the later section on sorting.

These definitions mentioned blankness several times, and that's the topic to cover in the next section.

See **Sorting** *for reordering a table to put it into increasing or decreasing order of the entries in any column*

Start of Chapter 16: Tables
Back to §16.5. Choosing rows
Onward to §16.7. Blank entries
Example 272: **Port Royal 4** A cell window through which the player can see people who were in Port Royal in the current year of game-time.

---

§16.7. Blank entries

We are allowed to leave certain entries blank (perhaps to be filled in later, perhaps not) by writing "--" instead of the relevant value:

Table 2.1 - Selected Elements

Element	Symbol	Atomic number	Atomic weight
"Hydrogen"	"H"	1	1
"Iron"	"Fe"	--	56
"Zinc"	--	30	65
"Uranium"	"U"	92	238

In effect, blank entries don't exist. "--" is not a value, but only a hole where a value might be. It can be useful to check for this:

if there is (a table entry)**:**

This condition is true if the entry referred to exists, that is, that is, the space for it in the table is not blank. Examples:

```
if there is a symbol corresponding to an atomic number of 30 in the Table of Standard Elements ...
if there is an atomic number in row 2 of the Table of Standard Elements ...
```

if there is no (a table entry)**:**

This condition is true if the entry referred to does not exist, that is, the space for it in the table is blank. Examples:

```
if there is no symbol corresponding to an atomic number of 30 in the Table of Standard Elements ...
if there is no atomic number in row 2 of the Table of Standard Elements ...
```

Start of Chapter 16: Tables
Back to §16.6. Repeating through tables
Onward to §16.8. Blank columns

---

§16.8. Blank columns

An entire column of blank entries "--" is problematic:

Table 2 - Selected Elements

Element	Symbol	Atomic number	Atomic weight
"Hydrogen"	"H"	1	--
"Iron"	"Fe"	26	--
"Zinc"	"Zn"	30	--
"Uranium"	"U"	92	--

Inform is unable to work out what kind of value should go into the "atomic weight" column here, since it has no examples to guess from. We can get around this by writing in the name of a kind of value:

Table 2 - Selected Elements

Element	Symbol	Atomic number	Atomic weight
"Hydrogen"	"H"	1	a number
"Iron"	"Fe"	26	--
"Zinc"	"Zn"	30	--
"Uranium"	"U"	92	--

That top entry in the "atomic weight" column is also blank, but now Inform knows that anything put into the column in future will be a number.

If there are many rows, and perhaps several blank columns, it would become very tedious to have to keep typing out "--". So this is optional *at the end of a row*: it remains compulsory for a blank value appearing in between two values which aren't blank. This is the general idea:

Table 2 - Selected Elements

Element	Symbol	Atomic number	Density	Specific gravity
"Hydrogen"	"H"	1	a number	a number
"Iron"	"Fe"	26		
"Zinc"	"Zn"	30		
"Uranium"	"U"	92		

Start of Chapter 16: Tables
Back to §16.7. Blank entries
Onward to §16.9. Blank rows

---

§16.9. Blank rows

There is no difficulty about entirely blank rows: or rather, the only difficulty is once again that they are boring to type out. We can avoid the necessity by appending "with ... blank rows" at the foot of the table:

```
Table 2 - Selected Elements

Element      Symbol Atomic number Atomic weight
"Hydrogen"   "H"    1             a number
"Iron"       "Fe"   26            --
"Zinc"       "Zn"   30            --
"Uranium"    "U"    92            --
with 3 blank rows
```

(These words cannot be placed in between rows, but only at the bottom.) And indeed the table can start out completely empty:

```
Table 3 - Undiscovered Periodic Table

Element (text)  Symbol (text) Atomic number (a number) Atomic weight (a number)
with 92 blank rows
```

Blank rows are useful because they enable us to add new data to a table. In effect, they are invisible when not used. A repeat loop like

```
repeat through Table 3:
    ...
```

automatically skips blank rows, so it would initially do nothing at all. Similarly, choosing a "random" row will never choose a blank one.

A convenient way to test if a table contains non-blank rows is to use the built-in adjectives "empty" and "non-empty". So:

```
if the Undiscovered Periodic Table is empty, ...
```

tests whether all of its rows are blank; if even one cell contains a value then the table is "non-empty".


Start of Chapter 16: Tables


Back to §16.8. Blank columns


Onward to §16.10. Adding and removing rows


Example 273:  **If It Hadn't Been For...** A sound recording device that records the noises made by player and non-player actions, then plays them back on demand.

---

§16.10. Adding and removing rows

Writing in new rows is simple, once we can find space for them:

choose a/the/-- **blank row in**/from (table name)

This phrase chooses a row in the given table which is currently blank under every column. A run-time problem message is issued if no rows are blank. Example:

```
choose a blank row in Table 3;
now element entry is "Fluorine";
now symbol entry is "F";
now atomic number entry is 9;
now atomic weight entry is 19;
```

To avoid problem messages, it can be important to worry about free space. To that end we can not only find the number of rows (as we have already seen) but also the number currently blank and not blank:

number of blank rows in/from (table name) **... number**

This phrase produces the number of rows in the given table which are entirely blank (that is, blank under every column).

number of filled rows in/from (table name) **... number**

This phrase produces the number of rows in the given table which are not entirely blank (that is, at least one column has a value in this row).

"Filled" here really means "non-blank": a row can be filled in this sense even if only one of its values exists. Since every row is either blank or filled, it must be true that:

```
the number of blank rows in Table 3
the number of filled rows in Table 3
```

add up to "the number of rows in Table 3".

We've seen that blank entries can be filled with values using "now":

```
now symbol entry is "F";
```

But the same method can't be used to put blanks back, since a blank is not a value. Instead:

blank out (a table entry)

This phrase replaces the entry referred to with a blank, erasing any value previously stored there. Example:

```
choose row 1 in the Table of Fish Habitats;
blank out the salinity entry;
```

These more destructive phrases need a steady hand:

blank out the whole row

This phrase replaces the currently chosen row with blanks, erasing any value previously stored under any of the columns. Example:

```
choose row 1 in the Table of Fish Habitats;
blank out the whole row;
```

blank out the whole (table column) **column in** (table)

This phrase replaces the currently chosen column with blanks, erasing any value previously stored in any of the rows. Example:

```
blank out the whole salinity column in the Table of Fish Habitats;
```

blank out the whole of (table)

This phrase replaces every row of the currently chosen table with blanks, erasing any value previously stored anywhere in it. Example:

```
blank out the whole of the Table of Fish Habitats;
```

This is only really useful when a Table is being used to hold working space for some calculation or other.


Start of Chapter 16: Tables


Back to §16.9. Blank rows


Onward to §16.11. Sorting


Example 274:  **Odyssey** A person who follows a path predetermined and stored in a table, and who can be delayed if the player tries to interact with her.

---

§16.11. Sorting

The three ways to sort a table correspond loosely to the three different orders in which tables can be repeated through. First:

sort (table name) **in random order**

This phrase rearranges the rows of the given table so that the non-blank rows occur at the top, in a uniformly random order, and any blank rows at the bottom. Example:

```
sort the Table of Recent Monarchs in random order;
```

Secondly:

sort (table name) **in** (table column) **order**

This phrase rearranges the rows of the given table so that the non-blank rows occur at the top, so that the given column has ascending order, and any blank rows at the bottom. Example:

```
sort the Table of Recent Monarchs in accession order;
```

Ascending order means 1 up to 10, say, or A up to Z, with blank values coming last.

sort (table name) **in reverse** (table column) **order**

This phrase rearranges the rows of the given table so that the non-blank rows occur at the top, so that the given column has descending order, and any blank rows at the bottom. Example:

```
sort the Table of Recent Monarchs in reverse name order;
```

Descending order means 10 down to 1, say, or Z down to A, with blank values coming last.

How sorting is done depends on the contents of the column being sorted on. If it holds numbers then numerical order is used, with 2 coming before 7, and so on. (And similarly for real numbers, though the existence of infinities makes this more interesting.) If times are sorted then they are sorted from midnight to midnight, following the "is greater than" relation, not with 4 AM as the zero point, as with "is after".

If text is sorted then alphabetical order is used, though this doesn't always come out the way you might expect, because upper case and lower case letters are treated as different: A-Z come before a-z, and accented letters such as é come after the regular alphabet. (What's

happening here is that Inform is sorting on raw character values, not performing the full Unicode collation algorithm, which would be too slow at run-time.)

Note that blank values will always be placed below non-blank ones, and entirely blank rows last of all. This is true even if we use "reverse".

The method of sorting is "stable", that is, if two rows have the same value then they will stay the same way round in the sorted table, rather than being swapped over. For example, if we sort this into reverse index order:

```
Index Comment
1     "Originally row 1"
2     "Originally row 2"
2     "Originally row 3"
3     "Originally row 4"
```

then we get

```
Index Comment
3     "Originally row 4"
2     "Originally row 2"
2     "Originally row 3"
1     "Originally row 1"
```

As a result note that repeating through this sorted table goes through the original rows in order 4, 2, 3, 1; whereas repeating through the original table in reverse order goes through in order 4, 3, 2, 1. (This is all to explain the word "loosely" in the opening sentence of this section.)


Start of Chapter 16: Tables


Back to §16.10. Adding and removing rows


Onward to §16.12. Listed in...


Example 275:  **Jokers Wild** A deck of cards which can be shuffled and dealt from.

---

§16.12. Listed in...

Tables are especially useful for combining a run of basically similar rules in a simple and concise way. The "listed in" condition, as in

```
if the newfound object is an item listed in the Table of Treasures...
```

looks through a given table (here "table of treasures"), in a given column ("item"), to see if a given value is present ("the newfound object"). If this is successful, the row where it was found is automatically chosen; but if not, note that any existing row selection will be lost, so make use of the row only if the test succeeds.

We can similarly use "... listed in ..." in a description used when specifying an action. Thus:

```
After taking an item listed in the Table of Treasures:
    if there is no time entry:
```

```
        now the time entry is the time of day;
        increase the score by the value entry;
    say "Taken!"
```

This assumes a table in the following shape:

```
Table of Treasures

Item     Value Time
brooch   5     a time
tiara    8     --
coronet  10    --
```

In effect the table has allowed us to combine three very similar rules into one. The time column records the first time at which the item has been picked up, which starts out blank since at the start of play it has never been picked up. This enables us to award the appropriate number of points on the first occasion only.

 Start of Chapter 16: Tables

 Back to §16.11. Sorting

 Onward to §16.13. Topic columns

 Example 276:  **Noisy Cricket** Implementing liquids that can be mixed, and the components automatically recognized as matching one recipe or another.

---

§16.13. Topic columns

When double-quoted matter appears in a column of a table, Inform will normally treat that as text for printing out. The exception is when the column is called "topic", where it is treated as text for comparing against what the player has typed. There is really only one operation allowed with topic columns, the "...listed in..." construction, but fortunately it is the one most often needed.

Let us suppose that the Sybil has a penchant for telling passers-by which is the Greek muse for what. We might write:

```
After asking the Sybil about a topic listed in the Table of Sybil's Replies, say "The Sybil
declaims for a while, the gist being that the muse in question looks after [muse entry]."
```

We can then provide a simple table giving her responses:

```
Table of Sybil's Replies

Topic          Muse
"calliope"     "epic poetry"
"clio"         "history"
"erato"        "love poetry"
"euterpe"      "music"
"melpomene"    "tragedy"
"polyhymnia"   "sacred poetry"
"terpsichore"  "dancing"
"thalia"       "comedy"
"urania"       "astronomy"
```

"monica"	"tidiness"
"phoebe"	"massage"
"rachel"	"oval hair-cuts"

Topics can use the full range of abilities of the "understanding" system which Inform uses to parse text, and which will be the subject of a later chapter. For now, note that the Sybil's topics might equally include "flora/eve" (matching the single word "flora" or the single word "eve"), or something more elaborate such as:

```
"Bridget" or "Bridge" or "Bridget Jones"
```

See **Understand** *for the system Inform uses to parse text*

Start of Chapter 16: Tables

Back to §16.12. Listed in...

Onward to §16.14. Another scoring example

Example 277: **Merlin** A REMEMBER command which accepts any text and looks up a response in a table of recollections.

Example 278: **Questionable Revolutions** An expansion on the previous idea, only this time we store information and let characters answer depending on their expertise in a given area.

Example 279: **The Queen of Sheba** Allowing the player to use question words, and using that information to modify the response given by the other character.

---

§16.14. Another scoring example

```
To record (T - text) as achieved:
    choose row with a citation of T in the Table of Tasks Achieved;
    if there is no time entry:
        now time entry is the time of day;
        increase the score by the points entry.
```

The phrase above expects to see a table like this one:

```
Table of Tasks Achieved
```

Points	Citation	Time
1	"pride"	a time
3	"anger"	
2	"avarice"	
4	"envy"	
1	"lust"	
2	"gluttony"	
3	"sloth"	

The middle column records the tasks to be achieved, the first column records the points on offer for each: the final column, initially blank, will store the times at which the tasks are first achieved.

```
Before eating, record "gluttony" as achieved.
```

The first time we record "gluttony" as achieved, 2 points will be awarded and the time will be logged in the Table, but on all subsequent occasions nothing will happen. So the combination of the phrase and the Table will look after a scoring system based on achieving specific goals (probably not the seven deadly sins, of course). We can, if we choose, use the same system to display a log of recent accomplishments:

```
repeat through the Table of Tasks Achieved in reverse time order:
    say "[time entry]: [citation entry] ([points entry])."
```


Start of Chapter 16: Tables


Back to §16.13. Topic columns


Onward to §16.15. Varying which table to look at


Example 280:  **Goat-Cheese and Sage Chicken** Implementing a FULL SCORE command which lists more information than the regular SCORE command, adding times and rankings, as an extension of the example given in this chapter.

---

§16.15. Varying which table to look at

So far, we have always used fixed table names when referring to tables: for instance in source like "sort the Table of Recent Monarchs in accession order", we refer to the "Table of Recent Monarchs", a definite and explicitly named table.

With a little care, however, we are allowed to have variables which themselves hold the names of tables. This opens up the possibility of more elaborate ways of storing and interconnecting information in table form, but is probably best avoided until it becomes necessary.

For example, suppose we have two different tables with the same basic structure:

```
Table 1 - Nifty Opening Plays in US Scrabble

word       score
"muzjiks" 128
```

```
Table 2 - Nifty Opening Plays in UK Scrabble

word       score
"quartzy"  126
"squeezy" 126
```

We could then record which one of these tables to use in a variable:

```
The lexicon is a table name that varies. The lexicon is Table 1.
```

Note that for this purpose, the kind of value is a special kind called "table name", not "table". (The word "table" already has too many meanings and we must be careful to avoid ambiguities here.) We could make use of this as follows, for instance:

```
To flip tables:
    say "You exchange dictionaries, lexically crossing the Atlantic. ";
    if the lexicon is Table 1, now the lexicon is Table 2;
    otherwise now the lexicon is Table 1;
    choose a random row in the lexicon;
    say "Did you know that according to [the lexicon], [word entry] scores [score entry]?"
```

which produces text such as

```
You exchange dictionaries, lexically crossing the Atlantic. Did you know that according to
Table 1 - Nifty Opening Plays in US Scrabble, muzjiks scores 128?
```

 Start of Chapter 16: Tables

 Back to §16.14. Another scoring example

 Onward to §16.16. Defining things with tables

 Example 281:  **Farewell** People who respond to conversational gambits, summarize what they said before if asked again, and provide recap of conversation that is past.

---

§16.16. Defining things with tables

Suppose we need to create a collection of items which differ in their properties, but are basically part of a larger pattern. For instance, here we set up what we need to make a collection of coloured shirts:

```
A jersey is a kind of thing. A jersey is wearable. A jersey has a number called year
established. A jersey has a text called citation. The description of a jersey is "Since [year
established], the Tour de France has awarded this jersey to the [citation]."
```

Now we have the pattern, but making the actual shirts is tedious and repetitive:

```
The yellow jersey is a jersey. The year established of the yellow jersey is 1919. The
citation of the yellow jersey is "race leader". The polkadot jersey...
```

And so on. Instead, we can use a table to abbreviate all of this:

```
"Tour des Maillots"

The Staging Area is a room. A jersey is a kind of thing. A jersey is wearable. Some
jerseys in the Staging Area are defined by the Table of Honorary Jerseys. The
description of a jersey is "Since [year established], the Tour de France has awarded this
jersey to the [citation]."

Table of Honorary Jerseys
```

jersey	year established	citation
a yellow jersey	1919	"race leader"
a polkadot jersey	1933	"King of the Mountains"
a green jersey	1953	"highest point scorer on sprints"
a white jersey	1975	"best cyclist aged 25 or less"

The first column provides names for the new things to be created. Subsequent columns provide property values. Note that we did not need to say that jerseys have a number called "year established" because Inform is able to infer this from the column heading and the presence of numbers in the column; similarly for "citation". Lastly, note that if any entry is blank (written "--") then that particular property is simply not set for that particular item.

Note that Inform reads articles such as "the" or "a" in the first column just as it would when something is created with any other sentence.

It's even possible to define kinds this way, though it's rare to need to create many kinds at once. (See the worked example "Reliques of Tolti-Aph" at the Inform website. There's no special syntax needed: rather than saying "Some jerseys are defined by..." we would say "Some kinds of jersey are defined by...")


Start of Chapter 16: Tables


Back to §16.15. Varying which table to look at


Onward to §16.17. Defining values with tables


Example 282:  **Sweeney** A conversation where each topic may have multiple questions and answers associated with it, and where a given exchange can lead to new additions to the list.


Example 283:  **Introduction to Juggling** Assortment of equipment defined with price and description, in a table.

---

§16.17. Defining values with tables

Just as we can define many similar things (or kinds) using a table, we can also define a whole run of new values. Again, this avoids unnatural prose like

```
The chemical elements are Hydrogen, Helium, Lithium, ..., and Ununquadium.
```

We can give these new values properties, too. For example:

```
Solar distance is a kind of value. 1000 AU specifies a solar distance. Planet is a kind of
value. The planets are defined by the Table of Outer Planets.

Table of Outer Planets

    planet    semimajor axis
    Jupiter   5 AU
    Saturn    10 AU
    Uranus    19 AU
    Neptune   30 AU
    Pluto     39 AU
```

creates five values of the kind "planet", but it also makes a property called "semimajor axis" which belongs only to these five values. Thus:

```
say "Pluto orbits at [semimajor axis of Pluto]."
```

produces "Pluto orbits at 39 AU." We can both use and change this value:

```
Praying is an action applying to nothing. Understand "pray" as praying.
Instead of praying:
    now the semimajor axis of Pluto is 1 AU;
    say "Your prayers are answered, and the Almighty moves Pluto in closer to the fire."
```

Similar properties would be made for each column of the table after the first (there can be any number of properties, including none). Because the values are created first, before the rest of the table is gone through, we can even use "planet" as one of the values of properties:

```
Table of Outer Planets

planet    semimajor axis    centre of government
Jupiter   5 AU             Jupiter
Saturn    10 AU            Saturn
Uranus    19 AU            Saturn
Neptune   30 AU            Pluto
Pluto     39 AU            Pluto
```

All of this is intended to be closely parallel to defining a whole run of things, such as the coloured jerseys, using a table, but there are two important restrictions: firstly, when a kind of value is defined by table, the table must contain all of its possible values; and secondly, the column names (after the first) cannot coincide with names of any properties held by any other value (or thing, for that matter). So it is a good idea to give the columns very specific names ("centre of government") rather than vague names which might cause clashes elsewhere ("owner").

Two technical footnotes. In a table used to define a kind of value, blank entries are not left blank: they are filled in with suitable default values. For instance, if the semimajor axis column had been all "--"s except for listing Neptune at "30 AU", say, Inform would deduce that the column was meant to hold a value of kind "solar distance", and would set the solar distances for all of the other planets to be "0 AU". It does this to ensure that "solar distance of P" exists for any planet P.

The second technical note is that we must not sort such a table, because it is used during play to store the properties, and if it were to get rearranged then so would the properties be - with probably disastrous results.


Start of Chapter 16: Tables


Back to §16.16. Defining things with tables


Onward to §16.18. Table continuations

---

§16.18. Table continuations

A table is an arrangement for putting information together concisely in a single place, so it might seem odd that we sometimes need to divide it up: but once in a while, we do. Suppose we have:

```
Table of Outer Planets

planet    semimajor axis
```

```
Jupiter   5 AU
Saturn    10 AU
Uranus    19 AU
Neptune 30 AU
Pluto     39 AU
```

But then someone in Chile with a telescope the size of God's own teacup notices something a long, long way out, and the newspapers get terribly excited. We can write an addendum:

```
Table of Outer Planets (continued)

planet   semimajor axis
Orcus    39 AU
Quaoar 43 AU
Xena     68 AU
Sedna   524 AU
```

This may seem unnecessary - why not simply add extra rows to the original table? - but it allows us to split the table between different parts of the source text, if we want to, or to continue a table which exists only in an extension. (Thus if we were using an extension which involved the planets, and had a table like this one, we would be able to add new planets without changing the extension.)

The name for the continuation must be identical to the original. The continuation has no existence in its own right: Inform simply splices the two (or more) pieces together, exactly as if the table were all in one piece at the place where it first occurred. Thus the above creates only one table, the "Table of Outer Planets", with nine rows. Each column in the continuation must exist in the original, but not every column need be given: those omitted are filled with blank entries. The columns need not be in the same order. Both original and continuations are allowed to quote a number of blank rows: if so, the combined total is used.

At time of writing the International Astronomical Union has not yet consented to name 2003 UB313 after Xena, the Warrior Princess, but this is surely only a bureaucratic delay. (Footnote: on 24 August 2006 it was demoted to dwarf planet status, like the luckless Pluto, and on 13 September renamed Eris; though its moon's official name, Dysnomia, is an ingenious double-meaning to do with the name of Xena's actress, Lucy Lawless.)


Start of Chapter 16: Tables


Back to §16.17. Defining values with tables


Onward to §16.19. Table amendments


Example 284:  **Food Network Interactive** Using a menu system from an extension, but adding our own material to it for this game.

---

§16.19. Table amendments

Tables can have amendments as well as continuations. The arrangement is much the same: a supplementary table supplies new rows for the original table. But instead of adding the new rows at the end of the original, as a continuation would, an amendment replaces matching rows in the original. (So the original stays the same size.)

The amendment table must have exactly the columns of the original and in the same order. Moreover, each row in the amended table must match exactly one row in the original. For instance:

```
Table of Plans

moment   outcome
10 AM    "takeover of Mars"
11:30 AM "canals reflooded"
11:45 AM "chocolate bar production doubled"

Table of Plans (amended)

moment   outcome
11:45 AM "volcanic cave production doubled"
```

creates a three-row Table of Plans, with reference to the chocolate bars struck out.

Amendment rows may be given in any order. The process of matching a row begins at the left-most column: Inform tries to see if any single row in the original table has a matching entry. If none does, a Problem is issued. If more than one do, Inform then looks at the second column, and so on. For instance:

```
Enthusiasm is a kind of value. The enthusiasms are pumped, wired and languid.

Table of Mental States

feeling   extent consequence
pumped 1        "you feel able to run for your life"
pumped 2        "you feel able to run for President"
wired     1     "you feel able to run"
languid   1     "you feel"

Table of Mental States (amended)

feeling   extent consequence
pumped 2        "you feel able to run for the Nebraska State Legislature"
```

Here the amendment is made to the second row of the original table. The value in the leftmost column, "pumped", matches two rows in the original, so Inform moves on to the next column, reads "2", and finds that only one row in the original still qualifies - so that is the one replaced.

For the present, at least, the columns used for matching may only contain: numbers, times, objects, action names, activities, figure names, sound names, truth states and any new kinds of value or units which have been declared.

Start of Chapter 16: Tables

Back to §16.18. Table continuations

Onward to Chapter 17: Understanding: §17.1. Understand

Example 285: ★★ **Trieste** Table amendment to adjust HELP commands provided for the player.

Examples from Chapter 16: Tables

Start of this chapter

Chapter 17: Understanding

Indexes of the examples

271 ★★★ **Example** **Dubai** *RB*

An elevator which connects any of 27 floors in a luxury hotel.

The problem of implementing an elevator that opens onto a large number of floors often challenges novice interactive fiction authors. It also raises a fundamental design problem -- how to implement a large building in an interesting way. It is tempting to write a hotel with an elevator and innumerable tedious and identical floors just for the sake of realism; in many cases it is a better idea simply to omit any locations that contribute nothing to the story or the play of the game.

In charity, though, let us assume that the author has a legitimate reason for wanting to implement an elevator that opens onto some generic floors. We will go whole-hog, and set this in the world's tallest hotel: Burj al-Arab, Dubai.

```
"Dubai"

The Burj al-Arab Lobby is a room. "The 202-suite Burj al-Arab - or Tower of the
Arabs - stands 321 metres (1,060 feet) high, and floats on its own man-made
island. It is shaped like the sail of a boat; just crossing the private bridge to reach
this place set you back $55."
```

(Since our budget did not run to visiting Burj al-Arab, the descriptions place implicit faith in the hotel's website.)

```
The Assawan Spa is a room. "Treatment rooms, hydrotherapy baths, oriental
massage, stand-up solarium, sauna, steam rooms and jacuzzi, two swimming
pools, squash court, two fully equipped fitness studios and an aerobics floor. To
the south is a shopping area, for those who do not find exercise adequately
therapeutic."

The Shopping Area is south of Assawan Spa. "In a setting that would make a
poet sigh, you may enjoy the services of (among others) Bulgari, Black Pearl
Caviar, Albarajeel Carpet Shop, Abdul Samed Al Qurashi (amber and Arabic
perfumes), Dianoor (jewellery), and the Commercial Bank of Dubai."
```

The Al Falak Ballroom is a room. "A palatial, two-tiered, domed ballroom crowned with a unique crystal chandelier."

Sahn Eddar is a room. "At the base of the world's tallest atrium, the Sahn Eddar restaurant offers light fine fare and Afternoon Tea. At the center, a 32-meter water column leaps toward the roof of the atrium."

Al Mahara is a room. "After the elevator, you must take a three-minute virtual submarine voyage to reach this seafood restaurant. A magnificent oval aquarium, full of sharks, is visible from every table."

Al Iwan is a room. "Middle-eastern food in an environment of dramatic black, red, and gold."

Al Muntaha is a room. "A top-floor restaurant with a magnificent view: the name means the ultimate or the highest, suitable for a place that stands at 200 metres above the Arabian Gulf. It offers modern European cuisine; and just off to the south is the Skyview Bar."

The Skyview Bar is south of Al Muntaha. "'A wonderful location for pre- and post dinner drinks,' claims the hotel's brochure, and certainly you can't fault the view."

The Juna Lounge is a room. "Two humidors that offer one of the largest selections of the finest Havana cigars in Dubai."

The Lift is a room. "This is not a mere elevator: it is an express panoramic lift, traveling six meters a second, and capable of taking you from lobby to the rooftop restaurant in an astonishingly short time. The illuminated number above the door says [current level of the Lift] - though you can make it move by pressing a numbered button."

The Presidential Suite is a room. "Astonishingly, this is not the most elaborate or expensive of suites; there is another, the Royal, upstairs of here."

The Royal Suite is a room. "The brochure described this as 'the last word in luxury,' and you have to admit that it is certainly the last word in [italic type]something[roman type]. A vast carpet of patterned red and gold stretches from you to the sofa; beyond which, in the distance, you make out several bedrooms and bathrooms outfitted in Carrera marble. There is also, of course, a private cinema."

Table of Floors

level	floor
0	Al Mahara
1	Burj al-Arab Lobby
2	Al Iwan
3	Juna Lounge
4	Sahn Eddar
15	Al Falak Ballroom
18	Assawan Spa
24	Presidential Suite
25	Royal Suite
27	Al Muntaha

The elevator exterior is a backdrop. It is not scenery. The initial appearance of the elevator exterior is "You can enter the elevator here." It is in Generic Floor, Al

Mahara, the Lobby, Al Iwan, Juna Lounge, Sahn Eddar, Al Falak, Assawan Spa, Presidential Suite, Royal Suite, and Al Muntaha.

Before entering the elevator exterior, try going inside instead.

Instead of going inside in the presence of the elevator exterior:
    if there is a level corresponding to a floor of the location in the Table of Floors:
        let the present level be the level corresponding to a floor of the location in the Table of Floors;
        now the current level of the Lift is the present level;
    otherwise:
        now the current level of the Lift is the current level of Generic Floor;
    move the player to the Lift.

The Lift has a number called current level. The current level of the Lift is 1. Instead of going up in the Lift: say "You'll have to select a specific floor; your options range from 0 to 27." Instead of going down in the Lift: try going up instead. The Lobby is outside from the Lift.

Before going outside in the Lift:
    if there is a floor corresponding to a level of the current level of the Lift in the Table of Floors:
        let the other place be the floor corresponding to a level of the current level of the Lift in the Table of Floors;
        move the player to the other place instead;
    otherwise:
        now the current level of the Generic Floor is the current level of the Lift;
        move the player to the Generic Floor instead.

The Generic Floor is a room. The Generic Floor has a number called current level. The printed name of the Generic Floor is "Floor [current level of the Generic Floor]". "A long hallway between suites, some of which run up to $15,000 a night."

Understand "push [number]" as pressing button. Understand "push [number] button" as pressing button. Understand "push button [number]" as pressing button. Pressing button is an action applying to one number.

Check pressing button:
    if the player is not in the Lift, say "You cannot control the express panoramic lift unless you are yourself inside." instead;
    if the number understood is the current level of the Lift, say "The lift pings politely and reopens its doors, since you are already on floor [number understood]." instead;
    if the number understood is greater than 27, say "There are only 27 floors." instead;
    if the number understood is less than 0, say "You cannot go below the ground floor in this elevator." instead.

Carry out pressing button:
    now the current level of the Lift is the number understood;
    say "You press button [the number understood]. The lift whirs into action and moves to the correct level."

```
Test me with "press 3 / in / press button 3 / look / out / in / press 27 / out / s / in /
n / in / press 15 / out / in / press 18 / out / s / in / n / in / press 4 button / out".
```

This will all work very well, unless the player has portable objects; in that case, anything he drops on the Generic Floor will be there every time he goes back -- whether it's masquerading as Floor 6 or Floor 23. There are there are two ways round this -- (i) the cheeky way. When we drop something, the unobtrusive yet ever-vigilant maids pick it up and take it down to the Foyer's lost property office; and (ii) the super-duper way, in which things are moved out of play but with their floor numbers remembered, so that the scenario can be reconstructed each time. (i) is probably in fact the more true-to-life, considering the hotel's boasts about its service, but we will demonstrate both methods.

Here is the version with vigilant maids:

```
The player carries a shopping bag. In the bag are a diamond necklace, a small
rug, and a jar of caviar.

Carry out pressing button:
    now every portable thing enclosed by the Generic Floor is in the Office.

The Office is south of the Lobby. "Here the maids collect everything abandoned
by careless guests." The printed name of the Office is "Lost and Found Office".

Test maids with "in / press 6 / out / drop all / in / out / in / press 23 / out / in /
press 1 / out / s / get all".
```

Notice that we tie the maid service to the pressing of the lift button, so that if the player just goes into the lift and comes out again, the maids will not have had a chance to clear his possessions.

Alternatively:

```
The player carries a shopping bag. In the bag are an evening gown, a bolero
jacket, and tickets to the Wild Wadi Animal Park.

Carry out pressing button when something is in the Generic Floor:
    repeat with item running through portable things in the Generic Floor:
        clear the item;
    repeat with item running through portable things enclosed by the Generic
Floor:
        clear the item.
```

The "enclosed by" line clears even things left on, say, small un-portable side-tables or whatever; but because we do "in" first, we make sure to move any containers or supporters undisturbed. The next bit could be more tidily incorporated into our previous "before going outside" rule, but since we are writing this code to be optionally pasted onto the end of the first bit, we'll express the rule separately:

```
Before going outside in the Lift when something is in Limbo:
    unless there is a floor corresponding to a level of the current level of the Lift in
the Table of Floors:
        repeat with item running through things in Limbo:
```

```
            if the current level of the item is the current level of the Lift, move the
item to the Generic Floor.

A thing has a number called current level.

To clear (item - a thing):
    now the current level of the item is the current level of the Lift;
    move item to Limbo.

Limbo is a room.

Test management with "get all from bag / in / press 22 / out / drop tickets / in /
press 23 / out / drop gown / in / press 22 / out / get tickets / in / press 23 / out /
get gown".
```

And now we have a situation in which the player's valuables are left untouched wherever in the hotel he happens to abandon them.

Incidentally, this example was almost set in an entirely different location: the largest hotel in the world may some day be the Ryugyong Hotel in Pyongyang, North Korea, with its 105 floors -- but for some years construction halted at the creation of the building's huge concrete shell.

272

Example **Port Royal 4**

A cell window through which the player can see people who were in Port Royal in the current year of game-time.

RB

Our protagonist is imprisoned in Port Royal, waiting out his years, and sometimes through the window of his cell he is able to see someone.

We are, however, obsessive about historical accuracy, so we provide a table of people who really lived in the city, together with the year in which their existence is attested. We want these people to appear in the description only in the year when they are known to have been present. (After all, mortality was high in Port Royal and new people were constantly arriving, so someone's presence one year is no guarantee of their continued existence the next.)

```
"Waiting for Godot, Chyrurgeon"

New New Prison is a room. "You have a not very interesting room. Through the
window you see passing [current denizen]."

Instead of waiting:
    increment the current year;
    say "It is now the year [the current year].";
    try looking.

When play begins: now the right hand status line is "[current year]".
```

```
Every turn:
    if the current year is 1692:
        say "It turns out you have remained imprisoned until the great earthquake
of 1692! Oops.";
        end the story.

Current year is a number that varies. The current year is 1664.

To say current denizen:
    repeat through the Table of Occupations and People:
        if the date attested entry is the current year:
            say "[nickname entry] [family entry], [trade entry]";
            blank out the whole row;
            rule succeeds;
    say "absolutely no one".
```

It is possible to look up a row corresponding to, say, a specific year value using "listed in", but repeat through is convenient here because we know that we will never wind up trying to print entries when no row can be successfully selected.

```
Table of Occupations and People
```

Trade	nickname	family	Date attested
"architect"	"Robert"	"Snead"	1684
"baker"	"William"	"Wingar"	1683
"barber"	"William"	"Alcocke"	1676
"blacksmith"	"William"	"Davidson"	1679
"bricklayer"	"Samuel"	"Richardson"	1683
"butcher"	"John"	"Dennis"	1676
"carpenter"	"John"	"Albert"	1675
"cabinet-maker"	"Robert"	"Avis"	1666
"joiner"	"Peter"	"Bartaboa"	1666
"chandler"	"William"	"Bates"	1674
"chyrurgeon"	"William"	"Axtell"	1674
"chyrurgeon"	"Thomas"	"Trapham"	1678
"combmaker"	"Paul"	"Bennett"	1673
"cooper"	"James"	"Hall"	1676
"cooper"	"Henry"	"Pullein"	1675
"cordwainer"	"George"	"Barnard"	1675
"cordwainer"	"Edward"	"Skannon"	1680
"cordwainer"	"John"	"Wilmott"	1675
"drugster"	"William"	"Mathews"	1682
"fisherman"	"Richard"	"Collingwood"	1674
"glazier"	"Thomas"	"Hudson"	1684
"goldsmith"	"Richard"	"Lord"	1677
"gunsmith"	"Stephen"	"Massey"	1664
"hatmaker"	"John"	"Rosewell"	1683
"ivory turner"	"William"	"Clifton"	1691
"labourer"	"John"	"Dennis"	1674
"limeburner"	"John"	"Hardwick"	1675
"mariner"	"Alexander"	"Bailing"	1680
"mariner"	"Thomas"	"Bowtell"	1675
"mariner"	"Peter"	"Claiton"	1675
"mariner"	"Joseph"	"Cupid"	1672
"mariner"	"Michael"	"Dunn"	1675
"mason"	"John"	"Stone"	1673
"merchant"	"John"	"Agard"	1680
"merchant"	"David Lopez"	"Narbona"	1674
"merchant"	"Abraham"	"Langford"	1675
"merchant"	"John"	"Sweeting"	1675
"merchant"	"Charles"	"Knight"	1680
"merchant"	"Cornelius"	"Vandananker"	1670
"merchant"	"Moses Jesurum"	"Cordova"	1675
"pewterer"	"Simon"	"Benning"	1667

"pipemaker"	"John"	"Pope"	1680
"porter"	"George"	"Paul"	1670
"poulterer"	"Richard"	"Jeffreys"	1677
"sailmaker"	"Adam"	"Brewer"	1671
"schoolmaster"	"Peter"	"Bird"	1677
"shipwright"	"William"	"Cavell"	1676
"tailor"	"William"	"Case"	1676
"tailor"	"Pewter"	"Ebden"	1683
"waterman"	"William"	"Brocke"	1674
"waterman"	"Joel"	"Clements"	1668
"wherryman"	"John"	"Grant"	1669
"victualler"	"Barnaby"	"Adams"	1675
"vintner"	"Gabriel"	"Adkins"	1668
"tavern-keeper"	"John"	"Baldwin"	1670
"tavern-keeper"	"Mary"	"Dayton"	1664
"tavern-keeper"	"James"	"Turpin"	1679
"tavern-keeper"	"Christopher"	"Mayham"	1664

Test me with "wait / wait / wait / wait / wait / wait / wait / wait / wait".

273

Example **If It Hadn't Been For...**

RB

A sound recording device that records the noises made by player and non-player actions, then plays them back on demand.

We start out by giving ourselves a capacious recording device:

```
"If It Hadn't Been For..."

The digital recorder is a device. The description is "A noise-activated recorder, which time-stamps each recording segment. It has space for about 60 short recordings."

Every turn:
    if the digital recorder is switched on and the number of blank rows in the Table of Recorded Content is 0, now the recorder is switched off.

Table of Recorded Content

time stamp    sound
a time        some text
with 60 blank rows.
```

And most of what follows is attaching sounds to various events. (We could have made noises associated with all the actions, but for simplicity we stuck to a few.)

The thing to note here is that the recording happens as part of Carry out, not as part of Report, so sounds will be recorded even when they are the result of non-player action when the player is not even in the room.

```
Carry out opening something in the presence of the switched on recorder:
    record "A creaking noise, as of something being opened."

Carry out someone opening something when the switched on recorder can see the noun:
```

```
    record "A creaking noise, as of something being opened."

Carry out closing something in the presence of the switched on recorder:
    record "A creaking followed by a slam."

Carry out someone closing something in the presence of the switched on
recorder:
    record "A creaking followed by a slam."

Carry out someone going to a room (called destination) in the presence of the
switched on recorder:
    if the destination is the holder of the recorder, record "Footsteps, growing
louder.";
    otherwise record "Footsteps, fading out."

Carry out going to a room (called destination) in the presence of the switched on
recorder:
    if the destination is the holder of the recorder, record "Footsteps, growing
louder.";
    otherwise record "Footsteps, fading out."

Carry out someone eating something in the presence of the switched on
recorder:
    record "Loud uncouth chewing sounds."

Carry out eating something in the presence of the switched on recorder:
    record "Distant muffled chewing sounds."

To record (noise - some text):
    if the number of blank rows in the Table of Recorded Content is 0, rule
succeeds;
    choose a blank row in the Table of Recorded Content;
    now time stamp entry is the time of day;
    now sound entry is noise.

Understand "play [something]" as listening.

Instead of listening to the recorder:
    if the number of filled rows in the Table of Recorded Content is 0, say "The
recorder remains blank." instead;
    repeat through the Table of Recorded Content:
        say "[line break][time stamp entry]: [sound entry]";
    say paragraph break.

The Haunted House is a room. The squeaky cupboard is an openable enterable
closed fixed in place container in the House. The ghost is a man in the
cupboard. The Lawn is outside from the Haunted House.

Instead of opening the closed cupboard when the ghost is in the cupboard: say
"The cupboard stubbornly refuses to open."

Every turn when the player is not in the House:
    if the ghost is in the cupboard:
        try the ghost exiting;
```

```
    otherwise if the cupboard is open:
        try the ghost closing the cupboard.

Before someone exiting when the person asked is in a closed container (called
the trap):
    try the person asked opening the trap.

Before someone entering a closed container: try the person asked opening the
noun.

Before going to the House when the House contains the ghost:
    try the ghost entering the cupboard;
    try the ghost closing the cupboard.

The player carries the recorder, chips, and a sandwich. The sandwich is edible.
The chips are edible.

Carry out someone eating the chips in the presence of the switched on recorder:
record "An incredible racket of a packet being opened." Carry out eating the
chips in the presence of the switched on recorder: record "An incredible racket of
a packet being opened."

Test me with "open cupboard / drop recorder / switch it on / eat chips / out / wait /
wait / wait / in / switch recorder off / play recorder".
```

Now we're at liberty to record evidence of the ghost getting out of the cupboard and getting back in, while we ourselves stand about on the lawn.

---

274 **Example Odyssey** *RB*

A person who follows a path predetermined and stored in a table, and who can be delayed if the player tries to interact with her.

```
"Odyssey"

Corinth is a room. Athens is east of Corinth. Epidaurus is southeast of Corinth
and east of Mycenae. Mycenae is south of Corinth. Olympia is west of Mycenae.
Argos is south of Mycenae. Thebes is northwest of Athens. Pylos is south of
Olympia. Sparta is east of Pylos and south of Argos. Delphi is northwest of
Thebes.

Athena is a woman in Athens.
```

Athena will proceed, unless delayed, through a list of locations stored in a simple table. Rather than using Inform's route-finding abilities ("the best route from..."), we simply move Athena from one location to the next, not even using the going action: she moves in mysterious ways, as befits a goddess.

```
Table of Athena's Movement
destination
Thebes
```

```
Delphi
Thebes
Athens
Corinth
Mycenae

Every turn when Athena is active:
    repeat through the Table of Athena's Movement:
        let last space be the location of Athena;
        if Athena can be seen by the player, say "Athena heads to [the destination
entry].";
        move Athena to destination entry;
        if Athena can be seen by the player, say "Athena arrives from [the last
space].";
        blank out the whole row;
        break.
```

By blanking out the table line by line, we make sure that we never lose our place in the path.

Since we want the player to be able to talk to Athena, we need a way to stall her in her path, as well.

```
Athena can be active or passive. Athena is active.

Before doing something to Athena:
    now Athena is passive;
    say "Athena waits around patiently, though you can tell she would like to
leave..."

Instead of telling Athena about something:
    say "She watches you patiently as though to say that she already knows."

Instead of asking Athena about something:
    say "Her response is inscrutably ancient and Greek. Afterwards you
remember only the flash of bright eyes."
```

Finally, we do need to wake Athena up again if she has become passive. The following rule will occur after the movement rule just because of code ordering, though we could make matters more explicit if we needed to:

```
Every turn when Athena is passive:
    now Athena is active.

Test me with "east / northwest / wait / examine athena / wait".
```

---

275  **Example** **Jokers Wild** *RB*

A deck of cards which can be shuffled and dealt from.

Suppose we want a deck of cards which the player can shuffle and draw from. Our first (rather tedious) task is merely to set up the deck as a table:

"Jokers Wild"

Suit is a kind of value. The suits are hearts, clubs, diamonds, and spades.

Table of Cards

suit	value
diamonds	1
diamonds	2
diamonds	3
diamonds	4
diamonds	5
diamonds	6
diamonds	7
diamonds	8
diamonds	9
diamonds	10
diamonds	11
diamonds	12
diamonds	13
spades	1
spades	2
spades	3
spades	4
spades	5
spades	6
spades	7
spades	8
spades	9
spades	10
spades	11
spades	12
spades	13
hearts	1
hearts	2
hearts	3
hearts	4
hearts	5
hearts	6
hearts	7
hearts	8
hearts	9
hearts	10
hearts	11
hearts	12
hearts	13
clubs	1
clubs	2
clubs	3
clubs	4
clubs	5
clubs	6
clubs	7
clubs	8
clubs	9
clubs	10
clubs	11
clubs	12
clubs	13

We're going to describe the higher numbers as face cards, so it helps to write a new "to say" phrase.

```
To say (count - a number) as a card value:
    choose row with a value of count in the Table of Value Names;
    say "[term entry]".

Table of Value Names

value term
1     "ace"
2     "deuce"
3     "three"
4     "four"
5     "five"
6     "six"
7     "seven"
8     "eight"
9     "nine"
10    "ten"
11    "jack"
12    "queen"
13    "king"
```

Now we get the shuffling of the deck from "sort in random order", so:

```
Understand "shuffle" as shuffling. Shuffling is an action applying to nothing.

Carry out shuffling:
    sort the Table of Cards in random order;
    say "You expertly rearrange the cards.".

When play begins:
    sort the Table of Cards in random order.
```

This will continue to work properly even as the deck is partially depleted. Speaking of which, suppose we want the player to be able to toss the cards one-by-one into a hat. They are going to need to be removed from the deck, so:

```
Understand "toss" or "toss a card" or "toss card" as tossing.

Tossing is an action applying to nothing.

Check tossing:
    if the number of filled rows in the Table of Cards is 0, say "The deck is empty."
instead.

Carry out tossing:
    repeat through the Table of Cards:
        let new value be value entry;
        let new suit be suit entry;
        say "You throw the [value entry as a card value] of [suit entry] at the top hat,
and [if a random chance of 1 in 3 succeeds]hit[otherwise]miss[end if].";
        blank out the whole row;
        rule succeeds.
```

If we wanted to simulate a slightly more stimulating game, we could instead have a second table to represent the player's hand of cards and record each card drawn. That would get long for the purposes of example, however, so instead we will just admit that the player's life is an empty husk of existence:

```
The Empty Room is a room. "It has come to this: sitting on the bare floor of
Lulu's apartment with nothing to amuse you but a deck of cards and the top hat
from last year's act. You reckon [the number of filled rows in the Table of Cards
in words] cardtosses are all that stand between you and the utter pointlessness
of existence.

Once again you curse Lulu for running off with that joker."

The player is carrying the deck of cards. The top hat is an open container in the
Empty Room. It is scenery.

Test me with "toss / again / again / again / again / again / again / again".
```

---

276  **Example Noisy Cricket** *RB*

Implementing liquids that can be mixed, and the components automatically recognized as matching one recipe or another.

Our previous experiments into liquid have not dealt with the possibility of mixing components, but that is because for most games, tracking the details of mixture is overkill.

But let's suppose that this time we do want to have mixed liquids; moreover, we want a way to describe the mixtures to the player inventively, so that if he hits specific combinations those combinations are recognized: calling the result a martini, say, rather than just "a mixture of vodka and vermouth".

The implementation that follows relies on a fairly simple idea from linear algebra. Any given liquid can be expressed as a vector in N-space, where N is the number of available ingredients and the length of the vector depends on how much of each ingredient is used; then we find the recipe that best describes the liquid by taking the dot product of our liquid vector with a bunch of sample vectors and selecting the one with the largest result.

If this does not make sense, don't worry: it's not necessary to understand the idea to use the code.

Any implementation involving a large number of place values is always a bit challenging in integer arithmetic. This examples assumes that no bodies of liquid will ever be very large, and that the proportions of ingredients in a mixture will not be vastly askew. (No 20-parts-to-1 proportions, for instance.) This probably works reasonably well for the cocktails that we make the basis of the example.

```
"Noisy Cricket"
```

Part 1 - Volumes and Mixtures

A volume is a kind of value. 15.9 fl oz specifies a volume with parts ounces and tenths (optional, preamble optional).

A fluid container is a kind of container. A fluid container has a volume called a fluid capacity. A fluid container has a volume called creme de menthe volume. A fluid container has a volume called vodka volume. A fluid container has a volume called cacao volume.

The fluid capacity of a fluid container is usually 12.0 fl oz. The creme de menthe volume of a fluid container is usually 0.0 fl oz. The vodka volume of a fluid container is usually 0.0 fl oz. The cacao volume of fluid container is usually 0.0 fl oz.

To decide what volume is the current volume of (item - a fluid container):
    let total be the creme de menthe volume of the item;
    increase total by the vodka volume of the item;
    increase total by the cacao volume of the item;
    decide on total.

Instead of examining a fluid container:
    if the noun is empty,
        say "You catch just a hint of [the nominal descriptor of the noun] at the bottom.";
    otherwise
        say "[The noun] contains [current volume of the noun in rough terms] of [adjectival descriptor of the noun] [nominal descriptor of the noun]."

Adjectival descriptor is a kind of value. The adjectival descriptors are strong, chocolatey, minty, perfect, and pure.

Nominal descriptor is a kind of value. The Nominal descriptors are creme de menthe, vodka, creme de cacao, grasshopper, chocolate vodka, mint vodka, chocolate martini, mintini, chocolate mint martini.

Our table of mixtures is expressed in parts: so if a recipe contains one part X and two parts Y, we would put "1" in the first column and "2" in the second column.

Table of Mixtures

rating	creme de menthe comp	vodka comp	cacao comp	adjectival descriptor	nominal descriptor
0.0 fl oz	1	0	0	minty	creme de menthe
0.0 fl oz	0	1	0	chocolatey	vodka
0.0 fl oz	0	0	1	chocolatey	creme de cacao
0.0 fl oz	1	2	0	chocolatey	mintini
0.0 fl oz	1	0	1	chocolatey	grasshopper
0.0 fl oz	0	2	1	chocolatey	chocolate martini
0.0 fl oz	0	3	1	chocolatey	chocolate vodka
0.0 fl oz	1	3	0	chocolatey	mint vodka
0.0 fl oz	1	2	1	chocolatey	chocolate mint martini

A fluid container has an adjectival descriptor. A fluid container has a nominal descriptor. Understand the adjectival descriptor property as describing a fluid container. Understand the nominal descriptor property as describing a fluid container.

```
To decide what number is (quantity - a number) squared:
    decide on quantity times quantity.

To score mixtures in (item - a fluid container):
    repeat through Table of Mixtures:
        let total line parts be creme de menthe comp entry squared;
        let total line parts be total line parts plus vodka comp entry squared;
        let total line parts be total line parts plus cacao comp entry squared;
        let creme de menthe score be creme de menthe comp entry times the
creme de menthe volume of item;
        let vodka score be vodka comp entry times the vodka volume of item;
        let cacao score be cacao comp entry times the cacao volume of item;
        let total score be creme de menthe score plus vodka score;
        let total score be total score plus cacao score;
        let total score be total score times calibration for total line parts;
        now rating entry is total score;
        if total line parts is 1, now adjectival descriptor entry is pure;
        otherwise now adjectival descriptor entry is perfect;
        [and for creme de menthe...]
        now creme de menthe comp entry is creme de menthe comp entry plus 1;
        let total line parts be creme de menthe comp entry squared plus vodka
comp entry squared;
        let total line parts be total line parts plus cacao comp entry squared;
        let creme de menthe score be creme de menthe comp entry times the
creme de menthe volume of item;
        let vodka score be vodka comp entry times the vodka volume of item;
        let cacao score be cacao comp entry times the cacao volume of item;
        let total score be creme de menthe score plus vodka score;
        let total score be total score plus cacao score;
        let total score be total score times calibration for total line parts;
        if total score is greater than rating entry, now adjectival descriptor entry is
minty;
        now creme de menthe comp entry is creme de menthe comp entry minus 1;
        [and for vodka...]
        now vodka comp entry is vodka comp entry plus 1;
        let total line parts be creme de menthe comp entry squared plus vodka
comp entry squared;
        let total line parts be total line parts plus cacao comp entry squared;
        let creme de menthe score be creme de menthe comp entry times the
creme de menthe volume of item;
        let vodka score be vodka comp entry times the vodka volume of item;
        let cacao score be cacao comp entry times the cacao volume of item;
        let total score be creme de menthe score plus vodka score;
        let total score be total score plus cacao score;
        let total score be total score times calibration for total line parts;
        if total score is greater than rating entry, now adjectival descriptor entry is
strong;
        now vodka comp entry is vodka comp entry minus 1;
        [and for cacao...]
        now cacao comp entry is cacao comp entry plus 1;
        let total line parts be creme de menthe comp entry squared plus vodka
comp entry squared;
        let total line parts be total line parts plus cacao comp entry squared;
        let creme de menthe score be creme de menthe comp entry times the
creme de menthe volume of item;
        let vodka score be vodka comp entry times the vodka volume of item;
        let cacao score be cacao comp entry times the cacao volume of item;
```

```
        let total score be creme de menthe score plus vodka score;
        let total score be total score plus cacao score;
        let total score be total score times calibration for total line parts;
        if total score is greater than rating entry, now adjectival descriptor entry is
chocolatey;
        now cacao comp entry is cacao comp entry minus 1.

To identify mixture in (item - a fluid container):
    score mixtures in item;
    sort Table of Mixtures in reverse rating order;
    choose row 1 in Table of Mixtures;
    now nominal descriptor of the item is nominal descriptor entry;
    let sample vodka be vodka comp entry; [Now keep track of all these]
    let sample creme de menthe be creme de menthe comp entry;
    let sample cacao be cacao comp entry;
    if rating entry divided by 100 is the current volume of the item:
        now adjectival descriptor of the item is pure;
    otherwise:
        now adjectival descriptor of the item is adjectival descriptor entry.

To decide what number is the raw quantity of (item volume - a volume):
    let raw be item volume divided by 0.5 fl oz;
    decide on raw.

To decide what number is calibration for (total - a number):
    if total is an initial listed in the table of Multipliers, decide on result entry;
    decide on 21.
```

Here we cheat on our arithmetic. The following chart just provides values corresponding roughly to 1/(sqrt (x)), but since Inform does not deal very gracefully with square roots or fractions, we will calculate this elsewhere and just supply the answers in the code:

```
Table of Multipliers

initial result
1    100
2    71
3    57
4    50
5    44
6    41
7    38
8    35
9    33
10   31
11   30
12   29
13   28
14   27
15   26
16   25
17   24
18   24
19   23
20   22

When play begins:
    repeat with item running through fluid containers:
```

```
        identify mixture in item.

To say (amount - a volume) in rough terms:
    if the amount is less than 0.6 fl oz:
        say "half an ounce or less";
    otherwise if tenths part of amount is greater than 3 and tenths part of amount
is less than 7:
        let estimate be ounces part of amount;
        say "[estimate in words] or [estimate plus 1 in words] fluid ounces";
    otherwise:
        if tenths part of amount is greater than 6, increase amount by 1.0 fl oz;
        say "about [ounces part of amount in words] fluid ounce[s]".

Before printing the name of a fluid container (called the target) while not drinking
or pouring:
    if the target is empty:
        say "empty ";
    otherwise:
        do nothing.

After printing the name of a fluid container (called the target) while not examining
or pouring:
    unless the target is empty:
        say " of [adjectival descriptor of the target] [nominal descriptor of the
target]";
        omit contents in listing.

Instead of inserting something into a fluid container:
    say "[The second noun] has too narrow a mouth to accept anything but
liquids."

Definition: a fluid container is empty if the current volume of it is 0.0 fl oz.
Definition: a fluid container is full if the current volume of it is the fluid capacity of
it.

Understand "drink from [fluid container]" as drinking.

Instead of drinking a fluid container:
    if the noun is empty:
        say "There is no more [nominal descriptor of the noun] within." instead;
    otherwise:
        let cacao loss be the consumed cacao of the noun out of sip volume;
        let creme de menthe loss be the consumed creme de menthe of the noun
out of sip volume;
        let vodka loss be the consumed vodka of the noun out of sip volume;
        decrease the cacao volume of the noun by the cacao loss;
        decrease the creme de menthe volume of the noun by creme de menthe
loss;
        decrease the vodka volume of the noun by vodka loss;
        say "You take a sip of [the nominal descriptor of the noun][if the noun is
empty], leaving [the noun] empty[end if].".

Sip volume is a volume that varies. Sip volume is 0.5 fl oz.
```

```
To decide what volume is the consumed cacao of (item - a fluid container) out of
(total consumption - a volume):
    let new volume be the cacao volume of the item times 100;
    let percentage be the new volume divided by the current volume of the item;
    let consumed volume be the percentage times total consumption;
    let consumed volume be consumed volume divided by 100;
    if consumed volume is greater than the cacao volume of the item, decide on
the cacao volume of the item;
    decide on consumed volume.

To decide what volume is the consumed creme de menthe of (item - a fluid
container) out of (total consumption - a volume):
    let new volume be the creme de menthe volume of the item times 100;
    let percentage be the new volume divided by the current volume of the item;
    let consumed volume be the percentage times total consumption;
    let consumed volume be consumed volume divided by 100;
    if consumed volume is greater than the creme de menthe volume of the item,
decide on the creme de menthe volume of the item;
    decide on consumed volume.

To decide what volume is the consumed vodka of (item - a fluid container) out of
(total consumption - a volume):
    let new volume be the vodka volume of the item times 100;
    let percentage be the new volume divided by the current volume of the item;
    let consumed volume be the percentage times total consumption;
    let consumed volume be consumed volume divided by 100;
    if consumed volume is greater than the vodka volume of the item, decide on
the vodka volume of the item;
    decide on consumed volume.

Part 2 - Filling

Understand the command "fill" as something new.

Understand "fill [something] with/from [something]" as filling it with.

Filling it with is an action applying to two things. Carry out filling it with: try
pouring the second noun into the noun instead.

Understand "pour [fluid container] in/into/on/onto [fluid container]" as pouring it
into. Understand "empty [fluid container] into [fluid container]" as pouring it into.

Understand "pour [something] in/into/on/onto [something]" as pouring it into.
Understand "empty [something] into [something]" as pouring it into.

Pouring it into is an action applying to two things.

Check pouring it into:
    if the noun is not a fluid container, say "You can't pour [the noun]." instead;
    if the second noun is not a fluid container, say "You can't pour liquids into [the
second noun]." instead;
    if the noun is the second noun, say "You can hardly pour [the noun] into itself."
instead;
    if the noun is empty, say "No more [nominal descriptor of the noun] remains in
[the noun]." instead;
```

if the second noun is full, say "[The second noun] cannot contain any more than it already holds." instead.

Carry out pouring it into:
let available capacity be the fluid capacity of the second noun minus the current volume of the second noun;
if the available capacity is greater than the current volume of the noun, now the available capacity is the current volume of the noun;
let cacao loss be the consumed cacao of the noun out of available capacity;
let creme de menthe loss be the consumed creme de menthe of the noun out of available capacity;
let vodka loss be the consumed vodka of the noun out of available capacity;
decrease the cacao volume of the noun by the cacao loss;
decrease the creme de menthe volume of the noun by creme de menthe loss;
decrease the vodka volume of the noun by vodka loss;
increase the cacao volume of the second noun by the cacao loss;
increase the creme de menthe volume of the second noun by creme de menthe loss;
increase the vodka volume of the second noun by vodka loss.

Report pouring it into:
identify mixture in noun;
identify mixture in second noun;
say "[if the noun is empty][The noun] is now empty; [otherwise][The noun] now contains [current volume of the noun in rough terms] of [nominal descriptor of the noun]; [end if]";
say "[the second noun] contains [current volume of the second noun in rough terms] of [adjectival descriptor of the second noun] [nominal descriptor of the second noun][if the second noun is full], and is now full[end if]."

Understand "of" as a fluid container.

Part 3 - Scenario

When play begins: say "When you decided to try Mixology WS102 (*cross-listed with Women's Studies), you envisioned yourself writing essays about gender discrimination during the Prohibition, say, or reading essays on male vs. female metabolism of alcohol. But no: MxWS102 turns out to be about... mixing the perfect chocolate mint martini."

The College of Mixology is a room. The bar is a supporter in the college.

The cocktail glass is a fluid container carried by the player. The fluid capacity of the cocktail glass is 4.0 fl oz.

The flask is a fluid container carried by the player. The vodka volume of the flask is 4.0 fl oz.

The jigger is a fluid container carried by the player. The fluid capacity of the jigger is 1.0 fl oz.

The small measure is a fluid container carried by the player. The fluid capacity of the small measure is 0.5 fl oz.

The decanter is a fluid container on the bar. The fluid capacity of the decanter is 32.0 fl oz. The creme de menthe volume of the decanter is 20.0 fl oz.

The bottle is a fluid container carried by the player. The cacao volume of the bottle is 10.0 fl oz.

Test me with "i / pour flask in jigger / pour jigger in glass / pour bottle in jigger / pour jigger in glass / pour bottle in jigger / pour jigger in glass / pour decanter in jigger / pour jigger in glass / drink glass / g / g / x glass / pour flask in glass".

277

Example Merlin

RB

A REMEMBER command which accepts any text and looks up a response in a table of recollections.

"Merlin"

Understand "remember [text]" as remembering.

Remembering is an action applying to one topic.

Carry out remembering:
    say "Nothing comes to mind.".

Instead of remembering a topic listed in the Table of Recollections:
    say "[response entry][paragraph break]".

Table of Recollections

Topic	Response
"rain/weather"	"You've seen worse, but not often: it's falling so hard now that the tin rattles and the runoff, on the low side of the roof, would be a tenable source of hydroelectric power."
"hydroelectric power" or "power/hydroelectric"	"It's not as though you have any sort of light bulb in here to turn on, even if you could power it."
"light bulb" or "light/bulb"	"Light bulbs, like so much else, are a thing of your past. Or is it your future? Tricky, the way the world loops round on itself."
"past/time/future"	"Living backwards has its drawbacks. A tendency to confuse and annoy your friends, being one; the total inability to maintain a stable relationship; and a deep dissatisfaction with most of the bodily processes people enjoy, since they ultimately make you hungrier, colder, or-- no point dwelling on it, really."
"backwards"	"It's not even exactly *backwards*, now is it? It's more like a series of forwardses stuck back to back. As though someone had taken each track of a CD and put them in the exactly wrong order. You miss that. The music on demand."

The Inadequate Shelter is a room. "A piece of corrugated tin, leaned on two sticks, and pathetically augmented with a tire (on one side) and a cardboard box (on the side towards the wind). And that's what you've got between you and the driving rain.

At the moment rain is all you can remember, in fact."

Test me with "remember rain / remember power / remember light bulb / remember future / remember backwards".

278

Example Questionable Revolutions

RB

An expansion on the previous idea, only this time we store information and let characters answer depending on their expertise in a given area.

```
"Questionable Revolutions"

Interrogative is a kind of value. The interrogatives are who, what, when, where, how, and why.

Current question is an interrogative that varies.

After asking someone about something: respond to the question. After answering someone that something: respond to the question.

After telling someone about something: say "You're here to ask questions."

Country is a kind of value. The countries are Czechoslovakia, Georgia, Sweden, Italy, Spain.

Table of Information

topic	date	place	definition
"velvet revolution"	1989	Czechoslovakia	"A bloodless revolution in Czechoslovakia, in which popular protests led to the resignation of the communist president Gustav Husak, and the election of Vaclav Havel in his place."
"rose revolution"	2003	Georgia	"A revolution in which President Eduard Shevardnadze was interrupted by protesters in the middle of his speech, and forced to flee."
"spanish revolution"	1936	Spain	"An anarchist and socialist movement during the Spanish civil war."

After reading a command:
	if the player's command includes "[interrogative]", now the current question is the interrogative understood.

To respond to the question:
	repeat through the Table of Information:
		if the topic understood includes topic entry:
			if the current question is what or the current question is who, say definition entry appropriately;
			if the current question is when, say date entry appropriately;
			if the current question is where, say place entry appropriately;
			rule succeeds;
	say "[The noun] shrugs."

Understand "ask [someone] [text]" as asking it about.

Comprehension is a kind of value. The comprehensions are vague, erroneous, and correct.

Table of Understanding

character	years	geography	general comprehension
Dr Tweedy	correct	correct	correct
Ms Finch	erroneous	erroneous	correct
Ms Clarion	vague	vague	erroneous
```

```
When play begins:
    say "Here you are in the first class cabin, but no matter how fancy the seats
are, you can still get bored circling over Zurich for three hours on end. To kill
time, you and the other passengers are playing a trivia game, and the final topic
is your specialty: revolutions."

First Class Cabin is a room. Dr Tweedy is a man in First Class. Ms Finch and Ms
Clarion are women in First Class.

To say (year - a number) appropriately:
    choose row with character of the noun in the Table of Understanding;
    if years entry is correct:
        say "'[year],' replies [the noun] promptly.";
        increment the quiz score of the noun;
    if years entry is erroneous:
        let guess be a random number between 1900 and 2005;
        say "'[guess]?' guesses [the noun], with an air of diffidence[if guess is the
year]. Which is right, as it happens[end if].";
        if guess is the year, increment the quiz score of the noun;
    if years entry is vague:
        let offset be a random number between -5 and 5;
        let year be year + offset;
        say "'I think [year]. About then. Close, anyway,' replies [the noun][if the
offset is 0], getting it right[end if].";
        if offset is 0, increment the quiz score of the noun.

To say (spot - a country) appropriately:
    choose row with character of the noun in the Table of Understanding;
    if geography entry is correct:
        say "'[spot],' replies [the noun] promptly.";
        increment the quiz score of the noun;
    if geography entry is erroneous:
        let guess be a random country;
        say "'Er... [guess]?' says [the noun][if guess is the spot]. Which is of course
correct[end if].";
        if guess is the spot, increment the quiz score of the noun;
    if geography entry is vague, say "'Europe,' replies [the noun] with confidence."

To say (explanation - some text) appropriately:
    choose row with character of the noun in the Table of Understanding;
    if general comprehension entry is correct:
        say "'[explanation]'[paragraph break]";
        increment the quiz score of the noun;
    otherwise:
        choose a random row in the Table of Information;
        say "'[definition entry]'[paragraph break]";
        if the definition entry is explanation:
            say "[A random other person who is not the noun] looks surprised that
this came out right.";
            increment the quiz score of the noun.

A person has a number called quiz score. The quiz score of Tweedy is 48. The
quiz score of Finch is 2. The quiz score of Clarion is 4.

Definition: a person is other if it is not the player.
```

```
When play begins:
    now left hand status line is "T: [quiz score of Tweedy] F: [quiz score of Finch]
C: [quiz score of Clarion]";
    now right hand status line is "[time of day]".

Test me with "dr tweedy, where was the velvet revolution located / ms finch,
when was the rose revolution / ms finch, what was the rose revolution / ms
clarion, when was the spanish revolution / g / g / ms finch, when was the spanish
revolution".
```

We have so far seen several ways to write conversational characters in Inform, and we will see more before the end of the manual. This naturally raises the question, which should we use? To which the answer is: it depends on the sort of game we're writing, and what we want our characters to do. The more rich and complex the system, the more likely that it will require a lot of content; if we add question types as well as keywords, for instance, we instantly multiply the number of responses we have to write by five or six. It is not worth doing this unless there is some corresponding advantage within the game.

---

279  **Example The Queen of Sheba** *RB*

Allowing the player to use question words, and using that information to modify the response given by the other character.

Suppose we want the player to ask questions of slightly more complexity - we might want to build in a system that understood "who", "what", "where", and "when", for instance. We could use a topic table for this, too:

```
"The Queen of Sheba"

Interrogative is a kind of value. The interrogatives are who, what, when, where,
how, and why.

Current question is an interrogative that varies.

After asking someone about something: respond to the question. After
answering someone that something: respond to the question.

After telling someone about something: say "You're here to ask questions and
test Solomon's wisdom, not to give him a sample of your own."

Table of Wise Answers
```

topic	question type	reply
"rain/weather/clouds/cloud/rains"	what	"'Clouds are a disturbance made by the paths of birds,' Solomon replies. 'The air beaten by their wings becomes agitated, as when a river is stirred and the mud churns up.'"
"rain/weather/clouds/cloud/rains"	where	"'Weather is contained in a great silk bag which holds in the heavens,' replies Solomon."
"hunger/food/eating"	when	"'Sorry, are you getting hungry?' he says, and rings a bell to summon servants."
"hunger/food/eating"	why	"'Men were made to need food in order that they must farm and cook and dine together,' Solomon replies. 'Otherwise, they might live apart,

		each sufficient in himself. But no man can feed himself alone all through his life.'"
"Solomon/he/himself"	who	"'As you see,' he says, holding out his arms to each side."
"Solomon/he/himself"	what	"'I am an ordinary man,' he answers."

One of the nice things about this system is that it only resets the "current question" when we get a new question word. For instance, this test will produce different replies to the question about Solomon himself, because the second time he is still in the mode of answering "what" questions:

```
Test me with "ask solomon about himself / ask solomon what rain is / ask
solomon about himself".
```

If Solomon is to live up to his reputation at all, his wisdom table will have to be quite a bit longer - though one also would want to be careful, because forcing the game to cycle through a really immense table could be quite time-consuming. In fact, for the sake of this example, let's reward the player for managing to stay within the (narrow) range of Solomon's knowledge:

```
The Hall of Almug Tree Pillars is a room. "The pillars of the room are made of
almug tree, the ceiling made of silk and the floor of glass." Solomon is a man in
the Hall of Almug Tree Pillars. Solomon has a number called wisdom. The
wisdom of Solomon is 0.

Every turn:
    if the wisdom of Solomon is 3:
        say "Truly, Solomon has answered all your questions, and his wisdom is
even as great as you had heard!";
        end the story saying "Your heart beats strangely fast".

When play begins, say "'Oh, you've arrived,' says Solomon."
```

In a real game we'd need to be a great deal subtler. All the same, if we have a character of quite limited resources to present to the player, it's a good idea to give the player some incentive to stay on topic, ask questions the character can answer, and generally interact within the parameters we're prepared for.

Now, this last bit requires some trickery from later chapters, particularly those on Understanding and Activities, to pull the question words out of the player's command:

```
After reading a command:
    if the player's command includes "[interrogative]", now the current question is
the interrogative understood.

To respond to the question:
    repeat through the Table of Wise Answers:
        if the topic understood includes topic entry:
            if the current question is the question type entry:
                say "[reply entry][paragraph break]";
                increment the wisdom of Solomon;
                rule succeeds;
    say "Solomon looks blank, appalled by a question for which he was not
```

```
prepared.";
    end the story saying "You have befuddled Solomon!"

Understand "ask [someone] [text]" as asking it about.
```

And now we have a game that will accept (though not always respond very sensibly to) questions of almost any form we might put to another character: ASK SOLOMON WHAT RAIN IS will be answered, but then again, it won't be distinguished from, say, ASK SOLOMON WHETHER THIS PERSISTENT RAIN IS A DIVINE PUNISHMENT OR WHAT.

All the same, a system that allowed the player a bit more specification of questions than simple keyword-use might be useful in a mystery game, for instance, where we might want to let our detective conduct inquiries into specific details. An alternative approach to the rather free one above would be to force the player to use only questions of the form WHAT IS RAIN? or WHO ARE YOU?: this would cut down on false-positive matches. But we might still choose to store the responses in a table of this type.

280

Example Goat-Cheese and Sage Chicken

RB

Implementing a FULL SCORE command which lists more information than the regular SCORE command, adding times and rankings, as an extension of the example given in this chapter.

Some games provide a FULL SCORE command that gives more information about the player's achievements than SCORE alone. Supposing we wanted to include a FULL SCORE in our game that gave the kind of score reading described in this chapter:

```
"Goat-Cheese and Sage Chicken"

Use scoring.

The story headline is "An interactive recipe"

Table of Tasks Achieved

Points Citation                Time
3      "sauteeing onions"      a time
3      "reconstituting apricots"
1      "flattening chicken"
1      "unwrapping goat cheese"

To record (T - text) as achieved:
    choose row with a citation of T in the Table of Tasks Achieved;
    if there is no time entry:
        now time entry is the time of day;
        increase the score by the points entry.
```

Requesting the full score is an action out of world. Understand "full" or "full score" as requesting the full score.

Carry out requesting the full score:
    if the score is 0, say "You have achieved nothing towards supper." instead;
    repeat through the Table of Tasks Achieved in reverse time order:
        say "[time entry]: [citation entry] ([points entry])."

Table of Rankings

Score	Rank
0	"Rank Amateur"
2	"would-be Bobby Flay"
5	"Alton Brown"
8	"Julia Child"

The Kitchen is a room. The description of the Kitchen is "Equipped with many familiar friends: refrigerator, stove, oven; countertop; cabinet for pans and bowls, and a drawer for your tools."

The stove is scenery in the kitchen. It is a supporter. The oven is a container. It is part of the stove. It is closed and openable. The stove's switch is a device. It is switched on. It is part of the stove. The oven's dial is a device. It is switched off. It is part of the oven.

A thing can be heatproof.

Instead of putting something which is not heatproof on the stove when the stove's switch is switched on:
    say "You catch yourself just at the last minute: not a good idea to put [the noun] directly on the stove while it's turned on."

Instead of switching on the stove, try switching on the stove's switch. Instead of switching off the stove, try switching off the stove's switch. Instead of switching on the oven, try switching on the oven's dial. Instead of switching off the oven, try switching off the oven's dial.

Before switching on the oven's dial when the oven is open:
    say "(closing the oven so that it will heat properly)[command clarification break]";
    try closing the oven.

The frying pan is a heatproof unopenable open container on the stove.

The cabinet is a closed openable container in the kitchen. It is scenery. It contains an open unopenable container called a mixing bowl. It contains a portable supporter called a platter. An open unopenable heatproof container called a Calphalon baking dish is in the cabinet. The baking dish has the description "One of those marvelous pieces of kitchen equipment which goes on the stove or in the oven, as you will. The chief thing is never ever to touch it when it is hot, since the handles are metal and the heat retention excellent."

The counter is a supporter in the kitchen. It is scenery. The kettle is a heatproof openable closed container on the counter. Some water is in the kettle.

The water can be cool, warm, or boiling. The printed name of the water is "[water condition] water".

The refrigerator is a closed openable container in the kitchen. It is scenery. Understand "fridge" as the refrigerator.

An ingredient is a kind of thing.

Some onions, some apricots, and some sage are ingredients on the counter. A chicken breast, an egg, and goat cheese are ingredients in the refrigerator.

The goat cheese can be wrapped, snipped open, or unwrapped. The printed name of the goat cheese is "[goat cheese condition] goat cheese".

The sage can be unwashed, clean, or julienned. The sage is unwashed. The printed name of the sage is "[sage condition] sage".

The apricots can be dried, reconstituted, or chopped. The apricots are dried. The printed name of the apricots is "[apricots condition] apricots".

The chicken breast can be whole, flattened, stuffed, rolled, coated, browned, or baked. The printed name of the chicken breast is "[chicken breast condition] chicken breast".

The onions can be unpeeled, peeled, diced, sauteed, or burnt. [The printed name of the onions is "[onions condition] onions".]

The can of chicken broth is a closed container on the counter. The bottle of white cooking wine and the bottle of Thurston Wolfe PGV are a closed containers in the refrigerator.

The description of the Thurston Wolfe is "A Washington State Pinot Gris-Viognier, 2003. It is said to have 'peach aromas', and, startlingly, the untutored person can detect these without resorting to fantasy.

(It is also supposed to possess a delicate perfume and a moderate body; the label author at least stopped short of 'good sense of humor and likes long walks on the beach')."

Understand the commands "wash" and "rinse" as "clean".

Instead of rubbing the unwashed sage:
    now the sage is clean;
    say "You rinse off the sage. There -- ready to slice."

Instead of cutting the sage:
    say "You'd need to have a knife in hand, first."

Instead of cutting the clean sage when the player is carrying the butcher knife:
    now the sage is julienned;
    say "You slice the sage into thin strips."

Instead of cutting the unwashed sage:
    say "It came from the garden, so it won't have any strange chemicals on it, but

you should still give it a rinse for dirt and bugs and so on before using it."

Instead of doing something other than examining or rubbing with the unwashed sage:
    say "It needs to be washed off."

Understand "peel [something]" as peeling.

Peeling is an action applying to one thing.

Instead of peeling the unpeeled onions:
    now the onions are peeled;
    say "You tear away the shining outer skin of the onions, leaving them pale and nekkid. Poor things."

Instead of cutting the diced onions:
    say "That seems unnecessary now."

Instead of cutting the sauteed onions:
    say "Too late; you're well past that stage."

Instead of cutting the burnt onions:
    say "There's no rescuing 'em -- the carbon isn't going to flake off, you know."

Instead of cutting the unpeeled onions:
    say "It would help to peel them first."

Instead of cutting the peeled onions:
    say "You'd need to have a knife in hand, first."

Instead of cutting the peeled onions when the player is carrying the butcher knife:
    now the onions are diced;
    say "You dice the onions neatly. Your own skill brings tears to your eyes."

Instead of opening the goat cheese:
    try peeling the goat cheese instead.

Instead of peeling the unwrapped goat cheese:
    say "The goat cheese is already unwrapped. (Stay focused, stay focused...)"

Before peeling the wrapped goat cheese when the shears are held by the player:
    try cutting the goat cheese.

Instead of peeling the snipped open goat cheese:
    now the goat cheese is unwrapped;
    record "unwrapping goat cheese" as achieved;
    say "Ah, success. The goat cheese is now free of its packet."

Instead of peeling the wrapped goat cheese:
    say "It would help to have a pair of scissors or something -- the packet resists being torn."

Instead of cutting the goat cheese:
    say "No need, at this point."

Before cutting the wrapped goat cheese when the shears are not held by the player and the shears are visible:
    say "(first picking up the shears)[command clarification break]";
    try taking the shears.

Instead of cutting the wrapped goat cheese:
    say "Something to cut with would be useful."

Instead of cutting the wrapped goat cheese when the shears are held by the player:
    now the goat cheese is snipped open;
    say "You neatly snip through the packaging with the shears."

Instead of examining the whole chicken breast:
    say "It is still entire and has yet to be pounded flat."

Instead of examining the flattened chicken breast:
    say "It has been hammered to a thickness of about a half inch. (The recipe said a quarter inch but you're pretty sure it was joking. You have never been able to achieve a quarter inch.)"

Instead of attacking the whole chicken breast:
    say "You need something heavy enough to flatten it with."

Instead of attacking the whole chicken breast when the player is holding the wooden mallet:
    now the chicken breast is flattened;
    record "flattening chicken" as achieved;
    say "You hammer away at the chicken breast, turning all your aggressions into culinary goodness. Several minutes pass. When you are done you have a broad flat chickeny pancake suitable for wrapping about a stuffing."

Before printing the name of onions:
    say "[onions condition] ".

The drawer is an openable closed container. It is part of the counter.

A tool is a kind of thing. A spatula, a spoon, a wooden mallet, some shears, and a ball of twine are tools in the drawer. A butcher knife is a tool carried by the player. Understand "scissors" as the shears.

Instead of burning something:
    say "You'll have to do that the hard way."

Some steam is fixed in place. "Dense clouds of steam fill the room."

Some smoke is fixed in place. "Smoke is beginning to collect near the ceiling."

Sauteeing Onions is a scene. Sauteeing Onions begins when the diced onions are in a hot container.

Definition: a container is hot if it is on the stove and the stove's switch is switched on.

Instead of touching the hot pan:
    say "Ow!"

Scorching Onions is a scene.

Preheating the Oven is a scene. Preheating the Oven begins when the oven's dial is heating.

Definition: a oven's dial is heating if the oven's dial has been switched on for exactly one turn.

Preheating the Oven ends when the time since Preheating the Oven began is five minutes.

When Preheating the Oven begins:
    say "The oven begins to warm up."

When Preheating the Oven ends:
    say "The oven beeps to inform you that it has reached the desired hotness."

Every turn during Sauteeing Onions:
    say "The onions sizzle in the pan."

Every turn during Scorching Onions:
    say "The onions are past their prime and are getting blacker by the moment."

Every turn during Hearing the Kettle Whistle:
    say "The kettle continues to whistle."

Instead of listening to during Hearing the Kettle Whistle:
    say "The only thing you can really hear just at the moment is the kettle."

Instead of smelling the Kitchen during Sauteeing Onions:
    try smelling the onions.

Instead of smelling the onions during Sauteeing Onions:
    say "The onions smell marvelous."

Instead of opening the oven during Preheating the Oven:
    say "It'll never heat if you open it up while it's warming."

Heating Kettle is a scene. Heating Kettle begins when the hot kettle contains cool water.

Before printing the name of the kettle when the kettle is hot:
    say "hot "

When Heating Kettle begins:
    say "The kettle begins to heat up."

```
Heating Kettle ends when the time since Heating Kettle began is 7 minutes.

Hearing the Kettle Whistle is a scene. Hearing the Kettle Whistle begins when
Heating Kettle ends. Hearing the Kettle Whistle ends when the kettle is not hot.

When Hearing the Kettle Whistle begins:
    now the water is boiling;
    say "The kettle begins to burble and whistle shrilly."

When Hearing the Kettle Whistle ends:
    say "The kettle's screaming dies off."

Idling is a scene. Idling begins when play begins. Idling ends when Sauteeing
Onions begins.

Sauteeing Onions ends in disaster when Scorching Onions begins.

Sauteeing Onions ends in success when the onions are sauteed and onions are
not in a hot container.

Definition: a thing is alone if it is in a container which contains exactly one thing.

Sauteeing Onions ends in mixture when the sauteed onions are not alone.

When Sauteeing Onions ends in mixture:
    say "The mixture of things in [the holder of the onions] stops them cooking
quite so fast."

When Sauteeing Onions ends in success:
    say "Nice work with the onions."

Every turn:
    if diced onions have been in a hot pan for ten turns:
        say "The onions are starting to look ready.";
        now the onions are sauteed.

Scorching Onions begins when Sauteeing Onions ends in disaster. Scorching
Onions begins when the alone sauteed onions are in a hot container.

Scorching Onions ends horribly when the time since Scorching Onions began is
three minutes. Scorching Onions ends in reprieve when the sauteed onions are
not in a hot container. Scorching Onions ends in mixture when the sauteed
onions are not alone.

When Scorching Onions ends in mixture:
    record "sauteeing onions" as achieved;
    say "The mixture of things in [the holder of the onions] stops them cooking
quite so fast."

When Scorching Onions ends horribly:
    move smoke to Kitchen;
    now the onions are burnt.
```

```
When Scorching Onions ends in reprieve:
    record "sauteeing onions" as achieved;
    say "You've got the onions off heat before they can scorch -- a good sign."

Instead of taking the onions when the onions are in the pan: try taking the pan.

Instead of smelling in the presence of the smoke:
    say "The scent of the late disaster lingers in the air."

Reconstituting the Apricots is a scene.

Reconstituting the Apricots begins when the dried apricots are in a container
which contains boiling water.

When Reconstituting the Apricots begins:
    say "The apricots slowly begin to plump up again."

Reconstituting the Apricots ends when the dried apricots are not in a container
which contains boiling water.

Every turn:
    if dried apricots have been in a container which contains boiling water for ten
turns:
        say "The apricots have turned plump(ish).";
        now the apricots are reconstituted;
        record "reconstituting apricots" as achieved.

Test sautee with "peel onions / cut onions / get onions / put onions in pan / get
sage / wash sage / cut sage / wait / wait / wait / wait / wait / wait / wait / get pan".

Test apricots with "get kettle / open kettle / get apricots / put apricots in kettle /
put kettle on stove / wait / wait / wait / wait / wait / wait / wait / wait / wait /
wait / wait / wait / wait / wait / wait / wait".

Test chicken with "open refrigerator / get chicken / open drawer / get mallet / hit
breast".

Test cheese with "get scissors / get cheese / unwrap cheese".

Test me with "full score / test sautee / full score / test apricots / full score / test
chicken / full score / test cheese / full score".
```

And... at that point you're a lot less close to being done than you think. The filling -- onions, sage, apricot, and cheese -- must be assembled and put in the chicken breasts; these tied up in string; each roll dipped in egg yolk and rolled in panko crumbs; these arranged in the Calphalon pan and baked. Then later, the whole retrieved from the oven, and the breasts transferred to a plate while we deglaze the pan and concoct the sauce with the chicken broth, wine, butter, etc. Then the chicken is sliced and plated, and the sauce poured over top. Usually one also wants a side dish or two. A number of things can go interestingly wrong in this process, of course, and implementing it would require, among other things, an intelligent management of all the possible mixtures that result.

281

Example Farewell

RB

People who respond to conversational gambits, summarize what they said before if asked again, and provide recap of conversation that is past.

We begin with the idea that each person comes with his own table of things to say:

```
"Farewell"

A person has a table name called conversation.

Instead of asking someone about something:
    let the source be the conversation of the noun;
    if topic understood is a topic listed in source:
        if there is a turn stamp entry:
            say "[The noun] has already told you that [summary entry].";
        otherwise:
            now turn stamp entry is the turn count;
            say "[reply entry][paragraph break]";
    otherwise:
        say "[The noun] stares at you blankly."
```

For the sake of simplicity, we'll conflate asking and telling here, though it would certainly be possible to have a more complex implementation if we want the characters to be told things as well.

```
Instead of telling someone about something:
    try asking the noun about it.
```

Now we might want to add a recap command to review conversation that has already occurred.

```
Definition: a person is other if it is not the player.

Understand "recap" or "recall" or "review" as recalling conversations.

Recalling conversations is an action applying to nothing.
```

Since we've been recording the turn count of each conversation bit, we can even present these in order by sorting the tables first.

```
Carry out recalling conversations:
    repeat with speaker running through other people:
        let source be the conversation of the speaker;
        sort source in turn stamp order;
        say "[The speaker] has so far told you: [line break]";
        let index be 0;
        repeat through source:
            if there is a turn stamp entry:
                let index be 1;
                say " [summary entry][line break]";
```

```
    if index is 0, say " absolutely nothing[line break]";
    say conditional paragraph break.
```

Now it remains only to create a couple of characters and provide them both with something to say:

```
The Farewell Bend Cafe is a room. "Beautiful Farewell Bend, Idaho -- or is it Oregon? An almost-abandoned truckstop, in any case, on one of those interminable American east-west highways."

Tina is a woman in the Farewell Bend Cafe. The conversation of Tina is the Table of Tina's Chatter. "Tina the waitress is slowly pouring coffee from the pot with a black neck into the pot with an orange neck."

George is a man in the Farewell Bend Cafe. The conversation of George is the Table of George's Chatter. "There is also a large man at table five. The tattoo on his arm says George. For the moment we will assume that it is his own name and not someone else's."

Table of Tina's Chatter
topic	reply	summary	turn stamp
"aasvogel"	"'Oh, it's a vulture.'"	"that an aasvogel is a vulture"	a number
"acaudate"	"She shrugs, mid-pour. 'Means something doesn't have a tail.'"	"that acaudate means 'tailless'"	--
"absorptiometer"	"'It's a thing that measures the solubility of gases in a liquid,' she explains gently, as to a child."	"that an absorptiometer measures solubility of gasses in a liquid"	--

Table of George's Chatter
topic	reply	summary	turn stamp
"baccaceous"	"'Something that has or bears berries,' says George, without looking up."	"that baccaceous means berry-bearing or berry-like"	a number
"bagheera"	"'Oh, that'd be a velvet-like textile.'"	"that bagheera is a velvet-like textile"	--
"balistarius"	"'That's a crossbow-man,' George replies instantly."	"that a balistarius is a crossbow-man"	--
```

A word of warning: this system does assume that every person in the game has a conversation table defined. If that were not the case, we would have to be a bit more careful.

As always, we can override specific words, too:

```
Instead of asking Tina about "advertisement" for the first time:
    say "Tina looks embarrassed. 'Of course! I almost forgot.' She hands you a brochure.";
    move the brochure to the player.

The encyclopedia sales brochure is a thing. The description is "A glossy flyer indicating that you can receive a free Volume A-Aalto of the New Idahoan Encyclopedia Set if you send back the business reply card, and then have the option of purchasing the remaining volumes at a very very reasonable price."

Test me with "recap / ask tina about aasvogel / recap / ask george about baccaceous / ask tina about absorptiometer / recap / ask tina about
```

advertisement / read brochure".

282

Example Sweeney

RB

A conversation where each topic may have multiple questions and answers associated with it, and where a given exchange can lead to new additions to the list.

"Sweeney"

A subject is a kind of thing. Some subjects are defined by the Table of Conversation Subjects.

Table of Conversation Subjects

subject	conversation
pies	Table of Pie Queries
employment	Table of Job Queries

Understand "job" as employment. Understand "meat" or "food" as pies.

Table of Job Queries

quip	discussion	label	subtopics
"whether there is a job available here"	"'Say, are you hiring?' you ask, as casually as you can manage. [The interlocutor] looks you over dubiously. 'I might be hiring someone, but I can't say it would necessarily be you.'"	0	--
"what happened to that boy that worked here"	"'Tell me, didn't you used to have a young assistant working here?' She shrugs. 'Young men these days are so unstable. He left-- who knows where he's gone? I haven't seen hair or fingernail of him for weeks.'"	0	--
with 3 blank rows.			

Table of Pie Queries

quip	discussion	label	subtopics
"what pie fillings are available"	"'What pies do you have in today, Mrs Lovett?' you ask. She starts, then smiles. 'Meat pies, of course.'"	0	Table of Pie Flavor Queries
with 3 blank rows.			

Table of Pie Flavor Queries

quip	discussion	label	subtopics
"what kind of meat"	"'What kind of meat goes into these pies, Mrs Lovett?' you ask pressingly. She looks shifty. 'Whatever the butcher brings this week,' she says. 'With the price of meat what it is, when you get it, you have to be glad of what you can get. If you get it.'"	0	--

To copy (first table - a table name) to (second table - a table name):
    repeat through first table:
        let copied quip be "blank";
        if there is a quip entry, now the copied quip is the quip entry;
        let copied discussion be "blank";
        if there is a discussion entry, now the copied discussion is the discussion entry;
        let copied subtopics be second table;
        if there is a subtopics entry, now the copied subtopics are the subtopics

```
entry;
        choose a blank row in the second table;
        if copied quip is not "blank", now quip entry is copied quip;
        if copied discussion is not "blank", now discussion entry is copied
discussion;
        if copied subtopics is not second table, now subtopics entry is copied
subtopics.

Current conversation table is a table name that varies. Current conversation
table is Table of Job Queries.

Interlocutor is a person that varies.

Understand "ask [someone] about [any subject]" as asking it about the subject.

Asking it about the subject is an action applying to two visible things.

Carry out asking it about the subject:
    say "You can't think of anything to say."

Instead of asking someone about the subject a subject listed in the Table of
Conversation Subjects:
    now interlocutor is noun;
    now current conversation table is the conversation of the second noun;
    if the number of filled rows in the current conversation table is 1:
        repeat through current conversation table:
            now label entry is 1;
        now number understood is 1;
        try selecting 1 instead;
    if the number of filled rows in the current conversation table is 0:
        say "You can think of nothing further to say on that topic.";
        stop the action;
    otherwise:
        let index be 0;
        let total be the number of filled rows in the current conversation table;
        say "Do you mean ";
        repeat through current conversation table:
            now index is index + 1;
            now label entry is index;
            say "([index]) [quip entry]";
            if index is total, say "?";
            if index is total - 1, say ", or ";
            if index is less than total - 1, say ", ".

Understand "[number]" as selecting.

Selecting is an action applying to one number.

Carry out selecting:
    say "No such option is available."

Instead of selecting a label listed in the current conversation table:
    say "[discussion entry][paragraph break]";
    if there is a subtopics entry:
        copy subtopics entry to current conversation table;
```

```
    choose row with label of number understood in the current conversation table;
    blank out the whole row.

Mrs Lovett's Meat Pies is a room. Mrs Lovett is a woman in Meat Pies.

Test me with "ask lovett about pies / ask lovett about employment / 1 / 2 / ask
lovett about pies".
```

---

283  **Example** **Introduction to Juggling** *RB*

Assortment of equipment defined with price and description, in a table.

Suppose we have a whole catalog-full of equipment that the player might want to purchase and use. We'll start by defining our purchasing rules:

```
"Introduction to Juggling"
```

We allow things to have prices, and the player's price to indicate how much money the player has:

```
Section 1 - Mail-ordering defined

Price is a kind of value. $100.99 specifies a price.

The player has a price. The price of the player is $60.00.
```

Because we're allowing the player to order things that he can't currently see, we need to borrow a special kind of grammar from the Understanding chapter. All our orderable items in this example are toys, so "any toy" means any object of the toy kind, whether or not it is in view at the moment:

```
Understand "buy [any toy]" as ordering. Understand the command "order" as
something new. Understand the command "order" as "buy".

Ordering is an action applying to one visible thing.

Check ordering:
    if the cost of the noun is greater than the price of the player, say "You only
have [price of the player], while [the noun] would cost [cost of the noun]."
instead.

Carry out ordering:
    move the noun to the player;
    decrease the price of the player by the cost of the noun.

Report ordering:
    say "You order [a noun], which is delivered instantly."
```

We should also handle the situation where the player orders another of something he has already bought and which is right in front of him:

```
Instead of buying something:
    say "You already have [a noun]."
```

So much for the general rules for this scenario. Now we move on to particulars: the actual items the player is allowed to order. Each item will have a description, a price, and a difficulty representing how skilled the player must be in order to make use of that item.

Since we are going to use price and difficulty in the table that defines our juggling equipment, we need to mention these kinds of value before the line that says how toys are defined.

```
Section 2 - The Scenario

Difficulty is a kind of value. The difficulties are easy, moderate, hard. The player has a difficulty. The difficulty of the player is easy.

The plural of toy is toys. A toy is a kind of thing. Some toys are defined by the Table of Juggling Equipment.

Table of Juggling Equipment
```

toy	cost	restriction	description	difficulty	outcome
an economy bounce ball set	$10.00	"comes in set of three"	"A fairly ordinary rubber ball, solid color."	moderate	"You create of the balls a cascade of moving color."
an acrylic contact ball	$14.00	"should be bought with ball polish"	"A large clear ball, not for throwing but for using in various hand tricks."	hard	"You rotate the ball between your fingers and pass it over the backs of your hands."
a UV-reactive contact ball	$55.00	"appears to glow in dark rooms"	"Similar to the ordinary acrylic contact ball, but UV-reactive."	hard	"The ball glows as it passes between your fingers and over the backs of your hands, rolls up to your wrist, snaps through the air-- all apparently of its own accord."
a ball polish set	$10.00	"useful only with acrylic contact balls"	"Three bottles of polish and a rag for keeping acrylic contact balls scratch-free."	hard	"You juggle the polish bottles with difficulty, since they are full of sloshing liquid."
a teaching beanbag set	$8.00	"set of three"	"Soft, easily-juggled bag."	easy	"You juggle the beanbags with basic competence."
a stage ball set	$13.50	"comes in set of three"	"Not much different in appearance from the economy bounce ball, but larger so as to be visible from a stage."	moderate	"You create of the balls a cascade of moving color, visible from quite a distance."
a fireball set	$33.00	"will not be sold to minors"	"A ball has wicking and a fuel-source inside so that it will burn while being juggled."	hard	"You juggle the fireballs rapidly, careful never to hold any of them a moment longer than necessary."

Notice that we are allowed to define "description" and other already-known properties in the table as well.

```
Backstage is a room. "A muffled black room with felt on the floors and walls. A glowing sign over the stage door says SHOW IN PROGRESS."

The Juggling Equipment Catalog is a thing in Backstage.
```

```
Instead of examining the Catalog:
    say "You read through the offerings, including: [paragraph break]";
    repeat through Table of Juggling Equipment:
        say "[bold type][toy entry][roman type]: [description entry] [cost entry],
[restriction entry]. [paragraph break]".

When play begins:
    now right hand status line is "Budget: [price of the player]";
    now left hand status line is "[location], feeling [if the difficulty of player is
easy]incompetent[end if][if the difficulty of player is moderate]moderately
skilled[end if][if the difficulty of player is hard]highly skilled[end if]".
```

And of course this will be no fun unless the player is allowed to use the equipment:

```
Understand "juggle [something]" as juggling.

Juggling is an action applying to one thing.

Check juggling:
    if the noun is not a toy listed in the Table of Juggling Equipment, say "You
can't juggle [a noun]!" instead;
    if the difficulty of the noun is greater than the difficulty of the player, say
"You're not quite ready to juggle something like [the noun]. Better to start with an
easier toy." instead.

Carry out juggling:
    if a random chance of 1 in 3 succeeds:
        if the difficulty of the player is less than hard and the difficulty of the player
is the difficulty of the noun:
            now the difficulty of the player is the difficulty after the difficulty of the
player.

Report juggling:
    say "[outcome of the noun][paragraph break]".

Instead of burning the fireball set:
    say "It will flame by itself when you use it."

Test me with "read catalog / buy economy / buy beanbag / juggle economy /
juggle beanbag / juggle beanbag / juggle beanbag / juggle beanbag / juggle
beanbag / juggle beanbag / juggle economy / juggle economy / juggle economy /
buy fireball set / juggle fireball".
```

---

284 **Example Food Network Interactive** *RB*

Using a menu system from an extension, but adding our own material to it for this game.

"Basic Help Menu" is an extension that uses tables to provide a menu of instructions. Suppose we wanted to include this menu in our own game, but add a few custom menu items of our own:

```
"Food Network Interactive"

Include Basic Screen Effects by Emily Short. Include Menus by Emily Short.
Include Basic Help Menu by Emily Short.

Table of Basic Help Options (continued)

title                    subtable          description                                                    toggle
"Recipes in This Game"   Table of Recipes  --                                                             --
"Contacting the Author"  --                "If you have any difficulties with [story title], please contact me at fakeaddress@gmail.com."  --
```

This table is one that is pre-defined by the extension. By continuing it, we add a few additional items to the list.

And since we've promised a sub-menu of recipes:

```
Table of Recipes

title                    subtable  description                                                    toggle
"Salmon Tartare"         --        "First, be sure to buy extremely fresh salmon. Raw fish should be served on the day it was caught, if possible. To guarantee this, visit an Asian supermarket or specialty store, and buy salmon marked 'sashimi grade'..."  --
"Pecan Brownies"         --        "Begin by shelling half a pound of pecans..."                  --

Whole Foods is a room.
```

To test it, type HELP and then experiment.

---

285 **Example Trieste** *RB*

Table amendment to adjust HELP commands provided for the player.

Suppose we are using an extension in which another author has defined some help topics for the player, and we want to amend them for our game.

We'll start with the portion of the text that we have inherited from the extension:

```
"Trieste"

Section 1 - Procedure

A help-topic is a kind of value. Some help-topics are defined by the Table of Standard Instructions.

Table of Standard Instructions

help-topic  reply
commands    "This game recognizes 150 common commands for forms of military attack. These include..."
saving      "To save the game, type SAVE. You will be prompted to supply a file-name for your saved game. If you'd like to return to play at that point again later, RESTORE the saved game."
```

```
Understand "help [help-topic]" as asking for help about. Asking for help about is
an action out of world, applying to one help-topic.
Understand "help" or "help [text]" as a mistake ("Help is available on the
following topics: [help-topics list]").

To say help-topics list:
    repeat through the Table of Standard Instructions:
        say "[line break] [help-topic entry]";

Carry out asking for help about:
    repeat through the Table of Standard Instructions:
        if the help-topic understood is the help-topic entry:
            say "[reply entry][paragraph break]";
            break.

Section 2 - Scenario
```

Now, let's imagine our game is a special one in which only a very limited supply of moves are allowed. In that case, we'll want to replace the information on commands:

```
Table of Standard Instructions (amended)

help-topic   reply
commands "The only commands this game recognizes are HOLD, MOVE, CONVOY, SUPPORT MOVE, and
          SUPPORT HOLD. No others are necessary."

Board Room is a room. Mark is a man in the Board Room. "Russia (played by
Mark) is also hovering over the board."

Guest Bathroom is south of Board Room. Lena and Rob are in the Guest
Bathroom. Lena is a woman. Rob is a man.

Rule for writing a paragraph about Lena when Lena is in the Guest Bathroom
and Rob is in the Guest Bathroom:
    say "[Lena] (Italy) and [Rob] (Great Britain) are having a hushed conversation
while leaning against your good towels. They stop and stare at you when you
come in."

Test me with "help / help commands / help saving".
```

Chapter 17: Understanding


Contents of *Writing with Inform*


Chapter 16: Tables


Chapter 18: Activities


Indexes of the examples

§17.1. Understand

During play, the computer and the player alternate in writing messages to each other: in the player's case, these are short instructions, usually saying what to do next. A wide range of such "commands" are automatically understood, but these only apply to the standard built-in actions. (This wide range is conveniently tabulated in the Commands part of the Actions index.) If we want the player to be able to command new actions, then we need to specify what is to be understood as what. For this, we supply special sentences starting with the word "Understand".

Suppose we return to the earlier example of a newly created action:

```
Photographing is an action applying to one visible thing and requiring light.
```

We then supply lines of grammar (as they are called) for Inform to recognise, like so:

```
Understand "photograph [someone]" as photographing.

Understand "photograph [an open door]" as photographing.
```

As usual, the square brackets indicate something which stands for text, rather than text to be taken verbatim. "[someone]" needs to be the name of anything of the kind "person", for instance (though as usual that person will need to be in sight of the player for the name to be accepted). The first word - in these examples "photograph" - must be something definite, not a substitution like this.

For obvious reasons, this pattern of words needs to match the expectations of the action. Photographing applies to "one visible thing" - the "visible" just means it does not need to be

touched, only seen - so neither of these would be allowable:

```
Understand "photograph" as photographing.
Understand "photograph [someone] standing next to [something]" as photographing.
```

The first is probably bad because it supplies no things at all, the second is certainly because it supplies two: what we want, of course, is just the one. (The reason the first is only probably bad is that it's possible to tell Inform how to choose the object if the player doesn't: see the "supplying a missing noun" activity.)


Start of Chapter 17: Understanding


Back to Chapter 16: Tables: §16.19. Table amendments


Onward to §17.2. New commands for old grammar


Example 286:  **Indirection** Renaming the directions of the compass so that "white" corresponds to north, "red" to east, "yellow" to south, and "black" to west.


Example 287:  **XYZZY** Basics of adding a new command reviewed, for the case of the simple magic word XYZZY.


Example 288:  **Xylan** Creating a new command that does require an object to be named; and some comments about the choice of vocabulary, in general.

---

§17.2. New commands for old grammar

In the photography example, we are providing entirely new grammar for an action not ordinarily built in to Inform. But we often want simply to provide alternative grammar for existing actions, or even to put new interpretations on commands that Inform already recognises. For instance:

```
Understand "deposit [something] in [an open container]" as inserting it into.
```

The inserting action is built in to Inform, but the command "deposit" is not, so this is created as new. It is occasionally useful to put a twist on this:

```
Understand "fill [an open container] with [something]" as inserting it into (with nouns reversed).
```

The clause "(with nouns reversed)" tells Inform to exchange the two nouns parsed, which is necessary because the inserting action expects the noun to be the item and the second noun to be the container, not vice versa.

The following example:

```
Understand "access [something]" as opening.
```

might look as if it makes "access" behave just like "open" when the player types it, but that's not so: "open" can also be used in constructions like "open the door with the brass key", in which case it is understood as the unlocking action. We could add another line to make

"access" behave this way too, but if what we really want is to make "access" behave just like "open", it's easier simply to say so:

```
Understand the command "access" as "open".
```

This is very useful when adding a new command which needs synonyms:

```
Understand the commands "snap" and "picture" as "photograph".
```

We can check the current stock of commands by looking at the table in the Actions index: for instance, before making "snap" synonymous with "photograph", it might be wise to check that it is not already defined as a command for breaking something.


Start of Chapter 17: Understanding


Back to §17.1. Understand


Onward to §17.3. Overriding existing commands


Example 289:  **Alpaca Farm** A generic USE action which behaves sensibly with a range of different objects.


Example 290:  **Anchorite** By default, Inform understands GET OFF, GET UP, or GET OUT when the player is sitting or standing on an enterable object. We might also want to add GET DOWN and DOWN as exit commands, though.


Example 291:  **Cloak of Darkness** Implementation of "Cloak of Darkness", a simple example game that for years has been used to demonstrate the features of IF languages.

---

§17.3. Overriding existing commands

Suppose we are devising specialist commands for a game of whist, and we want "discard" as one of them. Looking at the table of commands in the Action index, we find that, inconveniently enough, "discard" already has a meaning: it is synonymous with "drop", and while that might be sensible most of the time, it is perfectly wrong now. We need a way to free up "discard" for our own use. We can do that by:

```
Understand the command "discard" as something new.
```

This cuts it loose, so to speak, and ready to be given new meanings. If we check the Actions index again, we find no mention of "discard" - it is now a blank slate - but "drop" is still exactly as it was. We could now say something like:

```
Understand "discard [something]" as discarding.
```

(If we had declared that "drop" was something new, the whole thing would have happened in reverse, with "discard" retaining all of the original grammar. Inform does not distinguish between a command and its synonym.)

The "... as something new" sentence works even for a command which did not exist anyway, for instance with:

```
Understand the command "zylqix" as something new.
```

Of course this does nothing - but it is intentional that it generates no problem messages: it means that the sentence can be used to force a command to be fresh and untouched by previous definitions, which might be useful when working with extensions by other people.

It is also possible to clear out all the commands leading to a given action:

```
Understand nothing as taking.
```

The commands "take" and "get" will still exist, but now they'll only have their other senses (for taking off clothes, for getting out of boxes).

Start of Chapter 17: Understanding

Back to §17.2. New commands for old grammar

Onward to §17.4. Standard tokens of grammar

Example 292: **The Trouble with Printing** Making a READ command, distinct from EXAMINE, for legible objects.

Example 293: **Lanista 2** Randomized combat in which the damage done depends on what weapons the characters are wielding, and in which an ATTACK IT WITH action is created to replace regular attacking. Also folds a new DIAGNOSE command into the system.

---

§17.4. Standard tokens of grammar

We have already seen "[something]" and "[someone]", which are standard examples of "tokens of grammar" - patterns matched by suitable named things. There are several other standard tokens, provided not so much from necessity but to allow the story parser to be more graceful and responsive. "[someone]" matches the same possibilities as "[a person]" would, but the parser handles it a little better in cases of failure. These special tokens are best explained by looking at some of the examples in the standard grammar, which can be browsed in the Index of any story.

```
Understand "wear [something preferably held]" as wearing.
```

Here we expect that the named item will be one that is held by the player, and the parser will use this to resolve ambiguities between names of things carried and not carried. (If the action is one which positively requires that its noun be something carried, a command matching this token against something not carried will generate an automatic attempt to take it.)

```
Understand "take [things]" as taking.
Understand "drop [things preferably held]" as dropping.
```

"[things]" is like "[something]" but allows a list of items, or a vague plural like "all", to be typed. The result will be a sequence of actions, one for each item thus described. "[things preferably held]" is the analogous token for "[something preferably held]".

```
Understand "take [things inside] from [something]" as removing.
```

"[things inside]" matches only what is inside the second-named thing, and ensures that (for instance) the command "take all from box" does not also try to take the box.

```
Understand "put [other things] in/inside/into [something]" as inserting it into.
```

Similarly, "[other things]" will allow anything except the second-named thing. (Like "[things inside]" it is really only needed for handling containers.)

Finally there is "[any things]", which should be used only with care. This is like "[things]" but with no restriction at all on where the item comes from: it might be invisible, or from a different room, or out of play altogether. If we use this, we had better remember that it would match ALL, with quite extravagant consequences.

Start of Chapter 17: Understanding

Back to §17.3. Overriding existing commands

Onward to §17.5. The text token

Example 294: ★ **Shawn's Bad Day** Allowing the player to EXAMINE ALL.

Example 295: ★★★ **The Left Hand of Autumn** The possibility of using a [things] token opens up some interesting complications, because we may want actions on multiple items to be reported differently from actions on just one. Here we look at how to make a multiple examination command that describes groups in special ways.

---

§17.5. The text token

Most actions involve items: taking a vase, perhaps. As we shall see, they might also involve values, or a mixture of the two: turning a dial to 17 would involve both a thing (the dial) and a number (17). A few of Inform's built-in actions, however, can act on any text at all. For instance, asking the Sybil about the Persian army would involve a thing (the Sybil) and some text ("Persian army"). Inform does not try to understand automatically what that text might mean, or to relate it to any items, places or values it knows about: instead, Inform leaves that to the specific story to work out for itself, since the answer is bound to depend on the context. (In the chapter on Tables, we saw ways to compile tables of responses to particular topics of conversation.)

The token for "accept any text here" is just "[text]". For instance, if we create an action with:

```
Getting help about is an action applying to one topic.
```

We can then provide grammar for this action like so:

```
Understand "help on [text]" as getting help about.
```

When text like this is successfully matched, it is placed in a value called "the topic understood". (The term "topic" is used traditionally, really: most of the times one needs this feature, it's for a topic of conversation, or a topic being looked up in a book.)

The fact that "[text]" can match anything means that it's difficult to tell which version of a command was intended if they disagree only from a "[text]" onwards. For example, given:

```
Yelling specifically is an action applying to one topic. Understand "yell [text]" as yelling specifically. Understand "yell [text] at/to [someone]" as answering it that (with nouns reversed).
```

...Inform will in fact try the second possibility first, as being the more specific, but the result may freeze out the first possibility altogether due to autocompletion of commands.

Start of Chapter 17: Understanding

Back to §17.4. Standard tokens of grammar

Onward to §17.6. Actions applying to kinds of value

Example 296: **Ish.** A (very) simple HELP command, using tokens to accept and interpret the player's text whatever it might be.

Example 297: **Nameless** ASKing someone about an object rather than about a topic.

---

§17.6. Actions applying to kinds of value

Almost all actions apply to things: the player picks them up, pushes them, looks at them and so on. We only occasionally need to recognise other kinds of value, but when we do, we can. For instance:

```
Adjusting it to is an action applying to one thing and one number.

Understand "adjust [something] to [a number]" as adjusting it to.
```

The substitution "[a number]" matches any number (actually any whole number that is not too large) typed by the player. Inform checks the various kinds being used to make sure that everything matches, so, for instance, this would be disallowed:

```
Understand "adjust [something] to [something]" as adjusting it to.
```

Start of Chapter 17: Understanding

Back to §17.5. The text token

Onward to §17.7. Understanding any, understanding rooms

Example 298: **Safety** A safe whose dial can be turned with SPIN SAFE TO 1131, and which will open only with the correct combination.

Example 299: **Tom's Midnight Garden** A clock kind that can be set to any time using "the time understood"; may be turned on and off; and will advance itself only when running. Time on the face is also reported differently depending on whether the clock is analog or digital.

Example 300: **Ibid.** A system which allows the author to assign footnotes to descriptions, and permits the player to retrieve them again by number, using "the number understood". Footnotes will automatically number themselves, according to the order in which the player discovers them.

---

§17.7. Understanding any, understanding rooms

Ordinarily, if we write

```
Understand "manipulate [something]".
```

then the "[something]" will only match what is within reach or sight: this is the concept of "scope", which is what prevents a player from spookily acting on objects from a distance. The parser itself prevents the manipulation rules from ever being invoked on such distant items, which is as it should be.

Sometimes, though, we positively want to allow this possibility. If we use the special word "any", as in

```
Understand "manipulate [any door]".
```

then any door, anywhere in the model world, can be allowed in the player's command. (Of course, the manipulation rules may not do what the player hopes: all that has happened is that the command is now possible to type.) The "any" can be followed by any description of items or rooms, and the latter opens up new possibilities, since rooms are ordinarily never allowed to be named in the player's commands.

For example, the following gives the player the ability to walk between rooms without giving explicit directions of movement.

```
Going by name is an action applying to one thing.

Carry out going by name: say "You walk to [the noun]."; move the player to the noun.

Understand "go to [any adjacent visited room]" as going by name.
```

(This is really only a sketch: in a finished work, "go to" would produce helpful errors if non-adjacent but visited rooms were named, and we might also worry about rules applying to movement, because the method above will circumvent them.)

As might be expected, "[anything]" means the same as "[any thing]"; "[anybody]" and "[anyone]" mean the same as "[any person]"; and "[anywhere]" means the same as "[any room]".

Start of Chapter 17: Understanding

Back to §17.6. Actions applying to kinds of value

Onward to §17.8. Understanding names

Example 301: **One of Those Mornings** A FIND command that allows the player to find a lost object anywhere

Example 302: **Actaeon** A FOLLOW command allowing the player to pursue a person who has just left the room.

---

§17.8. Understanding names

So far in this chapter, Understand sentences have been used to give names to actions, but they can also be used to name objects - in particular, things and rooms.

This normally happens automatically. For instance, writing

```
The St Bernard is an animal in the Monastery Cages.
```

makes ST BERNARD refer to the dog, and MONASTERY CAGES refer to the room. But sometimes, as here, that isn't really enough. Why shouldn't the player type EXAMINE DOG? One way to allow this is to write:

```
Understand "dog" as the St Bernard.
```

Matters become more complicated when the player wants to refer to more than one object at once. When a kind is created, and the source text constructs multiple duplicate items of that kind, Inform generates a plural of the kind's name in order to understand commands referring to these multiples. For instance, given...

```
The Lake is a room. A duck is a kind of animal. Four ducks are in
  the Lake.
```

...the player can type TAKE DUCKS to try to pick up all four.

Once again the automatic behaviour can be enhanced:

```
Understand "birds" and "ruddy ducks" as the plural of duck.
```

Now TAKE BIRDS and TAKE DUCKS are equivalent. Plurals can even, strange as it may seem, be given for single things:

```
The magpie is in the Lake. Understand "birds" as the plural of the magpie.
```

And now TAKE BIRDS tries to take all four ducks and the magpie too.

Start of Chapter 17: Understanding
Back to §17.7. Understanding any, understanding rooms
Onward to §17.9. Understanding kinds of value

§17.9. Understanding kinds of value

In many cases, if K is the name of a kind of value, then Inform automatically makes an Understand token called "[K]" which matches only values of K. An example is "[number]", which matches text like 203 or SEVEN. There is a chart of the kinds of value in the Kinds index for a project, showing which ones can be understood in this way.

In particular, any newly created kind of value can always be understood. We make good use of that in the example story "Studious":

```
"Studious"

The Studio is a room. "The unreal world of the photographic studio, full of fake furniture, cantilevered stands and silver-white shades like miniature parachutes." The lumpy black camera is in the Studio. "A lumpy black camera hangs from a tripod."

The rake-thin model is a woman in the Studio. "A rake-thin model, exquisitely bored and boringly exquisite, angles herself indolently."

Limb is a kind of value. The limbs are left leg, left arm, right leg and right arm.

Detailing is an action applying to one limb and one visible thing, requiring light. Check detailing: if the camera is not carried then say "You can hardly photograph without a camera, now can you?" instead. Report detailing: say "Click! You take a detail photograph of the [limb understood] of [the second noun]."

Understand "photograph [limb] of [a person]" as detailing.

Test me with "get camera / photograph left leg of model".
```

Note the way we can refer to the limb mentioned by the player as the "limb understood". Similarly, we could talk about the "number understood" if the value parsed had been a number, and so on.

One of the built-in kinds of value is worth special note: time. A time can hold either a specific time of day, such as 10:23 PM, or a duration of something, such as 21 minutes. The "[a time]" token matches times of day, such as 10:15 AM or MIDNIGHT. But 10 MINUTES wouldn't be recognised by "[a time]" since it isn't a specific moment in the day. To get around this, an alternative version called "[a time period]" is available. So:

```
Understand "wait for [a time period]" as ...
```

would match WAIT FOR AN HOUR or WAIT FOR TWO HOURS 12 MINUTES.

Start of Chapter 17: Understanding

Back to §17.8. Understanding names

Onward to §17.10. Commands consisting only of nouns

Example 303: ★ **Pages** A book with pages that can be read by number (as in "read page 3 in...") and which accepts relative page references as well (such as "read the last page of...", "read the next page", and so on).

Example 304: ★★ **Down in Oodville** Offering the player a choice of numbered options at certain times, without otherwise interfering with his ability to give regular commands.

Example 305: ★★★ **Straw Into Gold** Creating a Rumpelstiltskin character who is always referred to as "dwarf", "guy", "dude", or "man" -- depending on which the player last used -- until the first time the player refers to him as "Rumpelstiltskin".

---

§17.10. Commands consisting only of nouns

In every example so far, and in almost all practical cases, the first word in a command which results in an action will be something fixed: a verb, in fact. When we write

```
Understand "photograph [something]" as photographing.
```

we are saying that the first word of such a command will always be "photograph". Occasionally, though, we would like to understand a noun as a command, perhaps in a situation where the command is obvious. If we say:

```
Understand "[something]" as examining.
```

then the command "examine" will be implicit when the player types a bare noun:

```
A red box and a blue ball are here.
> BALL
The blue ball is plaited from many small leather patches.
```

so that the command "ball" has resulted in the action "examining the blue ball".

This is a feature which should be used sparingly, since it could easily lead to confusion if not carefully explained to the player. By default, it is not used at all.

It also has what may be a serious limitation: verbless commands like this work only when typed by the player as actions to follow - they do not work as instructions for other people. So for instance SVEN, BALL would not ask Sven to try examining the ball - instead it would generate the action "answering ball to Sven". (This is because the Inform parser decides whether PERSON, SOME TEXT is a request or just conversation by looking at the first word after the comma to see if it's a command.)

Start of Chapter 17: Understanding

Back to §17.9. Understanding kinds of value

Onward to §17.11. Understanding values

Example 306: **Misadventure** A going by name command which does respect movement rules, and accepts names of rooms as commands.

Example 307: **Safari Guide** The same functionality, but making the player continue to move until he reaches his destination or a barrier, handling all openable doors on the way.

---

§17.11. Understanding values

"Understand" can be used to supply new ways to talk about both things and other values. For instance, if we create:

```
A brass lantern is in the Building.
```

then it can be called "brass", or "lantern", but not "lamp": Inform does not really know what these words mean, and has no grasp of synonyms. We can arrange for "lamp" to work as well like so:

```
Understand "lamp" as the lantern.
Understand "old lamp" as the lantern.
```

With care, we can do the same trick for entire kinds of thing at once. It is not ordinarily the case that a thing can be called by the name of its kind: if we put a woman called April into a room, then she can usually be called "April", but not "woman". (The exception is when we do not specify any name for her - in that case, Inform will give up and call her just "woman".) So there is not usually any form of words which can refer to anything of a given kind. If we should want this, we have to say so explicitly:

```
Understand "machine" as a device.
```

Device is a kind, so now the word "machine" can be used to refer to any device: if there are two in the same place, the result might play out like so:

```
>switch machine on
Which do you mean, the bale twiner or the grain thresher?
>twiner
You watch absorbed as a perfect cube of hay is trussed up like a parcel.
```

Similarly, we might conceivably want to allow new ways to recognise values - in this case, a number:

```
Understand "eleventy-one" as 111.
```

When making complicated names, we need to watch out for the possibility of writing a definition which will cause Inform to go around in circles (something which will show up as a "Too many activities at once" run-time problem). For instance,

```
Understand "[thing] substitute" as the placebo.
```

will fail because Inform, working left to right, needs to look for every possible object name before it can progress: one possibility is the placebo itself: to check that, it needs to look for every possible object name: and so on, never finishing. A definition like this one very likely matches too much in any case (would we really want to accept PLACEBO SUBSTITUTE or CIGARETTE SUBSTITUTE SUBSTITUTE SUBSTITUTE here, as the definition implies?).

Start of Chapter 17: Understanding

Back to §17.10. Commands consisting only of nouns

Onward to §17.12. This/that

Example 308: **Palette** An artist's workshop in which the canvas can be painted in any colour, and where painterly names for pigments ("cerulean") are accepted alongside everyday ones ("blue").

Example 309: **Baritone, Bass** Letting the player pick a gender (or perhaps other characteristics) before starting play.

---

§17.12. This/that

We have already seen "or" used in "Understand" sentences:

```
Understand "scarlet" or "crimson" as red.
```

In general, any number of alternative forms can be given which are to be understood as the same thing (in this case the colour red). When the alternatives are in any way complicated, "or" should always be used, but a shorthand form is allowed for simple cases where it is only a matter of a single word having several possibilities:

```
Understand "reach underneath/under/beneath [something]" as looking under.
```

This is shorthand for:

```
Understand "reach underneath [something]" or "reach under [something]" or "reach beneath [something]" as looking under.
```

Which in turn is shorthand for:

```
Understand "reach underneath [something]" as looking under. Understand "reach under [something]" as looking under. Understand "reach beneath [something]" as looking under.
```

It's possible also to make that second word optional:

```
Understand "reach underneath/under/beneath/-- [something]" as looking under.
```

because "--" is read by Inform as "no word at all". If "--" is an option, it can only be given once and at the end of the list of possibilities.

To recapitulate: the slash "/" can only be used between single, literal words, and is best for the wayward prepositions of English ("in/into/inside", and so forth). For anything more complex, always use "or".

Start of Chapter 17: Understanding

Back to §17.11. Understanding values

Onward to §17.13. New tokens

---

§17.13. New tokens

We have now made good use of square-bracketed tokens, such as "[something]", in a variety of "Understand..." sentences. It is sometimes convenient to create new tokens of our own, to match whatever grammar we choose: this enables complicated knots of grammar to be used in many different "Understand..." sentences without having to write it all out each time.

For instance, here are new tokens: one for each of two groups of alternative prepositions.

```
Understand "beneath/under/by/near/beside/alongside/against" or "next to" or "in front of" as "[beside]".

Understand "on/in/inside" or "on top of" as "[within]".
```

Again, note that the slash indicates a choice between words only, not between entire phrases. For instance, if we write:

```
Understand "red bird/robin" as "[robin]".
```

then the two alternative forms are "red bird" and "red robin", not "red bird" and "robin". By contrast,

```
Understand "red bird" or "robin" as "[robin]".
```

will understand either "red bird" or "robin" but not "red robin". If we want to capture all three forms, we might define

```
Understand "red bird/robin" or "robin" as "[robin]".
```

Start of Chapter 17: Understanding

Back to §17.12. This/that

Onward to §17.14. Tokens can produce values

Example 310: **Lies** Commands to allow the player to lie down in three different ways.

---

§17.14. Tokens can produce values

The examples just seen were tokens which simply matched specific words typed by the player, but newly created tokens can also produce values:

```
Colour is a kind of value. The colours are red, green and blue. Understand "colour [a colour]" or "[a colour] shade" as "[tint]".
```

Here the "[tint]" token matches, for instance, "colour red" and "blue shade", which would result in the values red and blue, respectively.

Tokens are not allowed to produce more than one value, and if several patterns are given to define them then those patterns have to be compatible. That means the following is disallowed, since it might work out to a colour, or to an object, leaving Inform unable to judge whether an action can safely be applied to the result.

```
Understand "colour [a colour]" or "[something]" as "[tint]".
```


Start of Chapter 17: Understanding


Back to §17.13. New tokens


Onward to §17.15. Understanding things by their properties

---

§17.15. Understanding things by their properties

Items are ordinarily understood only by their original given names. For instance, if we have:

```
In the Herb Garden is a china pot.
```

then the player could refer to this as "pot", "china pot" or "china". We can embellish this by adding extra forms:

```
Understand "chinese pot" or "chinese vase" as the china pot.
```

But suppose the pot changes its nature in the course of play? If we have:

```
The china pot can be unbroken or broken. The china pot is unbroken.

After dropping the china pot:
    say "Crack!";
    now the china pot is broken;
    now the printed name of the pot is "broken pot".
```

So now the player would reasonably expect to call it "broken pot", a wording which would have been rejected before. We can achieve this by writing:

```
Understand the unbroken property as describing the pot.
```

which allows "unbroken" or "broken" to describe the pot, depending on its state. And, since the player might well use a different adjective but with the same idea in mind, we can even add:

```
Understand "shattered" or "cracked" or "smashed" as broken. Understand "pristine" as
unbroken.
```

This is something of a toy example, but the feature looks rather more useful when there are more pots than just one:

```
"Terracotta"

A flowerpot is a kind of thing. A flowerpot can be unbroken or broken. Understand the
broken property as describing a flowerpot.

After dropping an unbroken flowerpot:
    say "Crack!";
    now the noun is broken;
    now the printed name of the noun is "broken flowerpot";
    now the printed plural name of the noun is "broken flowerpots".

The Herb Garden is a room. In the Herb Garden are ten unbroken flowerpots.
```

We then have the dialogue:

```
Herb Garden
You can see ten flowerpots here.
>get two flowerpots
flowerpot: Taken.
flowerpot: Taken.
>drop all
flowerpot: Crack!
flowerpot: Crack!
>look
Herb Garden
You can see two broken flowerpots and eight flowerpots here.
>get an unbroken flowerpot
Taken.
```

and so on and so forth.

There are in fact two slightly different forms of this kind of sentence:

```
Understand the broken property as describing a flowerpot.
Understand the broken property as referring to a flowerpot.
```

The only difference is that in the "describing" case, the property's name alone can mean the thing in question - so "take unbroken" will work; whereas, in the "referring to", the property's name can only be used as an adjective preceding the name of thing itself - so "take unbroken flowerpot" will work but "take unbroken" will not.

Start of Chapter 17: Understanding

Back to §17.14. Tokens can produce values

Onward to §17.16. Understanding things by their relations

Example 311: **Aspect** Understanding aspect ratios (a unit) in the names of televisions.

Example 312: **Hymenaeus** Understanding "flaming torch" and "extinguished torch" to refer to torches when lit and unlit.

Example 313: **Channel 1** Understanding channels (a number) in the names of televisions.

Example 314: **Terracottissima** The flowerpots once again, but this time arranged so that after the first breakage all undamaged pots are said to be "unbroken", to distinguish them from the others.

Example 315: **Peers** The peers of the English realm come in six flavours - Baron, Viscount, Earl, Marquess, Duke and Prince - and must always be addressed properly. While a peerage is for life, it may at the royal pleasure be promoted.

Example 316: **Channel 2** Understanding channels (a number) in the names of televisions, with more sophisticated parsing of the change channel action.

Example 317: **Terracottissima Maxima** Flowerpots with textual names that might change during play.

Example 318: **Tilt 1** A deck of cards with fully implemented individual cards, which can be separately drawn and discarded, and referred to by name.

---

§17.16. Understanding things by their relations

Sometimes it makes sense for the name of something to involve the names of other things to which it is related. For instance, if we say TAKE THE BOTTLE OF WINE, we mean that the bottle currently contains wine - if it were the very same bottle containing water, we would call it something else.

For names which must involve related names, a special form of token is provided. For instance, we could say:

```
A box is a kind of container. Understand "box of [something related by containment]" as a box.

The Toyshop is a room. The red box is a box in the Toyshop. Some crayons are in the red box.
```

and now TAKE BOX OF CRAYONS will work, because CRAYONS matches against "[something related by containment]" for the red box - or it does for as long as the crayons are there. We can have similar matches against relations of all kinds, but have to name the relation explicitly. (See the examples at the end of this section for plenty of cases.)

We can also reverse the sense. If we write:

```
A box is a kind of container. Understand "box in [something related by reversed containment]" as a box.
```

```
The Toyshop is a room. The crate and the hammock are in the Toyshop. In the crate is a
box. In the hammock is a box.
```

then TAKE THE BOX IN THE HAMMOCK will work: here, the relation goes the other way, because the box is being contained by the other-named item, rather than doing the containing.

 Start of Chapter 17: Understanding

 Back to §17.15. Understanding things by their properties

 Onward to §17.17. Context: understanding when

 Example 319:  **Cinco** A taco shell that can be referred to (when it contains things) in terms of its contents.

 Example 320:  **Puncak Jaya** When a character is not visible, responding to such commands as EXAMINE PETER and PETER, HELLO with a short note that the person in question is no longer visible.

 Example 321:  **Whither?** A door whose description says where it leads; and which automatically understands references such as "the west door" and "the east door" depending on which direction it leads from the location.

 Example 322:  **Claims Adjustment** An instant camera that spits out photographs of anything the player chooses to take a picture of.

---

§17.17. Context: understanding when

We have now seen several different forms of "Understand" sentence: for instance,

```
Understand the colour property as describing a building block.
Understand "mix [colour] paint" as mixing paint.
Understand "rouge" as red.
Understand "curious girl" as Alice.
```

Any of these may optionally have a condition tacked on: for instance,

```
Understand "mix [colour] paint" as mixing paint when the location is the Workshop.
Understand "rouge" as red when the make-up set is visible.
```

In principle, "when ..." can take in any condition at all. In practice a little care should be exercised not to do anything too slow, or which might have side-effects. (For instance, referring the decision to a phrase which then printed text up would be a bad idea.) Moreover, we must remember that the "noun" and "second noun" are not known yet, nor do we know what the action will be. So we cannot safely say "when the noun is the fir cone", for instance, or refer to things like "the number understood". (We aren't done understanding yet.) If we want more sophisticated handling of such cases, we need to write checking rules and so on in the usual way.

Contexts can be useful to make sense of things having different names depending on who is being spoken to, as here:

```
Understand "your" as a thing when the item described is held by the person asked.
```

With this rule in place FRODO, GIVE ME YOUR RING means that Frodo will know which ring is meant, even if there are a couple of dozen other rings present.

If the name of something has to change completely, perhaps because the player's understanding of events has changed completely, then Inform's standard way of handling names can be a nuisance. When an item or room is created, Inform automatically makes its name understood as referring to it (in fact, it makes each individual word in that name understood). For instance,

```
The Wabe is a room. The blue peacock and the sundial are in the Wabe.
```

means that the player can type EXAMINE BLUE PEACOCK or PUSH SUNDIAL or SHOWME WABE or TAKE BLUE, and so on. This is almost always a good thing, and here there's no problem, because peacocks and sundials are not usually disguised. But here is a case where a disguise is needed:

```
The secret document is a privately-named thing in the drawer.
The printed name of the secret document is "[if the secret document is handled]secret
document[otherwise]dusty paper".
Understand "dusty" and "paper" as the secret document.
Understand "secret" and "document" as the secret document when the secret document
is handled.
After taking the secret document for the first time: say "Heavens! It is the secret
document!"
```

As this demonstrates, the either/or property "privately-named" makes Inform create a thing or room which starts out with no automatic understandings at all. The name it happens to have in the source text is ignored. If we simply write:

```
The ungraspable concept is a privately-named thing in the Dining Room.
```

then nothing the player can type will ever refer to it; though he will see it, and even be able to pick it up by typing TAKE ALL.

The reverse property is "publicly-named", which all things and rooms are by default.

Inform has four built-in kinds of object (room, thing, direction and region), and all of those have this either/or property. When we create new kinds, they're normally kinds of those four fundamental ones, so they pick up the same behaviour. But if we create a new kind of object outside of these four, that won't be true unless we make it so:

```
A concept is a kind of object. A concept can be privately-named or publicly-named. A
concept is usually publicly-named.
```

(Privately-named is a property which only affects how Inform creates the object, and it can't usefully be given or taken away during play. "Understand ... when ..." is the way to change names during play.)

Start of Chapter 17: Understanding

Back to §17.16. Understanding things by their relations

Onward to §17.18. Changing the meaning of pronouns

Example 323: **Quiz Show** In this example by Mike Tarbert, the player can occasionally be quizzed on random data from a table; the potential answers will only be understood if a question has just been asked.

Example 324: **Bibliophilia** A bookshelf with a number of books, where the player's command to examine something will be interpreted as an attempt to look up titles if the bookshelf is present, but otherwise given the usual response.

---

§17.18. Changing the meaning of pronouns

The pronouns IT, HIM, HER and THEM are constantly adjusted during play, to save the player time when typing commands. If the player types EXAMINE NECKLACE on one turn, it's sufficient to type TAKE IT on the next, and IT will be understood as meaning whatever NECKLACE meant last turn.

All of that happens automatically, but once in a while the result can be unfortunate. Suppose that when the player examines the necklace, a security system automatically drugs her unconscious, and she wakes up in a cell, hours later, and is told that the cell is bare except for a key on the floor. If she types TAKE IT, she clearly doesn't mean IT to mean the necklace any more; she means the key. Inform's parser can't make guesses like this, so the following phrase can be used to help it.

set pronouns from (object)

This phrase adjusts the meaning of pronouns like IT, HIM, HER and THEM in the command parser as if the object mentioned has become the subject of conversation. Example: the combination of

```
set pronouns from the key;
set pronouns from Bunny;
```

might change IT to mean the silver key and HIM to mean Harry "Bunny" Manders, while leaving HER and THEM unaltered.

Start of Chapter 17: Understanding

Back to §17.17. Context: understanding when

Onward to §17.19. Does the player mean...

Example 325: **Pot of Petunias** Responding sensibly to a pot of petunias falling from the sky.

---

§17.19. Does the player mean...

When the player types an ambiguous reference, we need to work out what is meant. Consider the following source text:

```
The Champs du Mars is a room. The great Eiffel Tower is here. "The great Tower stands high over you." The souvenir model Eiffel Tower is here. "Comparatively tiny is the souvenir version."
```

Now suppose the player types GET TOWER. The response will be:

```
Which do you mean, the great Eiffel Tower or the souvenir model Eiffel Tower?
```

Which is a silly question, exposing our work of IF as something artificial. It's obvious to the author of the source text, and to the player, that the souvenir must be what is meant: but this is not obvious to the computer program running the story. Works of IF gain a subtle feeling of quality from being able to understand ambiguous references of the kind above, and Inform provides us with a way to do this by giving the parser clues in the form of "Does the player mean..." rules. For instance, if we add:

```
Does the player mean taking the great Eiffel Tower: it is very unlikely.
```

then the response to GET TOWER will now be:

```
(the souvenir model Eiffel Tower)
Taken.
```

"Does the player mean..." rules look at the actions which are possible interpretations of what the player typed, and grade them according to how likely they seem. (Note that these rules are only ever used to handle ambiguities: if the player unambiguously types GET GREAT EIFFEL TOWER, that will be the action. And the rules are only used where they are able to make a decision: if there are still multiple equally plausible meanings, the parser will ask about all possibilities, not just the most likely ones.) Rules in this rulebook can either decide nothing, or come up with one of the following verdicts:

```
it is very likely
it is likely
it is possible
it is unlikely
it is very unlikely
```

If there are no "does the player mean" rules, or the rules make no decision on a given possible action, it will be ranked as "it is possible".

We may use these rules to affect all sorts of interaction with a specific object or kind of object, as in

```
Does the player mean doing something with the cursed dagger of Thog: it is very unlikely.
Does the player mean doing something with the cursed dagger of Thog when the player is hypnotized: it is likely.
```

...and so on.

Notice that we can also make rules about actions that apply to two objects, so for instance:

```
Does the player mean throwing the can of shoe polish at the shoe polish vending
machine: it is likely.
```

which nicely clarifies THROW POLISH AT POLISH, but does not comment on the likelihood of throwing the can at other things or of throwing other things at the vending machine. Moreover, the (suspected) identity of the first item will be known when the rule is consulted; thus

```
Does the player mean tying the noun to the noun: it is very unlikely.
```

will tell Inform to prefer not to tie something to itself if other interpretations are available.

But there is a caveat. There are some cases where this mechanism will not in fact help Inform to choose its way out of an ambiguous command, because of the way it parses one noun at a time. It usually needs to understand the first noun before it will even try to make sense of the second. So a rule like:

```
Does the player mean throwing the can of shoe polish at the tree: it is likely.
```

may not work if the player types THROW POLISH AT TREE and POLISH is ambiguous, because when the parser is trying to understand POLISH, it hasn't yet seen to the end of the command and realised that the second noun will be the tree; so the second noun is unset and the rule won't match.

As a caveat to the caveat, the "inserting it into", "removing it from" and "putting it on" actions have this slightly back to front. These are parsed using the (little-used) "[other things]" or "[things inside]" tokens, and the Inform parser tries to detect the second noun before the first one, since the identity of the first has to depend on the second. So for instance if the situation contains "an oak tree" and also "an oak chest", we could write:

```
Does the player mean inserting into the oak chest:
    it is very likely.
```

which would successfully make PUT COIN IN OAK mean the chest, not the tree. (Note the way we write "inserting into" without saying anything about what's being inserted, not even that it's "something".)


Start of Chapter 17: Understanding


Back to §17.18. Changing the meaning of pronouns


Onward to §17.20. Multiple action processing


Example 326:  **Masochism Deli** Multiple potatoes, with rules to make the player drop the hot potato first and pick it up last.

---

§17.20. Multiple action processing

When the player types a command like DROP ALL, this is (usually) a request to carry out more than one action. After the command parser has decided what constitutes "ALL" (a process which can be influenced using the "deciding whether all includes" activity), it forms up a list and then runs through it, starting an action for each in turn. The result usually looks something like this:

```
>GET ALL
foxglove: Taken.
snake's head fritillary: Taken.
```

However, by adding rules to the rulebook:

```
multiple action processing rules
```

we can take a look at the actions intended, and rearrange or indeed change them before they take effect. To do that, we have to deal with a special list of objects. For two technical reasons this isn't stored as a "list of objects that varies" - first because it needs to exist even in low-memory situations where we can't afford full list-processing, and second because there are times when changing it might be hazardous. Instead, two phrases are provided to read the list and to write it back:

multiple object list ... list of objects

This phrase produces the current multiple object list as a value. The list will be the collection of objects found to match a plural noun like ALL in the most recent command typed by the player. If there is no multiple object, say if the command was TAKE PEAR, the list will be empty: it won't be a list of size 1.

alter the multiple object list to (list of objects)

This phrase sets the multiple object list to the given value. The list is ordinarily the collection of objects found to match a plural noun like ALL in the most recent command typed by the player, but using this phrase at the right moment (before the "generate action rule" in the turn sequence rules takes effect).


Start of Chapter 17: Understanding


Back to §17.19. Does the player mean...


Onward to §17.21. Understanding mistakes


Example 327:  **The Best Till Last** Reordering multiple objects for dramatic effect.


Example 328:  **Western Art History 305** Allowing EXAMINE to see multiple objects with a single command.

---

§17.21. Understanding mistakes

When inspiration strikes the player, he can usually be relied upon to make a good-faith effort to communicate the new idea: he will guess the right command. If he guesses wrongly, the mistake is probably the author's, because a good author will try to anticipate all possible wordings and make all of them work.

Nevertheless it is sometimes good practice to nudge the player towards the right wording - particularly if the player has the right idea but is not explicit enough: for instance, typing TALK TO JUDGE when we really want to know what is to be said (JUDGE, GUILTY); or if the player tries something like PLAY CHESS rather than MOVE PAWN TO KING 4. Similarly, if we make a casual reference such as "In your childhood days, you loved sliding in stocking feet across this hallway", a player might type SLIDE IN STOCKING FEET: a nice idea, and which deserves a nice response, even though it asks to do something beyond the scope of the story.

Inform provides a simple mechanism for recognising a command but at the same time recognising that *it does not properly specify an action*. Such commands are called "mistakes", for the sake of a memorable term, but the player has not really behaved badly, and should be helped rather than reproved. For instance:

```
Understand "act" as a mistake.
```

While that works - the command to "act" is indeed rejected - it is not very good, because no very helpful message is brought up. The following is much better:

```
Understand "act" as a mistake ("To join the actors, you have to adopt a role in the play! Try PLAY HAMLET or similar.").
```

Or we could once again insist on a given context:

```
Understand "act" as a mistake ("To join the actors, you have to adopt a role in the play! Try PLAY HAMLET or similar.") when the location is the Garden Theatre.
```

That still has the drawback that the command "act hamlet" will not be recognised: so the final version we want is probably

```
Understand "act [text]" as a mistake ("To join the actors, you have to adopt a role in the play! Try PLAY HAMLET or similar.") when the location is the Garden Theatre.
```

since the "[text]" part will soak up any words the player types (or none), meaning that any command at all whose first word is "act" will be matched.

We need to be careful to avoid circular things like this:

```
Understand "[text]" as a mistake ("'[the topic understood]' is something I really wish you wouldn't say.") when the topic understood is a topic listed in table 1.
```

This doesn't work because the topic understood isn't set until the line has been understood, but Inform checks the "when..." condition before it tries to understand the line. Indeed, even this:

```
Understand "[text]" as a mistake ("'[the topic understood]' is something I really wish you wouldn't say.").
```

is unsafe (quite apart from being unwise!) - again, "topic understood" doesn't exist for a mistake, because in a mistake, nothing is understood.

The following is often useful during beta-testing of a new work, though we would not want it in the final published edition. Many authors like to ask their testers not to try anything in particular, simply to play naturally: but to record the transcript of the session, and email it back to the author. The following command is a device to allow the tester to type a comment in to the transcript:

```
Understand "* [text]" as a mistake ("Noted.").
```

For instance, the tester might type "* DIDN'T WE SAY DARCY WAS TALL?", to which the story would reply "Noted." - and the author can search for such comments when receiving the transcript.

If we are careful, we can make the reply depend on what was typed in the mistaken command:

```
Understand "steal [something]" as a mistake ("Just TAKE [the noun] and leave without paying: that's stealing in my book.").
```

The care comes in because Inform applies much less checking to mistakes than to other actions, and odd errors will result if we try to refer to (say) "the second noun" in a command which did not have a second noun.

It's probably wise to take particular care if using "as a mistake" with any command which might include the mistake among what the player calls ALL: for example, if "take [sydney harbour bridge]" is understood as a mistake, then TAKE ALL will may result in this, even though the player doesn't intend any such thing.


Start of Chapter 17: Understanding


Back to §17.20. Multiple action processing


Onward to §17.22. Precedence


Example 329:  **Query** Catching all questions that begin with WHO, WHAT, WHERE, and similar question words, and responding with the instruction to use commands, instead.


Example 330:  **The Gorge at George** If the player tries to TALK TO a character, suggest alternative modes of conversation.


Example 331:  **Hot Glass Looks Like Cold Glass** Responding to references to a property that the player isn't yet allowed to mention (or when not to use "understand as a mistake").

---

§17.22. Precedence

When several different lines of grammar are supplied to meet the same circumstances, it makes a big difference what order they are tried in. For instance, suppose we have:

```
Understand "photograph [a door]" as photographing.

Understand "photograph [an open door]" as photographing.
```

The second line is more specific than the first, so Inform takes these grammar lines the other way around: it checks for "open door" before it checks for "door". That didn't matter here, since both lines came out with the same result (the action of photographing), but it matters very much in the next example:

```
Understand "employ [a door]" as opening.

Understand "employ [an open door]" as entering.
```

More subtle is a line already seen:

```
Understand "on/in/inside" or "on top of" as "[within]".
```

Here Inform puts "on top of" before "on/in/inside", since otherwise only the "on" of "on top of" will be recognised.

Mistakes always take precedence over non-mistakes: this is intended to make sure that

```
Understand "take umbrage" as a mistake ("Nobody takes umbrage in this story,
mister.").
```

will take precedence over

```
Understand "take [something]" as taking.
```

even if there is, in fact, a character called Mr Nimbus Umbrage so that the command could conceivably make sense.

Finally, there are a few grammars where the number of values produced is different in different lines. For example, the Standard Rules include these among the possible "put" commands:

```
Understand "put [something preferably held] on" as wearing.
Understand "put [other things] on/onto [something]" as putting it on.
```

One produces a single object, the other produces two. Inform gives precedence to the first of these, that is, it tries the one with fewer values first. This is important when reading commands like "PUT MARCH ON WASHINGTON SHIRT ON", and also prevents bogus auto-completions, in which PUT HAT ON might wrongly be auto-completed as if it were PUT HAT ON THE TABLE.

Start of Chapter 17: Understanding

Back to §17.21. Understanding mistakes

Onward to Chapter 18: Activities: §18.1. What are activities?

Example 332: ★ **Some Assembly Required** Building different styles of shirt from component sleeves and collars.

Example 333: ★★★ **Lakeside Living** Similar to "Lemonade", but with bodies of liquid that can never be depleted, and some adjustments to the "fill" command so that it will automatically attempt to fill from a large liquid source if possible.

Examples from Chapter 17: Understanding

Start of this chapter

Chapter 18: Activities

Indexes of the examples

286 ★ **Example** **Indirection** *RB*

Renaming the directions of the compass so that "white" corresponds to north, "red" to east, "yellow" to south, and "black" to west.

In Mayan culture, colours seem to have been used as names for the primary directions: for instance, "red" implies east as the colour of sunrise. So the following might be a stylish touch for a game in which the player has to get inside the Mayan world-view:

```
"Indirection"

Understand "white" and "sac" as north. Understand "red" and "chac" as east.
Understand "yellow" and "kan" as south. Understand "black" and "chikin" as
west.
```

We could also use a colour as a verb:

```
Understand "turquoise" and "yax" as looking.
```

And now a few extra rooms to try it out in:

```
The Square Chamber is a room. "A sunken, gloomy stone chamber, ten yards
across. A shaft of sunlight cuts in from the steps above, giving the chamber a
diffuse light, but in the shadows low lintelled doorways to east and south lead
into the deeper darkness of the Temple."

The Wormcast is east of the Square Chamber. The Corridor is south of the
Square Chamber.

Test me with "kan / white / chac / black".
```

287

Example XYZZY

RB

Basics of adding a new command reviewed, for the case of the simple magic word XYZZY.

We have seen before how to define a new action from scratch, but we may want to review here, using a simple command that requires no objects.

```
"XYZZY"

Understand "xyzzy" or "say xyzzy" or "cast xyzzy" as casting xyzzy.

Casting xyzzy is an action applying to nothing.

Check casting xyzzy:
    if the player does not wear the amulet of elocution, say "You are unable to
articulate the second 'z' separately from the first, and the spell fails in a
disdainful puff. Must be Parisian magic." instead;
    if the player has the plate, say "The plate of cheeses twitches uncomfortably,
aware that it should be doing something, but not sure what." instead.

Carry out casting xyzzy:
    move the plate to the player.

Report casting xyzzy:
    say "Under the influence of the Amulet of Elocution, you pronounce this as
Xhi-zee. And lo, from nowhere, a [plate] appears!"

The amulet of elocution is a wearable thing. It is carried by the player. The
description is "A heavy gold ring on a chain. If heated in an ordinary house fire, it
glows with the words, 'Moses Supposes His Toeses Are Roses.'"

The plate is a portable supporter. On the plate is a very ripe ooze. Instead of
smelling the ooze, say "It smells like socks. This is going to be wonderful." The
ooze is edible. The printed name of the plate is "plate[if the plate supports the
ooze] of cheese[end if]". The description of the ooze is "Definitely genuinely
cheese." Understand "cheese" as the ooze.

Instead of eating the ooze: now the ooze is nowhere; say "You are
transported..."; move the player to Paradise.

The Cheez Factory is a room. "All around you are squares of pressed orange
polymer, or possibly cheez. Your only hope is the magic word your uncle taught
you: XYZZY." The squares of pressed orange polymer are scenery in the
Factory. The description is "You see nothing special about the squares of
pressed orange polymer. Nothing special at all." Understand "square" or "cheez"
as the squares.

Paradise is a room. The description is "Well, it might just be one of the posh
upper rings of purgatory, if you're entirely honest with yourself."

Test me with "x squares / x amulet / x cheese / xyzzy / wear amulet / xyzzy / x
ooze / smell ooze / eat ooze".
```

XYZZY is a magic word from the original Adventure, and many other games respond to it with some sort of amusing message.

---

288  **Example Xylan** *RB*

Creating a new command that does require an object to be named; and some comments about the choice of vocabulary, in general.

If we wanted to define a brand new verb that did affect a specific object, we might begin like this:

```
"Xylan"

Understand "hydrolyze [something]" as hydrolyzing. Hydrolyzing is an action
applying to one thing.

Carry out hydrolyzing:
    say "[The noun] cannot be hydrolyzed."

Instead of hydrolyzing the xylan:
    move the xylose to the holder of the xylan;
    now the xylan is nowhere;
    say "At once the xylan becomes xylose."

Plant Cell Wall is a room.

There is a xylose sample. The xylan sample is a thing in Plant Cell Wall. The
description of the xylan is "A polysaccharide. Totally useless. If only you had
some xylose, instead!" The description of the xylose is "Awesome!"

Test me with "x xylan / hydrolyze xylan / x xylose".
```

Of course, how our players will ever solve this problem is another question (especially if their biology and chemistry are both rusty). When adding entirely new commands to a game, it is often a good idea to provide as many ways of phrasing the command as possible; to drop hints about the correct phrasing within the game's text; or even to tell the player about the expanded command list in some documentation or help at the beginning of the game. So for instance we might also add

```
Understand "break down [something] with water" or "break [something] down
with water" as hydrolyzing.
```

And these lines will also provide syntax for our new command, without interfering with the previous syntax. It's also good to anticipate alternative (British or American) spellings. People's typing habits are hard to overcome, even if they know you are spelling the word the other way. It is probably best not to annoy them unduly. So:

```
Understand "hydrolyse [something]" as hydrolyzing.
```

Then some text in-game might offer a clue, subtle or (since this is an example) blunt:

```
Instead of examining the player, say "You're a drop of water, which means that
you can break down certain chemicals!"

Understand "break down [something]" or "break [something] down" as
hydrolyzing.
```

And finally, we could try adding instructions explicitly:

```
Understand "help" or "hint" or "hints" or "instructions" or "info" or "about" as
asking for help. Asking for help is an action out of world. Carry out asking for
help: say "The following commands are understood, in addition to the standard
ones: EVAPORATE, FREEZE, HYDROLYZE, SUBLIME..."

Test more with "help / x me / break down xylan"
```

...though of course in fact these other commands won't be available until we define them, too.

This last approach, defining all the extra commands up front, is especially useful if these commands are very technical or unusual; if they are needed early in the game, before you've a chance to educate the player; or if they are not suggested by any in-game objects. A player who encounters a tool with an obvious use, such as a hairbrush, will likely think of trying to BRUSH things with it. It's harder to rely on his guessing actions that are both outside the range of usual commands and unrelated to any of the visible props, however.

289

Example **Alpaca Farm**

RB

A generic USE action which behaves sensibly with a range of different objects.

This example takes the ordering of grammar lines to its logical extreme, sorting the player's input into different categories depending on the kind and condition of the objects mentioned.

```
"Alpaca Farm"

Understand "use [an edible thing]" as eating.

Understand "use [a wearable thing]" as wearing.

Understand "use [a closed openable container]" as opening. Understand "use
[an open openable container]" as closing.

Understand "use [something preferably held] on [a locked lockable thing]" as
unlocking it with (with nouns reversed). Understand "use [something preferably
held] on [an unlocked lockable thing]" as locking it with (with nouns reversed).

Understand "use [a switched off device]" as switching on.
```

Understand "use [something]" as using. Using is an action applying to one thing. Carry out using: say "You will have to be more specific about your intentions."

Understand "use [a door]" as opening. Understand "use [an open door]" as entering.

The Llama Pen is a room. North of the Pen is the gate. The gate is a door. North of the gate is the Rocky Path. The brown llama is an animal in the Llama Pen.

Appearance is a kind of value. The appearances are muddy, scruffy, fluffy, and dapper. The brown llama has an appearance. The brown llama is muddy. Before printing the name of the brown llama, say "[appearance] ". Before printing the name of the brown llama while grooming: say "now-[if appearance of the brown llama is less than dapper]merely-[end if]".

A grooming tool is a kind of thing. Understand "use [a grooming tool] on [something]" as grooming it with (with nouns reversed). Grooming it with is an action applying to two things. Understand "groom [something] with [something]" as grooming it with.

Carry out grooming it with:
    if the appearance of the noun is less than dapper, now the appearance of the noun is the appearance after the appearance of the noun.

Report grooming it with:
    say "You attend diligently to the appearance and hygiene of [the noun]."

Instead of using a grooming tool in the presence of the brown llama:
    try grooming the brown llama with the noun.

The player carries some nail nippers, a slicker brush, and an apple. The apple is edible. The brush and the nippers are grooming tools. The player wears a sombrero.

The description of the nail nippers is "Ten inches long, to give you the necessary leverage to cut tough llama toenails. It still helps to soften them up by making the llama stand in a bucket of water first, though."

The description of the slicker brush is "Fine, angled soft bristles set into a broad back, perfect for removing mud from the coat of a long-woolled llama."

The industrial-strength blower is a fixed in place device in the Llama Pen. "Attached to the nearest wall, on its own movable boom, is an industrial-strength blower for doing llama hair."

Understand "use [switched off blower]" as switching on. Understand "use [switched on blower] on [brown llama]" as grooming it with (with nouns reversed). Instead of using the blower in the presence of the brown llama, try grooming the brown llama with the blower.

Test me with "use gate / use blower / use nippers / use brush / use apple / remove sombrero / use sombrero".

Whether we actually want a USE action is a subject of some theoretical debate in the IF community. On the one hand, it helps avoid guess-the-verb problems where the player cannot figure out what term to use in order to express a fairly simple idea. On the other, it encourages the player to think that all items have one and exactly one use, rather than getting him to consider the range of possibilities that arise from having a complex vocabulary.

---

290 **Example Anchorite** *RB*

By default, Inform understands GET OFF, GET UP, or GET OUT when the player is sitting or standing on an enterable object. We might also want to add GET DOWN and DOWN as exit commands, though.

With GET DOWN, we can replace the whole command, which will not interfere with the normal function of the TAKE verb, or allow the player to attempt to GET any other directions:

```
"Anchorite"

The Solitary Place is a room. "A glittering, shimmering desert without either
locusts or honey." The pillar is an enterable supporter in the Solitary Place. "The
broken pillar is short enough to climb and sit on." The description of the pillar is
"Once it was a monument: a long frieze of battles and lion-hunts spirals up the
side, in honor of an earthly king." The player is on the pillar.

Understand "get down" as exiting.
```

This doesn't cover the case where the player just types "DOWN", and we don't want to preempt the normal operation of the GO action here. So instead of writing a new understand instruction, we might catch this one at the action-processing level:

```
Instead of going down when the player is on a supporter:
    try exiting.

Test me with "down / enter pillar / get down / down / get down".
```

---

291 **Example Cloak of Darkness** *RB*

Implementation of "Cloak of Darkness", a simple example game that for years has been used to demonstrate the features of IF languages.

"Cloak of Darkness" is a brief example game that has been implemented in nearly every IF system currently used. It hasn't got much claim to complexity or richness, but it does exemplify many of the standard things one might want an IF language to be able to do: define descriptions and the results of actions, assign synonyms to nouns, create new verbs, handle darkness, track repeated acts, and so on.

Here is what the game looks like in Inform:

```
"Cloak of Darkness"

The story headline is "A basic IF demonstration."

Use scoring.

The maximum score is 2.
```

Whatever room we define first becomes the starting room of the game, in the absence of other instructions:

```
Foyer of the Opera House is a room. "You are standing in a spacious hall, splendidly decorated in red and gold, with glittering chandeliers overhead. The entrance from the street is to the north, and there are doorways south and west."

Instead of going north in the Foyer, say "You've only just arrived, and besides, the weather outside seems to be getting worse."
```

We can add more rooms by specifying their relation to the first room. Unless we say otherwise, the connection will automatically be bidirectional, so "The Cloakroom is west of the Foyer" will also mean "The Foyer is east of the Cloakroom":

```
The Cloakroom is west of the Foyer. "The walls of this small room were clearly once lined with hooks, though now only one remains. The exit is a door to the east."

In the Cloakroom is a supporter called the small brass hook. The hook is scenery. Understand "peg" as the hook.
```

Inform will automatically understand any words in the object definition ("small", "brass", and "hook", in this case), but we can add extra synonyms with this sort of Understand command.

```
The description of the hook is "It's just a small brass hook, [if something is on the hook]with [a list of things on the hook] hanging on it[otherwise]screwed to the wall[end if]."
```

This description is general enough that, if we were to add other hangable items to the game, they would automatically be described correctly as well.

```
The Bar is south of the Foyer. The printed name of the bar is "Foyer Bar". The Bar is dark. "The bar, much rougher than you'd have guessed after the opulence of the foyer to the north, is completely empty. There seems to be some sort of message scrawled in the sawdust on the floor."

The scrawled message is scenery in the Bar. Understand "floor" or "sawdust" as the message.

Neatness is a kind of value. The neatnesses are neat, scuffed, and trampled.
The message has a neatness. The message is neat.
```

We could if we wished use a number to indicate how many times the player has stepped on the message, but Inform also makes it easy to add descriptive properties of this sort, so that the code remains readable even when the reader does not know what "the number of the message" might mean.

```
Instead of examining the message:
    increment score;
    say "The message, neatly marked in the sawdust, reads...";
    end the story finally.
```

This second rule takes precedence over the first one whenever the message is trampled. Inform automatically applies whichever rule is most specific:

```
Instead of examining the trampled message:
    say "The message has been carelessly trampled, making it difficult to read.
You can just distinguish the words...";
    end the story saying "You have lost".
```

This command advances the state of the message from neat to scuffed and from scuffed to trampled. We can define any kinds of value we like and advance or decrease them in this way:

```
Instead of doing something other than going in the bar when in darkness:
    if the message is not trampled, now the neatness of the message is the
neatness after the neatness of the message;
    say "In the dark? You could easily disturb something."

Instead of going nowhere from the bar when in darkness:
    now the message is trampled;
    say "Blundering around in the dark isn't a good idea!"
```

This defines an object which is worn at the start of play. Because we have said the player is wearing the item, Inform infers that it is clothing and can be taken off and put on again at will.

```
The player wears a velvet cloak. The cloak can be hung or unhung. Understand
"dark" or "black" or "satin" as the cloak. The description of the cloak is "A
handsome cloak, of velvet trimmed with satin, and slightly splattered with
raindrops. Its blackness is so deep that it almost seems to suck light from the
room."

Carry out taking the cloak:
    now the bar is dark.

Carry out putting the unhung cloak on something in the cloakroom:
    now the cloak is hung;
    increment score.

Carry out putting the cloak on something in the cloakroom:
    now the bar is lit.

Carry out dropping the cloak in the cloakroom:
    now the bar is lit.
```

```
Instead of dropping or putting the cloak on when the player is not in the
cloakroom:
    say "This isn't the best place to leave a smart cloak lying around."

When play begins:
    say "[paragraph break]Hurrying through the rainswept November night, you're
glad to see the bright lights of the Opera House. It's surprising that there aren't
more people about but, hey, what do you expect in a cheap demo game...?"

Understand "hang [something preferably held] on [something]" as putting it on.

Test me with "s / n / w / inventory / hang cloak on hook / e / s / read message".
```

And that's all. As always, type TEST ME to watch the scenario play itself out.

---

292 **Example The Trouble with Printing** *RB*

Making a READ command, distinct from EXAMINE, for legible objects.

```
"The Trouble with Printing"

A thing has some text called printing. The printing of a thing is usually "blank".

Understand the command "read" as something new. Understand "read
[something]" as reading. Reading is an action applying to one thing, requiring
light. Check reading: if the printing of the noun is "blank", say "Nothing is written
on [the noun]." instead. Carry out reading: say "You read: [printing of the noun]
[line break]". Report reading: do nothing.

The Archive is a room.

Berkeley's report is a thing in the Archive. The description is "A report from
Governor Sir William Berkeley of Virginia, in 1671, in answer to the queries sent
by the Commissioners of Plantations the year previous. Of this report the better
part is burned and only a tail fragment remains." The printing of Berkeley's report
is "I thank God, [italic type]there are no free schools[roman type] nor [italic
type]printing[roman type], and I hope we shall not have these hundred years; for
[italic type]learning[roman type] has brought disobedience, and heresy, and
sects into the world, and [italic type]printing[roman type] has divulged them..."

Test me with "examine report / read report".
```

Since we defined reading as an action requiring light, we could further distinguish reading and examining (if we wanted) by writing some different visibility rules for it.

---

293 **Example Lanista 2** *RB*

Randomized combat in which the damage done depends on what weapons the characters are wielding, and in which an ATTACK IT

WITH action is created to replace regular attacking. Also folds a new DIAGNOSE command into the system.

Back in the chapter on randomization, we explored a way to create a randomized combat system. That system didn't allow for multiple weapons, though. Here we explore how to create an ATTACK IT WITH action that will let the player choose between weapons with different maximum powers.

We're also going to rewrite that original "instead of attacking:" rule into an attacking it with action that can be performed equally by the player or by any of the player's enemies.

```
"Lanista, Part Two"

The Arena is a room. "Sand, blood, iron. These festivals are normally held on hot days, but the sun has gone behind a cloud and fat drops of rain now and then spatter the arena floor." The gladiator is a man in the Arena. "A bare-chested Scythian gladiator faces you, wielding [a list of weapons carried by the gladiator]."

Section 1 - Hit Points

A person has a number called maximum hit points. A person has a number called current hit points.

The maximum hit points of the player is 35. The maximum hit points of the gladiator is 25.
```

In our simpler version of this example we set the current hit points by hand, but in a game with many characters this would get dull and repetitive, so here we'll use a "when play begins" to set all current hit point values automatically to maximum:

```
When play begins:
    repeat with victim running through people:
        now the current hit points of the victim is the maximum hit points of the victim.

Definition: a person is dead if his current hit points are less than 0.

Section 2 - Diagnosis

Diagnosing is an action applying to one visible thing. Understand "diagnose [something]" as diagnosing.

Check diagnosing:
    if the noun is not a person, say "Only people can have diagnoses." instead.

Carry out diagnosing:
    say "[if the noun is the player]You have[otherwise][The noun] has[end if] [current hit points of the noun] out of a possible [maximum hit points of the noun] hit points remaining."
```

```
Section 3 - Weapons

A weapon is a kind of thing. A weapon has a number called the maximum
damage. The maximum damage of a weapon is usually 4.

The gladiator carries a weapon called a trident. The maximum damage of the
trident is 5. The gladiator carries a weapon called a net. The maximum damage
of the net is 1.

The player carries a weapon called a mace. The maximum damage of the mace
is 3.

Section 4 - Attacking it with
```

In our new system, we want to specify what is being used for an attack. This means that we need to create a new "attacking it with" action, and also that we should disable the existing "attacking..." command.

Here's why: If we leave the default attack command in place, Inform will continue to accept commands like >ATTACK GLADIATOR, but reply foolishly with the default "Violence is not the answer..." response.

A somewhat better approach would be to change the reply of >ATTACK GLADIATOR to say something like "You must specify a weapon to attack with." But this is still less than ideal, because it means that the player has to then rewrite his entire command. If, on the other hand, we take out "ATTACK GLADIATOR" entirely, the game will always prompt "What do you want to attack the gladiator with?" -- which teaches the player the correct command structure for this particular game, and avoids pretending to understand any command that is not meaningful within this game.

This is a little bit of work because ATTACK has a lot of synonyms in the default library, but if we look through the actions index we can find them all:

```
Understand the commands "attack" and "punch" and "destroy" and "kill" and
"murder" and "hit" and "thump" and "break" and "smash" and "torture" and
"wreck" as something new.
```

Now we make our new command:

```
Attacking it with is an action applying to one visible thing and one carried thing.
Understand "attack [someone] with [something preferably held]" as attacking it
with.
```

Note that we've specified "one carried thing", because we want the player to pick up a weapon to use if necessary. And now we assign all the old attack vocabulary to apply to the new command:

```
Understand the commands "punch" and "destroy" and "kill" and "murder" and
"hit" and "thump" and "break" and "smash" and "torture" and "wreck" as "attack".
```

This may seem counter-intuitive, but order of source code matters here: we first get rid of the old, default vocabulary, then define our new action, then make the vocabulary apply to that new action. Inform will now understand >HIT GLADIATOR WITH TRIDENT, >BREAK GLADIATOR WITH TRIDENT, and so on.

Our new action is also a perfect place to use an action variable: we're going to need to choose an amount of damage done and refer to that several times in our action rules. So let's set that up first:

```
The attacking it with action has a number called the damage inflicted.

Setting action variables for attacking something with something:
    if the second noun is a weapon:
        let the maximum attack be the maximum damage of the second noun;
        now the damage inflicted is a random number between 1 and the maximum
attack.

Check an actor attacking something with something (this is the can't attack with
something that isn't a weapon rule):
    if the second noun is not a weapon:
        if the actor is the player, say "[The second noun] does not qualify as a
weapon.";
        stop the action.

Check an actor attacking something with something (this is the can't attack a
non-person rule):
    if the noun is not a person:
        if the actor is the player, say "[The noun] has no life to lose.";
        stop the action.

Carry out an actor attacking something with something (this is the standard
attacking it with a weapon rule):
    decrease the current hit points of the noun by the damage inflicted;
    if the noun is dead:
        now the noun is nowhere.
```

Though our checks and carry-out rules are similar regardless of who is acting, we're going to want actions to be described differently for different actors, so we'll use separate "report attacking" and "report someone attacking" rules. We'll also make some special cases for when the character has died as a result of the attack:

```
Report attacking a dead person with something (this is the death-report priority
rule):
    say "You attack with [the second noun], killing [the noun]!" instead.

Report attacking someone with something (this is the normal attacking report
rule):
    say "You attack [the noun] with [the second noun], causing [damage inflicted]
point[s] of damage!" instead.

Report someone attacking the player with something when the player is dead
(this is the player's-death priority rule):
    say "[The actor] attacks you with [the second noun], finishing you off!";
```

```
    end the story;
    stop the action

Report someone attacking the player with something (this is the standard report
someone attacking the player with rule):
    say "[The actor] attacks you with [the second noun], causing [damage
inflicted] point[s] of damage!" instead.

Report someone attacking something with something (this is the standard report
attacking it with rule):
    say "[The actor] attacks [the noun] with [the second noun], causing [damage
inflicted] point[s] of damage!" instead.

When play begins:
    now the left hand status line is "You: [current hit points of player]";
    now the right hand status line is "Gladiator: [current hit points of gladiator]".

Every turn (this is the gladiator-attack rule):
    if the gladiator is not dead, try the gladiator attacking the player with a random
weapon which is carried by the gladiator.

Test me with "hit gladiator with mace / kill gladiator / drop mace / attack gladiator
/ attack gladiator with mace / g / g".
```

Those devoted to role-playing will note that our form of randomization is still pretty naive: most RPG systems use multiple dice in order to create more interesting probability curves. For a system that simulates actual dice-rolling, see the full "Reliques of Tolti-Aph" game.

294

Example **Shawn's Bad Day**

RB

Allowing the player to EXAMINE ALL.

We can add the handling of multiple objects to an existing action simply by adding in a line of grammar using "[things]". In response, Inform will consider every object accepted by the token, and perform the action once for each of those objects. Thus:

```
"Shawn's Bad Day"

The Treasury is a room. The vault is a lockable locked closed openable
container in the Treasury. It is fixed in place. "A massive vault fills up one wall."
The description is "The vault's system includes [a list of things which are part of
the vault]."

A little green light, a little blue light, a little red light, a thin black pane of glass, a
laser beam, a retinal scanner, a thumbprint ID plate, a dial, and a large lever are
part of the vault.

The security guard is a man in the Treasury. The description is "His name is
Shawn, and he doesn't look happy."
```

```
The description of the green light is "Off." The description of the blue light is
"Tranquilly on." The description of the red light is "Angrily flashing."

Understand "examine [things]" as examining.

Test me with "examine all".
```

295

Example **The Left Hand of Autumn**

RB

The possibility of using a [things] token opens up some interesting complications, because we may want actions on multiple items to be reported differently from actions on just one. Here we look at how to make a multiple examination command that describes groups in special ways.

Suppose that we have a game in which groups of objects can have meaning apart from their individual significance -- perhaps there are spells that can only be cast by collecting just the right items in the same place.

In this case, one of the things the player might like to be able to do is look at several items together and get a special response, different from looking at the items individually.

To make this happen, we need to do several things:

```
(1) we need to create a version of the EXAMINE command that can apply to
multiple objects at once.
(2) we need to correct the way Inform normally deals with multiple-object
commands, because we want our group description to print only one time, and
we want to avoid stubs such as "pear: ... apple: ..." before or after the group
description.
(3) we need to define a way for Inform to identify interesting groups and describe
them.

"The Left Hand of Autumn"

Section 1 - Procedure

Understand "examine [things]" or "look at [things]" as multiply-examining.
Multiply-examining is an action applying to one thing.

Understand "examine [things inside] in/on [something]" or "look at [things inside]
in/on [something]" as multiply-examining it from. Multiply-examining it from is an
action applying to two things.

Group-description-complete is a truth state that varies.

Carry out multiply-examining it from:
    try multiply-examining the noun instead.
```

```
Check multiply-examining when group-description-complete is true:
    stop the action.

Carry out multiply-examining:
    let L be the list of matched things;
    if the number of entries in L is 0, try examining the noun instead;
    if the number of entries in L is 1, try examining entry 1 of L instead;
    describe L;
    say line break;
    now group-description-complete is true.

Before reading a command:
    now group-description-complete is false.
```

Now for step 2, overriding Inform's usual output of names of objects:

```
The silently announce items from multiple object lists rule is listed instead of the
announce items from multiple object lists rule in the action-processing rules.

This is the silently announce items from multiple object lists rule:
    unless multiply-examining or multiply-examining something from something:
        if the current item from the multiple object list is not nothing, say "[current
item from the multiple object list]: [run paragraph on]".

Definition: a thing is matched if it is listed in the multiple object list.
```

We'll save our "to describe" phrase until Section 2, when we can give the game specific instructions about how to report different lists of objects.

Now, the player might also want to be able to refer to a group of item by some kind of group name, so let's add the option of creating a Table of Collective Names which will interpret these:

```
After reading a command:
    repeat through the Table of Collective Names:
        let N be "[the player's command]";
        let Y be relevant list entry;
        while N matches the regular expression "[name-text entry]":
            replace the regular expression "(.*)[name-text entry](.*)" in N with
"\1[Y]\2";
        change the text of the player's command to N.

Report taking something:
    say "You pick up [the noun]." instead.
```

And as a bit of polish, because we'd like SEARCH TABLE to have the same effect as EXAMINE ALL ON TABLE:

```
Understand "look on [something]" as searching.

Instead of searching something which supports at least two things:
    let L be the list of things supported by the noun;
    describe L.
Instead of searching something which contains at least two things:
```

```
    let L be the list of things contained by the noun;
    describe L.

Section 2 - Scenario

Eight-Walled Chamber is a room. "A perfectly octagonal room whose walls are
tinted in various hues."

The display table is a supporter in the Chamber. A twig of rowan wood is on the
table.

The player carries an apple and a pear.

A glove is a kind of thing. A glove is always wearable. Understand "glove" as a
glove. The player carries a left glove and a right glove. The left glove and the
right glove are gloves.
```

Now we define a few actual lists of items:

```
Fruit list is a list of objects which varies. Fruit list is { apple, pear }.
Glove list is a list of objects which varies. Glove list is { right glove, left glove }.
Arcane list is a list of objects which varies. Arcane list is { left glove, twig, pear }.

To describe (L - a list of objects):
    sort L;
    if L is fruit list:
        say "Just a couple of fruits.";
    otherwise if L is glove list:
        say "It's a matched pair of fuzzy blue gloves.";
    otherwise if L is arcane list:
        say "To anyone else it might look like a random collection of objects, but
these three things -- [L with definite articles] -- constitute a mystic key known as
the Left Hand of Autumn. They practically hum with power.";
    otherwise:
        say "You see [L with indefinite articles]."

When play begins:
    sort fruit list;
    sort glove list;
    sort the arcane list.
```

We sort the lists so that regardless of how we change the rest of the code (and the order in which objects are coded), the resulting list will always be in sorted order and ready to compare with the list of items the player wants to look at. And thanks to the "Reading a command" code we wrote earlier, we can also teach the game to understand the player's references to "the left hand of autumn" as a specific collection of items.

```
Table of Collective Names

name-text               relevant list
"left hand of autumn"   "[arcane list]"
"gloves"                "[glove list]"
"pair of gloves"        "[glove list]"
```

```
Test me with "x apple and pear / x left and right / put pear on table / put left glove
on table / x all on table / put all on table / examine all on table / get apple, twig,
pear / x all on table / search table".
```

296

Example **Ish.**

RB

A (very) simple HELP command, using tokens to accept and interpret the player's text whatever it might be.

```
"Ish."

Ichiro's Dubious Sushi Hut is a room. "Despite the allure of the dusty plastic
sushi models in the window, you're beginning to have second thoughts about the
selection of this particular restaurant for your rendezvous with Agent Fowler.
There are no other patrons, for one thing. Afternoon sunlight filters lazily through
the window and illuminates a number of empty glass-topped tables, at each of
which is a chopstick dispenser (in form of cute ceramic cat) and a pitcher of soy
sauce (sticky).

The sushi bar itself is what gives the most pause, however. Behind it sits an
angry-looking Japanese woman, aggressively eating a Quarter Pounder with
Cheese."
```

We can, when necessary, accept any text at all as a token:

```
Understand "help [text]" or "help about [text]" as getting help about. Understand
the commands "instructions" or "hint" or "hints" or "menu" or "info" or "about" as
"help".

Getting help about is an action applying to one topic.
```

After that, we can use "the topic understood" to refer to the text we read:

```
Carry out getting help about:
    if the topic understood is a topic listed in the Table of Standard Help:
        say "[explanation entry][paragraph break]";
    otherwise:
        say "You're out of ideas."

Table of Standard Help
```

topic	title	summary	explanation
"sushi"	"sushi"	"Really it's just vinegary rice"	"Popular misconception says that sushi inevitably entails raw fish, but it is in fact just rice with rice vinegar on it. It's just that the really good kinds have raw fish in."
"cucumber roll" or "cucumber"	"Cucumber roll"	"Sushi for people who are afraid of sushi"	"It is just rice and slivers of cucumber in the middle, and as long as you don't go too crazy with the wasabi, all should be well."
"california roll" or "california"	"California roll"	"Travesty of the sushi concept"	"It's. Fake. Crab."
"monkfish liver"	"monkfish liver"	"Expert eaters only"	"The odds of Ichiro's having this unusual delicacy is near zero."
"microdot"	"microdot"	"What you came here to deliver"	"There'll be time enough for that later. If Fowler ever turns up. Where is she, anyway?"

Since the player may not know what all the help options are, we might as well let him get an overview, as well.

```
Understand "help" as summoning help. Summoning help is an action applying to nothing.

Carry out summoning help:
    say "Help is available about the following topics. Typing HELP followed by the name of a topic will give further information.[paragraph break]";
    repeat through the Table of Standard Help:
        say " [title entry]: [summary entry][line break]".

Test me with "help / help about microdot / help cucumber / help california roll".
```

---

297  **Example Nameless** *RB*

ASKing someone about an object rather than about a topic.

By default, ASK SOMEONE ABOUT... applies only to a text token. We might want also to offer the player the option of asking characters about pieces of physical evidence. This example implements an ASK PERSON ABOUT THING command that is mostly synonymous with SHOW, with the added nuance that the player can ask about things that are not currently visible, as long as he has encountered them at some time in the past.

```
"Nameless"

The Black Chamber is a room. "Despite its menacing name, it is quite an ordinary room, underlying the post office above. Here letters are brought each day, unsealed, transcribed, resealed, and sent again on their way; their contents then analyzed and recorded."

The Nameless Advisor is a woman in the Black Chamber. "A woman whose name has never been disclosed to you sits at the window, writing numbers on a sheet of paper." The Advisor carries a sheet of paper. Understand "woman" as the nameless advisor.

The player carries a letter from the emperor. The description of the letter is "Though its origin is obvious, its meaning is secret: the letters are an inexplicable jumble."
```

Now we create our new action, "interrogating it about". We write the grammar lines so that we can show any object in sight to someone, but also ask someone about any object that we have ever interacted with in the game, whether it is currently visible or not.

```
A thing can be known or unknown. The Nameless Advisor is known.

Understand "ask [someone] about [any known thing]" as interrogating it about. interrogating it about is an action applying to two visible things.
```

Now we replace and redirect the showing action. This gets rid of the requirement in the default library that the player be holding anything he shows to another character:

```
Understand the commands "show" and "display" and "present" as something new.

Understand "show [something] to [someone]" or "display [something] to [someone]" or "present [something] to [someone]" as interrogating it about (with nouns reversed). Understand "show [someone] [something]" as interrogating it about.
```

This bit keeps track of what the player has seen, for the purposes of "any known thing":

```
Before printing the name of something (called the target): now the target is known.
```

Here we define what happens by default when we interrogate someone about something; we use the same response we get to asking someone about something that isn't otherwise interesting:

```
Carry out interrogating someone about something:
    say "There is no reply."
```

Now redirect all asking to a topic table, and all interrogating to an object table:

```
Instead of asking Nameless Advisor about a topic listed in the Table of Nameless Advisor Topics:
    say "[reply entry][paragraph break]".

Instead of interrogating Nameless Advisor about an item listed in the Table of Nameless Advisor Items:
    say "[reply entry][paragraph break]".

Table of Nameless Advisor Items
item    reply
letter  "'It is enciphered,' she remarks[if the advisor can see the letter], glancing over the contents[otherwise], after you have offered a detailed description[end if]. 'A substitution cipher of some complexity, I believe.'"
Advisor "She listens to your inquiries about her identity and parentage with a placid smile, but does not answer."

Table of Nameless Advisor Topics
topic           reply
"cipher"        "'I know many dozens of ciphers,' she replies, smiling in a disquieting way."
"substitution cipher"   "'One letter is allowed to stand for another,' she explains, folding her hands together patiently. The backs of both hands are tattooed with silvery stars."
```

And just so that we can test what happens when asking someone about something out of sight:

```
The safe box is a container in the Chamber. It is fixed in place. It is openable and closed.
```

...and something unknown:

```
The poisonous apple is a thing.

Test me with "test sight / test knowledge".

Test sight with "i / x letter / ask Nameless Advisor about cipher / show cipher to
Nameless Advisor / ask Nameless Advisor about the letter / show the letter to
Nameless Advisor / show Nameless Advisor the letter".

Test knowledge with "open safe box / put letter in safe box / close safe box / ask
Nameless Advisor about the letter / show the letter to Nameless Advisor / ask
Nameless Advisor about the apple".
```

298

Example **Safety**

RB

A safe whose dial can be turned with SPIN SAFE TO 1131, and which will open only with the correct combination.

```
"Safety"

The Vault is a room. "Snug yet paranoid, this represents the state of the art in
cheerless security." The Safe is here. "A mammoth safe, with a dial which can
spin to any number, has pride of place. It must weigh about the same as a small
car, so don't get any ideas." Instead of opening the safe, say "The safe opens
only when turned to the correct combination."

In the Safe is a silver florin. The Safe is closed and fixed in place. Understand
"dial" as the Safe.

Spinning it to is an action applying to one thing and one number. Check spinning
it to: if the noun is not the Safe, say "[The noun] does not spin." instead. Report
spinning it to: say "Click! and nothing else happens."

Understand "spin [something] to [a number]" as spinning it to.

After spinning the closed Safe to 1384: now the Safe is open; say "Clonk! and
the safe door swings slowly open, revealing [a list of things in the Safe]."

Test me with "open safe / spin safe to 1131 / open safe / spin safe to 1384 / x
safe / get florin".
```

299

Example **Tom's Midnight Garden**

RB

A clock kind that can be set to any time using "the time understood"; may be turned on and off; and will advance itself only when running. Time on the face is also reported differently depending on whether the clock is analog or digital.

Time can also be understood as a token, and the time parsed will be recorded as "the time understood". So therefore, if we wish for clocks which may be set:

```
"Tom's Midnight Garden"

A clock is a kind of device. A clock has a time called the current time. A clock can be analog or digital. The current time of a clock is usually 9:01 AM. The description of a clock is "It shows the time to be [if analog]about [the current time to the nearest five minutes in words][otherwise][the current time][end if]."

Understand "set [clock] to [time]" as setting it by time. Setting it by time is an action applying to one thing and one time.

Instead of setting a clock to something:
    say "[The noun] can be set only to a time of day, such as 8:00 AM, or midnight."

Carry out setting a clock by time:
    now the current time of the noun is the time understood.

Report setting a clock by time:
    say "You set [the noun] to [time understood]."

Every turn:
    repeat with item running through switched on clocks:
        now the current time of the item is one minute after the current time of the item.

The Hall is a room. The grandfather clock is a fixed in place analog clock in the Hall. The travel clock is a switched on digital clock in the Hall. When play begins: now the right hand status line is "[time of day]".

Test me with "examine grandfather clock / set it to midnight / switch it on / wait / wait / wait / examine it / set travel clock to 4:12 / examine it".
```

---

300 **Example** **Ibid.** *RB*

A system which allows the author to assign footnotes to descriptions, and permits the player to retrieve them again by number, using "the number understood". Footnotes will automatically number themselves, according to the order in which the player discovers them.

"Hitchhiker's Guide to the Galaxy" introduced the idea of footnoted descriptions, and various IF games since have toyed with the idea. The recommended implementation in Inform 6 involved keeping an assortment of footnote objects around, but in Inform 7 the table is a much tidier way of handling the same problem.

```
"Ibid."

The Ship Inn is a room. "Here you are in a lovely pub which your guidebook assures you is extremely authentic. [1 as a footnote].
```

```
To your left sits a party of Italians, with their guidebook.

To your right is a silent, but not unappealing, young man.".

A party of Italians and a silent young man are people in the Ship Inn. The
Italians and the young man are scenery.

The table is a supporter in the Ship Inn. On the table is a mysterious pie. The
description of the pie is "Your waitress told you it was the specialty of the day,
Steak and Owl Pie. [2 as a footnote]." The pie is edible.

Table of Footnotes

    assignment note
    a number    "Francis Drake ate here, if the sign on the door is to be believed"
    --          "this is unlikely, considering that owls are protected animals in England these days [3 as a footnote]"
    --          "moreover, you can't imagine that owl would be very tasty"

Footnotes mentioned is a number that varies.
```

Whenever we mention a footnote for the first time, we need to assign it a number, which we will use consistently thereafter. And it's probably a good idea to protect ourselves against the author accidentally using a number too large for the footnote table, too. So:

```
To say (footnote - a number) as a footnote:
    if footnote > number of filled rows in the Table of Footnotes:
        say "Programming error: footnote assignment out of range.";
    otherwise:
        choose row footnote in the Table of Footnotes;
        if there is an assignment entry:
            say "([assignment entry])";
        otherwise:
            increment footnotes mentioned;
            choose row footnote in the Table of Footnotes;
            now assignment entry is footnotes mentioned;
            say "([assignment entry])".
```

Now, in order to let the player view these footnotes, we'll need to parse numbers.

```
Understand "footnote [number]" as looking up a footnote.

Looking up a footnote is an action applying to one number.

Check looking up a footnote:
    if the number understood > footnotes mentioned, say "You haven't seen any
such footnote." instead;
    if the number understood < 1, say "Footnotes are numbered from 1."

Carry out looking up a footnote:
    choose row with assignment of number understood in the Table of Footnotes;
    say "([assignment entry]): [note entry]."

Test me with "footnote 1 / examine pie / footnote 2 / footnote 3".
```

This method does require us to keep track of where a footnote appears in the table. If we found this inconvenient, we could add a column to the footnote table so that we could invoke it with tags like "[appearance quip as a footnote]".

301

Example **One of Those Mornings**

RB

A FIND command that allows the player to find a lost object anywhere

Suppose that, contrary to the usual rules of interactive fiction, we want to allow the player to discover the locations of things he hasn't actually seen yet:

```
"One of Those Mornings"

Understand "find [any thing]" as finding.

Finding is an action applying to one visible thing.

Carry out finding:
    if the player is carrying the noun:
        say "You're holding [the noun]!";
    otherwise:
        say "You left [the noun] [if the noun is on a supporter]on[otherwise]in[end if]
[the holder of the noun]."
```

The holder of the noun can be a room, a supporter, or a container: the phrase is not picky. We would want to be a little more careful if it were ever possible for an item to have been "removed from play" in our game, since then the holder could be nothing, and that would have odd results. In this particular example, though, that will not arise.

And that's it, as far as the find command goes. The rest is local color.

```
The Exhibition Room is a room. It contains a closed locked lockable transparent
openable container called the display case. The display case contains a
priceless pearl. The display case is scenery. The description of the Exhibition
Room is "By far the finest thing in the room is a priceless pearl in a glass display
case. It should of course be yours[if key is not visible], if only you can remember
where you hid the key[end if]."

The silver key unlocks the display case.

A jade vase, a teak chest, a bronze teakettle, and a child's burial casket are
openable closed containers in the Exhibition Room.

After taking the pearl:
    say "The pearl rolls into your hand, gleaming in the oblique light; your fortune
is made.";
    end the story finally.
```

If we want to have the key found in different places when the game is replayed:

```
When play begins:
    let the space be a random container which is not the display case;
    move the silver key to the space.

Every turn:
    say "Your watch ticks with maddening loudness."

The time of day is 1:02 AM.

At 1:08 AM: say "The security guard arrives to find you fumbling about with keys.
Curses."; end the story.

Test me with "find pearl / find teakettle / get teakettle / find teakettle / find key".
```

302

Example Actaeon

RB

A FOLLOW command allowing the player to pursue a person who has just left the room.

Suppose we want the player to be able to go after characters who are moving around the map. The trick, of course, is that once characters are gone they are no longer visible to "follow [person]", so we need "follow [any person]" to find them.

```
"Actaeon"

A person has a room called last location.

Understand "follow [any person]" as following. Understand the commands
"chase" and "pursue" as "follow".

Following is an action applying to one visible thing.

Check following:
    if the noun is the player, say "Wherever you go, there you are." instead;
    if the noun is visible, say "[The noun] is right here." instead;
    if the last location of the noun is not the location, say "It's not clear where [the
noun] has gone." instead.
```

Here again the best route comes in handy:

```
Carry out following:
    let the destination be the location of the noun;
    if the destination is not a room, say "[The noun] isn't anywhere you can
follow." instead;
    let aim be the best route from the location to the destination;
    say "(heading [aim])[line break]";
    try going aim.

Corinth is a room. Athens is east of Corinth. Epidaurus is southeast of Corinth
and east of Mycenae. Mycenae is south of Corinth. Olympia is west of Mycenae.
Argos is south of Mycenae. Thebes is northwest of Athens. Pylos is south of
```

```
Olympia. Sparta is east of Pylos and south of Argos. Delphi is northwest of
Thebes.

Artemis is a woman in Corinth.
```

We do also have to make sure that whenever we move a person from room to room, we record where they were moved from; otherwise, our clever restrictions about whom the player can pursue will not work properly.

```
To move (pawn - a person) tidily to (target - a room):
    now the last location of the pawn is the holder of the pawn;
    move the pawn to the target.

Every turn:
    let current location be the location of Artemis;
    let next location be a random room which is adjacent to the current location;
    if Artemis is visible, say "Artemis heads to [the next location].";
    move Artemis tidily to next location;
    if Artemis is visible, say "Artemis arrives from [the current location]."

Test me with "wait / follow artemis / follow artemis / follow artemis".
```

303

Example **Pages**

RB

A book with pages that can be read by number (as in "read page 3 in...") and which accepts relative page references as well (such as "read the last page of...", "read the next page", and so on).

Suppose we have a book that the player must consult page-by-page, and we want to be able to accept all of the following input:

```
> READ BOOK (to choose a random page and read it)
> READ PAGE 1 IN BOOK
> READ PAGE 2
> READ THE LAST PAGE OF THE BOOK
> READ THE NEXT PAGE
> READ PREVIOUS PAGE IN BOOK
> READ THE FIRST PAGE
```

One approach would be to write many different understand rules and actions: one action for reading randomly, one for reading a specific page, one for reading the first page, one for reading the previous page, one for reading the next page, and one for reading the last page. But this gets tedious to construct and maintain.

More usefully, we could consider that all of the last four options are essentially the same action at heart: the player is asking to read a page in the book using a name rather than a number, and we will have to perform a minor calculation to discover what the number should be. Here's an implementation using named values to resolve this problem:

```
"Pages"
```

```
The Library is a room. The sinister book is carried by the player. The sinister book has a number called the last page read. The sinister book has a number called the length. The length of the sinister book is 50.

Understand the command "read" as something new.

Understand "read [something]" or "consult [something]" or "read in/from [something]" as reading. Reading is an action applying to one thing, requiring light.

Understand "read [number] in/from/of [something]" or "read page [number] in/from/of [something]" or "look up page [number] in/from/of [something]" or "consult page [number] in/from/of [something]" as reading it in. Reading it in is an action applying to one number and one thing, requiring light.

Named page is a kind of value. The named pages are first page, last page, next page, previous page.

To decide what number is the effective value of (L - last page):
    decide on the length of the book.

To decide what number is the effective value of (F - first page):
    decide on 1.

To decide what number is the effective value of (N - next page):
    let X be the last page read of the book plus 1;
    decide on X.

To decide what number is the effective value of (P - previous page):
    let X be the last page read of the book minus 1;
    decide on X.

Understand "read [named page] in/from/of [something]" or "read the [named page] in/from/of [something]" as reading it relatively in. Reading it relatively in is an action applying to one named page and one thing, requiring light.

Does the player mean reading something in the sinister book: it is very likely.

This is the book requirement rule:
    if the player is not carrying the sinister book, say "You're not reading anything." instead.

Check reading it relatively in:
    if the second noun is not the sinister book, say "There are no pages in [the second noun]." instead;
    abide by the book requirement rule.

Carry out reading it relatively in:
    let N be the effective value of the named page understood;
    now the number understood is N;
    try reading N in the book.

Check reading it in:
    if the second noun is not the sinister book, say "There are no pages in [the
```

```
second noun]." instead;
    abide by the book requirement rule.

Check reading it in:
    if the number understood is greater than the length of the sinister book, say
"There are only [length of sinister book in words] pages in the book." instead;
    if the number understood is less than 1, say "The page numbering begins with
1." instead.

Carry out reading it in:
    read page number understood.

Check reading:
    if the noun is not the sinister book, say "There are no pages in [the noun]."
instead;
    abide by the book requirement rule.

Carry out reading:
    let N be a random number between 1 and the length of the sinister book; now
the number understood is N;
    say "You flip the pages randomly and arrive at page [the number understood]:
[paragraph break]";
    try reading the number understood in the sinister book.

Table of Book Contents

    page content
    2     "dhuma jyotih salila marutam / samnipatah kva meghah / samdes arthah kva patukaranaih / pranibhih
          prapaniyah"
    13    "amathesteron pws eipe kai saphesteron"
    50    "Rrgshilz maplot..."

To read page (N - a number):
    now the last page read of the sinister book is N;
    if there is a content corresponding to a page of N in the Table of Book
Contents:
        choose row with a page of N in the Table of Book Contents;
        say "You read: '[content entry]'[paragraph break]";
    otherwise:
        say "Page [N] appears to be blank."

To read page (N - 47):
    say "Your eyes burn; your ears ring. Beneath your gaze, the dreadful sigils
writhe, reminding you of that which lies outside the edges of the universe...";
    end the story saying "You have lost your remaining sanity".

Test me with "read from the sinister book / read the book / read the next page /
read page 2 / read previous page / g / read the first page / read the last page of
the book / read the next page / read 47 in book".
```

304

Example **Down in Oodville**

RB

Offering the player a choice of numbered options at certain times, without otherwise interfering with his ability to give regular commands.

Now and then in IF there is a situation where we need to ask the player for a numbered choice rather than an ordinary action command. What's more, that numbered choice might change during the game, so we don't want to just hard-wire the meanings of "1", "2", and "3" whenever the player types them.

A better trick is to keep a list or table (we'll use a table here because it involves slightly less overhead) recording what the player's numerical choices currently mean. Then every time the player selects a number, the table is consulted, and if the number corresponds to something, the player's choice is acted on.

In our example, we'll have a transporter pad that can take the player to any room in the game that he's already visited. (Just for the sake of example, we'll start him off with a few pre-visited rooms.)

```
"Down in Oodville"

Section 1 - Method

Understand "[number]" as selecting.

Selecting is an action applying to one number.

Check selecting: [assuming we don't want to be able to transport from just
anywhere]
    if the player is not on the transporter pad:
        say "You can transport only from the transporter pad. From other places
than the transporter room, you can HOME to your base ship, but not leap
sideways to other locations.";
        empty the transport options instead.

Check selecting:
    if the number understood is greater than the number of filled rows in the Table
of Transport Options or the number understood is less than one:
        say "[The number understood] is not a valid option. ";
        list the transport options instead.

Carry out selecting:
    let N be the number understood; [not actually a necessary step, but it makes
the next line easier to understand]
    choose row N in the Table of Transport Options;
    if the transport entry is a room:
        move the player to the transport entry;
    otherwise:
        say "*** BUG: Improperly filled table of transport options ***" [It should not
be possible for this to occur, but we add an error message for it so that, if it ever
does, we will know what is causing the programming error in our code]

To list the transport options:
    let N be 1;
    say "From here you could choose to go to: [line break]";
    repeat through the Table of Transport Options:
        say " [N]: [transport entry][line break]";
        increment N.
```

```
To empty the transport options:
    repeat through the Table of Transport Options:
        blank out the whole row; [first we empty the table]

To load the transport options:
    repeat with interesting room running through visited rooms which are not the
Transporter Room:
        choose a blank row in the Table of Transport Options;
        now the transport entry is the interesting room.

Table of Transport Options
transport
an object
with 3 blank rows. [In the current scenario, the number of blank rows need never
be greater than the number of rooms in the game, minus the transport room
itself.]

Understand "home" as homing. Homing is an action applying to nothing.

Check homing:
    if the player is in the Transporter Room:
        say "You're already here!" instead.

Carry out homing:
    move the player to the transporter room.

Section 2 - Scenario

The Transporter Room is a room.

Oodville is a visited room.

Midnight is a visited room. The Diamond City is west of Midnight.

The transporter pad is an enterable supporter in the Transporter Room. "The
transporter pad in the middle of the floor is currently dull blue: powered but
unoccupied."

After entering the transporter pad:
    say "The transporter beeps and glows amber as you step onto its surface. A
moment later a hologram displays your options. [run paragraph on]";
    empty the transport options;
    load the transport options;
    list the transport options.

Test me with "get on pad / 0 / -1 / 8 / 2 / look / w / home / get on pad / get off pad
/ 3".
```

If we wanted to replace the regular command structure entirely with numbered menus, or use menus to hold conversation options, we could: several Inform extensions provide these functions.

305  **Example** **Straw Into Gold** *RB*

Creating a Rumpelstiltskin character who is always referred to as "dwarf", "guy", "dude", or "man" -- depending on which the player last used -- until the first time the player refers to him as "Rumpelstiltskin".

```
"Straw Into Gold"

The Cell is a room. Rumpelstiltskin is an improper-named man in the Cell.
Rumpelstiltskin can be identified or unidentified. Rumpelstiltskin is unidentified.

R-name is a kind of value. The R-names are dwarf, guy, dude, and man-thing.
Rumpelstiltskin has an R-name. Understand "[R-name]" as Rumpelstiltskin.
```

Our example is slightly complicated by the fact that "man" is a name already known to Inform, so we can't re-use it as a kind of value. This is possible to work around, though:

```
Understand "man" as man-thing.
```

Now we borrow from the Activities chapter to look at the exact wording of the player's command:

```
After reading a command:
    if the player's command includes "[R-name]",
        now the R-name of Rumpelstiltskin is the R-name understood;
    if the player's command includes "Rumpelstiltskin":
        now Rumpelstiltskin is identified;
        now Rumpelstiltskin is proper-named.

Rule for printing the name of Rumpelstiltskin when Rumpelstiltskin is
unidentified:
    if the R-name of Rumpelstiltskin is man-thing:
        say "man";
    otherwise:
        say "[R-name]".

Test me with "x dwarf / x guy / x dude / look / x rumpelstiltskin / look / x man".
```

---

306  **Example** **Misadventure** *RB*

A going by name command which does respect movement rules, and accepts names of rooms as commands.

The original Adventure allowed the player to type the names of rooms in order to move to them, and it is now not too difficult for us to do the same. Adventure restricted this option to adjacent rooms, but we might want to be a bit more flexible, so we will accept any room:

```
"Misadventure"
```

```
Plover Room is a room. "You're in a small chamber lit by an eerie green light. An
extremely narrow tunnel exits to the west. A dark corridor leads northeast."

The Dark Corridor is northeast of Plover Room. Plover Room is south of the
Dark Corridor. The printed name of the Dark Corridor is "Dark Room". The
description of the Dark Corridor is "You're in the dark-room. A corridor leading
south is the only exit."

The Alcove is west of Plover Room. "You are in an alcove. A small northwest
path seems to widen after a short distance. An extremely tight tunnel leads east.
It looks like a very tight squeeze. An eerie light can be seen at the other end."

Northwest of the Alcove is the Misty Cavern. The description of Misty Cavern is
"You are following a wide path around the outer edge of a large cavern. Far
below, through a heavy white mist, strange splashing noises can be heard. The
mist rises up through a fissure in the ceiling. The path exits to the south and
west." West of Misty Cavern is the Alcove.

Understand "[any room]" as going by name. Understand "go to [any room]" as
going by name.

Going by name is an action applying to one thing.
```

We should reject movement to the player's current location, or to anywhere he hasn't been and can't see:

```
Check going by name:
    if the noun is the location, say "You're already in [the location]." instead;
    if the noun is not adjacent and the noun is unvisited, say "That noun did not
make sense in this context." instead.
```

The assumption here is that the player does know the names of the rooms adjacent to his current location, even if he hasn't been there yet.

Now for the travel itself. The simplest way to ensure that the usual movement rules will still apply is to convert GO BY NAME into a GO action, and here the best route comes to our aid:

```
Carry out going by name:
    let aim be the best route from the location to the noun, using doors;
    if aim is not a direction, say "You can't think how to get there from here."
instead;
    say "(heading [aim])[command clarification break]";
    try going aim;
    if the location is not the noun, say "You'll have to stop here."
```

This will allow the player to travel toward rooms he has already visited even if they are several moves away.

Finally, so that the player can also use the names of doors as commands:

```
Understand "[door]" as entering.
```

And in keeping with the original, we might add to our scenario a rule or two about restrictions on movement, just to test that it's all working right:

```
The player carries a plover egg and a platinum pyramid. The description of the
egg is "Plover's eggs, by the way, are quite large." The printed name of the egg
is "emerald the size of a plover's egg". Understand "emerald" as the egg. The
description of the pyramid is "The platinum pyramid is 8 inches on a side!"

Instead of going to the Plover Room from the Alcove when the player carries
something which is not the plover egg:
    say "Something you're carrying won't fit through the tunnel with you. You'd
best take inventory and drop something."

Test me with "go to misty cavern / go to dark corridor / go to plover room / go to
alcove / go to dark corridor / drop pyramid / go to dark corridor / g / go to alcove /
g / go to misty cavern".
```

---

307  **Example Safari Guide** RB

The same functionality, but making the player continue to move until he reaches his destination or a barrier, handling all openable doors on the way.

The foregoing example moves the player one location towards his destination, and requires that rooms have been visited before. But suppose we wanted to be a bit more lenient about movement, and let the player make as many steps as necessary per turn. We will also show consideration about doors, using the "Locksmith" extension supplied with Inform. (Now every time the code attempts opening a door, unlocking rules will also be invoked.)

```
"Safari Guide"

Include Locksmith by Emily Short.

The Monkey House is a room. The African Grasslands Exhibit is north of the
Monkey House. The bird door is north of the African Grasslands Exhibit and
south of the Aviary. The Ostrich Enclosure is west of the Aviary. The bird door is
a door. It is closed, lockable, and locked. The silver key is a passkey. It unlocks
the bird door. The player carries the silver key.

Understand "go to [any room]" as going by name. Understand "[any room]" as
going by name. Understand "[door]" as entering.

Going by name is an action applying to one thing.

Check going by name:
    if the noun is the location, say "You're already in [the location]." instead.

Carry out going by name:
    while the player is not in the noun:
        let heading be the best route from the location to the noun, using even
```

```
locked doors;
	if heading is not a direction, say "You can't think how to get there from
here." instead;
	let destination be the room heading from the location;
	say "(heading [heading])[command clarification break]";
	try going heading;
	if the player is not in the destination, rule fails.

Test me with "go to aviary / go to ostrich enclosure / african grasslands".
```

Notice that we continue the movement until one of two things happens: either the player reaches the room that is his destination, or the going attempt doesn't work. In the latter case we stop the action in order to avoid hanging the game up in a loop. This event might occur when the player runs into a locked door, for instance.

308

Example **Palette**

RB

An artist's workshop in which the canvas can be painted in any colour, and where painterly names for pigments ("cerulean") are accepted alongside everyday ones ("blue").

There are hundreds of traditional pigments, from lampblack to burnt sienna, so we will confine ourselves to just two:

```
"Palette"

The Atelier is a room. "The floridly untidy loft space used by a moderately
unsuccessful artist (you, that is)." The canvas, palette and paint brush are here.
Understand "painting" as the canvas.

Colour is a kind of value. The colours are white, red, blue and green.

The canvas has a colour. The canvas is white. The printed name of the canvas
is "largely [colour] canvas".

Painting is an action applying to one thing and one colour. Check painting: if the
noun is not the canvas, say "Centuries of tradition suggest that canvas is the
natural home of paint." instead. Carry out painting: now the colour of the canvas
is the colour understood. Report painting: say "You splash away at the now
[canvas]."

Understand "paint [something] [a colour]" as painting.

Understand "calico" as white. Understand "cerulean" or "cerulean blue" as blue.

Test me with "examine canvas / paint canvas red / examine canvas / paint
canvas cerulean / examine canvas".
```

Example **Baritone, Bass**

309 Letting the player pick a gender (or perhaps other characteristics) before starting play. RB

The "reading a command" activity is rather advanced; for the moment, what we need to understand is that we're intervening in commands at the start of play and insisting that the player's first instruction to the game consist of a choice of gender. After that point, the gender will be set and play will proceed as normal.

In order to do the parsing, we define gender as a kind of value, and give several alternate names to each gender.

```
"Baritone, Bass"

Getting Started is a room.

Gender is a kind of value. The genders are masculine, feminine, and unknown.
Understand "male" or "man" or "M" as masculine. Understand "female" or
"woman" or "F" as feminine.

A person has a gender. The gender of the player is unknown.

When play begins:
    now the command prompt is "Please choose a gender for your character. >".

After reading a command when the gender of the player is unknown:
    if the player's command includes "[gender]":
        now the gender of the player is the gender understood;
        if the gender of the player is unknown:
            say "This story requires a selection of male or female. [run paragraph
on]";
            reject the player's command;
        if the gender of the player is masculine, now the player is male;
        if the gender of the player is feminine, now the player is female;
        say "[line break]Thank you. We now begin...";
        now the command prompt is ">";
        move the player to Sandy Beach;
        reject the player's command;
    otherwise:
        say "Sorry, we're not ready to go on yet. [run paragraph on]";
        reject the player's command.

Sandy Beach is a room.

Instead of examining the player when the player is female:
    say "Congratulations, you are a girl!"

Instead of examining the player when the player is male:
    say "Congratulations, you are a boy!"
```

If we had a whole series of things to ask the player about, we might define a whole series of kinds of value

```
The vocal ranges are soprano, mezzosoprano, contralto...
```

and use a "construction stage" variable to keep track of the current stage of character-construction, as in

```
After reading a command when the current construction stage is choosing a
vocal range:
    ...
```

310

Example Lies

RB

Commands to allow the player to lie down in three different ways.

To set the scene, and make new actions to provide for two of these ways:

```
"Lies"

The Laundry is a room. "An old Limehouse haunt, the Chinese laundry used by
the down-trodden wives of the Tong of the Black Scorpion." The vast marble sink
is here. "There is nothing obviously oriental about the vast marble sink, which is
large enough to lie down inside. A wooden-rack floor, equipped for easy
drainage, turns out also to be equipped for snagging the shoes of passers-by."
The sink is an enterable container, fixed in place.

Lying down is an action applying to nothing. Report lying down: say "You lie
down for a while in the middle of the Laundry, wondering about the point of
existence, then get up again."

Lying near is an action applying to one thing. Report lying near: say "You lie
down next to [the noun] for a while, mumbling to yourself."

Instead of lying near the sink, say "Lying down close to the cool butcher's marble
slabs of the sink, your attention is caught by the sight of coolie shoes through a
floor-level grille for ventilation. The game is afoot!"
```

So far, so good. Now for the grammar, where we create two new tokens: one for each of two groups of alternative prepositions.

```
Understand "beneath/under/by/near/beside/alongside/against" or "next to" or "in
front of" as "[beside]".

Understand "on/in/inside" or "on top of" as "[within]".

Understand "lie down" as lying down.

Understand "lie down [within] [something]" as entering.

Understand "lie [beside] [something]" or "lie down [beside] [something]" as lying
near.
```

Test me with "lie down / lie down on top of the sink / get out / lie down inside the sink / get out / lie down in front of the sink".

---

311 **Example Aspect** *RB*

Understanding aspect ratios (a unit) in the names of televisions.

Named properties are not the only kind that Inform is able to understand referring to an object. We can also use unit and number properties to distinguish things from one another, as here, where televisions have aspect ratios:

"Aspect"

An aspect ratio is a kind of value. 16:9 specifies an aspect ratio.

A television is a kind of device. A television has an aspect ratio. Understand the aspect ratio property as referring to a television. Understand "European standard" as 16:9.

The Office is a room.

The widescreen TV is a television in the Office. The fifties TV is a television in the Office. The widescreen TV is 16:9. The fifties TV is 4:3.

Test me with "examine european standard tv / x 16:9 tv / x 4:3 tv".

---

312 **Example Hymenaeus** *RB*

Understanding "flaming torch" and "extinguished torch" to refer to torches when lit and unlit.

"Hymenaeus"

A torch is kind of thing. Understand the lit property as describing a torch. Understand "lighted" or "flaming" or "burning" as lit. Understand "extinguished" as unlit. A torch is usually lit.

Before printing the name of a lit torch, say "flaming ".
Before printing the name of an unlit torch, say "extinguished ".

The Wedding Procession is a room.

Orpheus is a man in the Wedding Procession. Orpheus carries a torch.
Eurydice is a woman in the Wedding Procession. Eurydice carries a torch.

Rule for writing a paragraph about someone (called target):
    say "[The target] carries [a list of things carried by the target]."

```
Every turn:
    if a random chance of 1 in 2 succeeds and a torch is lit:
        let target torch be a random lit torch;
        now the target torch is unlit;
        say "Aquilo blows down from the north, extinguishing the torch carried by
[the holder of the target torch]."

Instead of examining a lit torch:
    say "It casts a bright glow over [the holder of the noun]."
Instead of examining an unlit torch:
    say "[The holder of the noun] is looking at it disconsolately, obviously worried
about the omens."

Test me with "z / z / z / look / x flaming torch / x extinguished torch".
```

---

313  **Example Channel 1** RB

Understanding channels (a number) in the names of televisions.

We might want to allow every television to be tuned to a channel (a number property) which the player could refer to, so that

```
WATCH CHANNEL 13
TURN OFF CHANNEL 4
```

would be directed to the appropriate television object, if any television is turned on and tuned to the correct station. We might now write:

```
"Channel"

A television is a kind of device. A television has a number called the channel.
Understand the channel property as referring to a television. Understand
"channel" as a television.

The Office is a room. The widescreen TV is a television in the Office. The fifties
TV is a television in the Office.

Changing the channel of it to is an action applying to one thing and one number.

Understand "tune [something] to [number]" or "change channel of [something] to
[number]" as changing the channel of it to.

Check changing the channel of something to:
    if the noun is not a television, say "[The noun] cannot be tuned to a channel."
instead.

Carry out changing the channel of something to:
    now the channel of the noun is the number understood.

Report changing the channel of something to:
    say "You tune [the noun] to channel [number understood]."
```

```
Instead of examining a television:
    if the noun is switched off, say "[The noun] is currently turned off." instead;
    let the chosen channel be the channel of the noun;
    if the chosen channel is a current channel listed in the Table of Television
Channels:
        choose row with current channel of the chosen channel in the Table of
Television Channels;
        say "[output entry][paragraph break]";
    otherwise:
        say "Snow fills the screen of [the noun]."

Table of Television Channels

current channel | output
0 | "The screen of [the noun] is completely black."
4 | "A gloomy female news anchor describes the latest car bomb in Baghdad: 104 dead today, and no sign of change."
5 | "A couple of contestants in spangled scarlet outfits are performing an energetic paso doble."
13 | "On-screen, Ichiro is up to bat with one man on second and no outs."

Test me with "change channel of fifties tv to 4 / x channel 4 / switch on fifties / x
channel 4 / switch on widescreen / tune fifties tv to 5 / x channel 5 / x fifties tv / x
channel 4".
```

---

314  **Example Terracottissima** *RB*

The flowerpots once again, but this time arranged so that after the first breakage all undamaged pots are said to be "unbroken", to distinguish them from the others.

This easiest way to do this uses the "printing the name of" activity, which will come up in the following chapter:

```
"Terracottissima"

A flowerpot is a kind of thing. Understand "pot" as a flowerpot.

A flowerpot can be unbroken or broken. After dropping an unbroken flowerpot:
say "Crack!"; now the noun is broken. Understand the broken property as
describing a flowerpot.

Before printing the name of a broken flowerpot, say "broken ". Before printing the
name of an unbroken flowerpot: if a flowerpot is broken, say "unbroken ".

Before printing the plural name of a broken flowerpot, say "broken ". Before
printing the plural name of an unbroken flowerpot: if a flowerpot is broken, say
"unbroken ".

The Herb Garden is a room. In the Herb Garden are ten unbroken flowerpots.

Test me with "get three flowerpots / drop all / look".
```

315

Example **Peers**

RB

The peers of the English realm come in six flavours - Baron, Viscount, Earl, Marquess, Duke and Prince - and must always be addressed properly. While a peerage is for life, it may at the royal pleasure be promoted.

Almost all of this example is the flummery of pomp and circumstance: only the first two paragraphs really do anything.

```
"Peers" by Elizabeth II R

A title is a kind of value. The titles are Baron, Viscount, Earl, Marquess, Duke and Prince.

A peer is a kind of man. A peer has a title. A peer is usually a Baron. Before printing the name of a peer, say "[title] ". Understand the title property as describing a peer.

The House of Lords is a room. Maltravers, Pollifax, Omnium and St Vincent are peers in the House of Lords. Omnium is a Duke. St Vincent is an Earl.

Ennobling is an action applying to one thing and one title.

Check ennobling:
    if the noun is the player, say "The Sovereign is the fountain of honour, and may not be ennobled." instead;
    if the noun is not a peer, say "Commoners should remain so." instead;
    if the title of the noun is the title understood, say "But that is his title already." instead;
    if the title of the noun is greater than the title understood, say "As he is already of the rank of [title of the noun], any such letters patent are liable to be deemed invalid, following the precedent of the Buckhurst Peerage Case (1876). Best not." instead.

Carry out ennobling:
    now the title of the noun is the title understood.

Report ennobling:
    say "'Whereas Our Parliament for arduous and urgent affairs concerning Us the state and defence of Our United Kingdom and the Church is now met at Our City of Westminster We strictly enjoining Command you upon the faith and allegiance by which you are bound to Us that the weightness of the said affairs and imminent perils considered (waiving all excuses) you be at the said day and place personally present with Us and with the said Prelates Great Men and Peers to treat and give your counsel upon the affairs aforesaid And this as you regard Us and Our honour and the safety and defence of the said Kingdom and Church and dispatch of the said affairs in nowise do you omit Witness Ourself at Westminster the Fifth day of November in the 43rd year of Our Reign,' you say, with unpunctuated serenity. The new [noun] bows stiffly."

Understand "dub [someone] a/an [title]" as ennobling.
```

```
Test me with "dub st vincent a baron / dub maltravers a marquess / look /
examine marquess".
```

---

316  **Example Channel 2** *RB*

Understanding channels (a number) in the names of televisions, with more sophisticated parsing of the change channel action.

Our previous implementation of televisions ("Channel 1") doesn't allow the player to type things like

```
TUNE FIFTIES TELEVISION TO CHANNEL 4
```

nor does it deal with player input like

```
TUNE TO CHANNEL 4 ON FIFTIES TELEVISION
```

or

```
TUNE TO CHANNEL 4
```

where no television is specified. When we are designing commands which involve two elements (here, a television and a channel number), it's usually a good idea to allow the player to specify those elements in either order, as we saw demonstrated briefly in "New commands for old grammar".

We might, therefore, want to add a few refinements: first by defining a "[channel]" token that will accept input of the forms "[number]" and "channel [number]", and second by creating some additional "Understand" lines that will accept variant versions of the player's input.

```
"Channel 2"

Section 1 - Televisions in General

A television is a kind of device.

A television has a number called the channel. Understand the channel property
as referring to a television. Understand "channel" as a television.

Changing the channel of it to is an action applying to one thing and one number.

Understand "tune [television] to [channel]" or "change channel of [television] to
[channel]" as changing the channel of it to.

Understand "tune [something] to [channel]" or "change channel of [something] to
[channel]" as changing the channel of it to.

Understand "tune to [channel] on [television]" or "change to [channel] on
[television]" as changing the channel of it to (with nouns reversed).
```

```
Understand "tune to [channel] on [something]" or "change to [channel] on
[something]" as changing the channel of it to (with nouns reversed).

Understand "[number]" or "channel [number]" as "[channel]".

Check changing the channel of something to:
    if the noun is not a television, say "[The noun] cannot be tuned to a channel."
instead.

Carry out changing the channel of something to:
    now the channel of the noun is the number understood.

Report changing the channel of something to:
    say "You tune [the noun] to channel [number understood]."

Instead of examining a television:
    if the noun is switched off, say "[The noun] is currently turned off." instead;
    let the chosen channel be the channel of the noun;
    if the chosen channel is a current channel listed in the Table of Television
Channels:
        choose row with current channel of the chosen channel in the Table of
Television Channels;
        say "[output entry][paragraph break]";
    otherwise:
        say "Snow fills the screen of [the noun]."

Table of Television Channels

    current channel output
    0               "The screen of [the noun] is completely black."

Section 2 - The Scenario

The Office is a room.

The widescreen TV is a television in the Office. The fifties TV is a television in
the Office.
```

And we add the scenario-specific content to our Table of Television Channels; in the case of channel 13, we provide for a changing sequence of events using text variations.

```
Table of Television Channels (continued)

    current  output
    channel
    4        "A gloomy female news anchor describes the latest car bomb in Baghdad: 104 dead today, and no sign
             of change."
    5        "A couple of contestants in spangled scarlet outfits are performing an energetic paso doble."
    13       "[one of]On-screen, Ichiro is up to bat with one man on second and no outs.[or]Ichiro has singled to first
             and the other man is on third.[or]The next batter is in the middle of flying out.[or]Everything looks rosy
             until the men in black pull off a double-play and retire the side.[or]The channel has cut to a commercial.
             [stopping]"

Test me with "test one / test two".
```

Test one with "change channel of fifties tv to 4 / x channel 4 / switch on fifties / x channel 4 / switch on widescreen / tune fifties tv to channel 5 / x channel 5 / x fifties tv / x channel 4".

Test two with "tune to channel 13 / widescreen / tune channel 13 to channel 5 / tune channel 5 to channel 3 / widescreen / x channel 3".

---

317  **Example Terracottissima Maxima** RB

Flowerpots with textual names that might change during play.

Inform can also understand text properties:

"Terracottissima Maxima"

A flowerpot is a kind of thing. A flowerpot has a text called pattern. Understand the pattern property as describing a flowerpot. The printed name of a flowerpot is usually "[pattern] flowerpot". The printed plural name of a flowerpot is usually "[pattern] flowerpots".

The Herb Garden is a room. In the Herb Garden is a flowerpot with pattern "blue willow". In the Herb Garden is a flowerpot with pattern "striped". In the Herb Garden is a flowerpot with pattern "striped".

Test me with "x blue willow / get striped / look".

This may not seem very much different from having the pattern be a kind of value -- except that texts can, of course, hold almost anything. Further exploration of these possibilities may be found in the chapter on Advanced Text.

---

318  **Example Tilt 1** RB

A deck of cards with fully implemented individual cards, which can be separately drawn and discarded, and referred to by name.

We've simulated a deck of cards before, but only as entries in a table. This time we're going to do it more completely, with card objects that can be drawn and discarded, and referred to by name. The tedious way to do this would be to make 52 objects by hand and laboriously write out their names and understand rules.

A more sensible way is to make 52 identical card objects, assign them ranks and suits, and allow Inform to generate and parse their names automatically.

So:

"Tilt"

Section 1 - Cards

```
Suit is a kind of value. The suits are hearts, clubs, diamonds, and spades.
Understand "heart" as hearts. Understand "club" as clubs. Understand
"diamond" as diamonds. Understand "spade" as spades. [Providing the singular
forms means that Inform will also understand >EXAMINE SPADE, >DISCARD
CLUB, and so on.]

A card is a kind of thing. A card has a suit. A card has a number called rank.
Understand the suit property as describing a card. Understand the rank property
as describing a card.

52 cards are in the card repository.
```

Now, we're going to describe the higher numbers as face cards, so it helps to write a new "to say" phrase, just as we did in Jokers Wild. (A subsequent version of this example shows how to print card values with red and black symbols representing the different suits; see "Tilt 3".)

```
To say (count - a number) as a card value:
    choose row count in the Table of Value Names;
    say "[term entry]".

Rule for printing the name of a card (called target):
    say "[rank of the target as a card value] of [suit of the target]"

Table of Value Names

term      value topic
"ace"     "1"   "ace/A/one"
"deuce"   "2"   "deuce/two"
"three"   "3"   "three"
"four"    "4"   "four"
"five"    "5"   "five"
"six"     "6"   "six"
"seven"   "7"   "seven"
"eight"   "8"   "eight"
"nine"    "9"   "nine"
"ten"     "10"  "ten"
"jack"    "11"  "jack/knave/J"
"queen"   "12"  "queen/Q"
"king"    "13"  "king/K"
```

This is enough already to let inform understand things like "ten clubs", but we want to add a couple of refinements. For one thing, we'd like to accept "of" when it appears in phrases such as "ten of clubs" (but not generically otherwise); for another, we'd like the player to be able to use various names for ranks. To this end, we need to borrow from the Activities chapter and modify the player's command before attempting to understand it:

```
After reading a command:
    if the player's command includes "of [suit]":
        while the player's command includes "of":
            cut the matched text;
    repeat through the Table of Value Names:
        while the player's command includes topic entry:
            replace the matched text with value entry.
    [This allows Inform to understand "ace", "deuce", "king", etc., as numerical
ranks.]
```

It may be a bit confusing that the Table of Value Names has both a topic column and a term column, to all appearances essentially identical. But items in the topic column can be matched against the player's input, whereas items in other kinds of text column can be printed out; the two kinds of text are not treated identically by Inform, so we need to have both. Notice that the topic column contains entries like "jack/knave," which will match either "jack" or "knave" in the player's input.

Now to set up the deck at the outset. With some intelligent looping, we avoid having to declare every combination of suit and number individually:

```
When play begins:
    reconstitute deck.

To reconstitute deck:
    let current suit be hearts;
    now every card is in the card repository;
    while a card is in the card repository:
        repeat with current rank running from 1 to 13:
            let item be a random card in card repository;
            now rank of item is current rank;
            now suit of item is current suit;
            now item is in the deck of cards;
        now current suit is the suit after the current suit.
```

And now we need a simple setting and some actions to manipulate the deck with:

```
Section 2 - The Deck and the Discard Pile

The Empty Room is a room. "Nothing to see here."

The deck of cards is in the Empty Room. It is a closed unopenable container.
The description is "A standard poker deck."

The discard pile is a closed unopenable container. The description is "Cards in
this game are discarded face-down, so the discard pile is not very interesting to
see. All you can observe is that it currently contains [if the number of cards
which are in the discard pile is less than ten][the number of cards which are in
the discard pile in words][otherwise]about [the rounded number of cards which
are in the discard pile in words][end if] card[s]."

To decide what number is the rounded number of (described set - a description
of objects):
    let N be the number of members of the described set;
    let R be N divided by 5;
    let total be R times 5;
    decide on total.
```

The above phrase rounds a number to the nearest five, so that the player is not autistically able to count a large number of cards in the discard pile at a single glance.

This next bit is an optional borrowing from the Activities chapter: we want to prevent Inform printing things like "You can see a discard pile (closed) here.", since

we don't want the player to think of the piles as containers, even though Inform thinks of them in those terms.

```
Rule for printing room description details of something: do nothing instead.
```

Finally, we want the player to use "draw" and "discard" to manipulate his hand of cards:

```
Section 3 - Drawing and Discarding Actions

Understand the commands "take" and "carry" and "hold" and "get" and "drop" and "throw" and "discard" as something new.

Understand "take [text]" or "get [text]" or "drop [text]" as a mistake ("Here, you only draw and discard. Nothing else matters at the moment.").

Understand "draw" or "draw card" or "draw a card" as drawing. Drawing is an action applying to nothing. The drawing action has an object called the card drawn.

Setting action variables for drawing:
    now the card drawn is a random card which is in the deck of cards.

Check drawing:
    if the card drawn is nothing, say "The deck is completely depleted." instead.

Check drawing:
    if the number of cards carried by the player is greater than four,
        say "This is a five-card game; you must discard something before drawing anything further." instead.

Carry out drawing:
    move the card drawn to the player.

Report drawing:
    say "You draw [a card drawn]."

Understand "discard [card]" as discarding. Discarding is an action applying to one thing.

Check discarding:
    if the player does not carry the noun, say "You can only discard cards from your own hand." instead.

Carry out discarding:
    now the noun is in the discard pile;
    if the discard pile is not visible, move the discard pile to the location.

Report discarding:
    say "You toss [the noun] nonchalantly onto the discard pile."

Seeding is an action out of world. Understand "seed" as seeding. Carry out seeding: seed the random-number generator with 5681.
```

```
Test me with "seed / draw / g / g / g / g / i / discard seven of spades / draw /
discard six / draw / i / discard hearts / discard six of diamonds card / draw / draw
/ i / discard spades card / draw / discard king card".
```

319

Example Cinco

RB

A taco shell that can be referred to (when it contains things) in terms of its contents.

It's fairly common that we want to be able to refer to a container in terms of what it has in it: a bottle of wine, a salt shaker, a chicken sandwich. The player is free to remove the contents again, and the object will go back to using its usual name:

```
"Cinco"

Cinco de Mayo Fundraiser is a room.

The taco shell is an edible thing in the Fundraiser. It is a portable container. It
has carrying capacity 1.

Understand "[something related by containment] taco" as the taco.

Rule for printing the name of the taco shell while not inserting or removing:
    if the taco contains something (called filling), say "[filling] taco";
    otherwise say "taco shell";
    omit contents in listing.

The player carries shredded beef. It is edible.

The taking action has an object called source (matched as "from").

Setting action variables for taking:
    now source is the holder of the noun.

Report taking something from the taco shell:
    say "You gingerly pick [the noun] out of the taco shell." instead.

Test me with "x taco / put shredded beef in taco / get taco / i / x shredded beef
taco / get shredded beef / x shredded beef taco".
```

320

Example Puncak Jaya

RB

When a character is not visible, responding to such commands as EXAMINE PETER and PETER, HELLO with a short note that the person in question is no longer visible.

By default, when something is not present, Inform does not allow a player to refer to it. But there are times when we might like to acknowledge that the thing mentioned

in a command does exist somewhere in the game; it just happens not to be on hand right now.

One way to do this is to make an object that appears everywhere and responds to the name of its owner only when the owner itself is not in view.

```
"Puncak Jaya"

A ghost is a kind of person. A man-ghost is a kind of ghost. A man-ghost is
always male. A woman-ghost is a kind of ghost. A woman-ghost is always
female.
```

We make the ghost a person rather than some other kind of thing so that it will be able to respond to commands such as KISS BOB or (even trickier) BOB, JUMP: if Inform did not recognize the ghost as an animate creature, it would not accept such input.

```
Representation relates one ghost to one person. The verb to represent means
the representation relation.

One man-ghost represents every man. One woman-ghost represents every
woman.
```

This is, technically, an assembly -- except instead of saying that every device has a button part, or that there are three daffodils in every garden room, the assembly is based on a non-physical relation that we just designed.

Based on the "representation" relation, we now devise a conditional relation that applies only when the represented thing is not itself in view:

```
Indication relates a ghost (called X) to a person (called Y) when X represents Y
and Y is not visible.

Understand "[something related by indication]" as a ghost.

When play begins:
    now every ghost is in the concept-repository.

Instead of doing something to a ghost:
    say "You seem to have left [a random person which is represented by the
noun] behind."

Instead of doing something when the second noun is a ghost:
    say "You seem to have left [a random person which is represented by the
second noun] behind."

The concept-repository is an open unopenable transparent container. It is part of
the air. The air is a backdrop. It is everywhere.

Base of Puncak Jaya is a room. Temple, Kippax, and Huizenga are men in Base.
Peak of Puncak Jaya is above Base of Puncak Jaya.

Test me with "x kippax / up / x kippax / kiss kippax / kippax, hello".
```

Further complications of this example might require that the player meet a character before being able to refer to him or her.

---

321 **Example** **Whither?** *RB*

A door whose description says where it leads; and which automatically understands references such as "the west door" and "the east door" depending on which direction it leads from the location.

Here we expand on the simple examples When? and Whence?; this time we want the player to be able to refer to doors by their directions, as in "the west door" when the door in question does in fact lead west.

```
"Whither?"

The temporal vortex is an open door. It is west of Yesterday and east of Today.

The initial appearance of a door is usually "Nearby [an item described] leads [if
the other side of the item described is visited][direction of the item described
from the location] to [the other side][otherwise][direction of the item described
from the location][end if]."

Direction-relevance relates a door (called X) to a direction (called Y) when the
direction of X from the location is Y. The verb to be directionally-relevant to
means the direction-relevance relation.

Understand "[something related by direction-relevance] door" as a door.
```

As an added touch, we respond also to the case where the player postulates a door in some direction when there is no such thing at the moment:

```
Rule for printing a parser error when the player's command includes "[non-door
direction] door":
    say "There is no door in that direction." instead.

Definition: a direction (called direction D) is non-door:
    let the target be the room-or-door direction D from the location;
    if the target is a door:
        no;
    yes;

Test me with "examine west door / x east door / w / x w door / x e door / tie me to
the west door / tie the west door to me / push the west door east / push the east
door west".
```

---

322 **Example** **Claims Adjustment** *RB*

An instant camera that spits out photographs of anything the player chooses to take a picture of.

We start by creating a camera and a photograph object. As usual when we want to have a kind of object that can be dispensed in bulk, we start off with a bunch of identical instances of the object out of play (in this case, kept in an out-of-play container called "film roll"); we can then move them into play and give them characteristics when they're needed.

Each photograph can depict exactly one thing -- we're assuming that the player is not a landscape photographer here -- so we create a relation to indicate what is shown by each photograph. We'll then use that relation to determine how photographs are described, named, and parsed:

```
"Claims Adjustment"

A photograph is a kind of thing. 36 photographs are in the film roll.

Appearance relates one thing to various photographs. The verb to be shown by means the appearance relation.

The description of a photograph is usually "It shows [a random thing which is shown by the item described]."

Understand "of [something related by reversed appearance]" as a photograph.
```

This allows the player to refer to any photograph by its subject: useful if we have a large number of them.

Now we create an action to let the player use the camera and generate these photograph objects:

```
The player carries a cheap instant camera.

Understand "photograph [something] with [camera]" as photographing.
Understand "photograph [something] with [something preferably held]" as photographing. Photographing is an action applying to one visible thing and one carried thing, requiring light.

The photographing action has an object called the selected film.

Setting action variables for photographing:
    let N be a random photograph in the film roll;
    now the selected film is N.

Check photographing:
    if the second noun is not the camera, say "You need a camera for that purpose." instead.

Check photographing:
    if the noun is the camera, say "Sadly impossible." instead.

Check photographing:
    if the selected film is nothing, say "You're out of film." instead.
```

```
Carry out photographing:
    now the noun is shown by the selected film;
    move the selected film to the player.

Report photographing:
    say "Your camera instantly spits out [a selected film]."
```

Now we use two activities from the Activities chapter to describe the photographs to the player more elegantly:

```
After printing the name of a photograph (called target):
    say " of [a random thing which is shown by the target]".

After printing the plural name of a photograph (called target):
    let N be the holder of the target;
    say " of [a list of things which are shown by photographs which are held by
N]";
    if the number of things which are shown by photographs which are held by N
is greater than one, say " (variously)".
```

And finally we provide a brief scenario to give the player something to take pictures of:

```
The Treasure Room is a room. "Despite the fancy name, this is no more than a
closet -- albeit a closet with its own special circuit on the house alarm."

The Treasure Room contains a small Degas, a Ming vase, and a collection of
South African krugerrands. The player is carrying insurance forms, a first-class
stamp, and a security envelope.

The description of the forms is "Completely filled out in black ink in block letters:
now all you need to do is attach photographic evidence of the objects you wish
to insure."

Test me with "photograph degas / i / photograph degas / i / x photograph of
degas / photograph me / x photograph of me / i / photograph vase / photograph
camera / photograph collection / g / i / test more".

Test more with "x photograph of collection / x photograph of krugerrands / x
photograph of collection of south african krugerrands / photograph photograph of
degas / x photograph of photograph of degas".
```

323

Example Quiz Show

RB

In this example by Mike Tarbert, the player can occasionally be quizzed on random data from a table; the potential answers will only be understood if a question has just been asked.

```
"Quiz Show" by Mike Tarbert

Use scoring.
```

```
Answer mode is a truth state that varies.
Current state is a text that varies.

Guessing is an action applying to one topic.
Understand "[text]" as guessing when answer mode is true.
```

Because of the "...when" part of this line, random text is only treated as an answer when a question is being asked.

```
Check guessing (this is the default wrong answer rule):
    if the topic understood is not a topic listed in the Table of Dates of Statehood:
        say "Wrong!";
    now answer mode is false.

Carry out guessing a topic listed in the Table of Dates of Statehood:
    if state entry is the current state:
        say "Correct! ([comment entry], to be exact!)";
        increase the score by one;
    otherwise:
        say "Wrong!";
    now answer mode is false.
```

This next rule allows a player to do something other than answer the question, but then makes him wait for another question before answering.

```
Before doing anything other than guessing:
    if answer mode is true:
        say "(ignoring the question)[line break]";
    now answer mode is false.

Section 2 - Scenario

The Lab is a room. Sam is a man in the lab.

Every turn when the player is in the lab:
    if a random chance of 3 in 5 succeeds:
        choose a random row in the Table of Dates of Statehood;
        say "Sam asks you, 'In what year was [state entry] admitted into the
Union?'";
        now current state is state entry;
        now answer mode is true.

Table of Dates of Statehood

    State        Topic   Comment
    "Florida"    "1845"  "March 3rd"
    "Delaware"   "1787"  "December 7th"
    "Hawaii"     "1960"  "July 4th"

Test me with "1845 / z / z / 1787 / 1792 / z / 1845 / g".
```

Note that the situation will become a little more complicated if we have two or more identical topics in our trivia list; in that case, we would need to loop through the Table of Dates of Statehood explicitly, and only mark the player wrong if none of the

lines were found to match. (See the chapter on Tables for many more ways to manipulate table behavior.)

---

324  **Example Bibliophilia** *RB*

A bookshelf with a number of books, where the player's command to examine something will be interpreted as an attempt to look up titles if the bookshelf is present, but otherwise given the usual response.

Suppose we want a bookshelf with a very large number of books on it. They aren't to be taken or carried around in the game, but they should be mentioned, and the player should be allowed to look them up by name. Furthermore, the player's attempts to examine something unrecognized should be understood as an attempt to look up a title -- but only when the player is in the presence of the books. The rest of the time such requests should be rejected in the usual way.

```
"Bibliophilia"

The Graduate Lounge is a room. "Shabby sofas; plastic cups remaining from the afternoon's pre-lecture espresso; a collection of Xena and Hercules figurines posed for ironic effect. It's somewhat depressing at this hour, when everyone has gone home."

The Classics Reading Room is south of the Lounge. "Not as large a collection as the one in the Library, but it contains copies of everything really essential for reference."

Understand "examine [text]" as examining as a book when the player is in the Reading Room. Understand "look up [text]" as examining as a book when the player is in the Reading Room.

Examining as a book is an action applying to one topic.

Carry out examining as a book:
    say "You can't find any such text."

Instead of examining as a book a topic listed in the Table of Book Titles:
    say "[description entry][paragraph break]"

Table of Book Titles
topic                                                   title                   description
"Reading Greek Death" or "reading/greek/death" or "greek death"     "Reading Greek Death"   "A dense orange paperback treatise on the development of Greek eschatology."
"TAPA/Transactions/134-2"                               "TAPA 134-2"            "Transactions of the American Philological Association from 2004."
"Oxford Classical Dictionary" or "OCD/dictionary/classical/oxford"  "Oxford Classical Dictionary"  "A hefty reference with short articles on everything from Greek meter to ancient cosmetics."
"Collected Dialogues of Plato" or "Plato/dialogues/hamilton/cairns"  "Collected Dialogues of Plato"  "All the Platonic dialogues -- some, admittedly, in rather tired translations -- but still a useful single volume, ed. Edith Hamilton and Huntington Cairns."
"Adobe Illustrator CS User Guide" or "user guide" or "adobe illustrator" or  "Adobe Illustrator  "Hello, how did this get here? A suspiciously familiar name is scribbled inside the front cover..."
```

```
Some books are scenery in the Reading Room. Understand "copies" or "book"
or "shelf" or "shelves" as the books. Instead of examining the books:
    choose a random row in the Table of Book Titles;
    say "You scan the shelves and notice, among others, a volume entitled [italic
type][title entry][roman type]."

Test me with "south / examine ocd / examine books / examine books / examine
plato / n / x hercules / s / x hercules".
```

Now if we type >X HERCULES in the Lounge, we will get

```
>x hercules
You can't see any such thing.
```

thanks to our somewhat slovenly implementation of the Lounge scenery; but in the Reading Room,

```
>x hercules
You can't find any such text.
```

In practice we might also want to extend our coverage somewhat to handle a case where the player tried to take books from the bookshelf: currently that would not be understood.

325

Example **Pot of Petunias**

RB

Responding sensibly to a pot of petunias falling from the sky.

Suppose we have an object that makes a dramatic entrance on the scene, like so:

```
"Pot of Petunias"

Wide Open Field is a room. "A big field under a big sky. The clouds are puffy, the
trees are handsome."

Some clouds and some trees are scenery in Wide Open Field. The description of
the clouds is "That one looks like Yoda's head." The description of the trees is
"You've never been much good at botany, so it's anyone's guess what kind they
are."

A rock is in Wide Open Field. The description of the rock is "It looks like it's been
here from the dawn of time."

The broken flower pot is a thing. The description of the broken flower pot is "It
contains the remains of some abused petunias."

At 9:01 am:
    move the broken flower pot to the location;
```

```
    say "Quite unexpectedly, a flower pot falls from the sky and breaks open on
the ground. Good thing you weren't standing six inches to the left.";
    set pronouns from the broken flower pot.

Test me with "x it / x it / x it".
```

If we leave out the "set pronouns..." line here, we'll wind up with the following very unsatisfactory end to our test transcript:

```
Quite unexpectedly, a flower pot falls from the sky and breaks open on the
ground. Good thing you weren't standing six inches to the left.

>[3] x it
It looks like it's been here from the dawn of time.
```

326

Example **Masochism Deli**

RB

Multiple potatoes, with rules to make the player drop the hot potato first and pick it up last.

Here the player has several potatoes; we would like to make him more likely to drop the hot one, and more likely to pick up the cold one, all else being equal. At the same time, we want to phrase our rules so that they don't make the player try to take something he's already holding, or drop something he isn't.

So:

```
"Masochism Deli"

The Masochism Deli is a room. "Recent restructurings of corporate policy restrict
the 'lunch hour' to exactly thirty-two minutes, which means that no one has time
to go out. Instead, you and your coworkers eat here, in the company's very own
themed lunch room."

The plural of potato is potatoes. A potato is a kind of thing. A potato is edible.

Temperature is a kind of value. The temperatures are hot and cold. A potato has
a temperature. A potato is usually cold.

Understand the temperature property as describing a potato. Before printing the
name of a potato (called subject): say "[temperature of subject] ". Before printing
the plural name of a potato (called subject): say "[temperature of subject] ".

Does the player mean dropping a hot potato which is carried by the player: it is
very likely.

Does the player mean taking a cold potato which is not carried by the player: it is
very likely.

The player carries three potatoes.
```

```
After dropping a hot potato:
    say "The guy from Cube B sneers at your lack of potato-holding stamina."

When play begins: now a random potato is hot.

Test me with "inventory / drop potato / g / g / get potato / g / i / get potato".
```

---

327  **Example The Best Till Last** *RB*

Reordering multiple objects for dramatic effect.

If a single command asks to do many things, some dull and some exciting, we may want to save the good ones for the end.

```
"The Best Till Last"

The Funky Ignition Lounge is a room. "This is where all evenings end." The stick
of gelignite, the solid magnesium footstool, the vetiver candle, and the vodka
bottle are here.

The burn description of the vetiver candle is "It burns right down, expensively but
gothically."

The player carries an inexpensive firework. The description of the firework is "It
is a cardboard tube with red and green stripes along the outside, and a fuse
sticking out of the end." The burn description of the firework is "It ignites
gloriously! You take a few hasty steps back in order to avoid burning yourself,
and not a moment too soon. Red and green sparks fly out of the tube, and
there's a whistling noise punctuated by several loud cracks."

The player carries a lighter. The description of the lighter is "You don't smoke,
but you like to have access to flame now and then anyway."

Burning it with is an action applying to one thing and one carried thing.

Understand "burn [things] with [something preferably held]" as burning it with.

The block burning rule is not listed in any rulebook.

A thing has some text called the burn description.

Check burning something:
    if the player carries the lighter:
        try burning the noun with the lighter;
    else:
        try burning the noun with the noun.

Check burning something with something when the second noun is not the
lighter:
    say "Your trusty lighter is the best flame source available to you." instead.
```

```
Check burning something with something:
    if the burn description of the noun is "":
        say "Best not." instead.

Carry out burning something with something:
    remove the noun from play.

Report burning something with something:
    say "[burn description of the noun][line break]".

A multiple action processing rule when the action name part of the current action
is the burning it with action (this is the orderly burn rule):
    let L be the multiple object list;
    let dull list be a list of objects;
    let fun list be a list of objects;
    repeat with item running through L:
        if the burn description of the item is "":
            add item to dull list;
        else:
            add item to fun list;
    let F be the dull list;
    add fun list to F;
    alter the multiple object list to F.

Test me with "burn all with lighter".
```

---

328 **Example Western Art History 305** *RB*

Allowing EXAMINE to see multiple objects with a single command.

In a gallery, there are many individual things to look at, but you can also get a general impression by just examining them as a collection.

First, we'll make a kind for the paintings exhibited in the gallery, and then we'll also make a special object to represent all of them as a mass:

```
"Western Art History 305"

A painting is a kind of thing. A painting is usually fixed in place. Understand
"painting" as a painting. Understand "paintings" as the plural of painting.

The painting-collective is a thing. The printed name of the painting-collective is
"paintings". The description of the painting-collective is "There's [a list of visible
paintings]."
```

We could if we wanted tweak the description to be different in style in different rooms of the gallery, but this will do for now. Next we need to make it possible to type something like EXAMINE PAINTINGS, which normally wouldn't work because the Standard Rules don't tell Inform to recognise multiple objects with the EXAMINE command (unlike, say, DROP or TAKE). This is easy:

```
Understand "examine [things]" as examining.
```

Now to make use of the special object. If the player types EXAMINE PAINTINGS, the multiple object list will become a list of the visible paintings. The following rule looks at this list: if it contains more than one painting, it replaces them with the painting-collective instead. Now there's only one examining action, so we get a reply like "There's an abstract painting, a pointilist painting and a French academic painting." instead of a list of descriptions of each in turn.

```
A multiple action processing rule when the current action is examining (this is the
examine kinds rule):
    let L be the multiple object list;
    let F be L;
    let the painting count be 0;
    repeat with item running through L:
        if the item is a painting:
            increment the painting count;
            remove the item from F;
    if the painting count is greater than one:
        add the painting-collective to F;
        alter the multiple object list to F.
```

And now some art to try this out on:

```
Gallery is a room. "Various paintings hang on the walls of this gallery, awaiting
critical attention. A side chamber to the north contains smaller works."

The abstract painting, the pointilist painting, and the French academic painting
are paintings in the Gallery.

North of the Gallery is the Side Chamber. A handsome miniature is a painting in
the Side Chamber. The description of the handsome miniature is "The miniature
depicts a uniformed soldier of the late 18th century, with braid on his shoulders
and a curl in his beard."

The player carries a small notebook. The description of the notebook is "It
contains the notes you've taken so far towards a paper for Western Art History
305. So far you're still feeling a bit uninspired."

Test me with "x paintings / x all / n / x paintings / x all".
```

329

Example Query

RB

Catching all questions that begin with WHO, WHAT, WHERE, and similar question words, and responding with the instruction to use commands, instead.

First, we create a single "[query]" token so that we can capture all instances of such sentences in a single line:

```
"Query"
```

```
Blank Room is a room.

Understand "who" or "what" or "when" or "where" or "why" or "how" or "who's" or
"what's" or "when's" or "where's" or "why's" or "how's" as "[query]".

Understand "[query] [text]" as a mistake ("[story title] understands commands,
such as '[command prompt]examine [a random thing that can be seen by the
player]', but not questions. For more instructions, type HELP.").

Test me with "who am I? / who are you? / where is this place?".
```

Now the game will respond to all questions novice players might type with this reminder to look for help information.

330

Example The Gorge at George

RB

If the player tries to TALK TO a character, suggest alternative modes of conversation.

```
"Gorge at George"

The Dusty Lot is a room. "A few miles up the road from the concert venue, but at
least it's cheap to park here."

The motorcyclist is a man in the Dusty Lot. "A man clad in [a list of things worn
by the motorcyclist] leans against his Harley and watches you without saying
anything." The Harley is scenery in the Lot. The motorcyclist wears a black
leather jacket and shades. Understand "man" or "guy" as the motorcyclist.

Understand "talk to [someone]" as a mistake ("To start a conversation, try to
ASK [the noun] ABOUT something or TELL [the noun] ABOUT something.").

Instead of asking the motorcyclist about something:
    say "He smirks cryptically."

Instead of telling the motorcyclist about something:
    say "This does not seem to interest him much."

Test me with "talk to motorcyclist / ask motorcyclist about himself / tell
motorcyclist about me".
```

331

Example Hot Glass Looks Like Cold Glass

RB

Responding to references to a property that the player isn't yet allowed to mention (or when not to use "understand as a mistake").

Suppose we have a situation where the player is allowed to talk about the heat of an object only if he's properly equipped to detect it.

```
"Hot Glass Looks Like Cold Glass"

Use scoring.

Heat is a kind of value. The heats are hot, warm, room temperature, and cold. A
thing has a heat.

Understand the heat property as referring to a thing when the player wears the
infrared goggles.

The Test Kitchen is a room. "Your own personal lab, ready for scrupulously
scientific recipe research. You hope. The previous three runs of this did not go
well." The pair of infrared goggles is carried by the player. The description is "A
pair of head-mounted IR goggles which look very foolish when worn." The
goggles are wearable.

A glass dish is a kind of container. A glass dish is transparent. Three room
temperature glass dishes are on the counter. Two hot glass dishes are on the
counter. Two cold glass dishes are on the counter. The counter is scenery in the
Test Kitchen.

Instead of doing something other than examining to a hot glass dish:
    say "Ow! Crikey! You swear, and Claudia makes a sympathetic hiss. 'You're
going to have a mark from that for sure,' she comments.";
    decrease the score by 2.

Instead of doing something when a hot glass dish is the second noun:
    say "You brush [the second noun], and wince, but manage to conceal that
from Claudia.";
    decrement the score.

Before printing the name of a glass dish when the player wears the goggles: say
"[heat] "

Before printing the plural name of a glass dish when the player wears the
goggles: say "[heat] "
```

So far, so good. Now, what if the player tries to GET HOT DISH when the goggles are off? "You can't see any such thing." doesn't seem like quite the right response: he can see such a thing. He just doesn't know which it is.

We could go on to write a mistake rule that would scold the player for trying "get [heat] [text]" when not wearing the goggles. The problem is that this would not cover any other phrasing of the command, nor would it account for all the many other things the player might try to do with an object specified by heat.

What we really want is to catch all instances of the player using the property name when not allowed to do so; and for this purpose we can borrow a trick from the chapter on Activities:

```
After reading a command:
    if the player wears the goggles, make no decision;
    if the player's command includes "[heat]":
        say "Without the IR goggles on, you cannot tell hot things from cold at
```

```
sight.";
        rule succeeds.

Claudia is a woman in the Test Kitchen. "Your assistant Claudia stands by with [a
list of things carried by Claudia]." The description of Claudia is "Infinitely patient
and a very good stenographer. She is studiously avoiding giving you any sort of
look that might be construed as mocking." Claudia carries a notepad, a brined
chicken breast, a blowtorch, and a cup of heavy cream.

Instead of asking Claudia for something which is carried by Claudia:
    move the second noun to the player;
    say "'Check, [second noun],' repeats Claudia, in the tone of one who has seen
too many medical dramas. She does hand it over, though."

Instead of asking Claudia for something:
    say "She clears her throat faintly and glances at [the second noun], as though
to say that it's not hers to give."

Test me with "get all / drop all / look / wear goggles / look / ask claudia for cream
/ put cream in hot dish / put cream in cold dish / remove goggles / get hot dish".
```

332

Example Some Assembly Required

RB

Building different styles of shirt from component sleeves and collars.

We now have the mechanisms in place to do some fairly sophisticated renaming of objects. For instance:

```
"Some Assembly Required"

Garment type is a kind of value. The garment types are vest, t-shirt, polo shirt,
mandarin blouse, button-down, shell, experiment.

Every turn:
    assign identities.

When play begins: assign identities.

To assign identities:
    repeat with item running through torsos:
        reassess item.

To reassess (item - a torso):
    if the number of things which are part of the item is 0:
        now garment type of the item is vest;
        rule succeeds;
    if exactly two short sleeves are part of the item:
        if a collar is part of the item,
            now garment type of the item is polo shirt;
        otherwise now garment type of the item is t-shirt;
        rule succeeds;
    if exactly two long sleeves are part of the item:
```

```
        if a collar is part of the item,
            now garment type of the item is button-down;
        otherwise now garment type of the item is mandarin blouse;
        rule succeeds;
    if a collar is part of the item and the number of sleeves which are part of the
item is 0, now garment type of the item is shell;
    otherwise now garment type of the item is experiment.

Before cutting something which is worn by the player:
    try taking off the noun.

Instead of cutting something when something is part of the noun:
    say "You cut up [the noun], snipping off [a list of things which are part of the
noun].";
    now every thing which is part of the noun is in the holder of the noun.

Instead of cutting something which is part of something:
    say "You carefully snip [the noun] free.";
    now the player carries the noun.

Rule for printing the name of a torso: say "[garment type]".

A torso is a kind of thing. A torso is always wearable. Understand "shirt" or
"blouse" as a torso. A torso has a garment type. Understand the garment type
property as describing a torso. A sleeve is a kind of thing. A short sleeve is a
kind of sleeve. A long sleeve is a kind of sleeve. A collar is a kind of thing.

Understand "sew [something] to [something]" as affixing it to. Affixing it to is an
action applying to two things. Carry out affixing something to something: now the
noun is part of the second noun. Report affixing something to something: assign
identities; say "You sew [the noun] on, creating [a second noun]." Understand
the command "stitch" as "sew".

Instead of affixing something to something when the second noun is worn: say
"You're wearing [the second noun]!"

Instead of affixing a torso to something:
    if the second noun is a torso, say "Couture for Siamese twins is a daring field,
but a bit of a niche market.";
    otherwise try affixing the second noun to the noun.

Instead of affixing a sleeve to something when at least two sleeves are part of
the second noun: say "[The second noun] already sports [a list of sleeves that
are part of the second noun]."

Instead of affixing a collar to something when a collar is part of the second noun:
say "[The second noun] already sports [a list of collars that are part of the
second noun]."

Instead of examining something when something is part of the noun: say
"Stitched to [the noun] [is-are a list of things which are part of the noun]."
```

Here is where the issue of precedence arises. We want to encourage Inform to select a cuttable object that is part of something else, rather than one of the spares:

Definition: a thing is removable if it is part of something. Understand "cut [removable thing]" as cutting.

The Boutique is a room. "Still festively strewn with the confetti and streamers of the Grand Opening party, and still almost totally customer-free."

The player carries a torso. The player carries three short sleeves. The player carries two long sleeves. The player carries two collars.

Test me with "sew collar to shirt / i / sew short sleeve to shirt / g / i / x polo shirt / cut collar / i / cut shirt / sew long sleeve to shirt / i / sew long sleeve to shirt / i / sew collar to shirt / g / i / wear button-down".

---

333  **Example Lakeside Living** *RB*

Similar to "Lemonade", but with bodies of liquid that can never be depleted, and some adjustments to the "fill" command so that it will automatically attempt to fill from a large liquid source if possible.

Much of what follows is identical to "Lemonade" earlier; the new material begins at Part 2.

"Lakeside Living"

A volume is a kind of value. 15.9 fl oz specifies a volume with parts ounces and tenths (optional, preamble optional).

A fluid container is a kind of container. A fluid container has a volume called a fluid capacity. A fluid container has a volume called current volume.

The fluid capacity of a fluid container is usually 12.0 fl oz. The current volume of a fluid container is usually 0.0 fl oz.

Liquid is a kind of value. The liquids are water, absinthe, and iced tea. A fluid container has a liquid.

```
Instead of examining a fluid container:
    if the noun is empty,
        say "You catch just a hint of [the liquid of the noun] at the bottom.";
    otherwise
        say "[The noun] contains [current volume of the noun in rough terms] of
[liquid of the noun]."

To say (amount - a volume) in rough terms:
    if the amount is less than 0.5 fl oz:
        say "a swallow or two";
    otherwise if tenths part of amount is greater than 3 and tenths part of amount
is less than 7:
        let estimate be ounces part of amount;
        say "[estimate in words] or [estimate plus 1 in words] fluid ounces";
    otherwise:
```

```
        if tenths part of amount is greater than 6, increase amount by 1.0 fl oz;
        say "about [ounces part of amount in words] fluid ounce[s]".

Before printing the name of a fluid container (called the target) while not drinking
or pouring:
    if the target is empty:
        say "empty ";
    otherwise:
        do nothing.

After printing the name of a fluid container (called the target) while not examining
or pouring:
    unless the target is empty:
        say " of [liquid of the target]";
        omit contents in listing.

Instead of inserting something into a fluid container:
    say "[The second noun] has too narrow a mouth to accept anything but
liquids."

Definition: a fluid container is empty if the current volume of it is 0.0 fl oz.
Definition: a fluid container is full if the current volume of it is the fluid capacity of
it.

Understand "drink from [fluid container]" as drinking.

Instead of drinking a fluid container:
    if the noun is empty:
        say "There is no more [liquid of the noun] within." instead;
    otherwise:
        decrease the current volume of the noun by 0.2 fl oz;
        if the current volume of the noun is less than 0.0 fl oz, now the current
volume of the noun is 0.0 fl oz;
        say "You take a sip of [the liquid of the noun][if the noun is empty], leaving
[the noun] empty[end if]."

Part 2 - Filling

Understand the command "fill" as something new.
```

Here we want Inform to prefer full liquid sources to other containers when it chooses an end to a player's unfinished or ambiguous command. And so:

```
Understand "fill [fluid container] with/from [full liquid source]" as filling it with.
Understand "fill [fluid container] with/from [fluid container]" as filling it with.
```

Both grammar lines point to the same ultimate outcome; the purpose of specifying both is to tell Inform to check thoroughly for full liquid sources before falling back on other fluid containers when making its decisions.

```
Understand "fill [something] with/from [something]" as filling it with.

Filling it with is an action applying to two things. Carry out filling it with: try
pouring the second noun into the noun instead.
```

```
Understand "pour [fluid container] in/into/on/onto [fluid container]" as pouring it
into. Understand "empty [fluid container] into [fluid container]" as pouring it into.

Understand "pour [something] in/into/on/onto [something]" as pouring it into.
Understand "empty [something] into [something]" as pouring it into.

Pouring it into is an action applying to two things.

Check pouring it into:
    if the noun is not a fluid container, say "You can't pour [the noun]." instead;
    if the second noun is not a fluid container, say "You can't pour liquids into [the
second noun]." instead;
    if the noun is the second noun, say "You can hardly pour [the noun] into itself."
instead;
    if the liquid of the noun is not the liquid of the second noun:
        if the second noun is empty, now the liquid of the second noun is the liquid
of the noun;
        otherwise say "Mixing [the liquid of the noun] with [the liquid of the second
noun] would give unsavory results." instead;
    if the noun is empty, say "No more [liquid of the noun] remains in [the noun]."
instead;
    if the second noun is full, say "[The second noun] cannot contain any more
than it already holds." instead.

Carry out pouring it into:
    let available capacity be the fluid capacity of the second noun minus the
current volume of the second noun;
    if the available capacity is greater than the current volume of the noun, now
the available capacity is the current volume of the noun;
    increase the current volume of the second noun by available capacity;
    decrease the current volume of the noun by available capacity.

Report pouring it into:
    say "[if the noun is empty][The noun] is now empty;[otherwise][The noun] now
contains [current volume of the noun in rough terms] of [liquid of the noun]; [end
if]";
    say "[the second noun] contains [current volume of the second noun in rough
terms] of [liquid of the second noun][if the second noun is full], and is now
full[end if]."

Understand the liquid property as describing a fluid container. Understand "of" as
a fluid container.
```

And now we add our liquid source kind, which will represent lakes, absinthe fountains, and any other infinite supplies of liquid we might need. Note that 3276.7 is the largest possible number of fluid ounces available to us.

```
A liquid source is a kind of fluid container. A liquid source has a liquid. A liquid
source is usually scenery. The fluid capacity of a liquid source is usually 3276.7
fl oz. The current volume of a liquid source is usually 3276.7 fl oz. Instead of
examining a liquid source: say "[The noun] is full of [liquid of the noun]."

Carry out pouring a liquid source into something: now the current volume of the
noun is 3276.7 fl oz.
```

We want filling things from liquid sources to work the same way as usual, with the distinction that a) the liquid source never depletes in quantity (hence the carry-out rule resetting its fullness); and b) we should report the results a bit differently as well:

```
After pouring a liquid source into a fluid container:
    say "You fill [the second noun] up with [liquid of the noun] from [the noun]."
```

On the other hand, pouring liquids into a liquid source needs to work completely differently from pouring liquids into anything else. Let's say we're going to allow any liquid at all to be dumped into rivers and streams (environmental protections evidently are not very well-enforced in this scenario):

```
Instead of pouring a fluid container into a liquid source:
    if the noun is empty, say "[The noun] is already empty." instead;
    now the current volume of the noun is 0.0 fl oz;
    say "You dump out [the noun] into [the second noun]."
```

A couple of minor refinements:

```
Swimming is an action applying to nothing. Understand "swim" or "dive" as
swimming.

Instead of swimming in the presence of a liquid source:
    say "You don't feel like a dip just now."

Before inserting something into a liquid source: say "[The noun] would get lost
and never be seen again." instead.

Part 3 - Scenario

The Lakeside is a room. The Lakeside swing is an enterable supporter in the
Lakeside. "Here you are by the lake, enjoying a summery view."

The glass is a fluid container carried by the player. The liquid of the glass is
absinthe. The current volume of the glass is 0.8 fl oz.

The pitcher is a fluid container in the Lakeside. The fluid capacity of the pitcher is
32.0 fl oz. The current volume of the pitcher is 20.0 fl oz. The liquid of the pitcher
is absinthe.

The lake is a liquid source. It is in the Lakeside.

The player wears a bathing outfit. The description of the bathing outfit is
"Stylishly striped in blue and white, and daringly cut to reveal almost all of your
calves, and quite a bit of upper arm, as well. You had a moral struggle,
purchasing it; but mercifully the lakeshore is sufficiently secluded that no one
can see you in this immodest apparel."

Instead of taking off the outfit: say "What odd ideas come into your head
sometimes!"
```

Test me with "fill glass / empty absinthe into lake / fill glass / swim / drink lake / drink / x water / x lake".

Chapter 18: Activities

Contents of *Writing with Inform*
Chapter 17: Understanding
Chapter 19: Rulebooks
Indexes of the examples

§18.1. What are activities?

It is poor form to define with negatives, but the first thing to say about activities is that they are *not* actions. This needs saying because Inform often seems to treat them as if they are, by allowing us to write rules like so:

```
Before printing the name of a woman, say "Ms ".
```

With this rule in place, someone called "Daphne" will always be described as "Ms Daphne", and so on. The language looks as if we were imposing a rule on an action called "printing the name of", but there is no such action: instead, it is an "activity". To spell out the difference:

```
An action is a simulated task for the fictional protagonist.
```

An activity is a real task for the computer program doing the simulation.

Activities allow us to influence or change some of the standard habits of Inform, using rules as flexible and powerful as those applicable to actions, though activities are in several ways simpler and easier.

Start of Chapter 18: Activities

Back to Chapter 17: Understanding: §17.22. Precedence

Onward to §18.2. How activities work

Example 334: **Ant-Sensitive Sunglasses** What are activities good for? Controlling output when we want the same action to be able to produce very flexible text depending on the state of the world -- in this case, making highly variable room description and object description text.

---

§18.2. How activities work

All activities start, continue for a while and then finish: however, no activity ever runs on for more than a single turn. Several activities can be going on at the same time. For instance, suppose the following is printed as part of the description of a grocery:

```
You can see a banana, an apple and a star-fruit here.
```

At the moment when Inform prints "apple", two activities are under way: "listing contents of the Grocery", and "printing the name of the apple". The sequence of events was in fact:

```
say "You can see "
start listing contents of the Grocery
    say "a "
    start printing the name of the banana
        say "banana"
    finish printing the name of the banana
    say ", an "
    start printing the name of the apple
        say "apple"
    finish printing the name of the apple
    say " and a "
    start printing the name of the star-fruit
        say "star-fruit"
    finish printing the name of the star-fruit
finish listing contents of the Grocery
say " here."
```

The golden rule is: if activity B starts during activity A, it must also finish during activity A.

If we ever need to find out, we can always test:

```
if the printing the name activity is going on, ...
if the printing the name activity is not going on, ...
```

but as we shall see, it's usually simpler to attach "while printing the name" provisos to rules.

Start of Chapter 18: Activities
Back to §18.1. What are activities?
Onward to §18.3. Rules applied to activities

§18.3. Rules applied to activities

The activity "printing the name of something" is the process of printing up the name of something on screen: ordinarily, this means saying the text in its "printed name" property.

As with actions, rules can be attached to activities which change or augment what would normally happen. In fact the situation is simpler, because (unlike an action) an activity almost always finishes, so we almost always do reach its "after" stage. There are also only three rulebooks attached to an activity, as compared with the six affecting an action.

The three rulebooks for printing the name are called "before printing the name", "for printing the name" and "after printing the name", and this is the general pattern. What happens is:

1. All "before printing the name of" rules are considered;
2. The most specific, applicable "rule for printing the name of" is considered;
3. All "after printing the name of" rules are considered.

Whereas an action's later stages never take place if an early stage ends unexpectedly, an activity always goes through all three of its stages. Invoking the word "instead" in a before rule for an action will terminate not only the before rules but the whole action: the same thing for an activity will only terminate the before rules, and the for and after rules will take place as usual.

The actual task is usually carried out by one single rule tucked into the back of the "for..." rulebook: it is the rule for printing the name of whatever is concerned, hence the name. Inform's standard activities are all of this pattern: they start out with no "before" or "after" rules, and just one "for" rule.

Why the part about an activity only "almost always" finishing? One reason is that the story might end during it; but another is that it's possible, though uncommon, to abandon an activity partway. Very few of the activities supplied with Inform ever do this, and those that do are noted in the sections which follow.

Start of Chapter 18: Activities
Back to §18.2. How activities work
Onward to §18.4. While clauses

§18.4. While clauses

Rules applied to actions can become baroque ("after going through a door in the presence of an animal when -" and so on and so forth), but activities are again simpler: they only have

one possible clause attached, which is called "while". For instance, the following would provide a fairly sledgehammer hint that the sack should not lightly be thrown away:

```
The sack is a player's holdall. The sack is carried. Rule for printing the name of the sack
while the sack is not carried: say "your abandoned sack".
```

Any condition can be given after the "while", and we can also specify that another activity has to be going on. Thus:

```
Rule for printing the name of the lemon sherbet while listing contents: say "curious sort
of lemon sherbet sweet".
```

This nicely distinguishes between contexts where it's appropriate to be more verbose, and where it isn't. Thus:

```
You can see a teaspoon and a curious sort of lemon sherbet sweet here.
> TAKE ALL
teaspoon: Taken.
lemon sherbet: Taken.
```


Start of Chapter 18: Activities


Back to §18.3. Rules applied to activities


Onward to §18.5. New activities

---

§18.5. New activities

Activities are all about influencing the standard mechanisms which Inform uses, so it might at first seem that there is no need to create new activities: but on further reflection, quite a lot of the writing of interactive fiction involves creating new and systematic ways to do things, and as soon as we have a general rule, we will want to have exceptions. Inform therefore allows us to create our own activities, giving us ways to influence the operation of our own mechanisms.

There are two kinds of activity: those which relate to a specific value (usually an object but not necessarily), and those which do not. Here are some examples of activities being created:

```
Assaying is an activity.
Analysing something is an activity.
Announcing something is an activity on numbers.
```

Inform looks for the clue "something" (or "of something") after the activity's name to see if it will work on a value: so analysing and announcing will do, but assaying won't. If we don't specify a kind, Inform assumes the value will be an object, as if we had written:

```
Analysing something is an activity on objects.
```

As always in Inform, the names of activities are themselves values.

```
"assaying activity" has kind activity on nothing
"analysing activity" has kind activity on objects
"announcing activity" has kind activity on numbers
```

Creating an activity is like creating an action: it automatically makes new rulebooks - "before analysing", "for analysing" and "after analysing" - but they start out empty, so the activity does nothing yet. Just as it does for rulebooks, Inform defines the adjectives "empty" and "non-empty" for activities to test this state:

```
if the analysing activity is empty, ...
```

will be true only when all three of its rulebooks are empty.

A newly created activity never happens unless we take steps to make it do so. We can make an activity happen at any time by writing phrases like so:

carry out the (activity) **activity**

This phrase carries out the given activity, which must be one not applying to any value. Example:

```
carry out the assaying activity;
```

carry out the (activity on values) **activity with** (value)

This phrase carries out the given activity, which must apply to a kind of value matching the one supplied. Example:

```
carry out the analysing activity with the pitchblende;
carry out the announcing activity with the score;
```

To make the activity do something useful, we need to put a rule into its "for" rulebook:

```
Rule for announcing a number (called N): say "Ladies and gentlemen, [N]."

The last for assaying rule:
    say "Professionally, you cast an eye around mineral deposits nearby, noticing [list of
rocks in the location]."
```

"The last" is a technicality about rulebooks (see the next chapter) which, put briefly, guarantees that this rule comes last among all possible "for assaying" rules. This is good form because the whole point of an activity is to make it easy for further rules to interfere - so we deliberately hang back to last place, giving precedence to anybody else who wants it.

The "for" rulebook is one where rules stop the activity, by default, when they take effect - in the same way that the "instead" rules stop actions by default. If this causes problems, we can use:

continue the activity

This phrase should be used only in rules in activity rulebooks. It causes the current rule to end, but without result, so that the activity continues rather than stopping as a result of the rule. This is useful for rulebooks (like the "for" rulebook of an activity) where the default is that a rule does stop the activity.

Activities are more useful than they first appear. Every new one provides a context which other activities can observe. We could, for instance, define

```
Rule for printing the name of a rock while assaying: ...
```

so that during assays more technical names are used.

Start of Chapter 18: Activities

Back to §18.4. While clauses

Onward to §18.6. Activity variables

Example 335: **AARP-Gnosis** An Encyclopedia set which treats volumes in the same place as a single object, but can also be split up.

Example 336: **Aftershock** Modifying the rules for examining a device so that all devices have some specific behavior when switched on, which is described at various times.

Example 337: **Crusoe** Adding a "printing the description of something" activity.

---

§18.6. Activity variables

Just as actions can have variables, which are created when the action starts and disappear when it finishes, so activities can also have variables. They are visible to the rules for that activity, and nowhere else. (If the activity should happen a second time within its first run, that second occurrence gets its own copy of the variable, leaving the original untouched.)

Typically it will be useful to set a variable to some default value at the "before" stage, calculate some interesting value for it in the "for" stage, and make use of the outcome during the "after" stage. For instance:

```
Analysing something is an activity. The analysing activity has a text called first
impression. Instead of examining something (called the sample), carry out the analysing
activity with the sample.

Before analysing: now the first impression is "unremarkable".

Rule for analysing someone: now the first impression is "living tissue".

After analysing something (called the sample):
    say "Your professional opinion of [the sample] is that it is
        [first impression]."
```

Start of Chapter 18: Activities
Back to §18.5. New activities
Onward to §18.7. Beginning and ending activities manually

§18.7. Beginning and ending activities manually

If we have declared a new activity, like "analysing", the normal way to make it happen would be to write

```
carry out the analysing activity with the pitchblende;
```

which goes through the whole machinery of rules - before, for, after - and then resumes, the activity having started, taken place and come to an end.

But there are times when it is not convenient to write a suitable "for ..." rule, or where we need more control, and do not wish to hand the whole business over to a single phrase. For such times we are allowed to write:

begin the (activity) **activity**

This phrase causes the named activity to become active, and runs its "before" rulebook. The activity must be one which applies to nothing. Example:

```
begin the assaying activity;
```

In all cases a matching "end the ... activity" or else "abandon the ... activity" phrase must be reached.

begin the (activity on values) **activity with** (value)

This phrase causes the named activity to become active, and runs its "before" rulebook. The activity must be one which applies to a value of a matching kind. Example:

```
begin the analysing activity with the pitchblende;
```

In all cases a matching "end the ... activity with ..." or else "abandon the ... activity with..." phrase must be reached.

And when we are done:

end the (activity) **activity**

This phrase runs the "after" rulebook of the activity and then causes it to become inactive. The activity must be one which applies to nothing. Example:

```
end the assaying activity;
```

This must only happen to match an earlier "begin the ... activity" phrase.

end the (activity on values) **activity with** (value)

This phrase runs the "after" rulebook of the activity and then causes it to become inactive. The activity must be one which applies to a value of a matching kind. Example:

```
end the analysing activity with the pitchblende;
```

This must only happen to match an earlier "begin the ... activity with..." phrase.

So the usual structure is like so:

```
begin the analysing activity with the pitchblende;
...
end the analysing activity with the pitchblende;
```

This time the activity is ongoing throughout as many phrases as we care to write between the "begin" and "end". The before rules are considered at the time of the "begin ..." phrase; the after rules at the "end ...".

What, then, of the "for" rules? In the above setup, they would simply be ignored. But we can make them effectual thus

```
begin the analysing activity with the pitchblende;
...
if handling the analysing activity with the pitchblende:
    ...
...
end the analysing activity with the pitchblende;
```

We place the activity's normal behaviour inside the "if"; the condition, "if handling...", is true only if no rule has intervened. This means that we (or other authors using our activity) can create their own for rules to substitute here. If we elsewhere write

```
Rule for handling the analysing activity with the pitchblende when the player is not
sober:
    say "You can't seem to focus."
```

that rule will intervene and take the place of whatever we have placed inside the condition.

if handling (activity) **activity:**

This should be used only where the given activity has been started with "begin ..." and will be finished with "end ...". It runs the "for" rules for the activity, and then comes out true if none of those for rules intervened in the handling of that activity. (The activity must be one which doesn't apply to any value.)

if handling (activity on values) **activity with** (value)**:**

This should be used only where the given activity has been started with "begin ..." and will be finished with "end ...". It runs the "for" rules for the activity, and then comes out true if none of those for rules intervened in the handling of that activity. (The given value must be the one it is being applied to.)

It is also legal to force an early end to an activity with:

abandon the (activity) **activity**

This phrase ends an activity at once (without consulting any further rulebooks, including its "after" rulebook). It can only be used with an activity which has had its "begin" but not yet its "end" phrase; it is a drastic remedy best taken only if it is clear that circumstances have changed so that the activity now seems inappropriate. It must not be used during one of the rules for the activity: it can only be used between the begin and for stages, or between the for and end stages.

```
abandon the assaying activity;
```

abandon the (activity on values) **activity with** (value)

This phrase ends an activity at once (without consulting any further rulebooks, including its "after" rulebook). It can only be used with an activity which has had its "begin" but not yet its "end" phrase; it is a drastic remedy best taken only if it is clear that circumstances have changed so that the activity now seems inappropriate. It must not be used during one of the rules for the activity: it can only be used between the begin and for stages, or between the for and end stages.

```
abandon the analysing activity with the pitchblende;
```

We need to follow three golden rules: all activities must end, they must never last longer than a turn, and if activity B starts during activity A then it must also finish during activity A. We must also be careful to make sure that if an activity applies to something, then it begins and ends with the same something (the pitchblende, in the above example).

Start of Chapter 18: Activities
Back to §18.6. Activity variables
Onward to §18.8. Introduction to the list of built-in activities

§18.8. Introduction to the list of built-in activities

Activities tend to be about process, rather than outcome. Many of the things Inform does - printing up lists of items, reading commands from the keyboard, and so on - are done as activities, because that way the process can be nudged a little. Too many works of interactive fiction betray their mechanical nature by making it visible that the general machinery being used does not quite seem natural for this or that situation. Activities enable us to add the many graceful touches which avoid that: which contribute nothing to a work, and also everything.

The rest of this chapter covers every activity built in to Inform, with one section for each. It is intended primarily for reference, but may be worth skimming through at a first reading, to give a sense of the possibilities.

Start of Chapter 18: Activities
Back to §18.7. Beginning and ending activities manually
Onward to §18.9. Deciding the concealed possessions of something

§18.9. Deciding the concealed possessions of something

1. When it happens. Frequently - whenever Inform needs to check whether something is visible or not. Nothing should be printed, and the activity needs to run quickly, so it should not (for instance) calculate best routes through complicated maps before getting an answer.

2. The default behaviour. There is no concealment. The ordinary rules still apply, though: the contents of a closed opaque container are invisible because there is a barrier in the way which cannot be seen through, even though nobody is "concealing" anything.

3. Examples. To repeat a number of brief examples given at the end of Chapter 3, where this activity made an early appearance:

```
Rule for deciding the concealed possessions of the Cloaked Villain: if the particular possession is the sable cloak, no; otherwise yes.

The coin is in the Roman Villa. The face and inscription are parts of the coin. Rule for deciding the concealed possessions of the coin: if the coin is carried, no; otherwise yes.
```

The value "particular possession" is the one whose concealment is in question, of course. We can ignore this if someone is invariably secretive:

```
Rule for deciding the concealed possessions of the furtive ghost: yes.
```

In general a rule for deciding the concealed possessions of something will decide "yes" if finishes without making a decision, but it's better style to write such a rule in such a way that it always makes a decision.

Start of Chapter 18: Activities
Back to §18.8. Introduction to the list of built-in activities
Onward to §18.10. Printing the name of something
Example 338: **Hays Code** Clark Gable in a pin-striped suit and a pink thong.

---

§18.10. Printing the name of something

1. When it happens. Whenever the name of a thing or room is printed.

2. The default behaviour. For items other than the current player, the "printed name" property is printed out; but for the current player, "you" or "yourself" is printed. (That doesn't necessarily mean that the "printed name" of the player is never used. Suppose there are two people, Alice and Bob, and the narrative switches between them: when Alice is the player, she appears as "yourself" but Bob is "Bob"; but when Bob is the player, he is "yourself" and Alice is "Alice".)

3. Examples. (a) A pen which is described differently in inventories:

```
Rule for printing the name of the pen while taking inventory: say "useful pen".
```

"Taking inventory" is a condition which is true if that's the current action and not otherwise, so the effect is that the pen is called "a useful pen" only in inventory listings. "While looking" is a similarly useful one.

(b) Italicising the names of novels:

```
A novel is a kind of thing. Dr Zhivago and Persuasion are novels. Before printing the
name of a novel, say "[italic type]". After printing the name of a novel, say "[roman type]".
```

(c) Telling the time:

```
After printing the name of the wrist watch while taking inventory: say " (time: [the time of
day])".
```

(d) Merging containers with their contents:

```
Rule for printing the name of the bottle while not inserting or removing:
    if the bottle contains sand, say "bottle of sand";
    otherwise say "empty bottle";
    omit contents in listing.
```

This example makes use of a special phrase:

omit contents in listing

This phrase changes the form of an inventory listing, room description, etc., so that it will simply list "a bottle of sand" or "an empty bottle", rather than "a bottle (in which is sand)" or "a bottle (which is empty)". It should be used only when the listing is imminent, and does not have permanent effect.

The clause about not inserting or removing is to prevent messages like "You put the sand in the bottle of sand.", where it's confusing to refer to the bottle as anything other than "the bottle".

Start of Chapter 18: Activities

Back to §18.9. Deciding the concealed possessions of something

Onward to §18.11. Printing the plural name of something

Example 339: ★ **Shipping Trunk** A box of baking soda whose name changes to "completely ineffective baking soda" when it is in a container with something that smells funny.

Example 340: ★★ **Trachypachidae Maturin 1803** Bottles with removable stoppers: when the stopper is in the bottle, the bottle is functionally closed, but the stopper can also be removed and used elsewhere. Descriptions of the bottle reflect its state intelligently.

Example 341: ★★★★ **Chronic Hinting Syndrome** Using name-printing rules to keep track of whether the player knows about objects, and also to highlight things he might want to follow up.

---

§18.11. Printing the plural name of something

1. When it happens. Only when a group of identical items is present in the same place, and are being described jointly with text like "You can see five gold rings here." The activity happens after "five" and before "here." (See the activity "printing a number of something" if the whole phrase needs to be altered.)

2. The default behaviour. The plural name - in this case "gold rings" - is printed out.

3. Examples. (a) Suppose we want to emphasise how nice it is to have more than one gold ring:

```
Rule for printing the plural name of a gold ring: say "gleaming gold rings".
```

(b) If the number needs changing as well, it's necessary to use the "printing a number of something" activity instead.

Start of Chapter 18: Activities

Back to §18.10. Printing the name of something

Onward to §18.12. Printing a number of something

Example 342: **Hudsucker Industries** Letters which are described differently as a group, depending on whether the player has read none, some, or all of them, and on whether they are alike or unlike.

---

§18.12. Printing a number of something

1. When it happens. Only when a group of identical items is present in the same place, and are being described jointly with text like "You can see five gold rings here." The activity prints the "five gold rings" part. The variable "listing group size" contains the number, which in this example would be 5, and is always at least 2.

2. The default behaviour. The number of items is printed, in words ("five") and then the "printing the plural name" activity is run ("gold rings").

3. Examples. (a) Using this activity is for perfectionists, because the normal behaviour is almost always fine. Still:

```
Rule for printing a number of blocks when the listing group size is 3: say "all three blocks".
```

(b) Or perhaps:

```
Rule for printing a number of ants: say "altogether [listing group size in words] ants".
```

(c) If the only part needing variation is the plural name, it's simpler and tidier to use the "printing the plural name of something" activity instead.

Start of Chapter 18: Activities

Back to §18.11. Printing the plural name of something

Onward to §18.13. Listing contents of something

Example 343: **Prolegomena** Replacing precise numbers with "some" or other quantifiers when too many objects are clustered together for the player to count at a glance.

---

§18.13. Listing contents of something

1. When it happens. When taking inventory, the list is produced by the activity "listing contents of yourself"; when looking, a list of items which do not deserve their own paragraphs is produced by "listing contents of" the location.

And when it doesn't happen. (a) If the Storage Room contains a sideboard and an open shoe box, then "listing contents of the Storage Room" is used to produce the part of the room

description mentioning sideboard and box. But if the box in turn contains a pair of brogues, then "listing contents of the shoe box" is not used to say that part. So this works:

```
Rule for printing the name of the brogues while listing contents of a room: ...
```

But this won't affect room descriptions:

```
Rule for printing the name of the brogues while listing contents of the shoe box: ...
```

(b) The activity also doesn't happen when, for instance, "[a list of animals]" is printed, because that isn't a list of the contents of any room or location.

2. The default behaviour. The list is printed out.

3. Examples. (a) We have already seen that it can be elegant to elaborate on a description in the context of a list. Here we add "discarded" to a sweet wrapper which is found on the ground.

```
Rule for printing the name of the wrapper while listing contents of a room: say
"discarded sweet wrapper".
```

(b) Lists can be considerably shortened and tidied up if similar items are grouped together. We do this by specifying what should be grouped together before listing contents, using the special phrase "group ... together":

```
Utensil is a kind of thing. The knife, the fork and the spoon are utensils. Before listing
contents: group utensils together as "utensils".
```

The result will be, say, "two utensils (knife and spoon)", if both are found in the same place.

(c) We can less obtrusively group items together like so:

```
Before listing contents while taking inventory: group utensils together.
```

Three special phrases exist for this kind of list organisation:

group (description of objects) **together**

This phrase causes the objects described to be listed together in a single item as part of an inventory or room description. The effect is temporary, and the phrase should only be used when this list is imminent. Example:

```
Utensil is a kind of thing. The knife, the fork and the spoon are utensils. Before
listing contents: group utensils together.
```

This might produce the list item "fork and spoon".

group (description of objects) **together giving articles**

This phrase causes the objects described to be listed together in a single item as part of an inventory or room description, but giving each individual item its indefinite article. The effect is temporary, and the phrase should only be used when this list is imminent. Example:

> Utensil is a kind of thing. The knife, the fork and the spoon are utensils. Before listing contents: group utensils together giving articles.

This might produce the list item "a fork and a spoon".

group (description of objects) **together as** (text)

This phrase causes the objects described to be listed together in a single item as part of an inventory or room description, summarised with the given text. The effect is temporary, and the phrase should only be used when this list is imminent. Example:

> Utensil is a kind of thing. The knife, the fork and the spoon are utensils. Before listing contents: group utensils together as "utensils".

This might produce the list item "two utensils (fork and spoon)".

Start of Chapter 18: Activities

Back to §18.12. Printing a number of something

Onward to §18.14. Grouping together something

Example 344: **Unpeeled** Calling an onion "a single yellow onion" when (and only when) it is being listed as the sole content of a room or container.

---

§18.14. Grouping together something

1. When it happens. Only while listing contents, and only when a collection of items to be grouped together is reached. This in turn happens only if a "before listing contents" rule has chosen it (see previous section). The first item in the group is the one to which the activity formally applies. The variable "listing group size" gives the number of items grouped together in this way.

2. The default behaviour. The items grouped together are printed in an English phrase, such as "egg, chicken and farmer". In particular, they are not split onto separate lines even if the rest of the list is. (See previous section.)

3. Examples. (a) Here are Scrabble pieces which are described as "the tile W from a Scrabble set" or similar outside of lists, but which, when they turn up together in lists, are rolled together into "the tiles A, B and D from a Scrabble set".

```
A Scrabble piece is a kind of thing. The X, the Y and the Z are Scrabble pieces.

Before listing contents: group Scrabble pieces together.

Before printing the name of a Scrabble piece while not grouping together, say "tile ".
After printing the name of a Scrabble piece while not grouping together, say " from a
Scrabble set".

Before grouping together Scrabble pieces, say "the tiles ". After grouping together
Scrabble pieces, say " from a Scrabble set".
```

(b) Maybe we only want an abbreviated form when there are five or more tiles in one place:

```
A Scrabble piece is a kind of thing. The X, the W, the F, the Y and the Z are Scrabble
pieces in the Lounge.

Before listing contents: group Scrabble pieces together.

Before grouping together Scrabble pieces when the listing group size is greater than 4:
    say "some [listing group size in words] tiles (".
After grouping together Scrabble pieces when the listing group size is greater than 4:
    say ") from a Scrabble set".
```

(c) We can throw out all pretence at listing and say whatever we like, in fact:

```
Before listing contents while taking inventory: group utensils together. Rule for grouping
together utensils: say "the usual utensils".
```


Start of Chapter 18: Activities


Back to §18.13. Listing contents of something


Onward to §18.15. Issuing the response text of something

---

§18.15. Issuing the response text of something

1. When it happens. When Inform prints a text marked with a response letter (A), (B), (C), ..., in a rule making use of them. For example, in this rule:

```
Carry out taking inventory (this is the print empty inventory rule):
    if the first thing held by the player is nothing,
        say "[We] [are] carrying nothing." (A) instead.
```

Or, less directly,

```
let R be the print empty inventory rule response (A);
say "To be frank: [text of R].";
```

2. The default behaviour. To print the current textual value of the response, making any substitutions in the ordinary way.

3. Examples. This activity is not the best way to amend responses or make them dynamic; the whole idea of responses is that they can be changed just as if they were text variables. This activity should be used only if it's important to amend blocks of responses in some systematic way.

(a) With that said, some interesting effects can be achieved. This is a way to see which responses are being printed, for example:

```
Before issuing the response text of a response (called R): say "[R]: ".
```

whence:

```
>WAIT
standard report waiting rule response (A): Time passes.
```

(b) And this intercepts the activity in order to re-run it in each of the six viewpoints. (Note the way a variable is used to prevent the rule from applying to all of those re-runs as well.)

```
The response inhibition is initially false.

Rule for issuing the response text of a response (called R) when the response inhibition
is false:
    now the response inhibition is true;
    let the current viewpoint be the story viewpoint;
    repeat with P running through narrative viewpoints:
        now the story viewpoint is P;
        say "[P]: [text of R][command clarification break]";
    now the story viewpoint is the current viewpoint;
    now the response inhibition is false.
```

With that in place,

```
>EAST
first person singular: I can't go that way.
second person singular: You can't go that way.
third person singular: He can't go that way.
first person plural: We can't go that way.
second person plural: You can't go that way.
third person plural: They can't go that way.
```

Start of Chapter 18: Activities

Back to §18.14. Grouping together something

Onward to §18.16. Printing room description details of something

Example 345: ★ **Wesponses** Parser messages that are delivered with a speech impediment.

---

§18.16. Printing room description details of something

1. When it happens. When an item is listed in the miscellaneous collection of items present in a room (the ones which do not deserve their own paragraphs): this is normally the last paragraph of a room description.

2. The default behaviour. A bracketed piece of extra information is added for certain items such as containers:

```
You can also see Po and a cage (empty) here.
```

The " (empty)" (note initial space) was added by this activity. (Note that this activity is not responsible for describing further items visible because of the item in question: that is, it does not print the text such as "(in which is a notepad)" which would appear if there were contents. If we want to remove such text, we should use "omit contents in listing": see the activity "for printing the name of something".)

3. Examples. (a) To get rid of such addenda entirely, try:

```
Rule for printing room description details: stop.
```

(b) To add a new form of addendum:

```
Rule for printing room description details of a person:
    say " (at last, someone to talk to)" instead.
```

If both examples (a) and (b) are in place at once, we might now read:

```
You can also see Po (at last, someone to talk to) and a cage here.
```

Start of Chapter 18: Activities

Back to §18.15. Issuing the response text of something

Onward to §18.17. Printing inventory details of something

Example 346: **Rules of Attraction** A magnet which picks up nearby metal objects, and describes itself appropriately in room descriptions and inventory listings, but otherwise goes by its ordinary name.

---

§18.17. Printing inventory details of something

1. When it happens. When an item is listed in an inventory of items carried by the player.

2. The default behaviour. A bracketed piece of extra information is added for certain items such as containers:

```
a flaming branch (providing light)
```

The " (providing light)" (note initial space) was added by this activity.

3. Examples. (a) To get rid of such addenda entirely, try:

```
Rule for printing inventory details: stop.
```

(b) To add a new form of addendum:

```
Rule for printing inventory details of something edible:
    say " (yummy!)[run paragraph on]".
```

Start of Chapter 18: Activities

Back to §18.16. Printing room description details of something

Onward to §18.18. Printing a refusal to act in the dark

---

§18.18. Printing a refusal to act in the dark

1. When it happens. When an action which requires light is tried, and the visibility rules decide that not enough light is present.

2. The default behaviour. To print "It is pitch dark, and you can't see a thing."

3. Examples. (a) This might do for some twilit, penumbral room:

```
Rule for printing a refusal to act in the dark: if we are examining something, say "It's not totally dark here, perhaps, but certainly too dim for close-up examination of anything." instead.
```

Start of Chapter 18: Activities

Back to §18.17. Printing inventory details of something

Onward to §18.19. Printing the announcement of darkness

Example 347:  **Zorn of Zorna** Light levels vary depending on the number of candles the player has lit, and this determines whether or not he is able to examine detailed objects successfully.

---

§18.19. Printing the announcement of darkness

1. When it happens. Inform frequently calculates to see if the player is in light or darkness: this activity happens on the change from light to darkness.

2. The default behaviour. To print "It is now pitch dark in here!".

3. Examples. (a) The most obvious use is to change the text:

```
Rule for printing the announcement of darkness: say "Ooh-er! It's now very nearly pitch dark in here." instead.
```

(b) But we could also use this activity for sneakier purposes, silently moving things around:

```
Before printing the announcement of darkness: now all of the gremlins are in the
kitchen.
```

(c) A special description for occasions when the player has climbed into a container and shut it (so that the darkness is the result of his own actions, rather than some external circumstance):

```
Rule for printing the announcement of darkness when closing a container which
contains the player:
    say "Congratulations: now you can't see a thing." instead.
```


Start of Chapter 18: Activities


Back to §18.18. Printing a refusal to act in the dark


Onward to §18.20. Printing the announcement of light

---

§18.20. Printing the announcement of light

1. When it happens. Inform frequently calculates to see if the player is in light or darkness: this activity happens on the change from darkness to light.

2. The default behaviour. To try the looking action, which usually prints up a room description.

3. Examples. (a) Perhaps the player is initially too disoriented to look around in any coherent way:

```
Rule for printing the announcement of light in the Dazzling Temple: say "You are almost
blinded by the suffusion of white light, and have spots before your eyes." instead.
```


Start of Chapter 18: Activities


Back to §18.19. Printing the announcement of darkness


Onward to §18.21. Printing the name of a dark room

---

§18.21. Printing the name of a dark room

1. When it happens. When looking in darkness, or writing the (default) status line in darkness.

2. The default behaviour. To print "Darkness".

3. Examples. (a) One might modify the darkness with some adjective:

```
Before printing the name of a dark room, say "Near ".
```

(Note that this activity does not come in different forms for different dark rooms: the wording is fixed at "printing the name of a dark room", and we are not allowed to substitute particular dark rooms or assign a "(called ...)" onto the mention of the dark room.)

Start of Chapter 18: Activities

Back to §18.20. Printing the announcement of light

Onward to §18.22. Printing the description of a dark room

---

§18.22. Printing the description of a dark room

1. When it happens. When looking in darkness.

2. The default behaviour. To print "It is pitch dark, and you can't see a thing."

3. Examples. (a) A simple variation of wording:

```
Rule for printing the description of a dark room: say "Your eyes can barely make
anything out." instead.
```

(b) More stylishly,

```
Rule for printing the description of a dark room: try listening instead.
```

which produces, for instance,

```
Darkness
You hear nothing unexpected.
```

(Note that this activity does not come in different forms for different dark rooms: the wording is fixed at "printing the description of a dark room", and we are not allowed to substitute particular dark rooms or assign a "(called ...)" onto the mention of the dark room.)

Start of Chapter 18: Activities

Back to §18.21. Printing the name of a dark room

Onward to §18.23. Constructing the status line

Example 348:  **Hohmann Transfer** Changing the way dark rooms are described to avoid the standard Inform phrasing.

Example 349:  **Four Stars 1** An elaboration of the idea that when light is absent, the player should be given a description of what he can smell and hear, instead.

---

§18.23. Constructing the status line

1. When it happens. Just before input is accepted from the keyboard, Inform constructs a "status line" at the top of the window which is normally displayed in reverse colours (white

on black instead of black on white, say).

2. The default behaviour. Makes the status line up out of two pieces, the "left hand status line" and the "right hand status line". Since these can freely be changed, note that the status line is already very customisable without using rules applied to this activity.

3. Examples. (a) The most useful thing about this activity is that it allows us to vary descriptions in the status line. This is especially helpful to abbreviate unduly long room names, which might not otherwise fit:

```
The Temple Of A Thousand Mightily Peeved Deities is a room. Rule for printing the
name of the Temple while constructing the status line: say "Temple".
```

(b) Again, it's usually not necessary to apply activity rules to this, but occasionally amusing effects are possible if we do:

```
The blindfold is wearable and carried. Rule for constructing the status line while the
blindfold is worn: do nothing.
```


Start of Chapter 18: Activities


Back to §18.22. Printing the description of a dark room


Onward to §18.24. Writing a paragraph about


Example 350:  **Ways Out** A status line that lists the available exits from the current location.


Example 351:  **Guided Tour** A status line that lists the available exits from the current location, changing the names of these exits depending on whether the room has been visited or not.

---

§18.24. Writing a paragraph about

1. When it happens. Just *before* writing a paragraph about some item in a room description.

2. The default behaviour. Is to do nothing. However, if a rule is supplied which prints something up, then this replaces the paragraph which would otherwise have been printed. Moreover, any items whose names are said in the course of this rule - for instance, by being listed - are then excluded from the remainder of the room description, because they are considered as having been described sufficiently already.

Warning: because we often want a "for" rule for this activity to make some calculation and then possibly choose to do nothing (see the example "Otranto"), Inform suppresses the usual paragraph not when a "for" rule took effect but when it detected a paragraph having been printed. This can get confused if a text substitution affecting paragraph breaks, say "[line break]", is within the final "say" of a "for writing a paragraph about" rule.

3. Examples. (a) This is a neat way to wrap several things together into the same paragraph:

```
Rule for writing a paragraph about Mr Wickham:
    say "Mr Wickham looks speculatively at [list of women in the location]."
```

because now "Mr Wickham looks speculatively at Velma and Daphne" will now prevent the appearance of the subsequent text "You can also see Velma and Daphne."

Inform keeps track of which objects have already been named with an either/or property called "mentioned", which it assigns whenever the name of an object has been automatically printed. So in this case, Velma and Daphne are now mentioned. Note "automatically printed", though: if the text printed had just been "Mr Wickham looks speculatively at Velma and Daphne", rather than the text-substitution list used above, then Inform would not know that Velma and Daphne have been described.

If we ever need to override this - say, we want to list all the women but make sure that Velma gets another paragraph anyway - we could change Velma to unmentioned again after the listing.


Start of Chapter 18: Activities


Back to §18.23. Constructing the status line


Onward to §18.25. Listing nondescript items of something


Example 352:  **Reflections** Emphasizing the reflective quality of shiny objects whenever they are described in the presence of the torch.


Example 353:  **Emma** Social dynamics in which groups of people form and circulate during a party.


Example 354:  **Air Conditioning is Standard** Uses "writing a paragraph about" to make person and object descriptions that vary considerably depending on what else is going on in the room, including some randomized NPC interactions with objects or with each other.

---

§18.25. Listing nondescript items of something

1. When it happens. This activity prints up the also-ran paragraph at the end of a room description. These are nondescript items because they don't merit paragraphs of their own: if, as sometimes happens, there are none in the room, then no such paragraph is printed and this activity does not happen. (So to add a further paragraph to a room description, a simpler "after looking" rule should be used, not an "after listing nondescript items" rule.)

2. The default behaviour. The paragraph ordinarily reads as "You can also see a cask and a clock." or similar. Before the activity begins, those objects which are nondescript - in this case the cask and the clock - are given the property of being "marked for listing".

If it turns out that nothing is marked for listing, because of before rules like the one in the example below, then nothing is printed and **the activity is abandoned**, so that the rules for and after are never reached.

3. Examples. (a) Promoting something out of the nondescript category, by unmarking it.

```
Before listing nondescript items:
    if the watch is marked for listing:
        say "The watch catches your eye.";
        now the watch is not marked for listing.
```

(b) Changing the normal phrasing of the paragraph. Note that we can also change the listing style; the one below is the default.

```
Rule for listing nondescript items of the Distressingly Messy Room:
    say "Strewn carelessly on the floor";
    list the contents of the Distressingly Messy Room, as a sentence,
        tersely, listing marked items only, prefacing with is/are,
        including contents and giving brief inventory information;
    say "."
```

Start of Chapter 18: Activities

Back to §18.24. Writing a paragraph about

Onward to §18.26. Printing the locale description of something

Example 355: **Rip Van Winkle** A simple way to allow objects in certain places to be described in the room description body text rather than in paragraphs following the room description.

Example 356: **Happy Hour** Listing visible characters as a group, then giving some followup details in the same paragraph about specific ones.

Example 357: **The Eye of the Idol** A systematic way to allow objects in certain places to be described in the room description body text rather than in paragraphs following the room description, and to control whether supporters list their contents or not.

---

§18.26. Printing the locale description of something

1. When it happens. A "locale description" is Inform jargon for the part of a room description which catalogues the visible items in the room. When looking, Inform will normally print the description of the room itself, followed by a locale description for the room. But if the player is in a cage in the room, there will be two locale descriptions: one for the room, then another for the cage. This activity is used to write the locale description for a single domain, and the "something" can be either a room, an enterable container, or an enterable supporter.

2. The default behaviour. Is quite complicated, and is written up in full in the typeset form of the Standard Rules downloadable from the Inform website. Briefly, though: we first run the "choosing notable locale objects" activity to find out what ought to be mentioned here. That assembles a list of things to mention, sorted into priority order. Items with priority 1 go first, then those with priority 2, and so on. The "printing a locale paragraph" activity is run for each, and in practice that usually hands the job over to "writing a paragraph about". Sometimes a paragraph will indeed be written, but not always. Sometimes there is nothing interesting to say, and an item is left until a final, single paragraph which gathers up the leftovers ("You can also see a scarlet fish, a harmonium and a kite here."), the printing of which is done by the "listing nondescript items of" activity. As soon as any item picks up the either/or property "mentioned", by having its name printed, it is struck out so that it will not appear subsequently, whatever its priority.

3. Examples. As general advice: if the effect wanted can be got using "writing a paragraph about" and "listing nondescript items of" alone, use those; if it's necessary to meddle further, use "choosing notable locale objects" and "printing a locale paragraph" to alter the normal

processes; use the all-powerful "printing the locale description" activity only when the whole process needs to be altered, not the item-by-item workings.

(a) In the Very Misty Moorlands, nothing on the ground can ordinarily be seen through the swirling mist, so the locale description is suppressed entirely:

```
Rule for printing the locale description of the Very Misty Moorlands:
    say "Mist coils around your feet, thick as a blanket. You cannot even see the ground
you walk upon." instead.

Report taking something in the Very Misty Moorlands:
    say "You grope blindly in the mist and pick up [the noun]." instead.
```

(b) Here we take the chance to insert an additional paragraph into the locale description. This does relate to an item which might be described later, but where the player doesn't know that:

```
The Horological Workshop is a room. The marble table is fixed in place in the Workshop.

The parcel is a closed opaque container on the marble table. The alarm clock is a
device in the parcel. The alarm clock is switched on.

Before printing the locale description of a room (called the locale):
    if the locale encloses the alarm clock and the alarm clock is switched on, say "A faint
ticking noise can be heard."
```

Start of Chapter 18: Activities

Back to §18.25. Listing nondescript items of something

Onward to §18.27. Choosing notable locale objects for something

Example 358: **Priority Lab** A debugging rule useful for checking the priorities of objects about to be listed.

---

§18.27. Choosing notable locale objects for something

1. When it happens. See "printing the locale description". This activity is expected to decide which items ought to be mentioned in a locale description for a given room, enterable container or enterable supporter, and to give each item a priority, which is a number ranging upwards from 1 (which is the top priority). The lower the priority number, the earlier the mention, or at least, the earlier the opportunity to be mentioned: it's up to other activities whether to give it a paragraph of its own or not. This activity only makes something a candidate, and decides what order the candidates will be tried in.

2. The default behaviour. By default, this activity contains only the "standard notable locale objects rule". This chooses exactly those items directly contained by the locale, assigning all of them priority 5. Note that this includes scenery, and other probably unwanted items - those will be excluded later.

3. Examples. (a) In the Misty Moorlands, only large items on the ground are visible through the mist:

```
A thing can be large or small. A thing is usually small. The stepladder is a large thing in
the Misty Moorlands.

Rule for choosing notable locale objects for the Misty Moorlands:
    repeat with item running through large things in the Misty Moorlands:
        set the locale priority of the item to 5.

Report taking a small thing in the Misty Moorlands:
    say "You grope blindly in the mist and pick up [the noun]." instead.
```

Note the special phrase

```
set the locale priority of the item to 5;
```

which should be used only in rules for locale activities. It makes the given item a candidate and sets its priority. (Setting the priority to 0 forces an item not to be a candidate, and can thus undo the effect of previous rules.)

It's best to avoid situations where an item has a locale priority which is higher than that of something it is on top of, or inside, since this can result in an oddly-worded description.


Start of Chapter 18: Activities


Back to §18.26. Printing the locale description of something


Onward to §18.28. Printing a locale paragraph about


Example 359:  **Low Light** An object that is only visible and manipulable when a bright light fixture is on.


Example 360:  **Casino Banale** Creating room descriptions and object descriptions that change as the player learns new facts and pieces things together.

---

§18.28. Printing a locale paragraph about

1. When it happens. See "printing the locale description". By this point, the locale description process has identified a number of items as candidates to be described, and worked out a priority order. This activity is then called for each candidate in turn, starting with the highest priority items and working downwards. It can either print some text or not, and can either mark the item as "mentioned" or not: if it does, then the item won't appear subsequently in the locale description. If the activity does nothing, the item becomes "nondescript" and falls through into the final "You can also see..." paragraph, unless another rule mentions it in the mean time.

2. The default behaviour. Is provided by a sequence of seven rules:

```
(1) The "don't mention player's supporter in room descriptions rule" excludes anything
the player is directly or indirectly standing on or, less frequently, in. The header of the
```

room description has probably already said something like "Boudoir (on the four-poster bed)", so the player can't be unaware of this item.

(2) The "don't mention scenery in room descriptions rule" excludes scenery.

(3) The "don't mention undescribed items in room descriptions rule" excludes the player object. (It's redundant to say "You can also see yourself here.") At present nothing else in I7 is "undescribed" in this sense.

(4) The "set pronouns from items in room descriptions rule" adjusts the meaning of pronouns like IT and HER to pick up items mentioned. Thus if a room description ends "Mme Tourmalet glares at you.", then HER would be adjusted to mean Mme Tourmalet.

(5) The "offer items to writing a paragraph about rule" gives the "writing a paragraph about" activity a chance to intervene. We detect whether it does intervene or not by looking to see if it has printed any text.

(6) The "use initial appearance in room descriptions rule" prints the "initial appearance" property of an item which has never been handled as a paragraph, if it has one.

(7) The "describe what's on scenery supporters in room descriptions rule" is somewhat controversial. It prints text such as "On the mantelpiece is a piece of chalk." for items which, like the mantelpiece, are scenery mentioned - we assume - in the main room description. (It is assumed that scenery supporters make their contents more prominently visible than scenery containers, which we do not announce the contents of.)

3. Examples. If all that's required is to supply an interesting paragraph of room description about something then it's always better to use the "writing a paragraph about" activity, not this one. This activity should only be used when the mechanism itself needs to be adjusted.

(a) The following excludes doors from room descriptions:

```
For printing a locale paragraph about a door (called the item)
    (this is the don't mention doors in room descriptions rule):
    set the locale priority of the item to 0;
    continue the activity.
```

(It's usually a good idea to "continue the activity" at the end of rules for this activity, since usually they all need to take effect for a happy outcome to the process. Here it doesn't really matter, since we were trying to stop anything from happening about the door, but it doesn't do any harm either.)

(b) Here's how to abolish what may be the most contentious rule in the whole Standard Rules:

```
The describe what's on scenery supporters in room descriptions rule is not listed in any
rulebook.
```

Start of Chapter 18: Activities

Back to §18.27. Choosing notable locale objects for something

Onward to §18.29. Deciding the scope of something

Example 361: ★ **Kiwi** Creating a raised supporter kind whose contents the player can't see or take from the ground.

Example 362: ★★★ **Copper River** Manipulating room descriptions so that only interesting items are mentioned, while objects that are present but not currently useful to the player are ignored.

---

§18.29. Deciding the scope of something

1. When it happens. "Scope" is a term of art in interactive fiction programming: it means the collection of things which can be interacted with at any given moment, which depends on who you are and where you are. Commands typed by the player will only be allowed to go forward into actions if the things they refer to are "in scope". Inform also needs to determine scope at other times, too: for instance, when deciding whether a rule conditional on being "in the presence of" something is valid. It is a bad idea to say anything during this activity.

2. The default behaviour. Is complicated: see the Inform Designer's Manual, 4th edition, page 227. Briefly, the scope for someone consists of everything in the same place as them, unless it is dark.

3. Examples. (a) We very rarely want to forbid the player to refer to things close at hand, but often want to allow references to distant ones. For instance, a mirage of something which is not present at all:

```
After deciding the scope of the player while in the Shrine: place the holy grail in scope.
```

Two different phrases enable us to place unusual items in scope:

place (object) **in scope**

This phrase should only be used in rules for the "deciding the scope of..." activity. It places the given object in scope, making it accessible to the player's commands, regardless of where it is in the model world. Examples:

```
place the distant volcano in scope;
place the lacquered box in scope, but not its contents;
```

Ordinarily if something is placed in scope, then so are its parts and (in the case of a supporter or a transparent or open container) its contents; using the "but not its contents" option we can place just the box itself in scope.

place the/-- contents of (object) **in scope**

This phrase should only be used in rules for the "deciding the scope of..." activity. It places the things inside or on top of the given object in scope, making them accessible to the player's commands, but it does nothing to place the object itself in scope. (It might of course be in scope anyway, and if it is then this phrase won't remove it.) Example:

```
place the contents of the lacquered box in scope;
place the contents of the Marbled Steps in scope;
```

Note that the object in question can be a room, as in this second example.

(b) Another useful device is to be able to see, but not touch, another room:

```
The Cloakroom is a room. "This is just a cloakroom, but through a vague, misty mirror-window you can make out the Beyond." After looking in the Cloakroom, say "In the mirror you can see [list of things in the Beyond]."

After deciding the scope of the player while in the Cloakroom: place the Beyond in scope.

The Beyond is a room. Johnny Depp is a man in the Beyond.
```

(This must, however, also be a mirage, as at time of writing Mr Depp is alive and as well as can be expected following the reviews of "Charlie and the Chocolate Factory".) Note that "place the Ballroom in scope" doesn't just allow the player to talk about the dancers, the chamber musicians and so forth, also allows, say, "EXAMINE BALLROOM". To get one but not the other, use "place the contents of the Ballroom in scope" or "place the Ballroom in scope, but not its contents".

(c) In darkness, the scope of someone is ordinarily restricted to his or her possessions (and body), but we can override that:

```
After deciding the scope of the player while in darkness: place the location in scope.
```

4. A note about actions. This activity takes place during the process of understanding the player's command, when the action that will take place is not fully known. So if the player types "TAKE SHOEBOX", this activity would happen when SHOEBOX is being examined for meaning. Inform knows the action it would be taking if the current line of command grammar were to be accepted, but it does not yet know to what objects that command would be applied. That means attaching a proviso like "... while taking a container" to a rule for this activity will cause the rule to have no effect - whereas "... while taking" would be fine.

Start of Chapter 18: Activities

Back to §18.28. Printing a locale paragraph about

Onward to §18.30. Clarifying the parser's choice of something

Example 363: ★ **Peeled** Two different approaches to adjusting what the player can interact with, compared.

Example 364: ★ **Four Stars 2** Using "deciding the scope" to change the content of lists such as "the list of audible things which can be touched by the player".

Example 365: ★★ **Ginger Beer** A portable magic telescope which allows the player to view items in another room of his choice.

Example 366: ★★ **Rock Garden** A simple open landscape where the player can see between rooms and will automatically move to touch things in distant rooms.

Example 367: ★★★ **Stately Gardens** An open landscape where the player can see landmarks in nearby areas, with somewhat more complex room descriptions than the previous example, and in which we also account for size differences between things seen at a distance.

---

§18.30. Clarifying the parser's choice of something

1. When it happens. When the player has typed an ambiguous noun reference, and Inform has made a decision about what was meant, and it matters what this decision is. (If the decision is between three identical gold coins, say, then it doesn't matter, and this activity does not take place.) There are a couple of limitations on this: the activity applies only to the first noun, and only if it's an object. So for a command like SELECT BLUE, where BLUE is a noun referring to a colour value, say, this activity isn't used. But the simple case where the activity does play a part is nevertheless very useful.

2. The default behaviour. Text in brackets such as "(the laminated mahogany box)" is printed, on its own line.

3. Examples. (a) In the following, asking to TAKE TOWER results in the parser choosing the souvenir model (because of the "does the player mean..." rule making the alternative unlikely), and then explaining itself by saying "(The little one, obviously.)" instead of "(the souvenir model Eiffel Tower)".

```
The Champs du Mars is a room. The great Eiffel Tower is here. "The great Tower stands high over you." The souvenir model Eiffel Tower is here. "Comparatively tiny is the souvenir version." The great Eiffel Tower is fixed in place. Does the player mean taking the great Eiffel Tower: it is very unlikely.

Rule for clarifying the parser's choice of the model tower: say "(The little one, obviously.)"
```

4. A note about actions. This activity takes place during the process of understanding the player's command, when the action that will take place is not fully known. So if the player types "TAKE SHOEBOX", this activity would happen when SHOEBOX is being examined for meaning. Inform knows that the action will be taking, but nothing else. That means attaching a proviso like "... while taking a container" to a rule for this activity will cause the rule to have no effect - whereas "... while taking" would be fine.

Start of Chapter 18: Activities
Back to §18.29. Deciding the scope of something
Onward to §18.31. Asking which do you mean

---

§18.31. Asking which do you mean

1. When it happens. When the player has typed an ambiguous noun reference, and Inform has not been able to decide what was meant.

2. The default behaviour. A question such as "Which do you mean, the laminated mahogany box or the boom box?" is printed. (This activity shapes the question: it is not responsible for parsing the answer. It would be very mysterious to write rules for this activity such that nothing is printed, because the player would then have no idea what to type.)

3. Examples. The question is harder to print than may first appear, since one must not simply list the options, but also take into account collections of plural objects ("Which do you mean, the gold-tipped pen or a gold coin?"). It is probably better not to try to rewrite this.

(a) But we can place notes before or after: here is a verbose explanation for beginners to IF.

```
Before asking which do you mean: say "Okay, so I'm going to have to ask a question now: you've typed something ambiguous, and I don't know which noun you're referring to."

After asking which do you mean: say "(Just type a word or two to give me more information.)"
```

(b) We can also use this activity as a context for other activities. For instance:

```
The Champs du Mars is a room. The great Eiffel Tower is here. "The great Tower stands high over you." The souvenir model Eiffel Tower is here. "Comparatively tiny is the souvenir version." The great Eiffel Tower is fixed in place. Understand "actual" as the great Tower.

Rule for printing the name of the great Tower while asking which do you mean: say "actual Tower". Rule for printing the name of the souvenir tower while asking which do you mean: say "souvenir".
```

causes TAKE TOWER (for instance) to produce a nice tidy question in reply: "Which do you mean, the actual Tower or the souvenir?"

4. A note about actions. This activity takes place during the process of understanding the player's command, when the action that will take place is not fully known. So if the player types "TAKE SHOEBOX", this activity would happen when SHOEBOX is being examined for meaning. Inform knows that the action will be taking, but nothing else. That means attaching a proviso like "... while taking a container" to a rule for this activity will cause the rule to have no effect - whereas "... while taking" would be fine.

Start of Chapter 18: Activities

Back to §18.30. Clarifying the parser's choice of something

Onward to §18.32. Supplying a missing noun/second noun

Example 368: ★ **Apples** Prompting the player on how to disambiguate otherwise similar objects.

Example 369: ★ **Originals** Allowing the player to create models of anything in the game world; parsing the name "model [thing]" or even just "[thing]" to refer to these newly-created models; asking "which do you mean, the model [thing] or the actual [thing]" when there is ambiguity.

Example 370: ★★★ **Walls and Noses** Responding to "EXAMINE WALL" with "In which direction?", and to "EXAMINE NOSE" with "Whose nose do you mean, Frederica's, Betty's, Wilma's or your own?"

---

§18.32. Supplying a missing noun/second noun

1. When it happens. (Two different activities here, but identical except for applying to different nouns.) This happens when an Understand sentence fails to supply a noun for an action which requires one. For example, in the sentence 'Understand "seize" as taking.' - the "taking" action is incompletely specified, because it requires a noun, and there's no noun in the command to be understood.

Note that this is not what happens if it's the player who fails to supply the noun. That is, suppose the player types a half-finished command like TAKE, which can't be matched against (for example) 'Understand "take [things]" as taking.' because the player didn't name any thing(s). Typically a story will reply to such a command with a question asking for clarification, but sometimes it makes guesses about what was meant. The "supplying a missing noun" activity plays no part in this guesswork, and can't influence it: that's the task of the "does the player mean" rulebook.

Suppose we do have the first of these cases, then. "Supplying a missing noun" takes place to remedy the problem. It can either:

(i) Set a noun, printing text like "(presumably the black bag)" if it wants, in which case the action goes forward, though it is still subject to the full rules on accessibility exactly as any other action would be; or

(ii) Make no choice, in which case no action takes place and the player's command is rejected. If the activity printed nothing, Inform will produce a generic reply to the player that "You must supply a noun.".

2. The default behaviour. In the default grammar for Inform, only three such half-finished actions are ever Understood. One is "going" with no direction, for which this activity simply prints a refusal. The other two are the two undirected senses, "smelling" and "listening". In each case, the "supplying a missing noun" activity sets the noun to the current location: so, for instance, typing the bare command "listen" might generate the action "listening to the Shoreline".

3. Examples. (a) This is the definition Inform uses to make "listen" work as outlined above:

```
Rule for supplying a missing noun while listening (this is the ambient sound rule):
    now the noun is the location.
```

(b) It can be elegant to allow second nouns to be dropped with habitual actions, or where the choice is obvious:

```
Understand "unlock [something]" as unlocking it with.

Rule for supplying a missing second noun while unlocking:
    if the skeleton key is carried, now the second noun is the skeleton key;
    otherwise say "You will have to specify what to unlock [the noun] with."
```

Note that, in order for our activity to succeed, we do need to supply a grammar line allowing the player to try "unlocking it with" using only one noun. Otherwise, the command "unlock something" will still produce the question "What do you want to unlock the door with?"


Start of Chapter 18: Activities


Back to §18.31. Asking which do you mean


Onward to §18.33. Reading a command


Example 371:  **Latin Lessons** Supplying missing nouns and second nouns for other characters besides the player.


Example 372:  **Minimal Movement** Supplying a default direction for "go", so that "leave", "go", etc., are always interpreted as "out".

---

§18.33. Reading a command

1. When it happens. When reading a command from the keyboard.

2. The default behaviour. Print the prompt text; wait for the player to type something and press return. Reject an entirely blank line, and treat a command beginning "oops" as a correction to the previous one. This is a fairly complicated business, so it is probably best not to change the "for" rules for this activity: "before", and especially "after", are another matter. (Note, however, that if Inform does reject a blank line and ask for another then this all happens inside the "for" rules: no "after" occurs after the blank line, nor does a "before" happen before the second attempt by the player. It is all a single round of the activity, not two.)

3. Examples. (a) To lead absolute beginners in gently:

```
Before reading a command while the turn count is 1, say "(This is your chance to say
what the protagonist should do next. After the '>', try typing 'take inventory'.)"
```

(b) The following responds politely but firmly if the player tries to type "please look", say, instead of just "look":

```
After reading a command:
    if the player's command includes "please":
```

```
say "Please do not say please.";
reject the player's command.
```

To explain. Fragments of what the player has typed are called snippets: "the player's command" is the entire thing. We can test if a snippet matches a given pattern like so:

if (snippet) **matches** (topic)**:**

This condition is true if the given snippet exactly matches the specification. Example:

```
if the player's command matches "room [number]", ...
```

will be true if the command is ROOM 101, but not if it's EXPLORE ROOM 7.

if (snippet) **does not match** (topic)**:**

This condition is true if the given snippet does not exactly match the specification.

if (snippet) **includes** (topic)**:**

This condition is true if the given snippet includes words matching the specification, either at the beginning, in the middle, or at the end. Example:

```
if the player's command includes "room [number]", ...
```

will be true if the command is ROOM 101, EXPLORE ROOM 7, or ROOM 22 AHOY, but not if it's VISIT ROOM GAMMA 7.

if (snippet) **does not include** (topic)**:**

This condition is true if the given snippet does not include any run of words which matches the specification.

Lastly, we took drastic action with another new phrase:

reject the player's command

This phrase should be used only in rules for the "reading a command" activity. It tells Inform not to bother analysing the text further, but to go back to the keyboard. (No time passes; no turn elapses; nothing happens in the simulated world.)

(c) An improved version takes commands like "please drop the coin" and strips "please" from them, but then allows them to proceed normally:

```
After reading a command:
    if the player's command includes "please":
        say "(Quelle politesse! But no need to say please.)";
        cut the matched text.
```

"Matched text" is a snippet containing the words which matched against the pattern in the most recent "includes" condition, so in this case it contains just the single word "please". Two phrases allow snippets to be altered:

replace (snippet) **with** (text)

This phrase should be used only in "after" rules for the "reading a command" activity; it replaces the snippet of command, usually the "matched text" found immediately before, with the given text. Example:

```
    if the player's command includes "room [number]":
        replace the matched text with "office".
```

cut (snippet)

This phrase should be used only in "after" rules for the "reading a command" activity; it removes the snippet of command. Example:

```
    if the player's command includes "or else":
        cut the matched text.
```

Note that "replace" and "cut" can only be used in "after reading a command" rules: not when an action has been chosen and has gone ahead into its rulebooks. Once the "reading a command" activity has finished, the command is final.

(d) To make the word "grab" an abbreviation for "take all":

```
After reading a command:
    if the player's command matches "grab", replace the player's command with "take all".
```

("Snippet" is actually a kind of value, so we could say "Ah, you typed '[the player's command]'!" or some such if we liked. But in practice only three snippets are likely to be useful: the two mentioned above, "player's command" and "matched text", and the "topic understood", used when matching the "[text]" token in command grammar.)

(e) Finally, we can make still more detailed alterations to the text of the command using the techniques presented in the Advanced Text chapter. For instance:

change the text of the player's command to (text)

This phrase should be used only in "after" rules for the "reading a command" activity; it replaces the current command text entirely. Example:

```
After reading a command:
    let T be "[the player's command]";
    replace the regular expression "\p" in T with "";
    change the text of the player's command to T.
```

This converts the player's command to text, which is then manipulated by searching for any punctuation mark and replacing it with blank text (that is, deleted), and then put back again as the new command.

Start of Chapter 18: Activities

Back to §18.32. Supplying a missing noun/second noun

Onward to §18.34. Implicitly taking something

Example 373: **Cloves** Accepting adverbs anywhere in a command, registering what the player typed but then cutting them out before interpreting the command.

Example 374: **Fragment of a Greek Tragedy** Responding to the player's input based on keywords only, and overriding the original parser entirely.

Example 375: **North by Northwest** Creating additional compass directions between those that already exist (for instance, NNW) -- and dealing with an awkwardness that arises when the player tries to type "north-northwest". The example demonstrates a way around the nine-character limit on parsed words.

Example 376: **Complimentary Peanuts** A character who responds to keywords in the player's instructions and remarks, even if there are other words included.

---

§18.34. Implicitly taking something

1. When it happens. When an action is tried which requires the actor (normally the player, of course) to be carrying something, but which is not in fact carried by that person. For instance, if the player types WEAR OVERCOAT in reference to a Moroccan overcoat currently draped over a chair.

2. The default behaviour. To print text such as "(first taking the Moroccan overcoat)" and then silently try taking the object in question (the overcoat). If the take succeeds, the silence means that nothing else is printed: if it fails, it will say why.

No matter what rules are written for this activity, it is impossible to use it to allow the action to go ahead even without the item. The activity allows us to change how, or if, an implicit take will happen, but not to change the consequences of failure. (To do that, we would need to say that "The ignore the carrying requirements rule does nothing", but this kind of unstitching of the action machinery needs to be done with caution.)

3. Examples. (a) Forbidding implicit takes for certain dangerous items. (This seems especially fair if taking such items might cause death: the player will not wish to be killed on the strength only of our guess as to what he might be intending to do.)

```
Rule for implicitly taking the curare:
    say "Ordinarily you'd pick up the curare in order to be able to do that, but this seems
like a good moment for caution." instead.
```

(b) Changing the way the implicit action is reported for the player:

```
Rule for implicitly taking something (called target):
    try silently taking the target;
    if the player carries the target, say "You appropriate [the target] first, of course. [run
paragraph on]"
```

(c) Combining implicit takes when the noun and second noun must both be carried:

```
Rule for implicitly taking the noun when the second noun is a thing and the second noun
is not carried by the player:
    try silently taking the noun;
    try silently taking the second noun;
    say "(first taking both [the noun] and [the second noun])[line break]"
```

(d) Making another character reply amusingly:

```
Rule for implicitly taking something which is carried by the player when the person
asked is Clark:
    say "'I don't see how I'm supposed to do that when you're holding [the noun],' remarks
Clark sulkily." instead.
```

(e) Causing implicit takes which wouldn't otherwise happen. Suppose we have a photographing action, and there are very small flowers which can't conveniently be snapped without being first picked. We then want an implicit take to occur, even though we wouldn't want this for other sorts of photography. So:

```
Check an actor photographing a flower:
    if the actor is not carrying the noun:
        carry out the implicitly taking activity with the noun;
        if the actor is not carrying the noun, stop the action.
```

Note that if the activity doesn't succeed in taking the item, it's expected to print some text explaining this, which is why we don't need to say anything further.

Start of Chapter 18: Activities

Back to §18.33. Reading a command

Onward to §18.35. Printing a parser error

Example 377: ★ **The Big Sainsbury's** Making implicit takes add a minute to the clock, just as though the player had typed TAKE THING explicitly.

Example 378: ★ **Pizza Prince** Providing a pizza buffet from which the player can take as many pieces as he wants.

Example 379: ★★★ **Lollipop Guild** Overriding the rules to allow the player to show something to another character without first taking it.

---

§18.35. Printing a parser error

1. When it happens. The parser is the part of the run-time software, included in all works produced by Inform, which tries to match the player's command against the grammar provided by the work. When it is unable to make a valid match, the parser prints an error to the player: for instance,

```
> BIFURCATE TREE
That's not a verb I recognise.
```

There are more than twenty possible messages. The one which the parser wants to say is stored in the variable "latest parser error", which has the convenient kind "command parser error". This has the following possible values:

```
didn't understand error
only understood as far as error
didn't understand that number error
can't see any such thing error
said too little error
aren't holding that error
can't use multiple objects error
can only use multiple objects error
not sure what it refers to error
excepted something not included error
can only do that to something animate error
not a verb I recognise error
not something you need to refer to error
can't see it at the moment error
didn't understand the way that finished error
not enough of those available error
nothing to do error
referred to a determination of scope error
I beg your pardon error
noun did not make sense in that context error
can't again the addressee error
comma can't begin error
can't see whom to talk to error
can't talk to inanimate things error
didn't understand addressee's last name error
```

2. The default behaviour. Prints the message in question.

3. Examples. (a) Perhaps for newcomers:

```
After printing a parser error:
    say "If you are new to interactive fiction, you may like to try typing HELP."
```

(b) Or to give the parser a certain amount of character:

```
Rule for printing a parser error when the latest parser error is the I beg your pardon
error:
    say "What's that? Speak up, speak up." instead.

Rule for printing a parser error:
    say "That's a rum thing to say, and no mistake." instead.
```

(c) This can be helpful for seeing what's going on:

```
Rule for printing a parser error:
    say "The [latest parser error] happened.";
    continue the activity.
```

 Start of Chapter 18: Activities

 Back to §18.34. Implicitly taking something

 Onward to §18.36. Deciding whether all includes

 Example 380:  **WXPQ** Creating a more sensible parser error than "that noun did not make sense in this context".

 Example 381:  **Xot** Storing an invalid command to be repeated as text later in the game.

---

§18.36. Deciding whether all includes

1. When it happens. When parsing a command such as "take all", where the player uses "all" to signify everything within reach.

2. The default behaviour. The actual method used is complicated, as "all" is not as simple as it seems - "take all" would not include the player's own body, for instance, or the crescent moon. The point of this activity is to allow the normal method to be changed for given objects, or given kinds of object.

3. Examples. (a) Removing scenery from "all" (but see (4) below):

```
Rule for deciding whether all includes scenery: it does not.
```

The phrases "it does" and "it does not" make a decision.

(b) Ensuring that a given thing, which might otherwise be excluded, is included:

```
Rule for deciding whether all includes the oval roof: it does.
```

4. The Standard Rules already uses this. Note that the Standard Rules already stocks this activity with several rules:

```
exclude scenery from take all rule
exclude people from take all rule
exclude fixed in place things from take all rule
```

5. A note about actions. This activity takes place during the process of understanding the player's command, when the action that will take place is not fully known. So if the player types "TAKE SHOEBOX", this activity would happen when SHOEBOX is being examined for meaning. Inform knows that the action will be taking, but nothing else. That means attaching a proviso like "... while taking a container" to a rule for this activity will cause the rule to have no effect - whereas "... while taking" would be fine.


Start of Chapter 18: Activities


Back to §18.35. Printing a parser error


Onward to §18.37. Printing the banner text

---

§18.37. Printing the banner text

1. When it happens. The banner is the bibliographic masthead text, which typically looks something like this:

```
Relations
An Interactive Fiction by Emily Short
Release 1 / Serial number 050630 / Inform 7 build 2U98 (I6/v6.30 lib 6/10N) SD
```

(The serial and build numbers are those applying when the story file was last made: these ones are from the mid-2000s.) The banner is printed at the start of play, and when the player types "version" at the command line, and when say "[banner text]" occurs.

say "[the/-- banner text]"

This text substitution expands to the banner text giving bibliographic details of the current story file, rather like the opening credits of a movie, or the title page of a book.

2. The default behaviour. Prints the text above, giving the title, the headline, the author, the release number, the date of compilation (that's the serial number: YYMMDD), and version numbers of the Inform components used to put the story together.

3. Examples. (a) Adding a line to the banner:

```
After printing the banner text, say "DRM authentication code: 13S-451-
2034u75y65u%%a1248."
```

(b) Simplifying the banner:

```
Rule for printing the banner text: say "Welcome." instead.
```

Start of Chapter 18: Activities

Back to §18.36. Deciding whether all includes

Onward to §18.38. Printing the player's obituary

Example 382: **Bikini Atoll** Delaying the banner for later.

---

§18.38. Printing the player's obituary

1. When it happens. The obituary is the text "*** You have died ***" or similar, sometimes followed by a final score, if the appropriate use option ("Use scoring.") is in force.

2. The default behaviour. Printing the aforementioned text, then the final score, and reducing the status line to a largely blank state.

3. Examples. Here's one way to add to the verdict of history:

```
After printing the player's obituary: say "And you visited [number of visited rooms] place[s]."
```

Start of Chapter 18: Activities

Back to §18.37. Printing the banner text

Onward to §18.39. Amusing a victorious player

Example 383: **Battle of Ridgefield** Completely replacing the endgame text and stopping the game without giving the player a chance to restart or restore.

Example 384: **Finality** Not mentioning UNDO in the final set of options.

Example 385: **Jamaica 1688** Adding a feature to the final question after victory, so that the player can choose to reveal notes about items in the game.

---

§18.39. Amusing a victorious player

1. When it happens. When the player chooses "AMUSING" from the short menu of choices after a story has been won. Traditionally, this is where the author gets to point out quirky by-ways of the story, or make some final acknowledgements, or simply salute the player's perseverance. Note that the AMUSING option is only offered when the story has ended finally, and that it is only offered if there is at least one rule present in the "for amusing a victorious player" rulebook.

2. The default behaviour. None. The "for amusing a victorious player" rulebook is empty by default, and no amusement is available.

3. Examples. The format would be like so:

```
Rule for amusing a victorious player: say "Hmm. You're easily amused."
```

Start of Chapter 18: Activities

Back to §18.38. Printing the player's obituary

Onward to §18.40. Starting the virtual machine

Example 386: ★★ **Xerxes** Offering the player a menu of things to read after winning the game.

---

§18.40. Starting the virtual machine

1. When it happens. This activity is provided solely as a "hook" for any low-level tasks which need to be performed when the virtual computer which runs Inform story files is starting up. This happens much earlier than "when play begins" rules, and should be used only as a last resort.

It should be remembered that Inform can produce story files for several different virtual computers. On some of these, it will not be safe to print any text during this activity, as the windows which would display such text do not yet exist.

2. The default behaviour. None.

3. Examples. No detailed examples will be given here, but the activity might be used (for instance) to set styles for the Glulx windows shortly to be brought into existence.

Start of Chapter 18: Activities

Back to §18.39. Amusing a victorious player

Onward to Chapter 19: Rulebooks: §19.1. On rules

Example 387: ★ **Blankness** Emptying the status line during the first screen of the game.

Examples from Chapter 18: Activities

Start of this chapter

Chapter 19: Rulebooks

Indexes of the examples

334 ★ **Example** **Ant-Sensitive Sunglasses** *RB*

What are activities good for? Controlling output when we want the same action to be able to produce very flexible text depending on the state of the world -- in this case, making highly variable room description and object description text.

Suppose we want to create an object -- or maybe even a series of objects -- that warp the player's perception of every room description and object around him.

We've already seen some ways to create variations in text. For instance, we could make a room description with if substitutions in it, like so:

```
The Kitchen is a room. "[if the player is wearing the sunglasses]Are ants coming out of the sink? No, probably no.[otherwise]A small kitchen tucked into a corner of the vacation house. There is storage space for five or six cups, a sink, a two-ring stove; nothing else to speak of.[end if]"
```

That works fine if we have one or two variations we want to add; it's not so good if we're going to have several items that work like the sunglasses, or if we want the sunglasses to override the description of every room in the house.

A slightly more flexible method is to use a substitution that calls out to a say phrase, like this:

```
The Kitchen is a room. "[kitchen-description]"

To say kitchen-description:
    if the player is wearing the sunglasses:
        say "Are ants coming out of the sink? No, probably no.";
    otherwise:
        say "A small kitchen tucked into a corner of the vacation house. There is storage space for five or six cups, a sink, a two-ring stove; nothing else to speak of."
```

But again this doesn't handle the case of overriding multiple rooms at once very well.

When we reach a point where we need a given piece of text to be very flexible depending on the world model, it's time to use an activity.

Activities offer several advantages. One, we can create an activity like this:

```
Printing the room-description of something is an activity.
```

and then write a rule that applies to multiple rooms at once, like:

```
Rule for printing the room-description of a room when the player wears the sunglasses:
    say "The walls look like they're covered with ants. Just a coincidence, I'm sure."
```

Inform's usual rule-ranking also means that more-specific rules will override less-specific ones, so we could add

```
    Rule for printing the room-description of the Kitchen when the player wears the sunglasses:
    say "Are ants coming out of the sink? No, probably not."
```

and have that rule override the behavior of the activity just in the kitchen. Meanwhile, our base room descriptions remain straightforward and uncluttered by if-

statements.

Several other examples will show how to hook activities into existing actions: Crusoe goes into detail about how how to make the descriptions of things more variable, and Aftershock demonstrates activities for describing the behavior of switchable devices.

Here, we preview all of those methods, just to get a sense of how they work and why they might be useful in controlling a game. Subsequent chapters go into more detail about the syntax of creating activities and the list of activities that are already defined by Inform.

```
"Ant-Sensitive Sunglasses"

Part 1 - Procedure
```

To add a new activity to an existing Inform rule, we need to do three things:

1) Define our new activity.

2) Give a basic rule that says what is supposed to happen when that activity occurs, as in "Rule for..."

3) Replace the existing rule in Inform's rulebooks with a new one that calls on our activity.

Here we do this with examining:

```
Section 1 - Item Description

Printing the description of something is an activity.
```

Now, by default, we want to print the description property; we just want the option to write some extra rules overriding that property. So we tell Inform that our most basic rule for printing the description of something is just to give that description text:

```
Rule for printing the description of something (called item):
    if the description of the item is not "":
        say "[description of item][paragraph break]";
    otherwise:
        say "You see nothing special about [the item].".
```

Next, we need the standard examining rule to look at our printing-the-description activity:

```
The activity-based examining rule is listed instead of the standard examining rule
in the carry out examining rules.

This is the activity-based examining rule:
    carry out the printing the description activity with the noun;
    rule succeeds.
```

Now we do the same thing to room descriptions.

```
Section 2 - Room Description

Printing the room-description of something is an activity.

Rule for printing the room-description of a room (called item):
    if the description of the item is not "":
        say "[description of item][paragraph break]";
    otherwise:
        do nothing instead.

The activity-based room description body text rule is listed instead of the room
description body text rule in the carry out looking rules.
```

Our replacement rule this time around is a little bit trickier just because the rule that we're replacing is a complicated one: describing a room already checks to see whether there's light to see by, whether the player has turned off room descriptions when he enters a room for the second time, and whether the player character is (say) inside a closed box he can't see out of.

But all of those details are re-copied from the standard rules, and the important thing is that, at the end, we again carry out our activity.

```
This is the activity-based room description body text rule:
    if the visibility level count is 0:
        if set to abbreviated room descriptions, continue the action;
        if set to sometimes abbreviated room descriptions and
            abbreviated form allowed is true and
            darkness witnessed is true,
            continue the action;
        begin the printing the description of a dark room activity;
        if handling the printing the description of a dark room activity,
            say "It is pitch dark, and you can't see a thing.";
        end the printing the description of a dark room activity;
    otherwise if the visibility ceiling is the location:
        if set to abbreviated room descriptions, continue the action;
        if set to sometimes abbreviated room descriptions and abbreviated form
            allowed is true and the location is visited, continue the action;
        carry out the printing the room-description activity with the location.

Section 3 - Device Description

Showing action of something is an activity.

Rule for showing action of something (called item):
    if the item is switched on, say "[The item] is switched on.";
    otherwise say "[The item] is switched off."

The activity-based described devices rule is listed instead of the examine
devices rule in the carry out examining rules.

This is the activity-based described devices rule:
    if the noun is a device:
```

```
        carry out the showing action activity with the noun;
        now examine text printed is true.

Report switching on something:
    say "You flip a switch. ";
    carry out the showing action activity with the noun instead.
```

Part 2 - Scenario

```
The Kitchen is a room. "A small kitchen tucked into a corner of the vacation
house. There is storage space for five or six cups, a sink, a two-ring stove;
nothing else to speak of."

The microwave is a device in the Kitchen.

South of the Kitchen is the Living Area. The description of the Living area is "A
whitewashed living/dining/reclining area in what used to be a shepherd's stone
hut, but now costs vacationers 600 euros a week. It offers no mod cons, only a
straight view of the Mediterranean and a wobbly writing table."

Rule for printing the room-description of a room when the player wears the
sunglasses:
    say "The walls look like they're covered with ants. Just a coincidence, I'm
sure[antsy]."

Rule for printing the room-description of the Kitchen when the player wears the
sunglasses:
    say "Are ants coming out of the sink? No, probably not[antsy]."

Rule for printing the description of something (called the item) when the player
wears the sunglasses:
    say "[The item] [are] [one of]ant-colored[or]ant-legged[or]covered in ants[at
random][antsy]."

Rule for showing action of the microwave:
    say "The microwave hums meaningfully to itself."

Rule for showing action of the microwave when the player wears the sunglasses:
    say "The microwave hums as though inhabited by a billion ants[antsy]."

The player carries sunglasses of freakiness and an apple. The apple is edible.
The sunglasses are wearable.

ant-paranoia is a number that varies.

To say antsy:
    increase ant-paranoia by 1;

Every turn:
    if the ant-paranoia is greater than 3:
        say "Augh! AUUUGH! GET THEM OFF--";
        end the story saying "You have lost your mind."
```

Test me with "look / turn on microwave / turn off microwave / x apple / x sunglasses / s / wear sunglasses / look / x apple / n / turn on microwave".

335

Example AARP-Gnosis

RB

An Encyclopedia set which treats volumes in the same place as a single object, but can also be split up.

Suppose we have a complete Encyclopedia in our game. The player is allowed to pick up the whole set (there must not be too many volumes), but also to do things with individual volumes, and indeed to scatter these volumes all over the place. Putting a volume back in the same place as the rest of the Encyclopedia should, however, restore it to the collective. We will start out by defining general rules for collectives like this:

```
"AARP-Gnosis"

Fitting relates various things to one thing (called the home). The verb to fit
means the fitting relation. Definition: a thing is missing if it is not part of the home
of it.

A collective is a kind of thing.

Before doing something to something which is part of a collective:
    let space be the holder of the home of the noun;
    move the noun to the space.

Instead of examining a collective:
    say "[The noun] consists of [the list of things which are part of the noun]."
```

Now the real work begins. One reason to make this an activity is that we might easily want to override it for specific objects; for instance, the generic collecting activity here would not deal properly with collectives of clothing where some items might be worn and others not. In that case, we would want to write another, more specific "collecting" activity to handle the complexities of fashion.

```
Collecting something is an activity.

Every turn:
    repeat with item running through collectives:
        carry out the collecting activity with the item.

To remove (item - a thing) when empty:
    let space be the holder of the item;
    if the number of things which are part of the item is 0:
        now the item is nowhere;
    if the number of things which are part of the item is 1:
        let the last thing be a random thing which is part of the item;
        move the last thing to the space;
        now the item is nowhere.
```

```
Before collecting a thing (called the item):
    remove item when empty;
    let space be the holder of the item;
    if space is not a thing and space is not a room:
        if something (called the other space) contains at least two things which fit
the item, move item to the other space;
        if a room (called the other space) contains at least two things which fit the
item, move item to the other space;
        if someone (called the owner) carries at least two things which fit the item,
move item to the owner.

Rule for collecting a thing (called the item):
    let space be the holder of the item;
    if space is a thing or space is a room:
        repeat with component running through things held by the space:
            if the component fits the item, now the component is part of the item;
        remove item when empty.
```

And now for a cheerful scenario:

```
The Boise Memorial Library is a room. "A concrete box of a room, roughly eight
feet by fourteen, which contains all the fallout shelter has to offer by way of
entertainment. Someone with a grim sense of humor has tacked a READ!
literacy poster to the door, as though there were anything else to do while you
await the calming of the Geiger counters." The shelf is a supporter in the Library.
"A battered utility shelf stands against the south wall."

The New Idahoan Encyclopedia Set is a collective. Volume A-Aalto fits the
Encyclopedia. It is part of the Set. Volume AAM-Aardvark fits the Encyclopedia.
It is part of the Set. Volume Aarhus-Aaron fits the Encyclopedia. It is part of the
Set. Volume AARP-Gnosis fits the Encyclopedia. It is part of the Set. Volume
Gnu-Zygote fits the Encyclopedia. It is part of the Set. The Set is on the shelf.
```

Let's have the Encyclopedia describe itself differently depending on whether it's all in one place:

```
After printing the name of the Set when something missing fits the Set:
    say " (missing [a list of missing things which fit the Set])"

Before printing the name of the Set when the number of missing things which fit
the set is 0:
    say "complete ".

Test me with "get aarhus-aaron / look / inventory / get aam-aardvark / look / get
gnu-zygote / look / get aarp-gnosis / look / inventory / drop set / look / get set /
get a-aalto / inventory".
```

---

336  **Example Aftershock** *RB*

Modifying the rules for examining a device so that all devices have some specific behavior when switched on, which is described at various times.

The built-in behavior of Inform is to print a line after a device is examined, saying whether the item is on or off. This is often inappropriate, and we could simply turn off that behavior in general by instructing Inform to ignore the "examine described devices rule" (see the chapter on rulebooks).

Perhaps, though, we would like continue to have a short passage about the action of any switched on device; we'd just like a little more control over what it says from time to time. And in that case, we might change the rule to give a new activity control over that portion of the description:

```
"Aftershock"

Section 1 - Showing actions

Showing action of something is an activity.

Rule for showing action of something (called item):
    if the item is switched on, say "[The item] is switched on.";
    otherwise say "[The item] is switched off."
```

Borrowing from the rulebooks chapter, we can replace the standard "examine described devices" rule with something that uses this activity.

```
The new described devices rule is listed instead of the examine devices rule in
the carry out examining rules.

This is the new described devices rule:
    if the noun is a device:
        carry out the showing action activity with the noun;
        now examine text printed is true.
```

Thus far we have essentially replicated the original behavior, but we've made it possible to write specialized behavior for devices, and to invoke that behavior in other places:

```
Report switching on something:
    say "You flip a switch. ";
    carry out the showing action activity with the noun instead.
```

This might be useful for an electric lamp kind:

```
Section 2 - Electric Lamps

An electric lamp is a kind of device.

Rule for showing action of an electric lamp (called item):
    if the item is switched on, say "[The item] is lit[if the number of visible lit things
is greater than 1], competing with [the list of visible lit things which are not the
item][end if].";
    otherwise say "[The item] is dark."

Carry out switching on an electric lamp: now the noun is lit. Carry out switching
off an electric lamp: now the noun is unlit.
```

```
Section 2 - The Scenario

The time of day is 3:47 AM. When play begins, now the right hand status line is "[time of day]".

The Downstairs Hallway is a dark room. "The only room in the house with no furniture and almost nothing on the walls. At times like this you always notice the crack in the plaster, originating near the light fixture and running almost all the way to the wall."

A plastic jug of filtered water is in the Downstairs Hallway. The description is "Five gallons, not that that will last you very long, hot as it has been lately."

The crack is scenery in the Hallway. The description is "No, the ceiling isn't going to fall on you today."

The light fixture is an electric lamp in the Hallway. It is switched on, lit, and scenery. The description is "A plain globe of frosted glass containing the light bulb. Nothing special, and you never think about it except when, as now, you are forced to spend hours in this room."

The flashlight is an electric lamp carried by the player. The description is "A shiny red flashlight." The portable radio is a device carried by the player. The description is "A small battery-operated radio which you received for free with your subscription to US News & World Report. It has served you well through many earthquakes past."
```

And with our activity, we can override the flashlight's electric lamp behavior with new behavior:

```
Rule for showing action of the flashlight:
    if the flashlight is switched on, say "A strong, narrow beam of light shines from the flashlight.";
    otherwise say "It is currently switched off."
```

...or give special actions for the radio:

```
Rule for showing action of the radio:
    if the radio is switched on, say "Through the static, you pick up pieces of discussion: a 6.7 on the Richter scale, epicenter... something about Topanga... but it crackles out again.";
    otherwise say "The radio is silent. You're saving the batteries."

Instead of listening in the presence of the switched on radio:
    carry out the showing action activity with the radio instead.

Test me with "examine light / switch light off / switch flashlight on / switch radio on / examine radio / examine flashlight".
```

---

337 **Example Crusoe** RB

Adding a "printing the description of something" activity.

Suppose we want to add rules so that any time we examine a charred object (and most of our objects can be charred), a line about the charring is appended to the end of the object description. We could use "after examining...", but perhaps we would prefer for the sentence about the charring not to appear in its own paragraph.

This is an ideal occasion for a new activity. We look at the action index for "examining" to identify the rule that causes the old behavior (in this case, the "standard examining rule"); replace this with a new rule that calls our activity; and write our "printing the description" activity in such a way that it uses an object's description without forcing a paragraph return afterward.

Then we will use "after printing the description" to add our line about charring, and make sure that the paragraph return does occur before the prompt.

So:

```
"Crusoe"

Section 1 - Creating our New Activity

The fancy examining rule is listed instead of the standard examining rule in the
carry out examining rules.
```

This instruction replaces a normal piece of the examine action, the standard examining rule, with another one of our own devising. (The replacement of the standard examining rule will be explained in more detail in the chapter on rulebooks.)

```
Printing the description of something is an activity.

This is the fancy examining rule:
    carry out the printing the description activity with the noun;
    rule succeeds.
```

All we have done here is enclose what is usually just a rule inside an activity. This means that we can now write before and after rules for the activity, and also add special instructions like "Rule for printing the name of something while printing the description of something" -- this may not be likely to arise often, but Inform now has the concept of "printing the description of something" as a separate context of action. Next we add the modification that lets us append to the description without a new line:

```
Rule for printing the description of something (called item):
    if the description of the item is not "":
        say "[description of item] [run paragraph on]";
    otherwise:
        say "You see nothing special about [the item]. [run paragraph on]".
```

"run paragraph on" here will mean that we do not get a paragraph break following the description, even if it ends with a period. We also insert a space, so that our follow-on comments will be properly punctuated.

```
After printing the description of something charred:
    say "It is charred." instead.
```

The instead at the end of this line stops Inform for going on with any other "after printing the description of..." rules.

The standard library also has rules for printing additional text about containers and supporters with visible contents, and devices that are switched on; with this current system, we could add those as "after printing the description" rules as well, building up a complete paragraph if we wanted. But for simplicity we won't exemplify all of that here. The effects would be much the same as with the "charred" line.

Now, because we want to make sure that we always do get a paragraph break after our description, we add this rule last after all the other rules. "Last" and "first" rules are covered in more detail in the chapter on rulebooks.

```
Last after printing the description of something:
    say paragraph break.
```

```
Section 2 - The Scenario

The Desert Isle is a room. "A pale expanse of sand, here and there developing into hillocks of grass, and a small clump of palms. The water is shallow here, and there are other islands within swimming distance -- or even wading distance, perhaps -- but none of them is any larger than your island, so it doesn't seem worth the trouble of visiting.

A few hundred feet out, the water turns darker blue, the sea floor drops away, and there is nothing to be seen all the way down to the horizon, except a couple of fluffy clouds, and an occasional bird.

The remains of your fire smolder in the stone-lined pit."

A thing can be charred or whole. A thing is usually whole. Instead of burning something: say "You hold [the noun] to the fire until it flares and chars."; now the noun is charred.

The player carries a stick. The description of the stick is "A strip of palm from the woodiest part of the leaf, about a foot and a half long."

The player carries a glass bottle and a piece of paper. The description of the paper is "A single blank sheet." In the glass bottle is a grain of sand. The glass bottle is openable and open. Instead of burning the glass bottle: say "You hold the bottle to the flame, but it grows uncomfortably warm."

Instead of burning the grain of sand: say "You drop the grain into the fire pit, where it becomes indistinguishable from all the others."; now the grain of sand is nowhere. Instead of dropping the grain of sand: now the grain of sand is nowhere; say "You return the grain of sand to its brethren."
```

The player's description is handled in an unusual way, and this will produce a space paragraph break there where it should not. Instead, therefore, we will add an instead for examining the player (probably a good idea anyway):

```
Instead of examining the player:
    say "You are sunburned and there is sand in cracks you didn't know existed."

Test me with "i / x stick / x bottle / x sand / x paper / x me / burn stick / x stick /
burn paper / x paper".
```

The "printing a description" activity may be useful for other games, and can be imported just by lifting section 1.

---

338 **Example Hays Code** *RB*

Clark Gable in a pin-striped suit and a pink thong.

The following burlesque was considered too much for the tender readers of Chapter 3, since it involved explicit use of listing and persuasion:

```
"Hays Code"

The Movie Set is a room. Clark Gable is a man in the Movie Set. "Clark leans on
a polystyrene pillar, wearing [a list of unconcealed things worn by Clark] with his
usual aplomb." Persuasion rule for asking Clark to try doing something:
persuasion succeeds.

Clark is wearing a pin-striped suit and a pink thong. Rule for deciding the
concealed possessions of Clark: if the particular possession is the thong and
Clark is wearing the suit, yes; otherwise no.

Test me with "clark, remove suit / look / clark, remove thong / look".
```

---

339 **Example Shipping Trunk** *RB*

A box of baking soda whose name changes to "completely ineffective baking soda" when it is in a container with something that smells funny.

First to lay some groundwork:

```
"Shipping Trunk"

A chest is a kind of container. A chest is always openable. A lid is a kind of
supporter. A lid is part of every chest.

Before opening a chest when something (called the obstruction) is on a lid which
is part of the noun:
    say "Better remove [the obstruction]." instead.

A thing can be innocent or smelly.
```

```
The Storage Unit is a room. The shipping trunk is a closed chest in the Storage
Unit. The trunk contains some garlic, a loaf of moldy sourdough, a mildewy
bathtowel, a pair of unwashed socks, two dead trout, and a box of baking
powder. The garlic, trout, sourdough, bathtowel, and socks are smelly. The
baking powder is innocent.

The shipping trunk's lid supports a small card. The description of the small card
is "'Please, please do not open this trunk.'"

After opening the trunk:
    if the trunk had been open:
        say "You steel yourself...";
        continue the action;
    otherwise:
        say "There roils up from inside an indescribable funk, which, when you can
see straight, you have no trouble attributing to the presence of [a list of smelly
things in the trunk]. You also note [a list of innocent things in the trunk] in the
corner.".
```

And now, with that preparation:

```
Before printing the name of the baking powder when the powder is in a container
which contains a smelly thing: say "completely ineffective ".

Test me with "open trunk / examine card / get card / open trunk / get powder /
inventory".
```

340

Example **Trachypachidae Maturin 1803**

RB

Bottles with removable stoppers: when the stopper is in the bottle, the bottle is functionally closed, but the stopper can also be removed and used elsewhere. Descriptions of the bottle reflect its state intelligently.

```
"Trachypachidae Maturin 1803"

A bottle is a kind of container. Bottles are usually openable, transparent, and
closed. A cork is a kind of thing. A cork is in every bottle.

Understand "cork [something]" as corking.

Understand the command "stopper" as "cork".

Understand "uncork [something]" as uncorking.

Corking is an action applying to one thing.

Check corking:
    if the noun is not a bottle, say "[The noun] cannot be corked." instead.

Carry out corking:
    try closing the noun.
```

```
Uncorking is an action applying to one thing.

Check uncorking:
    if the noun is not a bottle, say "[The noun] cannot be uncorked." instead.

Carry out uncorking:
    try opening the noun.

Understand "close [something] with [something preferably held]" as corking it
with.

Understand "cork [something] with [something preferably held]" as corking it
with.

Corking it with is an action applying to one thing and one carried thing.

Check corking it with:
    if the noun is not a bottle, say "[The noun] cannot be corked." instead;
    if the second noun is not a cork, say "[The second noun] will not fit in [the
noun]." instead.

Carry out corking it with:
    try inserting the second noun into the noun instead.

Instead of closing a bottle:
    if a cork (called the item) is carried by the player, try inserting the item into the
noun instead;
    otherwise say "You need a stopper of some kind."

Instead of opening a bottle:
    if a cork (called the item) is in the noun, try taking the item instead;
    otherwise say "[The noun] has no stopper."

Carry out inserting a cork into a bottle:
    now the second noun is closed.

After inserting a cork into a bottle:
    say "You stopper [the second noun] with [the noun]."

Before taking a cork when the noun is in a closed bottle (called the item):
    now the item is open.

Instead of taking a cork when the noun is in a bottle (called the item):
    move the noun to the player;
    say "You pull [the noun] from [the item]." instead.

Before printing the name of a bottle (called target) while not inserting, taking,
searching, or removing:
    if the target is closed, say "sealed ";
    otherwise say "now open ".

After printing the name of a bottle (called target) while not inserting, searching,
examining, or removing:
    if the target contains a noncork thing, say " containing [a list of noncork things
```

```
in the target]";
    omit contents in listing.

Instead of examining a bottle:
    say "[The noun] contains [a list of noncork things in the noun]."

Definition: a thing is noncork if it is not a cork.

The Doctor's Cabin is a room. "A dark, cramped triangle, like a slice of cake,
except that its sharp end has been cut off: and so low that a moderately tall man
would strike his head on the deck above if he were to stand upright. Every free
surface is covered with sheets of best Venetian looking-glass, to increase the
light filtering in. Long use and the carpenter's ingenuity have packed in a folding
cot and table, and lockers are built into unlikely places: lockers filled with
specimens, skeletons, sketches, drafts and serial letters." The jug is a bottle in
the Doctor's Cabin. The jug contains a beetle. The description of the beetle is
"The doctor assures you that it is a nondescript."

Test me with "get jug / x jug / open jug / x jug / i / x cork / cork jug / i / uncork jug /
i / x jug / get beetle / i / close jug / i / x jug".
```

---

341  **Example Chronic Hinting Syndrome** *RB*

Using name-printing rules to keep track of whether the player knows about objects, and also to highlight things he might want to follow up.

Suppose we have a conversation system in which it is important to keep track of which subjects the player has heard mentioned. If we're careful to mark subjects in brackets, we can use the "printing the name of" activity to record which things have been mentioned so far:

```
"Chronic Hinting Syndrome"

A subject is a kind of thing.

Knowledge relates various people to various subjects. The verb to know means
the knowledge relation.

Awareness relates various people to various subjects. The verb to be aware of
means the awareness relation.

Definition: a subject is pending if the player is aware of it and it is not known by
the player.

Instead of thinking:
    if the number of pending subjects is 0, say "You have no fresh leads at the
moment.";
    otherwise say "You recall that thus far you have not followed up with questions
about [the list of pending subjects]."
```

```
After printing the name of a subject (called idea):
    now the player is aware of the idea.
```

Now suppose that as an added convenience for the player, we let him turn on a mode in which useful conversation topics are always automatically highlighted in the text, so he doesn't waste his time trying to follow up dead leads:

```
Setting is a kind of value. The settings are bright and dull. Understand "on" as
bright. Understand "off" as dull.

Highlighting is a setting that varies. Highlighting is dull.

Understand "highlighting [setting]" as setting highlighting. Setting highlighting is
an action out of world, applying to one setting.

Carry out setting highlighting:
    now highlighting is the setting understood.

Report setting highlighting:
    say "Highlighting is now [if highlighting is dull]off[otherwise]on[end if]."

Before printing the name of a subject (called idea) when highlighting is bright:
    unless the player knows the idea:
        say "[bold type]".

After printing the name of a subject when highlighting is bright:
    say "[roman type]".
```

...And the rest is peripheral.

```
The Sickbay is a room. "A place arranged for Nathan's comfort, since his
sickness has been prolonged and because he becomes so irritating when not
comfortable." The Hallway is outside from the Sickbay.

A supporter can be untried or rejected. A supporter is usually untried.

The Sickbay contains a wobbly pedestal, a table, and a sickbed. Understand
"bed" as the sickbed. The pedestal, the table, and the sickbed are supporters.
Nathan is a man on the sickbed. The sickbed is scenery. The initial appearance
of the wobbly pedestal is "A wobbly pedestal near the door has sometimes been
known to support vases of flowers, but is currently bare." The initial appearance
of the table is "There is also a table of a more ordinary sort."

Nathan can be active or passive.

After printing the name of Nathan: now Nathan is passive.

Instead of putting the sculpture on the table:
    now the table is rejected;
    say "'[Not there],' [Nathan] snaps. 'The table is way too far from the sickbed.'"

Instead of putting the sculpture on the sickbed:
    now the sickbed is rejected;
```

say "'[Not there],' [Nathan] rebukes you. 'You don't want me knocking it over if I roll around. In pain.'"

Instead of putting the sculpture on the pedestal:
    now the pedestal is rejected;
    say "The pedestal starts to wobble so ominously that you don't dare let go.

'[Not there],' says [Nathan]. 'That thing is falling apart.'"

Before putting something on the down: try dropping the noun instead.

To say not there:
    if all the supporters are rejected:
        say "Look, the floor would be fine";
    otherwise if the number of rejected supporters is 1:
        say "Yeah, anywhere but there, I'm afraid";
    otherwise:
        say "Come on, use your head -- I can't be watching you all the time, I'm sick".

Instead of going outside when the player is carrying the sculpture:
    say "You've still got this sculpture to get rid of."

Instead of going outside when the breakage is pending:
    say "You can't very well smash in front of [Nathan] his prize sculpture and then just scamper off without saying something. Appealing though the thought is at the moment."

Instead of going outside when a subject which is not the breakage is pending:
    say "'Yeah, go ahead,' says [Nathan], with a martyr-like air. 'It's probably best that you don't hear about [the random pending subject]. It's not something I'd go into normally.'"

The breakage is a subject. The description is "It's not a big deal. I'll just buy a new [mental wave generator].' A slight awkward pause. 'I mean, this one was a [gift], but don't worry about it". Understand "accident" or "smashing" or "breaking" or "shard" or "mishap" or "shards" or "mistake" as the breakage. Understand "sculpture" as the breakage when the player is not carrying the sculpture.

Instead of saying sorry in the presence of Nathan when the player is aware of the breakage:
    try asking Nathan about the matter of breakage.

Instead of asking Nathan to try saying sorry when the player is aware of the breakage:
    try asking Nathan about the matter of breakage.

The mental wave generator is a subject. The description is "They're kind of expensive but I can save up. I really need one, though, because of my [dreams]".

The dreams is a subject. The description is "They're not the kind of dream you want to have.' He closes his eyes and contemplates these undesirable dreams. 'Have you ever woken up convinced you were dead? No, probably not. Well... At

any rate, I need the [generator]. Oh, don't worry, they're expensive but not so expensive that I won't be able to save up for another, in a few months".

The gift is a subject. The description is "[The mental wave generator] was a present from a girl named [Shari]. Actually I'm not sure she'd take to being called a girl".

Shari is a subject. The description is "Look, let's just not go into it, okay? I don't really want to relive all that right now. I still have a six-inch [scar] shaped like a banana in the middle of my back".

The scar is a subject.

Instead of asking Nathan about the matter of the scar:
    say "Nathan clears his throat, lowers his voice, and begins to tell you the story...";
    end the story saying "End of Demo -- Register to Continue!!"

Understand "ask [someone] about [any subject]" as asking it about the matter of.

Asking it about the matter of is an action applying to one thing and one visible thing.

Check asking it about the matter of:
    if the player is not aware of the second noun, say "What [second noun]?" instead;
    if the noun does not know the second noun, say "'I've no idea,' replies [the noun]." instead;
    if the player knows the second noun, say "You've already covered that. The response was '[description of the second noun].'" instead.

Carry out asking it about the matter of:
    now the player knows the second noun.

Report asking it about the matter of:
    say "'[description of the second noun],' says [the noun]."

Instead of telling Nathan about something:
    say "He pinches the bridge of his nose. 'I can't really follow this right now,' he says. 'I'm sorry.'"

Instead of asking Nathan about something:
    say "He shrugs weakly."

When play begins:
    say "'Just put that down anywhere,' says [Nathan], as you come into the room. He's sitting in the sickbed with his legs straight out in front of him. 'I don't care where.' His voice is weak, but it sharpens up for the last remark: 'And don't make a lot of noise about it.'

Considering that he woke you from a sound slumber to beg you to bring this thing over, his attitude is a bit much. You stare dubiously at the awkward crystal sculpture in your hands.";
    now Nathan knows every subject.

Instead of asking Nathan about something while the player carries the sculpture, say "[Nathan] moans dramatically and refuses to be drawn into conversation."

The player is carrying an awkward crystal sculpture. Understand "objet" or "objet de hideous" as the sculpture. The description of the sculpture is "It might possibly be natural, or it might be man-made. It might appeal to someone, but it is certainly not to your tastes."

Instead of showing the sculpture to Nathan:
    say "'Please put it anywhere,' he says."

Instead of giving the sculpture to Nathan:
    say "'No, it doesn't work if I touch it. That's why I couldn't-- well, just put it down.'"

After dropping the sculpture:
    now the player is aware of the breakage;
    now the sculpture is nowhere;
    say "You are incredibly careful, but somehow the sculpture slips -- you might almost say slithers -- from your fingers and crashes into a thousand shards at the feet of [Nathan].

There is a tense silence."

Before reading a command: now Nathan is active.

Every turn while not asking:
    if Nathan is passive, rule succeeds;
    if the player is carrying the sculpture:
        if showing or giving, rule succeeds;
        say "[Nathan] opens one eye and stares at you meaningfully. He is waiting for you to deposit his objet de hideous somewhere.";
        rule succeeds;
    if the breakage is pending:
        if dropping, rule succeeds;
        say "You're not quite sure where to begin, but you can't very well leave without making at least some remark on the smashing of the sculpture.";
        rule succeeds;
    if a subject is pending:
        choose a random row in the Table of Offhand Reminiscences;
        say "[line entry][paragraph break]".

Table of Offhand Reminiscences
line
"'It actually is kind of a funny story about [the random pending subject],' [Nathan] remarks casually."
"[Nathan] chuckles under his breath. 'Man, I hadn't thought about [the random pending subject] in ages.'"
"He glances sideways at you. 'It's nothing personal, you know, but I don't feel comfortable discussing [the random pending subject] with just anyone.'"
"'I don't know why I brought up [the random pending subject] just now,' [Nathan] comments. 'Don't mention it to anyone, if you don't mind.'"
"'Okay, see, the thing about [the random pending subject] is...' [paragraph break]'Yes?' you ask, on cue.[paragraph break]'...never mind.'"
"[Nathan] makes an explosive exasperated sound. 'Don't you want to ask me about [the random pending subject]?' he demands."

Test me with "highlighting bright / put sculpture on pedestal / put it on table / put it on sickbed / drop it / think / ask nathan about breakage / think / ask him about generator / ask him about dreams / ask him about gift / ask him about shari / ask him about scar".

342

Example Hudsucker Industries

RB

Letters which are described differently as a group, depending on whether the player has read none, some, or all of them, and on whether they are alike or unlike.

In this scenario, the player starts with a bag full of unsorted letters. These can be polite or rude, but he won't know which until he has examined them. What's more, he is allowed to sort the letters, in which case a group of letters will be shown as (for instance) "two polite letters"; but a group of mixed letters, even if they have all been read, will be called "unsorted letters".

Further, the player should be allowed to refer to sorted letters by tone, but not unsorted letters.

To do this, we'll need printing the name... and printing the plural name..., as well as some special understanding rules.

```
"Hudsucker Industries"

Tone is a kind of value. The tones are effusive, affectionate, polite, curt, and
flamingly rude.

A letter is a kind of thing. The description of a letter is usually "On inspection, it
turns out to be quite [tone]." A letter has a tone. The tone of a letter is usually
polite.

A letter can be read or unread. A letter is usually unread. Carry out examining a
letter: now the noun is read.

Before printing the name of a read letter: say "[tone] ".

Before printing the name of an ungrouped letter: say "random ".

Before printing the plural name of a letter (called the subject):
    if the subject is grouped:
        say "[tone] ";
    otherwise if the number of unread letters which are next to the subject is 0:
        say "unsorted ".

After printing the plural name of a letter (called the subject):
    if the number of read letters which are next to the subject is 0, say " (all
unread, at the moment)" instead;
    if the number of unread letters which are next to the subject is greater than 0,
say " (some as yet unread)" instead.
```

```
Proximity relates a thing (called X) to a thing (called Y) when the holder of X is
the holder of Y. The verb to be next to means the proximity relation.

Definition: a letter is grouped:
    if it is unread, no;
    if the number of unread letters next to it is greater than 0, no;
    repeat with item running through letters which are next to it:
        if the tone of item is not the tone of it, no;
    yes.

Definition: a letter is ungrouped if it is not grouped.

The Mailroom is a room. "Usually a thrumming hive of bee-like workers, but you
got in early to get a jump on the day's work."

The satchel is carried by the player. Two flamingly rude letters are in the satchel.
Five polite letters are in the satchel.

The mail wall are fixed in place in the mailroom. "Before you is a wall of
mailboxes, including [a list of mailboxes which are part of the mail wall]."

The plural of mailbox is mailboxes. A mailbox is a kind of container. The CEO
box is a mailbox. The Hold box is a mailbox. The Trash box is a mailbox.
Understand "mailbox" as a mailbox.
```

Now, there's a good bit of interaction to streamline. We intend that the player will be taking letters from the satchel, reading them, and putting them (perhaps grouped) into boxes. Our interaction rules should assist in this process as much as possible. To start with, the player will be most likely to examine letters he hasn't read yet:

```
Does the player mean examining a letter (called the subject):
    if we have examined the subject, it is very unlikely;
    it is very likely.
```

The rules about taking are more subtle: the player is more likely to want to take an ungrouped letter than a grouped one; he is more likely to want one from the satchel than not; and he is most unlikely to want to take a letter (grouped or ungrouped) that he is already holding.

```
Does the player mean taking a letter (called subject) which is grouped:
    if the player carries the subject, it is very unlikely;
    if the subject is in the satchel, it is possible;
    it is unlikely.

Does the player mean taking a letter (called subject) which is ungrouped:
    if the player carries the subject, it is very unlikely;
    if the subject is in the satchel, it is very likely;
    it is possible.
```

And finally, we will assume by default that anything other than examining or taking is most likely to apply to a letter he's already identified:

```
Does the player mean doing something other than examining or taking with a
letter (called the subject):
    if we have examined the subject, it is likely;
    it is unlikely.
```

And we would also like to understand properties under the same circumstances as printing -- a letter will be identifiable as "polite" if it's already been read and it is either by itself or in a sorted stack of polite letters, but otherwise not. What's more, to make it possible to disambiguate commands in the other direction, we'll call any unsorted letter "random", to represent that the player doesn't know what it is.

```
Understand the tone property as referring to a letter when the item described is
grouped. Understand "random" as a letter when the item described is
ungrouped.

When play begins:
    now every mailbox is part of the mail wall;
    repeat with switch count running from 1 to 5:
        move a random letter to the satchel.

Test me with "inventory / examine letter / get letter / i / put letter in ceo box /
inventory / get letter / x letter / g / g / i / x letter / g / g / i / put letter in hold box /
get letter / g / g / i".
```

That last "repeat" is merely a device to shuffle the order of items in the satchel so that the player will not always encounter the letters in a neatly presorted order, despite our defining them that way. (Of course, that means that the test produced by TEST ME cannot be very exciting...)

343

Example **Prolegomena** *RB*

Replacing precise numbers with "some" or other quantifiers when too many objects are clustered together for the player to count at a glance.

Room descriptions often make the player character out to be a bit of a savant, able to count whole stacks of items at a glance: "You see 27 paper clips here."

We can adjust this behavior to our liking, though, with the printing a number... activity, as follows:

```
"Prolegomena"

The Editor's Office is a room. The desk is a supporter in the Editor's Office.

A red pencil is a kind of thing. 12 red pencils are on the desk.

A letter is a kind of thing. 12 letters are on the desk. Understand
"correspondence" as a letter.

Rule for printing the plural name of a letter:
    if the listing group size is greater than 7, say "correspondence";
```

```
    otherwise say "letters".

Rule for printing a number of something (called the target) when the listing group
size is greater than 7:
    say "[one of]some [or]various [or]an assortment of [at random]";
    carry out the printing the plural name activity with the target.
```

This general rule can of course be overridden by more specific ones; for instance, if we want to take the opportunity to comment on the viewpoint character's appetite for instruments of correction:

```
Rule for printing a number of red pencils (called the target) when the listing
group size is greater than 10:
    carry out the printing the plural name activity with the target;
    say " in nearly-sufficient quantity".

Test me with "get two letters / look / get a pencil / i / get pencil / g / g / look / i /
get all / i".
```

---

344 **Example Unpeeled** RB

Calling an onion "a single yellow onion" when (and only when) it is being listed as the sole content of a room or container.

```
"Unpeeled"

Scullery is a room. A sack is carried by the player. The sack contains a yellow
onion. The player carries a cork.

Before printing the name of the onion while listing contents:
    if the holder of the onion contains exactly 1 thing, say "single ".

Test me with "i / put cork in sack / i".
```

---

345 **Example Wesponses** RB

Parser messages that are delivered with a speech impediment.

If we want to change individual responses to player action, then the best thing to do is to use the response facility to modify those selections, as shown in the chapter on Responses.

However, suppose what we want is to give the parser a speech impediment that slightly alters all of the responses it issues. For this purpose, we might need to do a bit of text replacement...

```
"Wesponses"
```

```
The Office is a room. Barry Kripke is a man in the Office.

The response inhibition is initially false.

Rule for issuing the response text of a response (called R) when response
inhibition is false:
    now response inhibition is true;
    let output be "[text of R]";
    now response inhibition is false;
    replace the text "r" in output with "w";
    replace the text "R" in output with "W";
    say "[output]".

Test me with "i / x barry / listen / waffle / jump"
```

Notice that this doesn't affect the printed names of objects in the room description or other kinds of output text -- only those that are issued by the response mechanism.

It would also not work to try to give our parser a nervous personality by simply adding "Um, " to the beginning of each response, because responses are not guaranteed to be full standalone sentences. For example, we could imagine writing

```
Rule for issuing the response text of a response (called R) when response
inhibition is false:
    now response inhibition is true;
    let output be "Um, [text of R]";
    now response inhibition is false;
    say "[output]".
```

but here is what the room description would say as a result:

```
Office
Um, You Um, can see Barry KripkeUm, here
```

346

Example **Rules of Attraction**

RB

A magnet which picks up nearby metal objects, and describes itself appropriately in room descriptions and inventory listings, but otherwise goes by its ordinary name.

Often we have some salient features of an object that we want to make sure the player notices whenever looking at the item in a room or in inventory. At other times, we may prefer to allow the name of the item to be printed bare. So for instance:

```
"Rules of Attraction"

A metal form is a kind of thing. A magnet is a kind of metal form.

Every turn:
    repeat with item running through nonmagnetic metal forms which are not part
of something:
```

```
        if item is in a container which contains a magnet (called attractor):
            say "[The item] sticks to [the attractor].";
            now the item is part of the attractor.
```

The horseshoe magnet is a magnet carried by the player. The nail is a metal form carried by the player. The Barn is a room. In the Barn is a bucket. In the bucket is a metal form called the iron hook.

Definition: a thing is nonmagnetic if it is not a magnet.

Rule for printing room description details of a magnet (called attractor): if something is part of the attractor, say " (stuck to which [is-are the list of things which are part of the attractor])".

After printing the name of a magnet (called attractor) while taking inventory:
    if something is part of the attractor, say " (stuck to which [is-are the list of things which are part of the attractor])".

Before taking a touchable thing which is part of a magnet (called attractor):
    move the noun to the holder of the attractor.

Test me with "i / put horseshoe in bucket / look / get horseshoe / i / drop horseshoe / i / look / get all / put all in bucket / i / x magnet / get nail / i".

347

Example **Zorn of Zorna**

RB

Light levels vary depending on the number of candles the player has lit, and this determines whether or not he is able to examine detailed objects successfully.

```
"Zorn of Zorna"

Visibility rule:
    if examining:
        if the detail of the noun is fine and the number of visible lit candles is less than 5, there is insufficient light;
        if the detail of the noun is ordinary and the number of visible lit candles is less than 3, there is insufficient light;
    there is sufficient light.

Detail is a kind of value. The details are fine, ordinary, and gross. A thing has detail.

A candle is a kind of thing. Before printing the name of a candle while not burning or blowing out: say "[if lit]lit [otherwise]unlit [end if]". A candle is usually lit. Before printing the plural name of a candle while not burning or blowing out: say "[if lit]lit [otherwise]unlit [end if]". A candle is usually lit. Understand the lit property as describing a candle. A candle is usually gross.

Instead of burning a candle: now the noun is lit; say "You light [the noun]."
```

Understand "blow out [something]" or "extinguish [something]" or "put out [something]" as blowing out. Understand the command "snuff" as "extinguish". Blowing out is an action applying to one thing.

Understand "burn [unlit candle]" as burning.

Instead of blowing out a candle:
    now the noun is unlit;
    say "You put out [the noun]."

Rule for printing a refusal to act in the dark:
    if we are examining something, say "The details of [the noun] are too fine to make out in the light of only [the number of visible lit candles in words] candle[s]." instead.

Every turn when the Todal is visible:
    if the number of visible lit candles is greater than 1:
        say "The brightness of the room wakens the Todal from slumber, and with you unarmed...";
        end the story;
    otherwise:
        say "Todal sleeps fitfully, troubled by even that faint light."

A room is usually dark.

The Palace is a room. "The Duke is out; the way is clear. East is Saralinda's Chamber; north, a hallway zigs and zags down to the gate that leads out." A finely-written placard is in the Palace. "A finely-written placard is on the wall next to this exit." The placard is fine. The description of the placard is "You read: 'Beware the Todal: its bite is worse than its gleep.

No more than one candle!'"

The candle-stand is a supporter in the Palace. Understand "stand" as the candle-stand. The description of the candle-stand is "The candle-stand is a tall metal branch for holding lights, but someone has quite practically added casters to the bottom." It is pushable between rooms. Three candles are on the candle-stand. Instead of removing something from the candle-stand: say "[The noun] is fixed quite firmly in place." Instead of taking something which is on the candle-stand: say "[The noun] won't come out of the holder." Instead of putting something on the candle-stand: say "[The candle-stand] is full."

Saralinda's Chamber is east of the Palace. "Now that Saralinda herself is gone, there is no real radiance in this place." Two unlit candles are in Saralinda's Chamber.

A large-print romantic novel is in Saralinda's Chamber. The novel is ordinary. The description of the novel is "'She Was Only The Chimney-Sweep's Daughter', by Marie Swelldon."

The Zig-Zag Hallway is north of the Palace. "The Hallway goes left, then right, then left again..." Two unlit candles are in the Hallway.

Todal is an animal in the Zig-Zag Hallway.

```
Rule for printing the description of a dark room when the Todal is in the location:
    try listening.

Instead of listening when in darkness and the Todal is in the location:
    say "In the darkness something softly gleeps."

Instead of going north from the Hallway when in darkness:
    say "You stumble and cannot find your way."

North of the Hallway is Freedom. Instead of going to Freedom: say "You make it
out into the cool night air at last!"; end the story finally.

Test me with "examine placard / get placard / n / listen / n / s / examine candle-
stand / push candle-stand east / examine novel / get unlit candle / light it / light
unlit candle / examine placard / push candle-stand west / e / examine novel / w /
n / n".
```

---

348  **Example Hohmann Transfer** *RB*

Changing the way dark rooms are described to avoid the standard Inform phrasing.

Inform automatically keeps track of light and darkness, handling such questions as whether a room is lit, whether the player can see any light sources, etc., and then managing the descriptions accordingly. When the room is dark and no light sources are visible, the player is said to be "in darkness".

If we don't specify otherwise, Inform will describe our surroundings in a dark room thus:

```
Darkness
It is pitch dark, and you can't see a thing.
```

This is fine in many situations, but we may sometimes want to replace this phrase with something else.

```
"Hohmann Transfer"

The Western Hemisphere is a dark room. "The cloud mass covers much of the
land on this side of the planet, and a particularly nasty storm is brewing off to the
south."

The Eastern Hemisphere is west of the Western Hemisphere. The Eastern
Hemisphere is east of the Western Hemisphere. The Eastern Hemisphere is
north of the Western Hemisphere. The Eastern Hemisphere is south of the
Western Hemisphere. "This side of the planet is more ocean than land, with only
two continents worthy of the name, and a volcanic archipelago in the north
seas."

Use full-length room descriptions.
```

```
Rule for printing the description of a dark room:
    say "It's night on this side of the planet, so you can make out only the glow of urbanized areas along the seacoasts." instead.

Rule for printing the name of a dark room:
    say "Dark Side" instead.
```

And now a few minor refinements so that we can see what happens when one room becomes dark and the other light:

```
Carry out going:
    say "You fire the thrusters and loop around to the other side of the planet before settling into a new geosynchronous orbit. Six months and one minute later..."

The time of day is 4:55 PM.

At 5 PM:
    now the Eastern Hemisphere is dark;
    now the Western Hemisphere is lit.

Rule for printing the announcement of darkness:
    say "The planet abruptly spins itself over, exposing its cool underbelly to the sun."

Test me with "e / z / z / w / z / z / e".
```

349

Example **Four Stars 1**

RB

An elaboration of the idea that when light is absent, the player should be given a description of what he can smell and hear, instead.

```
"Four Stars"

A thing has some text called sound. The sound of a thing is usually "silence".

The report listening rule is not listed in the report listening to rules.

Carry out listening to something:
    say "From [the noun] you hear [the sound of the noun]."

Instead of listening to a room:
    if an audible thing can be touched by the player, say "You hear [the list of audible things which can be touched by the player].";
    otherwise say "A merciful peace prevails."

Definition: a thing is audible if the sound of it is not "silence".

Before printing the name of something audible while listening to a room:
    say "[sound] from the "
```

A thing has some text called scent. The scent of a thing is usually "nothing".

The report smelling rule is not listed in the report smelling rulebook.

Carry out smelling something:
    say "From [the noun] you smell [scent of the noun]."

Instead of smelling a room:
    if a scented thing can be touched by the player, say "You smell [the list of scented things which can be touched by the player].";
    otherwise say "The place is blissfully odorless."

Definition: a thing is scented if the scent of it is not "nothing".

Before printing the name of something scented while smelling a room: say "[scent] from the "

The Waning Moon Resort is a dark room. "A spacious room with a flagstone floor, and a dreamcatcher hung over the king-size bed." The dreamcatcher is scenery in the Resort. The description is "The usual web of threads and crystals, feathers and beads." Instead of taking the dreamcatcher, say "Ah, ah -- you might be tempted to take it as a souvenir, except that the price list in the minibar clearly states they charge $65 apiece if you walk off with one. Cheaper than stealing the Frette bathrobes, but still probably not a good idea."

The king-size bed is an enterable supporter in the Resort. The description is "200-thread-count Egyptian cotton sheets, according to the website. You would make fun, only they really are extraordinarily comfortable." The player is on the bed. A Lindt chocolate is on the bed. It is edible. The scent of the chocolate is "chocolate-hazelnut smell".

Instead of exiting: say "You are too weary to move."

The suitcase is an openable closed container in the Resort.

An electric light is a kind of device. Carry out switching on an electric light: now the noun is lit. Carry out switching off an electric light: now the noun is unlit. Understand "light" as an electric light.

The solar lamp is an electric light in Waning Moon Resort. The description is "Specially designed to give light in a spectrum resembling sunlight, to improve the mood and make a person energetic." The lamp is switched on and lit.

An electric noisemaker is a kind of device. An electric noisemaker has some text called usual sound. The usual sound of an electric noisemaker is usually "beepbeepbeep". Carry out switching on an electric noisemaker: now the sound of the noun is the usual sound of the noun. Report switching on an electric noisemaker: say "[The noun] goes [usual sound of the noun]!" instead. Report switching off an electric noisemaker: say "You switch off [the noun], silencing the [usual sound of the noun]." instead.

Carry out switching off an electric noisemaker: now the sound of the noun is "silence".

The bedside table is in the Resort. The table supports a potted plant and a Bose speaker. The scent of the potted plant is "rosemary"

The Bose speaker is an electric noisemaker. The usual sound of the speaker is "soothing whalesong". The sound of the speaker is "soothing whalesong". The speaker is switched on.

Instead of touching a device: say "You feel the surface of [the noun] and discover the switch."

Instead of touching a scented thing: say "The brush of your fingers stirs loose a fresh cloud of [scent of the noun] smell."

Rule for printing the description of a dark room: try listening; try smelling; rule succeeds.

Instead of examining an audible thing while in darkness: try listening to the noun.
Instead of examining something while in darkness: try touching the noun.

Before touching something when in darkness:
    say "You grope about..."

After inserting the plant into something:
    say "You unceremoniously dump [the noun] into [the second noun], hoping it sustains no important damage thereby."

Before printing the name of a dark room: if the player can touch an audible thing, say "Noisy "; if the player can touch a scented thing, say "Perfumed ".

Visibility rule when in darkness:
    if examining something, there is sufficient light;
    there is insufficient light.

Rule for printing the announcement of darkness: say "It is now pleasantly lightless in here." instead.

Rule for deciding the scope of the player while in darkness: place the location in scope.

To decide whether in daylight:
    if in darkness, no;
    yes.

Instead of sleeping when in daylight:
    say "You've never been able to sleep with the light on."

Instead of sleeping when the player can touch an audible thing (called the irritant):
    say "The steady [sound of the irritant] from [the irritant] prevents your slumber."

Instead of sleeping when the player can touch a scented thing (called the irritant):
    if the irritant is chocolate, say "The smell of chocolate continues to tantalize

```
you, keeping you from sleep.";
    otherwise say "You sniffle. [The irritant] is probably acting on your allergies."

Instead of sleeping:
    say "You slip easily into the arms of Morpheus.";
    end the story finally saying "At last..."

When play begins:
    say "You have at last escaped from the airport and gotten through customs;
survived an unnerving taxi ride over icy highways; stared down the impertinent
concierge; endured the bellhop's catalog of features in your room; and achieved,
finally, a moment of peace. Time for a good night's slumber!"

Test me with "x dreamcatcher / switch lamp off / look / sleep / eat chocolate /
sleep / get plant / examine plant / open suitcase / put plant in suitcase / close
suitcase / sleep / look / examine bose / switch bose off / sleep".
```

350

Example Ways Out

RB

A status line that lists the available exits from the current location.

A not-uncommon device in games with large maps is a list of available exits printed in the status bar. We might do this so:

```
"Ways Out"

When play begins:
    now left hand status line is "Exits: [exit list]";
    now right hand status line is "[location]".

To say exit list:
    let place be location;
    repeat with way running through directions:
        let place be the room way from the location;
        if place is a room, say " [way]".
```

We may find that printing out full directions makes the status line unpleasantly crowded. Fortunately, it isn't hard to provide a set of abbreviations to use in this context:

```
Rule for printing the name of a direction (called the way) while constructing the
status line:
    choose row with a heading of the way in the Table of Abbreviation;
    say "[shortcut entry]".

Table of Abbreviation

heading    shortcut
north      "N"
northeast  "NE"
northwest  "NW"
east       "E"
southeast  "SE"
```

```
south      "S"
southwest "SW"
west       "W"
up         "U"
down       "D"
inside     "IN"
outside    "OUT"
```

```
Dome is a room. North of Dome is North Chapel. South of the Dome is South Chapel. West of the Dome is Western End. Quiet Corner is northwest of the Dome, north of Western End, and west of North Chapel. Loud Corner is east of North Chapel, northeast of Dome, and north of Eastern End. Eastern End is north of Dim Corner and east of Dome. Dim Corner is southeast of Dome and east of South Chapel. Ruined Corner is southwest of Dome, west of South Chapel, and south of Western End.

The church door is east of Eastern End and west of the Courtyard. The church door is a door.

Test me with "w / n / e / e / s / e".
```

Everywhere else, the names of directions will still be printed out in full in the usual way.

---

351  **Example Guided Tour** *RB*

A status line that lists the available exits from the current location, changing the names of these exits depending on whether the room has been visited or not.

It may sometimes be helpful to prompt the player with a list of exits printed up in the status line. For instance, here is a status line that will print the names of nearby rooms, as well as all the doors the player can see:

```
"Guided Tour"

When play begins:
    now left hand status line is "Nearby: [if a room is adjacent][the list of adjacent rooms][end if][if a room is adjacent and a door is visible] and [end if][if a door is visible][the list of visible doors][end if]";
    now right hand status line is "".
```

Of course, we may not want to tell the player what glories are to be found in locations he hasn't yet explored.

```
Rule for printing the name of an unvisited room (called the target) while constructing the status line:
    let aim be the best route from the location to the target;
    say "something [aim]".
```

Even when we have seen a room, we might still want a reminder about how to get there:

```
After printing the name of a visited room (called the target) while constructing the status line:
    let aim be the best route from the location to the target;
    say " ([aim])".
```

We may also find that printing out full directions makes the status line unpleasantly crowded. Fortunately, it isn't hard to provide a set of abbreviations to use in this context:

```
Rule for printing the name of a direction (called the aim) while constructing the status line:
    choose row with a heading of the aim in the Table of Abbreviation;
    say "[shortcut entry]".

Table of Abbreviation

heading    shortcut
north      "N"
northeast  "NE"
northwest  "NW"
east       "E"
southeast  "SE"
south      "S"
southwest  "SW"
west       "W"
up         "U"
down       "D"
inside     "in"
outside    "out"
```

Everywhere else, the names of directions will still be printed out in full in the usual way. And now we give it a little map to work with:

```
Dome is a room. North of Dome is North Chapel. South of the Dome is South Chapel. West of the Dome is Western End. Quiet Corner is northwest of the Dome, north of Western End, and west of North Chapel. Loud Corner is east of North Chapel, northeast of Dome, and north of Eastern End. Eastern End is north of Dim Corner and east of Dome. Dim Corner is southeast of Dome and east of South Chapel. Ruined Corner is southwest of Dome, west of South Chapel, and south of Western End.

The church door is east of Eastern End and west of the Courtyard. The church door is a door.

Test me with "n / w / s".
```

Note that while this looks fine in some places, other locations exceed the limits of what the status-line can hold: if any given room is going to have a large number of exits, this kind of listing will almost certainly not fit. So apply cautiously.

352

Example **Reflections** *RB*

Emphasizing the reflective quality of shiny objects whenever they are described in the presence of the torch.

"Reflections"

Behind the Waterfall is a room. "Though one wall of the cave is open to the waterfall, the quantity of water is so great that barely any light comes through from the outside." Behind the Waterfall is dark.

Surface is a kind of value. The surfaces are shiny and dull. A thing has a surface. A thing is usually dull.

The player carries a reflecting ball, a canopic jar, an abacus, a plumbline, a piece of chalk, and a torch. The reflecting ball is shiny.

Aladdin's lamp is a shiny thing in Behind the Waterfall.

Brightness is a kind of value. The brightnesses are guttering, weak, radiant and blazing. The torch has a brightness. The torch is blazing. The torch is lit.

Understand "blow out [something]" or "blow [something]" or "extinguish [something]" as blowing out. Blowing out is an action applying to one thing.

Carry out blowing out: say "Futile."

Instead of blowing out the torch:
    now brightness of torch is the brightness before the brightness of the torch;
    say "The light of the torch dies to [brightness of torch]."

Instead of blowing out the guttering torch:
    say "Fool! Do you want to put it out entirely?"

Rule for writing a paragraph about a shiny thing:
    say "The [brightness of the torch] light of [the torch] reflects in the surface[if the number of shiny things in the location > 1]s[end if] of [the list of shiny things in the location]."

Before printing the name of the torch while writing a paragraph about something:
    if the torch is in the location, say "fallen ".

Test me with "drop ball / look / blow torch / look / drop torch / look".

353

Example Emma

RB

Social dynamics in which groups of people form and circulate during a party.

To start with, let's understand "room" to mean "a group of people talking". These groups can grow and shrink as people come and go, so we'll want to name and rename them; and we're also going to need some rules to motivate people moving around, and a description to narrate how they behave when we're with them.

"Emma"

```
by the banquet table is a room. at the corner is a room. next to the doorway is a room. by the window is a room.

Social clump is a kind of value. The social clumps are vacancy, lone person, couple, cluster, group.

A room has a social clump. Understand the social clump property as describing a room.

Before printing the name of a room:
    say "a [social clump] ".

After looking:
    assign clumping;
    say "Elsewhere in the room, you can see [the list of rooms which are not the location]."

Understand "go to [any room]" as joining. Joining is an action applying to one visible thing. Carry out joining: move player to the noun. Report joining: do nothing.

Understand "examine [any room]" as looking toward. Looking toward is an action applying to one visible thing. Carry out looking toward a room: say "In that direction you see [a list of other people in the noun]."

When play begins: assign clumping. Every turn: assign clumping.

To assign clumping:
    repeat with space running through rooms:
        now the social clump of the space is vacancy;
        if the space contains exactly 1 person, now the social clump of the space is Lone person;
        if the space contains exactly 2 people, now the social clump of the space is Couple;
        if the space contains more than 2 people and the space contains fewer than 5 people, now the social clump of the space is cluster;
        if the space contains more than 4 people, now the social clump of the space is group.

The room description heading rule is not listed in the carry out looking rules.

A person has a number called longevity. The longevity of a person is usually 0. A person can be active or passive.

Definition: a person is other if it is not the player.

Every turn:
    repeat with mover running through other people:
        now the mover is active;
        increment the longevity of mover;
        if longevity of mover is greater than 3 or the mover is bored:
            assign value of spaces for the mover;
            let destination be the nicest room;
            if the destination is not the location of the mover:
                if the player can see the mover, say "[The mover] makes excuses and
```

```
drifts off to join [the destination].[paragraph break]";
            move the mover to the destination;
            now the mover is complacent;
            now the longevity of the mover is 0;
            if the player can see the mover, say "[The mover] wanders over.
[paragraph break]";
            assign clumping;
            now mover is passive.
```

A room has a number called attractiveness.

Definition: a room is nice if its attractiveness is 1 or more.

```
To assign value of spaces for (mover - a person):
   repeat with space running through rooms:
      now attractiveness of the space is 0;
      repeat with figure running through people in the space:
         if the mover is bored, decrease attractiveness of the space by 2;
         if the mover likes the figure, increment attractiveness of the space;
         if the mover dislikes the figure, decrement attractiveness of the space;
         if the mover desires the figure, increase attractiveness of the space by 2.
```

Liking relates various people to various people. The verb to like means the liking relation.

Disliking relates various people to various people. The verb to dislike means the disliking relation.

Attraction relates various people to various people. The verb to desire means the attraction relation.

Mr Weston, Mr Woodhouse, Mr Elton, Mr Knightley, and Frank Churchill are men. Mrs Weston, Mrs Bates, Miss Bates, Harriet Smith, Emma Woodhouse, and Jane Fairfax are women.

Harriet Smith likes Mr Elton. Harriet Smith desires Mr Elton. Harriet Smith likes Emma Woodhouse.

Mr Elton desires Emma Woodhouse.

Emma Woodhouse likes Harriet Smith and Mr Knightley. Emma Woodhouse dislikes Jane Fairfax.

Mr Knightley likes Emma Woodhouse, Mr Weston, and Mrs Weston. Mr Knightley desires Emma Woodhouse.

Jane Fairfax desires Frank Churchill. Jane Fairfax likes Frank Churchill.

Frank Churchill desires Jane Fairfax and Emma Woodhouse. Frank Churchill likes Jane Fairfax.

Miss Bates likes Jane Fairfax, Emma Woodhouse, and Mrs Bates.

Mr Weston likes Frank Churchill, Emma, Knightley, and Mrs Weston.

```
Mrs Weston likes Frank Churchill, Emma, Knightley, and Mr Weston.

Mrs Bates likes Miss Bates.

A person can be complacent or bored.

When play begins:
    repeat with character running through other people:
        let space be a random room;
        move character to space.
```

And now we use writing a paragraph about... to describe character behavior in groups, when we join them:

```
Rule for writing a paragraph about Frank Churchill:
    if the location contains a woman (called flirt) who is desired by Frank:
        say "[Frank Churchill] is talking with great animation and slightly more than
becoming warmth to [the flirt][if an unmentioned other person is in the location],
while [the list of unmentioned other people in the location] look on with varying
degrees of amusement or irritation[end if].";
        repeat with character running through people in the location:
            if the character is not Churchill and the character is not the flirt, now the
character is bored.

Rule for writing a paragraph about Mr Elton:
    if the location contains an unmentioned woman (called flirt) who is desired by
Elton:
        say "[Mr Elton] hangs on the sleeve of [the flirt], offering an assortment of
studied gallantries that make you wonder about his good sense.";
        repeat with character running through people in the location:
            if the character is not Elton and the character is not the flirt, now the
character is bored.

Rule for writing a paragraph about Harriet Smith:
    if the location contains Emma and Emma is unmentioned:
        say "[Harriet] and [Emma] are conversing in low tones -- Harriet, apparently,
being too shy to speak so that everyone can hear her."

Rule for writing a paragraph about Mr Knightley:
    if the location contains an unmentioned man (called the listener) who is not Mr
Knightley:
        say "[Mr Knightley] is speaking with [the listener] about agricultural
matters.";
        now the listener is complacent.

Rule for writing a paragraph about Miss Bates:
    say "[Miss Bates] is giggling about the weather[if an unmentioned other
person is in the location]. This does not seem to compel the interest of [the list of
unmentioned other people in the location][end if].";
    repeat with character running through people in the location:
        if the character is not Miss Bates and character is not Mrs Bates, now the
character is bored.
```

Since this is just an example, we'll stop here, but there's no reason we couldn't write such paragraphs for everyone.

Test me with "z / z / z / look / x corner / x doorway / x window / x table / go to the table".

---

354  **Example Air Conditioning is Standard** *RB*

Uses "writing a paragraph about" to make person and object descriptions that vary considerably depending on what else is going on in the room, including some randomized NPC interactions with objects or with each other.

"Air Conditioning is Standard"

Section 1 - The Garage

A person has some text called current occupation. The current occupation of a person is usually "None".

Mood is a kind of value. The moods are bemused, bored, attentive, rapt, and blushing. A person has a mood. A person is usually attentive.

```
Instead of examining a person:
    now every thing is unmentioned;
    carry out the writing a paragraph about activity with the noun.

Rule for writing a paragraph about a person (called X):
    let the subsequent mention be "Name";
    if the current occupation of X is not "None":
        say "[current occupation of X]. ";
        let the subsequent mention be "He";
        if X is female, let the subsequent mention be "She";
    if X wears something unmentioned:
        if the subsequent mention is "Name", say "[The X] ";
        otherwise say "[subsequent mention] ";
        say "is wearing [a list of unmentioned things worn by X]";
        if X carries something unmentioned, say " and carrying [a list of
unmentioned things carried by X]";
        say ".";
    otherwise:
        if X carries something unmentioned:
            if the subsequent mention is "Name", say "[The X] ";
            otherwise say "[subsequent mention] ";
            say " is carrying [a list of unmentioned things carried by X]."
```

The Garage is a room. "Above the street door is a spectacular art nouveau fanlight, wherein a stained-glass Spirit of Progress bestows the gift of Transportation on mankind.

The sun, gleaming through the hair of Progress, throws amber curls on the macadam floor."

The fanlight is scenery in the Garage. The description is "A semi-circle of stained glass as wide as the garage door, designed by Louis Comfort Tiffany himself. No

expense has been spared."

The gift of Transportation is part of the fanlight. The description is "The gift of Transportation is envisioned as a cornucopia disgorging a steam locomotive. And that blue bit of glass might be the Montgolfier balloon."

The Spirit of Progress is part of the fanlight. The description is "It is part of her character to have bare shoulders like that."

The machinist is a bored man. He is in the Garage. He is wearing a grimy pair of overalls. He carries a wrench and a screwdriver. The current occupation of the machinist is "[The machinist] is making some adjustments to [the random thing which is part of the Victorian Car] with his [random thing carried by the machinist]"

The Victorian Car is a device in the Garage. A cast-iron steering wheel, a leather bucket seat, a horn, and a combustion engine are part of the Victorian Car. The seat is an enterable supporter.

Rule for writing a paragraph about a device (called X):
    let the subsequent mention be "Name";
    if the X is unmentioned:
        say "[The X] is here. ";
        let the subsequent mention be "It";
    if something is part of X:
        if the subsequent mention is "Name", say "[The X] ";
        otherwise say "[subsequent mention] ";
        say "[if a mentioned thing is part of X]also [end if]features[if a mentioned thing is part of X], in addition to [the list of mentioned things which are part of X], [end if] [a list of unmentioned things which are part of X]";
        say ".".

Rule for printing the name of the steering wheel while writing a paragraph about a person:
    say "steering wheel".

A supporter has some text called position. The position of a supporter is usually "None".

The Office is west of the Garage. The Office contains a desk. The desk has the position "A [desk] with several dozen drawers stands in the center of the room". On the desk are some papers.

After printing the name of a supporter (called X) which supports an unmentioned thing:
    now X is unmentioned.

Rule for writing a paragraph about a supporter (called X):
    let the subsequent mention be "Name";
    if the position of X is not "None":
        say "[position of X]. ";
        let the subsequent mention be "It";
    if a mentioned thing is on X:
        say "Besides [the list of mentioned things which are on X], ";
        let the subsequent mention be "it";

```
    if the subsequent mention is "Name", say "[The X] ";
    otherwise say "[subsequent mention] ";
    say "holds [a list of unmentioned things which are on X]."

Section 2 - Schedule

The time of day is 4:38 PM.

At 4:42 PM:
    move the machinist to the Office;
    say "The machinist wanders into the Office to get some paperwork.";
    now every thing carried by the machinist is on the desk;
    now the current occupation of the machinist is "[The machinist] rifles through
[the papers] on [the desk]".

At 4:43 PM:
    move the young lady to the Garage;
    if the young lady can be seen by the player,
        say "An attractive young lady walks in from the street, and glances around
as though she has never been here before."

At 4:45 PM:
    if the young lady can be seen by the player,
        say "With a not-quite-convincing air of innocence, [the young lady] happens
to lean upon [the horn], which bleats loudly.";
    otherwise say "There is a honk from the Garage[if the machinist can be seen
by the player]. The machinist looks up with a frown[end if].";
    now the horn is mentioned.

At 4:46 PM:
    move the machinist to the Garage;
    say "The machinist strolls from the Office into the Garage to find out what is
going on.";
    now the current occupation of the machinist is "[The machinist] is chatting with
[the young lady]. He seems to be demonstrating the various features of [the car],
including [the random thing which is part of the car]";
    now the current occupation of the young lady is "[The young lady] is asking
[the machinist] a number of questions about [the car]".

At 4:49 PM:
    if the young lady can be seen by the player, say "[The machinist] gives [the
young lady] his arm to climb into [the seat].";
    move the young lady to the seat;
    now the young lady is rapt;
    now the current occupation of the young lady is "[The young lady] is turning
[the steering wheel] from side to side";
    now the current occupation of the machinist is "[The machinist] is leaning on
the door of [the car], pointing out features to [the young lady]";
    move the besotted expression to the machinist;
    now the machinist is wearing the besotted expression.

At 4:52 PM:
    now the sober grey gown is unbuttoned at the neck;
    if the young lady can be seen by the player, say "[The young lady] murmurs
something about the wilting heat, and undoes a button or two of her gown. The
machinist's expression is comical, or would be, if you weren't annoyed."
```

```
Every turn when the player is in the Garage and young lady is on the seat:
    say "You are beginning to feel a little unnecessary in this scene."

Every turn when the player is in the Office and the young lady is on the seat:
    say "There's no sound at all from the other room, not even conversation."

Before going to the Garage when the young lady is on the seat:
    now the sober grey gown is tellingly dishevelled;
    move the young lady to the Garage;
    now the young lady is blushing;
    say "There is a flurry of movement as you enter the room.";
    now the current occupation of the young lady is "[The young lady] stands near
the door, tapping her foot nervously";
    now the besotted expression is nowhere;
    now the current occupation of the machinist is "[The machinist] is leaning
against [the car], looking smug".

Section 3 - Initially Out of Play

The besotted expression is a wearable thing. The description is "It looks foolish,
doesn't it?"

The young lady is a bemused woman. She is wearing a sober grey gown and a
pair of black boots. The current occupation of the young lady is "[The young
lady] is running a gloved finger along the chassis of [the victorian car]"

Before printing the name of the young lady while writing a paragraph about a
person:
    say "[mood of the young lady] "

The description of the grey gown is "Something about the perfect row of tiny
buttons has the wrong effect -- at any rate, it is natural to wonder how long they
take to undo." The gown can be buttoned almost to the chin, unbuttoned at the
neck, or tellingly dishevelled.

Rule for printing the name of the gown when writing a paragraph about a person:
    say "sober grey gown ([sober grey gown condition])"

Test me with "z / look / look / z / look / west / east / z / look / z / look / z / look /
west / east".
```

355

Example Rip Van Winkle

RB

A simple way to allow objects in certain places to be described in the room description body text rather than in paragraphs following the room description.

There are times when, for greater elegance of prose, we'd like to mention an object in the main body text of a room. For instance:

```
Here is a lovely, secluded fold in the mountains, far from civilization: as though to
prove it, Rip Van Winkle is sleeping under a tree.
```

As we've already seen, that's no problem if Rip is scenery. He'll stay there motionless.

But what if something in the game allows Rip to wake up? Or what if we want to use the same technique on a portable object that the player should be allowed to take? Clearly in that case it's not appropriate to make the mentioned thing be scenery, and at the same time, we need to keep Inform from adding a superfluous

```
You can see Rip Van Winkle here.
```

to the end of our description.

Here is how:

```
"Rip Van Winkle"

A person can be asleep.

The Catskills is a room. "Here is a lovely, secluded fold in the mountains, far
from civilization[if Rip Van Winkle is asleep]: as though to prove it, Rip Van
Winkle is sleeping under a tree[end if]."

A tree is scenery in the Catskills.

Rip Van Winkle is a man in the Catskills. Rip Van Winkle is asleep.

Before listing nondescript items of the Catskills:
    if Rip Van Winkle is marked for listing:
        now Rip Van Winkle is not marked for listing;
        if Rip Van Winkle is not asleep,
            say "Rip Van Winkle stands here, looking mightily confused."

Instead of waiting:
    say "Rip Van Winkle wakes up with a snort.";
    now Rip Van Winkle is not asleep.

Test me with "look / z / look".
```

356

Example Happy Hour

RB

Listing visible characters as a group, then giving some followup details in the same paragraph about specific ones.

Often it is best to have an entire paragraph about the characters present in a room, but suppose we're narrating a large party with a lot of people moving around. In that case, it might be better to list everyone together, then add a few salient details by way of follow-up, like this:

```
"Happy Hour"
```

```
Before listing nondescript items:
    say "You can see [a list of people who are marked for listing] here. ";
    repeat with named party running through people:
        now the named party is not marked for listing;
    let count be the number of visible other people who are carrying something;
    if count is 0:
        say paragraph break;
        continue the action;
    let index be count;
    repeat with holder running through visible other people who are carrying
something:
        if index is count, say "[The holder]";
        otherwise say "[the holder]";
        say " has [a list of things carried by the holder]";
        decrement index;
        make delimiter index of count.
```

The next part could be simpler, but for rigor we will write it in such a way that it will work whether or not the serial comma is in use. This requires some extra work.

```
To make delimiter (index - a number) of (count - a number), continuing or halting:
    if index is 0:
        if continuing, say ". [run paragraph on]";
        otherwise say ".";
    otherwise if index is 1:
        if count is 2, say " and ";
        otherwise say "[optional comma] and ";
    otherwise:
        say ", ".

To say optional comma:
    if the serial comma option is active:
        say ",".
```

And now the scene:

```
The Banquet Hall is a room. "A large cheery banner over the door (which,
incidentally, vanishes when you approach it) reads: HELLO NEW INDUCTEES!
WELCOME TO THE AFTERLIFE!"

Fred, George, and Larry are men in the Banquet Hall. Fred carries a dry martini.
Larry carries a cream puff. Matilda and Louise are women in the Banquet Hall.

Definition: a person is other if it is not the player.

Every turn:
    let wanderer be a random other person;
    let place be the holder of the wanderer;
    let next place be a random room adjacent to the place;
    let the way be the best route from the place to the next place;
    try the wanderer going the way.

The Kitchen is west of the Banquet Hall. "Dominated by a pile of dirty plates
which you imagine it will be someone's privilege to wash, later." Vanessa is a
```

```
woman in the Kitchen. Vanessa carries a tray. On the tray is a salmon roll. The
roll is edible.

Test me with "z / look / g / g / g ".
```

357

Example The Eye of the Idol

RB

A systematic way to allow objects in certain places to be described in the room description body text rather than in paragraphs following the room description, and to control whether supporters list their contents or not.

```
"The Eye of the Idol"

Section 1 - Reusable Material
```

We start by defining relations that let us know where items "belong", with the understanding that if something is where it belongs, it will be described in the main room description and therefore should not be separately listed. Thus:

```
Positioning relates various things to various things. The verb to be placed in
means the positioning relation. The verb to be placed on implies the positioning
relation.

Room-positioning relates various things to various rooms. The verb to be room-
placed in means the room-positioning relation.
```

We can't make relations relate various objects to various objects, and rooms are not things, so two separate cases are necessary. An alternative approach would be to say "A thing has an object called the initial placement", which would allow a thing to have an initial placement that was a room, a supporter, or a container; an advantage of using relations, though, is that that way we can if we like specify multiple placements for the same object, so that, e.g., a sparkling diamond can be described in the main description paragraph as "half-buried in dust" in the beginning of the game, and then at the end as "in the eye of the idol" at the end.

Now we define, based on these relations, an "in-place" adjective, which will identify whether something is in a location which will specially describe it:

```
Definition: a thing (called prop) is in-place:
    if the prop is in the location and the prop is room-placed in the location, yes;
    if the holder of the prop is a thing and the prop is placed in the holder of the
prop, yes;
    no.

Definition: a thing is out-of-place if it is not in-place.
```

With that done, removing these items automatically from the room description is actually pretty easy:

```
Before listing nondescript items:
    now every marked for listing in-place thing is not marked for listing.
```

One tricky case remains: when something is placed on a supporter that is scenery, it can be mentioned even if we have marked that object "not marked for listing". What matters here is not whether the object itself is marked for listing but whether the supporter has been "mentioned". (A fuller description of how room descriptions are assembled is available in the Looking section of the Commands chapter in the Recipe Book.) So let's also add a feature whereby we can easily suppress the descriptions of these supporters when appropriate:

```
A supporter can be quiet.
```

A quiet supporter is one that is never mentioned itself and which only mentions its contents if they are out of place. This allows for maximum flexibility in incorporating it into the body of room descriptions.

```
Rule for writing a paragraph about a quiet supporter (called chosen table):
    if an out-of-place thing is on the chosen table:
        if an in-place thing is on the chosen table,
            say "On [the chosen table], in addition to [the list of in-place things on the
chosen table], [is-are a list of out-of-place things which are on the chosen
table].";
        otherwise say "On [a chosen table] [is-are a list of out-of-place things which
are on the chosen table].";
    now the chosen table is mentioned.
```

Notice that we can still override this with writing a paragraph rules about specific supporters in our game, if we decide that we want something a little different in some cases.

Now, an example to test this out:

```
Section 2 - A Sample Scenario

The Sand-Floored Chamber is a room. "The constant wind has filled this
chamber with a layer of fine red sand, as soft as powder snow[if the diamond is
in the Sand-floored Chamber]. Something sparkling is half-buried in the
corner[end if]. A doorway lies open to the north."

The sparkling diamond is in the Sand-floored Chamber. The sparkling diamond
is room-placed in the Sand-floored Chamber. The description is "It is a vast
diamond; the front is faceted, the back smoothed to fit in some sort of socket."

The Hexagonal Temple is north of the Sand-Floored Chamber. "The temple walls
are great ashlar blocks rising to a hundred feet overhead, perhaps more; the roof
is a scarlet awning only, through which the sun filters down in blood hues.
Overseeing all is a sculpture in stone and ivory[if the sparkling diamond is in the
idol's eye], in whose single eye a vast diamond gleams[end if][mat-and-incense
text].".

To say mat-and-incense text:
    if the mat is in the Temple and the incense stick is on the pedestal:
```

```
        say ". A prayer mat at the idol's feet, and an incense stick still burning on
    the pedestal, indicate that someone was only recently consigning her grievances
    to the care of the deity";
    otherwise if the mat is in the Temple:
        say ". At the idol's feet, some worshipper has left a prayer mat";
    otherwise if the incense stick is on the pedestal:
        say ". At the idol's side is a pedestal, on which incense still smolders".
```

We could have done all this with text conditions in the main room description, but it becomes difficult to read when there are too many conditions operating in the same text property, so we break it out into a clearer set of conditions.

```
The idol is scenery in the Hexagonal Temple. Understand "sculpture" or "stone"
or "ivory" as the idol. The description is "The idol is perhaps three times the
height of an ordinary man."

The idol's eye is part of the idol. It is a container. The description is "[if the
diamond is in the idol's eye]It gleams with purpose and righteous
wrath[otherwise]A round socket in the center of the idol's forehead from which
something seems to be missing[end if]."

The pedestal is a quiet supporter in the Hexagonal Temple. On the pedestal is
an incense stick. The incense stick is placed on the pedestal.

A mat is in the Hexagonal Temple. It is room-placed in the Hexagonal Temple.
The description is "Woven of assorted grasses."

Test me with "get diamond / look / n / get mat / look / drop diamond / look / get
diamond / put diamond in eye / look / get incense / look / drop mat / look / get
mat / put mat on pedestal / look / put incense on pedestal / look".
```

358

Example **Priority Lab**

RB

A debugging rule useful for checking the priorities of objects about to be listed.

When it comes time to start manipulating the priorities of items, it is useful to be able to check the table for debugging purposes; the problem is that printing the names of the objects can itself affect the way the room description is generated, foiling our debugging efforts.

What follows is a rule to help with debugging safely, and a sample of how priorities work:

```
"Priority Lab"

Section 1 - Procedure

Before printing the locale description (this is the dump locale table rule):
    say "Locale Priority list:";
    repeat through Table of Locale Priorities:
```

```
        let the flag be whether or not the notable-object entry is mentioned;
        say "[line break] [notable-object entry]: [locale description priority entry]";
        if the flag is false, now the notable-object entry is not mentioned;
    say line break.
```

Now, let's look at some items put in a specific order. Things with low priority numbers list towards the beginning; things with high priority numbers list towards the end. (It helps to think of it as though we were making a numbered list of the paragraphs to appear in the description.) Anything numbered 0 doesn't appear at all, and the default priority of an object is 1.

```
A thing can be early-described, late-described, latest-described, never-
described, sightline-described, or ordinarily-described. A thing is usually
ordinarily-described.

After choosing notable locale objects (this is the apply early and late description
rule):
    repeat with item running through early-described things:
        if there is a notable-object of item in the Table of Locale Priorities:
            set the locale priority of the item to 1; [list before everything else -- this
would work with any number lower than 5 and higher than 0]
    repeat with item running through late-described things:
        if there is a notable-object of item in the Table of Locale Priorities:
            set the locale priority of the item to 10; [list after everything else -- this
would work with any number larger than 5]
    repeat with item running through never-described things:
        set the locale priority of the item to 0; [don't list at all]
    continue the activity.
```

An important cautionary note: priorities are only honored if the objects are going to get their own paragraphs (with "writing a paragraph about..." or because they have initial appearances). Priorities do not affect the order in which items appear in the final "You can see..." list, except that items with priority 0 or lower are omitted. (If we want to order the items in that list, we may want to resort to the Complex Listing extension by Emily Short.)

There are further refinements available to us: for instance, we could make some things that are only visible if the player is raised above ground level.

```
After choosing notable locale objects (this is the sightline-described things are
visible from supporters rule):
    if the player is not on a supporter:
        repeat with item running through sightline-described things:
            if there is a notable-object of item in the Table of Locale Priorities:
                set the locale priority of the item to 0; [remove objects that can only be
seen from higher objects.]
    continue the activity.
```

It may also be useful to know about the "parameter-object", which refers to the thing whose contents we are currently describing: the standard rules consider how to describe the contents of the location and then also check the contents of any supporter or container the player may be inside, so in the first case "parameter-object" would be the location, and then in the second the supporter in question.

In practice this is rarely useful, but should we need to change priorities in the case of both player and object being inside a particular container, we might make use of it, for instance:

```
A thing can be tasteful or icky. A thing is usually tasteful.

After choosing notable locale objects (this is the icky things next to players rule):
    if the player is on the parameter-object:
        repeat with item running through icky things :
            if there is a notable-object of item in the Table of Locale Priorities:
                set the locale priority of the item to 10; [remove objects that can only
be seen from higher objects.]
    continue the activity.
```

The other thing to note is that by default that final collection of generic objects ("You can also see...") appears at the end, regardless of the priority of everything else. If we really wanted to, though, we could force something to appear even after that paragraph, by adding a new listing rule to the locale description rules:

```
After choosing notable locale objects (this is the latest-described items priority
rule):
    repeat with item running through latest-described things:
        if the item is a notable-object listed in the Table of Locale Priorities:
            now the item is mentioned;
            now the item is marked for late listing.

The late listing rule is listed after the you-can-also-see rule in the for printing the
locale description rules.

A thing can be marked for late listing. A thing is usually not marked for late
listing.

This is the late listing rule:
    if something is marked for late listing:
        say "Oh! And also [a list of things which are marked for late listing].";
        now everything is not marked for late listing;
    continue the activity.

Section 2 - Scenario

The Priority Lab is a room. The early bird, the worm, the leaf, the unseen object,
the pebble, the twig, and the late edition are things in the Priority Lab.

The early bird is early-described. The late edition is late-described. The unseen
object is never-described.

The worm is icky.

The high window is in Priority Lab. It is sightline-described and fixed in place.
The initial appearance of the high window is "There's a tiny high window up near
the ceiling that you can't see unless you're on top of something."
```

In order for the priorities we just set to be interesting, let's give out some initial appearances and writing a paragraph rules:

```
The initial appearance of the worm is "A worm inches along the ground."
The initial appearance of the late edition is "Finally, the late edition lies at your
feet."

Rule for writing a paragraph about the early bird when the early bird is in a room:
say "The early bird always appears first, and here it is."
Rule for writing a paragraph about the leaf: say "Look, there's [a leaf][unless the
leaf is in the location] on [the holder of the leaf][end if]!"
Rule for writing a paragraph about an icky thing (called icky item) which is on
something which supports the player: say "Ew, [an icky item] is right next to you."
```

This procedure also means (as you can test by experiment) that after the late edition has been picked up and dropped again, it lists in no special order in the "you can see..." paragraph (since initial appearances only print when the object has not yet been moved).

```
The afterthought is a thing in the Priority Lab. It is latest-described.

The bar stool is an enterable supporter in Priority Lab.

Test me with "get leaf / drop leaf / look / x unseen object / get pebble / look / get
twig / look / get afterthought / look / drop twig / look / get late edition / look / drop
late edition / sit on bar stool / look / get all / put all on stool / look".
```

359

Example Low Light

RB

An object that is only visible and manipulable when a bright light fixture is on.

Suppose we want a different treatment of lighting than the usual: the room isn't totally dark, but there's something we can't see unless we turn on a bright light.

```
"Low Light"
```

First we make our environment and its light:

```
The Workroom is a room. The desk is in the Workroom. The brilliant lamp is a
device on the desk.

To decide whether the light level is high:
    if the brilliant lamp is switched off, no;
    if the player cannot see the brilliant lamp, no;
    yes.

To decide whether the light level is low:
    if the light level is high, no;
    yes.
```

Now we make a shadow so that the player can only refer to it if the shadow is in inventory or the light is on:

```
The shadow is a privately-named thing on the desk.

Understand "barely-visible" or "barely visible" or "shadow" as the shadow when
the light level is high. Understand "invisible" or "shadow" as the shadow when
the player encloses the shadow.
```

And finally a couple of extra touches to make it clear why we're able to interact with the shadow when it's in inventory, even if the light is low:

```
Before printing the name of the shadow:
    if the light level is high:
        say "barely-visible ";
    otherwise if the player encloses the shadow:
        say "invisible (but tangible) "

After dropping the shadow when the light level is low:
    say "You let it go and it fades into the ambient gloom."
```

To handle the appearance of the object, we want to set its locale priority to 0: that will prevent it being named in room descriptions.

```
After choosing notable locale objects:
    unless the light level is high:
        set locale priority of the shadow to 0.

Test me with "look / get shadow / turn on lamp / look / get shadow / i / turn off
lamp / i / drop shadow / look / get shadow / turn on lamp / look".
```

360

Example Casino Banale

RB

Creating room descriptions and object descriptions that change as the player learns new facts and pieces things together.

In a work of interactive fiction that involves many new discoveries, we might want to change the way we narrate room descriptions and describe objects as the player learns new information.

One approach to this is to create a model of the facts we want the player to find out, and attach some narrative text to each. When a fact becomes relevant to the story, that narrative text is shown to the player. So:

```
"Casino Banale"

Section 1 - Procedure
```

First we create the concept of facts, and the idea that facts can make some things more important than others.

```
A fact is a kind of thing. A fact can be known or unknown. A fact can be ready to
learn or hidden. A fact has some text called the narration.

Definition: a thing is narratively significant if it conveys an interesting fact.

Definition: a thing is narratively dull if it is not narratively significant.

Conveyance relates various things to various facts. The verb to convey means
the conveyance relation.

Definition: a fact is interesting if it is unknown and it is ready to learn.
```

Now, we also need a way to tell Inform to introduce certain new facts when the right previous ones have been introduced. We'll create a "following" relation, according to which a new fact can be told to the player when the player has already learned all the facts it follows. This way, we can simulate the effect of putting together several pieces of evidence to come to a conclusion:

```
Following relates various facts to various facts. The verb to follow means the
following relation.

To say (new fact - a fact):
    say "[narration of the new fact]";
    now the new fact is known;
    repeat with possible outcome running through facts which follow the new fact:
        if every fact which is followed by possible outcome is known:
            now the possible outcome is ready to learn.
```

Next we need a way for the game to introduce these new facts. Let's say we want them to come up when the player examines something appropriate, or sees it in the room:

```
After examining something which conveys an interesting fact (called discovery):
    say "[discovery][paragraph break]".

After choosing notable locale objects:
    repeat through the Table of Locale Priorities:
        if the notable-object entry is narratively significant:
            set the locale priority of the notable-object entry to 1.

For writing a paragraph about a narratively significant thing (called item):
    now the item is mentioned;
    let chosen fact be a random interesting fact which is conveyed by the item;
    say "[chosen fact][paragraph break]".
```

The "after choosing notable locale objects" line here handles things so that any interesting conclusions we want to draw are always given first, followed by the less interesting description.

And finally, we need to give the player a little evidence to piece together:

```
Section 2 - Scenario
```

```
The Casino is a room.

Frince is a man in the Casino. The description is "Frince is a friend of yours -- if you reckon friendship on the same terms that one reckons a cat as a pet. He spends time with you when he wants to, but if your wishes or convenience ever run counter to a whim of his, it's the whim that wins. Always. [paragraph break]He's also wearing a somewhat ludicrous shirt."

Frince wears a ludicrous shirt. The description of the ludicrous shirt is "Fine white fabric with satiny white pinstripes: it's that expensive, effeminate look that Frince is so fond of, and which -- combined with his name -- gives people completely the wrong idea about him."

Tim is a man in the Casino. The description is "You don't know Tim well. Kind of wall-flowerish. The only thing that seems to excite him is craps."

Penny is a woman in the Casino. The description is "Loud. Brash. Hot, probably, if you can look past the loud and brash."

Rule for writing a paragraph about a narratively dull person:
    let is-are-n be "is";
    if the number of unmentioned narratively dull people is not 1:
        let is-are-n be "are";
    say "[A list of unmentioned narratively dull people] [is-are-n] [one of]watching the croupier[or]following the spin of the roulette[or]chattering[at random][one of] breathlessly[or] impatiently[or][at random]."

Penny-annoying is a fact.
    It is ready to learn.
    The narration is "[if looking]Penny grimaces at you-- [end if]Penny is the same woman who stepped on your toe in the buffet line. The third time, she blurted, 'You have big shoes, don't you?'"
    Penny conveys penny-annoying.

lipstick-smudges is a fact.
    It is ready to learn.
    The narration is "There are a couple of smudges of coral-colored lipstick on the collar."
    The ludicrous shirt conveys lipstick-smudges.

penny-wears-coral is a fact.
    It follows penny-annoying.
    The narration is "[if looking]Penny catches your eye again. [end if]The bright coral lipstick was really not a wise choice."
    Penny conveys penny-wears-coral.

Affair-with-penny is a fact.
    It follows lipstick-smudges and penny-wears-coral.
    The narration is "You avoid [if examining Frince]his[otherwise]Frince's[end if] eye. You need some time to adjust to the image of him making out with Penny in a storage closet before you can talk to him without appalled giggling."
    Frince conveys affair-with-Penny.

Test me with "x penny / x frince / x shirt / look".
```

361

Example Kiwi

RB

Creating a raised supporter kind whose contents the player can't see or take from the ground.

Suppose we want there to be some high shelves in our game, which the player can't get at unless he's standing on a prop of some kind. (This is a pretty hoary and over-used puzzle, but there may still be occasions when it becomes useful again.)

In order to resolve this, we want to set up a raised supporter kind. When something is on a raised supporter, it should be mentioned to the player only if the player is in the right position (i.e., standing on something) and otherwise omitted from the description entirely.

```
"Kiwi"

Section 1 - Procedure

A raised supporter is a kind of supporter.

For printing a locale paragraph about a raised supporter (called the high place):
    if the player is on a supporter (called the riser):
        say "Up on [the high place] (and only visible because you're on [the riser])
[is-are a list of things on the high place].";
    otherwise:
        say "The [high place] is above you."
```

Note that here we don't continue the activity because we want to completely replace the normal behavior of describing what is on supporters.

```
Definition: a thing (called target item) is out of reach:
    if the player is on a supporter, no;
    if the target item is on a raised supporter, yes;
    no.
```

Now we also need to prevent the player from interacting with things that are out of reach:

```
Before doing something:
    if the noun is out of reach or the second noun is out of reach:
        say "You can't reach from down here." instead.
```

...or restoring things to the shelves while the player is in the wrong position...

```
Instead of putting something on a raised supporter when the player is not on a
supporter:
    say "You can't reach from down here."
```

And raised supporters shouldn't be searchable from the ground either:

```
Instead of searching or examining a raised supporter when the player is not on a
supporter:
    say "You can't see from down here."
```

Finally, we need to tackle the case where the player types GET ALL FROM SHELF, because we don't want to list the objects up there if the player can't even see them. We use a rule for deciding whether all includes in order to tell Inform not to consider items that can't be reached, and then we adjust the parser error so that it's a little more instructive than "There are none at all available!", which is what the response would otherwise be:

```
Disallowed-all is a truth state that varies. Disallowed-all is false.

Rule for deciding whether all includes an out of reach thing:
    now disallowed-all is true;
    it does not.

Rule for printing a parser error when the latest parser error is the nothing to do
error and the player is not on a supporter:
    if disallowed-all is true:
        say "Whatever might be up there, you can't see or reach from down here.";
    otherwise:
        make no decision.

A first action-processing rule:
    now disallowed-all is false.

Section 2 - Scenario

The Bottom of the Nursery is a room. "Ever since you ate that mysterious cake,
you've been even shorter than usual."

The high shelf is a raised supporter in the Nursery. It is scenery. On the high
shelf are a kiwi-green ball and a stuffed dodo.

The step-ladder is an enterable supporter in the Nursery. Understand "ladder" as
the step-ladder.

Test me with "x shelf / search shelf / get dodo / get all from shelf / stand on
ladder / get all from shelf / search shelf / get off / put all on shelf / get all from
shelf / stand on ladder / put all on shelf".
```

362

Example Copper River

RB

Manipulating room descriptions so that only interesting items are mentioned, while objects that are present but not currently useful to the player are ignored.

In a very dense environment, we might want to offer the player room descriptions in which only the currently-interesting items are mentioned, while other objects are suppressed even if they are present. In effect, this takes the idea of scenery and

makes it more flexible: different things might become background objects or foreground objects at different times during play.

There are a wide range of possible reasons to do this -- to shift the narrative emphasis, to change the mood of the game by highlighting different parts of the environment, to show the game from the perspective of different viewpoint characters -- but in the following example, our goal is to show the player only the objects that are currently useful for puzzles.

To do this, we need some notion of what puzzles are currently available and unsolved, so we make an "unsolved" adjective; we also need to know which things solve the puzzle, so we create a "resolving" relation, to indicate which objects resolve which problems.

Given that information, we can create rules about which objects in the game world are currently interesting, which are currently dull, and describe accordingly:

```
"Copper River"

Use scoring.

Section 1 - Procedure

Resolving relates various things to various things. The verb to resolve means the
resolving relation.

Definition: a thing is interesting if it is not dull.

Definition: a person is dull:
    no.

Definition: a thing is dull:
    if it is unsolved, no;
    if it resolves an unsolved thing, no;
    yes.

Definition: a supporter is dull:
    if it is unsolved, no;
    if it resolves an unsolved thing, no;
    if it supports an interesting thing, no;
    yes.

Definition: a container is dull:
    if it is unsolved, no;
    if it resolves an unsolved thing, no;
    if it contains an interesting thing, no;
    yes.

After choosing notable locale objects:
    repeat with item running through unsolved things:
        set the locale priority of the item to 1.
```

```
For printing a locale paragraph about a dull thing (called item):
    now the item is mentioned.

Before printing a locale paragraph about a supporter (called item):
    now every dull thing on the item is mentioned.

Before printing a locale paragraph about a container (called item):
    now every dull thing on the item is mentioned.

Instead of searching a supporter:
    if the noun supports something interesting:
        say "[A list of interesting things on the noun] [are] on [the noun]";
        if the noun supports something dull:
            say " (alongside [a list of dull things on the noun])";
        say ".";
    otherwise if the noun supports something dull:
        say "There's nothing very useful here, only [a list of dull things on the
noun].";
    otherwise:
        say "[The noun] [are] completely bare."

Instead of searching a container:
    if the noun contains something interesting:
        say "[A list of interesting things in the noun] [are] in [the noun]";
        if the noun contains something dull:
            say " (alongside [a list of dull things in the noun])";
        say ".";
    otherwise if the noun contains something dull:
        say "There's nothing very useful here, only [a list of dull things in the
noun].";
    otherwise:
        say "[The noun] [are] completely empty."

Before listing contents when not taking inventory: group dull things together.

Rule for grouping together dull things: say "assorted dull items".
```

Section 2 - Scenario World and Objects

```
The Kitchen is a room. "Your Aunt Fiona's kitchen looks as though it has been at
the eye of a glitter storm. Fine, sparkling grit dusts every surface. The appliances
are slightly askew, too, as though they hadn't quite settled after a vigorous
earthquake."

The shelf is a scenery supporter in the Kitchen. On the shelf is a can of beans, a
can of potato leek soup, and a tin of deflating powder.

The cabinet is a scenery container in the Kitchen. In the cabinet is a book of
matches, a bottle of descaling solution, a fish hook, and a rusty knife. It is
openable and closed.

The counter is a scenery supporter in the Kitchen. On the counter is an espresso
machine, a blender, and a mortar. The blender and the mortar are containers. In
the mortar is a pestle. Understand "countertop" as the counter.
```

The stove is a scenery supporter in the Kitchen. The oven is part of the stove. The oven is a closed openable container.

The refrigerator is a fixed in place container in the Kitchen.
    Understand "fridge" as the refrigerator.
    The description is "The refrigerator is a dull blue-green, and has a puffy, marshmallow texture on the outside, which means that it's no good for sticking magnets to. Aunt Fiona has never been willing to explain where she got it." The refrigerator is openable and closed.

In the refrigerator are a bottle of ice wine, a bag of carrot sticks, and an egg.

Aunt Fiona is a woman in the Kitchen. Aunt Fiona can be inflated or deflated. Aunt Fiona is inflated. "[if Aunt Fiona is inflated]Aunt Fiona stands nearby. Or perhaps 'stands' is the wrong word: she has been sort of puffed up in her own skin like a balloon, and is now propped in a corner of the room with her head lolling back[otherwise]Aunt Fiona stands -- on her own two slender legs -- at the center of the room[end if]."

Every turn when Fiona is unsolved and Fiona can see the player:
    if a random chance of 1 in 3 succeeds:
        say "[one of]Aunt Fiona's eyes follow you, wide and desperate, but it doesn't look like she's able to do anything[or]Aunt Fiona is still looking reproachful[or]A faint gurgling comes from Aunt Fiona[or]Aunt Fiona makes a funny croak noise[or]Aunt Fiona is still having trouble speaking. Perhaps her throat is as swollen as the rest of her[or]Aunt Fiona twitches[stopping]."

There is a thing called a salmon. Understand "fish" as the salmon. The salmon can be scaly or prepared. The salmon is scaly. The description is "[if scaly]It looks delicious, but is still covered with scales[otherwise]The salmon has been scaled and is ready to eat[end if]."

Before printing the name of the salmon when the salmon is scaly:
    say "very scaly ".

Section 3 - Scenario Puzzles

Definition: Aunt Fiona is unsolved if she is inflated.
Definition: the salmon is unsolved:
    if the salmon is off-stage, no;
    if the salmon is scaly, yes;
    no.

The deflating powder resolves Aunt Fiona.

Instead of putting the deflating powder on Aunt Fiona:
    try throwing the deflating powder at Aunt Fiona.
Instead of giving the deflating powder to Aunt Fiona:
    try throwing the deflating powder at Aunt Fiona.

Instead of throwing the deflating powder at Aunt Fiona:
    if Aunt Fiona is inflated:
        say "You toss some of the powder in Aunt Fiona's direction, and with a sudden gaseous HUFF! she returns to her usual shape and size. [paragraph break]'Well!' she says, brushing herself off. 'That was bracing!' [paragraph

```
break]You give her an embarrassed smile, to apologize for not curing her
faster.";
        now Aunt Fiona is deflated;
        increase the score by 2;
    otherwise:
        say "[one of]You throw another hefty dose of the powder at your aunt.
[paragraph break]'Thank you, child,' she says, sneezing. 'But I think you've done
enough now.'[or]You throw another hefty dose of the powder at your aunt.
[paragraph break]'You're too kind,' she wheezes, through a cloud of glittering
dust.[or]You've probably done enough with the powder.[stopping]".

Every turn when Aunt Fiona is deflated and the salmon is off-stage:
    move the salmon to the counter;
    say "'At least they didn't get this,' she says, producing from somewhere on her
person a fresh-caught salmon. An odd pattern around its eye sockets makes it
looks comically as though it wears spectacles. 'It's the Salmon of Knowledge,'
she explains casually. 'We just need to scale and cook it.'"

The bottle of descaling solution resolves the salmon.

Does the player mean putting the descaling solution on the fish hook: it is
unlikely.
Does the player mean putting the descaling solution on the salmon: it is very
likely.

Instead of putting the bottle of descaling solution on the salmon:
    if the salmon is scaly:
        now the salmon is prepared;
        say "With just a single squirt of the descaling solution (which confusingly
has a picture of bathroom tiles on the label), you remove the scales from the
salmon, leaving its pink flesh ready for preparation.";
        increase the score by 2;
    otherwise:
        say "'Don't do that,' Aunt Fiona warns you. 'Excessive applications could
damage the flesh.'"

Test me with "look / get powder / drop powder / look / look in cabinet / get
powder / put powder on fiona / look / open cabinet / look in cabinet / get solution
/ open fridge / put solution in fridge / look / get solution / put solution on salmon /
look".
```

363

Example **Peeled**

RB

Two different approaches to adjusting what the player can interact with, compared.

Suppose we have a situation where the player is in darkness, but is allowed to feel and interact with (except for examining) any large objects. In that case, we write a scope rule that puts those large objects into scope all the time, and trust the "requires light" aspect of verbs like examining to prevent the player from doing any actions that he shouldn't:

```
"Peeled"

A thing can be large or small.

Before touching a large thing when in darkness: say "You grope for [the noun]..."

After deciding the scope of the player:
    repeat with item running through large things in the location:
        place item in scope.

Some generic surroundings are backdrop. They are everywhere. Understand
"walls" or "wall" or "ceiling" or "ground" or "floor" or "area" or "room" or "here" as
the generic surroundings. Instead of touching the generic surroundings: say "You
encounter nothing extraordinary." Instead of touching the generic surroundings
when in darkness: say "You try feeling your way around and reach [a list of large
things in the location]." After deciding the scope of the player when in darkness:
place the surroundings in scope.

The Room of Mystery is a dark room. The bearskin rug is a large thing in the
Room of Mystery. Instead of touching the rug: say "It feels furry!"

The peeled grape is a small thing in the Room of Mystery. Instead of touching
the peeled grape: say "Gosh, is that an eyeball?"

Test me with "feel floor / feel rug / eat rug / examine rug / get grape".
```

Sadly, because the grape is small, the player will never encounter this horror.

Alternatively, suppose we have a situation in which the player can use one command to interact with a kind of thing that isn't normally in scope. It's usually most convenient to write the "understand" rule appropriately rather than use the scope activity.

(Note that we define "inquiring about" as applying to one *visible* thing; otherwise we would be required to be able to touch the catsuit in order to inquire about it. More on this restriction may be found in the Advanced Actions chapter on the topic of visible, touchable, and carried things.)

```
"Peeled"

Mr Steed's Flat is a room.

Understand "ask about [any subject]" as inquiring about. A subject is a kind of
thing. The skintight catsuit is a subject. Inquiring about is an action applying to
one visible thing.

Carry out inquiring about something:
    say "'What can you tell me about [the noun]?' you demand. Mr Steed raises
his eyebrows, but does not reply."

Test me with "ask about catsuit / x catsuit".
```

All this said, there do arise certain complex situations when we want an activity-specific scoping.

364

Example Four Stars 2

RB

Using "deciding the scope" to change the content of lists such as "the list of audible things which can be touched by the player".

As we have seen, a well-written understand rule will often solve the problem of allowing the player to apply specific actions to objects not normally in scope. When we need to adjust scope for some other reason than reading the player's command, though, "deciding the scope of..." may come in handy.

For instance, suppose we wanted to extend Four Stars 1 to add a tomcat on the balcony that will be heard whenever the player listens from the next room, as in:

```
>listen
You hear the soothing whalesong from the Bose speaker and the yowling from
the tomcat.
```

To do this, we need to make sure that in the rule that assembles our listening description,

```
Instead of listening to a room:
    if an audible thing can be touched by the player, say "You hear [the list of
audible things which can be touched by the player].";
    otherwise say "A merciful peace prevails."
```

now includes the tomcat in the "list of audible things which can be touched by the player".

To this end, we're going to change the way we assess scope, but only during the listening action. Otherwise the tomcat remains in the other room and off-limits. The new source text is marked out below:

```
"Four Stars"

Section 1 - Procedure

A thing has some text called sound. The sound of a thing is usually "silence".

The report listening rule is not listed in the report listening to rules.

Carry out listening to something:
    say "From [the noun] you hear [the sound of the noun]."

Instead of listening to a room:
    if an audible thing can be touched by the player, say "You hear [the list of
audible things which can be touched by the player].";
    otherwise say "A merciful peace prevails."
```

```
Definition: a thing is audible if the sound of it is not "silence".

Before printing the name of something audible while listening to a room:
    say "[sound] from the "

A thing has some text called scent. The scent of a thing is usually "nothing".

The report smelling rule is not listed in the report smelling rulebook.

Carry out smelling something:
    say "From [the noun] you smell [scent of the noun]."

Instead of smelling a room:
    if a scented thing can be touched by the player, say "You smell [the list of
scented things which can be touched by the player].";
    otherwise say "The place is blissfully odorless."

Definition: a thing is scented if the scent of it is not "nothing".

Before printing the name of something scented while smelling a room: say "
[scent] from the "
```

Here is our addition:

```
After deciding the scope of the player while listening or sleeping or looking:
    if in darkness:
        repeat with locale running through adjacent rooms:
            place locale in scope.

A reaching inside rule while listening or sleeping or looking:
    rule succeeds.

Section 2 - Scenario

The Waning Moon Resort is a dark room. "A spacious room with a flagstone
floor, and a dreamcatcher hung over the king-size bed." The dreamcatcher is
scenery in the Resort. The description is "The usual web of threads and crystals,
feathers and beads." Instead of taking the dreamcatcher, say "Ah, ah -- you
might be tempted to take it as a souvenir, except that the price list in the minibar
clearly states they charge $65 apiece if you walk off with one. Cheaper than
stealing the Frette bathrobes, but still probably not a good idea."
```

And now our threat to the player's peace:

```
The Balcony is outside from the Resort. In the Balcony is a tomcat. The sound of
the tomcat is "yowling". After printing the name of the tomcat when the tomcat is
not visible: say " outside on the balcony".
```

From here we continue with the same scenario as before:

```
The king-size bed is an enterable supporter in the Resort. The description is
"200-thread-count Egyptian cotton sheets, according to the website. You would
make fun, only they really are extraordinarily comfortable." The player is on the
```

bed. A Lindt chocolate is on the bed. It is edible. The scent of the chocolate is "chocolate-hazelnut smell".

Instead of exiting: say "You are too weary to move."

The suitcase is an openable closed container in the Resort.

An electric light is a kind of device. Carry out switching on an electric light: now the noun is lit. Carry out switching off an electric light: now the noun is unlit. Understand "light" as an electric light.

The solar lamp is an electric light in Waning Moon Resort. The description is "Specially designed to give light in a spectrum resembling sunlight, to improve the mood and make a person energetic." The lamp is switched on and lit.

An electric noisemaker is a kind of device. An electric noisemaker has some text called usual sound. The usual sound of an electric noisemaker is usually "beepbeepbeep". Carry out switching on an electric noisemaker: now the sound of the noun is the usual sound of the noun. Report switching on an electric noisemaker: say "[The noun] goes [usual sound of the noun]!" instead. Report switching off an electric noisemaker: say "You switch off [the noun], silencing the [usual sound of the noun]." instead.

Carry out switching off an electric noisemaker: now the sound of the noun is "silence".

The bedside table is in the Resort. The table supports a potted plant and a Bose speaker. The scent of the potted plant is "rosemary"

The Bose speaker is an electric noisemaker. The usual sound of the speaker is "soothing whalesong". The sound of the speaker is "soothing whalesong". The speaker is switched on.

Instead of touching a device: say "You feel the surface of [the noun] and discover the switch."

Instead of touching a scented thing: say "The brush of your fingers stirs loose a fresh cloud of [scent of the noun] smell."

Rule for printing the description of a dark room: try listening; try smelling; rule succeeds.

Instead of examining an audible thing while in darkness: try listening to the noun.
Instead of examining something while in darkness: try touching the noun.

Before touching something when in darkness:
    say "You grope about..."

After inserting the plant into something:
    say "You unceremoniously dump [the noun] into [the second noun], hoping it sustains no important damage thereby."

Before printing the name of a dark room: if the player can touch an audible thing, say "Noisy "; if the player can touch a scented thing, say "Perfumed ".

```
Visibility rule when in darkness:
    if examining something, there is sufficient light;
    there is insufficient light.

Rule for printing the announcement of darkness: say "It is now pleasantly
lightless in here." instead.

Rule for deciding the scope of the player while in darkness: place the location in
scope.

To decide whether in daylight:
    if in darkness, no;
    yes.

Instead of sleeping when in daylight:
    say "You've never been able to sleep with the light on."

Instead of sleeping when the player can touch an audible thing (called the
irritant):
    say "The steady [sound of the irritant] from [the irritant] prevents your
slumber."

Instead of sleeping when the player can touch a scented thing (called the
irritant):
    if the irritant is chocolate, say "The smell of chocolate continues to tantalize
you, keeping you from sleep.";
    otherwise say "You sniffle. [The irritant] is probably acting on your allergies."

Instead of sleeping:
    say "You slip easily into the arms of Morpheus.";
    end the story finally saying "At last..."

When play begins:
    say "You have at last escaped from the airport and gotten through customs;
survived an unnerving taxi ride over icy highways; stared down the impertinent
concierge; endured the bellhop's catalog of features in your room; and achieved,
finally, a moment of peace. Time for a good night's slumber!"

Test me with "listen / x dreamcatcher / switch lamp off / look / sleep / eat
chocolate / sleep / get plant / examine plant / open suitcase / put plant in
suitcase / close suitcase / sleep / look / examine bose / switch bose off / sleep".
```

Of course, this new version is less happy for the player, as we haven't included any way to silence the cat.

365

Example **Ginger Beer**

RB

A portable magic telescope which allows the player to view items in another room of his choice.

Suppose we want to have a pair of linked lenses so that the player can look into one of them and see things which occur in room containing the other lense.

We begin simply with a bit of environment for the player to wander around:

"Ginger Beer"

The Ginger Beer Factory is a room. "In the center of the room is an enormous pot filled with crushed ginger, which seems to be bubbling slightly on its own. The fumes are overwhelming."

The pot is scenery in the Ginger Beer Factory. The description of the pot is "Cast iron." In the pot is a bubbling brew.

Instead of smelling the Ginger Beer Factory: try smelling the brew.

Instead of smelling the brew, say "You blink back tears."

The Storeroom is south of the Ginger Beer Factory. "The walls here are lined with a prodigious number of small, rounded bottles, each with a screw top and a smiling pirate on the label."

The Clippings Room is west of the Ginger Beer Factory. "A clean room lined with steel tables, for preparing ingredients."

Some steel tables are a supporter in the Clippings Room. They are scenery. The description is "They are roughly the size and height of laboratory worksurfaces."

The quantity of dandelion is on the steel tables. The description is "Horrible common weed."

The wooden box is on the steel tables. It is openable and closed. The description is "A large wooden box with a lid, used for ingredient storage. There is a label on the lid."

The label is part of the box. The description is "BURDOCK: the root beaten with a little salt and laid on the place suddenly easeth the pain thereof, and helpeth those that are bit by a mad dog:... the seed being drunk in wine 40 days together doth wonderfully help the sciatica: the leaves bruised with the white of an egg and applied to any place burnt with fire, taketh out the fire, gives sudden ease and heals it up afterwards.... The root may be preserved with sugar for consumption, stone and the lax."

The quantity of burdock is in the box. The description is "It looks like a kind of thistle."

Some bottles are in the Storeroom. They are scenery. The description is "They are smaller than the average bottle, because more potent." Instead of taking the bottles, say "Take one away and the whole lineup will cascade to the floor."

Now for the lenses themselves:

A lense is a kind of thing.

The large end of the telescope is a lense in the Ginger Beer Factory. "There is a large glass lense propped against the wall, in which are reflected all the contents of the room." Understand "glass" or "lense" as the large end.

```
The small end of the telescope is a lense in the Storeroom. "There is a small
glass lense sitting on the floor. Due to some curious effect of the optics, it
appears to be giving a view of somewhere else entirely." Understand "glass" or
"lense" as the small end. The description is "A gleaming lense about the size of
a pound coin."
```

Here is the critical bit, which needs to be somewhat flexible, since the large end of the telescope could in theory be left anywhere in the game (and should still work).

```
After deciding the scope of the player while the small end is carried by the
player:
    let there be the holder of the large end;
    place there in scope.

Before searching the small end when the small end is not carried by the player:
    say "(first picking up [the small end] and holding it to your eye)";
    silently try taking the small end.

Instead of searching the small end when the player is not carrying the small end:
    say "It's too hard to look through the small end from a distance."

Instead of searching the large end,
    say "You see only your own reflection."
```

We also want to make sure that the player who looks through the small lense does not see the large lense listed among the contents of the other location:

```
Definition: a thing is recognizable if it is not a lense.

Instead of searching the small end:
    let the far side be the holder of the large end of the telescope;
    say "You peer into the little lense and through it see, in [the far side], [the list
of recognizable things in the far side]."

Test me with "examine lense / south / examine lense / look through lense / north
/ look through small lense".
```

And we're done.

366

Example **Rock Garden**

RB

A simple open landscape where the player can see between rooms and will automatically move to touch things in distant rooms.

A map of linked rooms works well for modeling enclosed or indoor space, and somewhat less well for modeling large open spaces, where a person should reasonably be able to see things which are much too far away to touch. With some modifications to scoping, though, we can create an environment where objects in nearby rooms are described and viewable, and where the player will automatically move towards distant items before interacting with them physically.

```
"Rock Garden"

Section 1 - General Rules

Intervisibility relates rooms to each other in groups. The verb to be connected
with means the intervisibility relation.

Definition: a room is inter-visible if it is connected with more than one room.

After deciding the scope of the player when the location is an inter-visible room:
    repeat with other place running through rooms which are connected with the
location:
        unless the other place is the location, place the other place in scope.

Rule for reaching inside a room (called target) which is connected with the
location:
    let way be the best route from the location to the target;
    if the way is not a direction:
        say "You can't get over to [the target] from here.";
        deny access;
    say "(first heading [way])[command clarification break]";
    try going way;
    if the player is in the target, allow access;
    otherwise deny access.

After looking when the location is an inter-visible room:
    repeat with other place running through rooms which are connected with the
location:
        if the other place is not the location, describe locale for other place.

Section 2 - The Scenario

Rock Garden West is west of Rock Garden East. Rock Garden East contains a
rake. Rock Garden West contains a bench and a maple leaf. The bench is an
enterable supporter.

Rock Garden West is connected with Rock Garden East.

Test me with "get rake / drop rake / sit on bench / get rake".
```

367

Example **Stately Gardens**

RB

An open landscape where the player can see landmarks in nearby areas, with somewhat more complex room descriptions than the previous example, and in which we also account for size differences between things seen at a distance.

This time we're going to assume that the player can see into any room that is on a line of sight within one or two steps of travel.

```
"Stately Gardens"
```

```
Chapter 1 - Laying Out Rooms

A room can be indoors or outdoors.

Use full-length room descriptions.

After deciding the scope of the player:
    repeat with the way running through directions:
        let first step be the room the way from the location;
        if the first step is a room:
            place the first step in scope;
            let second step be the room the way from the first step;
            if the second step is a room, place the second step in scope;
    place the obelisk in scope.
```

The obelisk is so large that it can be seen from every room. If we had a number of such large monuments we might want to write a systematic routine to handle them, but this will do for now.

```
The room description heading rule is not listed in the carry out looking rules.
```

Now, we set things up so that the surrounding areas are described automatically as part of the room description:

```
Building description is a truth state that varies. Building description is false.

After looking when the location is an outdoors room:
    now count of sentences is 0;
    now building description is true;
    repeat with way running through directions:
        let space be the room way from the location;
        if space is an outdoors room, silently try looking toward space;
    if the obelisk is not in the location and the obelisk is unmentioned:
        let the way be the best route from location to the Upper Terrace;
        if the way is a direction, say "[The obelisk] is proudly visible on [the way]
horizon. [run paragraph on]";
        increment the count of sentences;
    now building description is false;
    unless the count of sentences is 0:
        say paragraph break.
```

But perhaps there are a few rooms where we do not wish that to happen, so we'll build in exceptions for those.

```
After looking in the rose garden:
    say "Otherwise, you are quite cut off.".

After looking in the Ha-ha:
    do nothing.
```

And suppose we want to allow the player to look in any direction:

```
Understand "look [direction]" or "look to/toward [direction]" as facing.
```

```
Facing is an action applying to one visible thing.

Carry out facing:
    let the viewed item be the room noun from the location;
    if the viewed item is not a room:
        if the location is indoors, say "Your view is restricted by the lack of doors or
windows in that direction." instead;
        otherwise say "You can't see anything promising that way." instead;
    try looking toward the viewed item.

Instead of facing up:
    say "Above you is bright sky."
```

We also need to tell distant rooms how to describe themselves.

```
Understand "look toward [any adjacent room]" as looking toward.

Looking toward is an action applying to one visible thing.

Check looking toward a room which does not contain something mentionable:
    if building description is false:
        say "You can't make out anything of interest that way." instead.

Carry out looking toward:
    now every thing is unmentioned;
    now the chosen direction is the best route from the location to the noun;
    now the second noun is the room the chosen direction from the noun;
    if the noun contains something mentionable:
        repeat with item running through mentionable things in the noun:
            carry out the writing a distant paragraph about activity with the item;
    if the noun contains something mentionable:
        increment the count of sentences;
        choose row count of sentences in the Table of Distance Sentences;
        if the second noun is an outdoors room and the second noun contains
something mentionable, say "[both entry] [run paragraph on]";
        otherwise say "[here entry] [run paragraph on]";
    otherwise:
        if the second noun is an outdoors room and the second noun contains
something mentionable:
            increment the count of sentences;
            choose row count of sentences in the Table of Distance Sentences;
            say "[there entry] [run paragraph on]";
    if building description is false:
        say paragraph break.
```

And again, some exception needs to be made for seeing what's in the dip in the ground:

```
Instead of looking toward the Ha-ha:
    now the chosen direction is the best route from the location to the noun;
    now the second noun is the room the chosen direction from the noun;
    if the second noun is an outdoors room and the second noun contains
something mentionable:
        increment the count of sentences;
```

```
        choose row count of sentences in the Table of Distance Sentences;
        say "[there entry] [run paragraph on]".
```

The following is to account for cases where the player types "look toward obelisk" or similar, rather than looking toward a room:

```
Understand "look toward [something]" as examining.
```

The following is arguably an unnecessary refinement, but the listing of items in the distance gets a bit repetitive unless we vary the sentence structure.

```
Chosen direction is a direction that varies.

Count of sentences is a number that varies.

Table of Distance Sentences
```

both	here	there
"From here, you make out [a list of mentionable things in the noun] a little way [chosen direction], and, further on, [a list of mentionable things in the second noun]."	"From here, you make out [a list of mentionable things in the noun] [if the noun is not adjacent to the location]some distance [end if]to [the chosen direction]."	"From here, you make out [a list of mentionable things in the second noun] some distance [chosen direction]."
"To [the chosen direction] there [is-are a list of mentionable things in the noun], partly obscuring your further view of [a list of mentionable things in the second noun]."	"To [the chosen direction] there [is-are a list of mentionable things in the noun]."	"Quite a way [chosen direction] [is-are a list of mentionable things in the second noun]."
"Then [chosen direction] [is-are a list of mentionable things in the noun], and beyond [a list of mentionable things in the second noun]."	"Meanwhile, to [the chosen direction] [is-are a list of mentionable things in the noun]."	"Meanwhile, [chosen direction] in the middle distance [is-are a list of mentionable things in the second noun]."
"When you turn [chosen direction], you see [a list of mentionable things in the noun], and somewhat further on [a list of mentionable things in the second noun]."	"When you turn [chosen direction], you see [a list of mentionable things in the noun]."	"If you turn [chosen direction], you see [a list of mentionable things in the second noun] some way off."
"Somewhere generally [chosen direction] [is-are a list of mentionable things in the noun], beyond which, [a list of mentionable things in the second noun]."	"Roughly [chosen direction] [is-are a list of mentionable things in the noun]."	"Moreover, in the [chosen direction] distance [is-are a list of mentionable things in the second noun]."
"[The chosen direction] shows [a list of mentionable things in the noun] and then [a list of mentionable things in the second noun]."	"And to [the chosen direction] [a list of mentionable things in the noun]."	"Meanwhile, [chosen direction] in the middle distance [is-are a list of mentionable things in the second noun]."
"Then, [chosen direction], [is-are a list of mentionable things in the noun], and beyond [a list of mentionable things in the second noun]."	"Meanwhile, to [the chosen direction] [is-are a list of mentionable things in the noun]."	"Meanwhile, [chosen direction] in the middle distance [is-are a list of mentionable things in the second noun]."
"Finally, [chosen direction], [is-are a list of mentionable things in the noun], somewhat nearer than [a list of mentionable things in the second noun]."	"Finally, to [the chosen direction] [is-are a list of mentionable things in the noun]."	"Finally, [chosen direction] in the middle distance [is-are a list of mentionable things in the second noun]."

Now, our ability to view things at a distance should be determined by the size of the things we're trying to see:

```
Chapter 2 - Height

A height is a kind of value. 10 feet 11 inches specifies a height. 10 feet 11
specifies a height. The verb to stand means the height property. The verb to
measure means the height property. A thing has a height. The height of a thing
is usually 3 feet 0.
```

```
Definition: a thing is tiny if its height is 0 feet 6 inches or less.

Definition: a thing is short if its height is 3 feet 0 or less.

Definition: a thing is tall if its height is 6 feet 0 or more.

The height of a man is usually 5 feet 10 inches. The height of a woman is usually
5 feet 6 inches.

Definition: a thing is monumental if it is taller than 25 feet 0 inches.

Definition: a thing is mentionable if it stands tall enough to see.

To decide whether (item - a thing) stands tall enough to see:
    if the item is in the Rose Garden and the item is shorter than the roses, no;
    if the item is mentioned, no;
    if the item is in an adjacent room and item is taller than 2 feet 0, yes;
    if the item is taller than 4 feet 0, yes;
    no.

Instead of examining something which is within a room (called the space) which
is not the location:
    if the location is adjacent to the space:
        if the noun is tiny, say "It is too far from here for you to make out much
detail about [the noun]." instead;
        let way be the best route from the location to the space;
        if the way is a direction, say "You gaze off [way] at [the noun]...";
        continue the action;
    otherwise:
        if the noun is short, say "It is too far from here for you to make out much
detail about [the noun]." instead;
        let way be the best route from the location to the space;
        if the way is a direction, say "You gaze off [way] into the distance at [the
noun]...";
        continue the action.
```

We might also want to be able to override, manually, the way distant things are described.

```
Writing a distant paragraph about something is an activity.

Rule for writing a distant paragraph about the lily pond:
    if the second noun is a room and something mentionable is in the second
noun, say "A [lily pond], [chosen direction], patchily reflects [a list of mentionable
things in the second noun] on the far side. [run paragraph on]";
    otherwise say "To [the chosen direction], [a lily pond] shimmers in the sunlight.
[run paragraph on]"

Rule for writing a distant paragraph about the roses:
    if something in the Rose Garden is taller than the roses,
        say "Over the tops of [the roses], [chosen direction], you see [a list of
mentionable things in the rose garden]. [run paragraph on]";
    otherwise say "Immediately [chosen direction] is [the roses]. [run paragraph
on]"
```

```
Rule for writing a distant paragraph about the obelisk:
    if a mentionable thing in the Upper Terrace is shorter than the obelisk,
        say "A stupidly grand [obelisk], [chosen direction], towers over [a list of
mentionable things in the Upper Terrace]. [run paragraph on]";
    otherwise say "To [the chosen direction], you can't help noticing [the obelisk],
which is much larger than any object really needs to be. [run paragraph on]".

After writing a distant paragraph about something:
    increment the count of sentences.
```

Moreover, proximate things might have special descriptions too.

```
Rule for writing a paragraph about something tiny when the location is outdoors:
    if the location is the Gravel Circle,
        say "Abandoned in the gravel [is-are a list of unmentioned tiny things in the
location]. [run paragraph on]";
    otherwise say "Half trampled into the grass, and easy to miss, [is-are a list of
unmentioned tiny things in the location]. [run paragraph on]"

Before doing something other than examining or approaching to something
which is not within the location:
    if the player has the noun, continue the action;
    say "(first going over to [the noun])[line break]";
    try approaching the noun;
    if the noun is not within the location, stop the action.

Understand "go toward/to/towards/near [something]" or "approach [something]"
as approaching.

Approaching is an action applying to one visible thing.

Check approaching:
    if the player is in something, say "You'll have to get up." instead;
    if the noun is within the location, say "You're as close to [the noun] as you can
get." instead;
    let space be the location of the noun;
    if the space is not a room, say "You don't quite see how to get there." instead;
    let way be the best route from the location to the space;
    unless way is a direction,
        say "You can't see how to get over there from here." instead.

To head to (space - a room):
    let the way be the best route from the location to the space;
    if the space is adjacent to the location,
        try going way;
    otherwise silently try going way.

Carry out approaching:
    let space be the location of the noun;
    while the space is not the location:
        head to space.
```

This is a bit primitive, since if we had an occasion where going was blocked, we could get stuck in a loop. So we would need to be careful, but for this example it won't arise.

Going state is a truth state that varies. Going state is false.

Check going:
    now going state is true.

The description of a room is usually "[if going state is true]You drift [noun] across the open lawn[direction relative to obelisk]. [end if]An absolutely phenomenal quantity of manicured turf stretches from where you stand in almost every direction."

Before reading a command:
    now going state is false.

To say direction relative to obelisk:
    if obelisk is in the location:
        say ", as though drawn magnetically to the foot of the monument";
    otherwise:
        let way be the best route from the location to the Upper Terrace;
        if way is the noun, say ", drawn towards [the obelisk]";
        if the way is the opposite of the noun, say ", keeping [the obelisk] more or less at your back".

Chapter 3 - The Grounds

When play begins:
    now the left hand status line is "Idyllic";
    now the right hand status line is " ".

The Gravel Circle, the Ha-ha, the Sheep Field, the Open Lawn, the Croquet Ground, the Rose Garden, the Upper Terrace, the Middle Terrace, and the Lower Terrace are outdoors.

The Middle Terrace is north of the Lower Terrace and south of the Upper Terrace. The lily pond is fixed in place in the Middle Terrace."You [if going state is true]come to[otherwise]are at[end if] the north edge of a perfectly round lily pond, bordered with stones. Its surface patchily reflects [the marble anteater] on the south bank." A tent peg and a wilted orchid are in the Middle Terrace. The tent peg measures 0 feet 6. The orchid measures 0 feet 4.

The description of the Lower Terrace is "[if going state is true]You climb [noun] up a small hillock[direction relative to obelisk][otherwise]You stand on a short, round, entirely artificial hillock[end if]."

The marble anteater is a fixed in place thing in the Lower Terrace. The height of marble anteater is 6 feet 2 inches."A marble anteater stands on a pedestal at the top of the hill. In the bright sunlight the white marble makes a striking contrast with [the obelisk] in the distance." The description is "The anteater is very much more than life-size."

The obelisk of black granite is a fixed in place thing in the Upper Terrace."Now that you are at the foot of it, you can properly appreciate the stupid immensity of the obelisk, pointing stonily at heaven." The height of the obelisk is 50 feet 0 inches. The description of the obelisk is "It stands ridiculously tall, and has an inscription on the face."

The inscription is part of the obelisk. The height of the inscription is 0 feet 3 inches. The description of the inscription is "You can't read the squirming, pointed letters, but they make you uneasy.".

The Gravel Circle is west of the Upper Terrace, northwest of the Middle Terrace, and north of the Croquet Ground. The description of the Gravel Circle is "[if going state is true]You head [noun] until the lawn thins and[otherwise]Here the lawn[end if] gives way to a circle of raked gravel, which crunches pleasingly beneath you."

Instead of going northwest in the Upper Terrace, try going north.

The Ha-ha is north of the Gravel Circle and northwest of the Upper Terrace. The description of the Ha-ha is "[if going state is true]The land dips here so suddenly that you do not know the dip is there until you're in it; but it prevents livestock from crossing barriers, and that is the important thing[otherwise]You are at the base of a steep-sided depression, so the lawn continues north and south more or less at the level of your head[end if].

The tip of [the obelisk] is the only thing you can make out from this depression, off to the southeast.". North of the Ha-ha is the Sheep Field. In the Sheep Field is an animal called a black sheep. The black sheep stands 4 feet 3 inches."A black sheep grazes placidly nearby." The description of the black sheep is "It reminds you of your Uncle Tim."

Before going from the Ha-ha:
    say "It's a bit of a scramble to get back up the side of the depression, and you keep slipping in the damp grass. But you manage at last."

The Rose Garden is southwest of the Lower Terrace. The thicket of red roses is a fixed in place thing in the Rose Garden. The thicket stands 4 feet 2 inches."Heavy red roses grow over a roughly horseshoe-shaped wall around you. Over this barrier, the head of [the marble anteater] is visible to the northwest, and the tip of [the obelisk] in the distance."

The description of the Rose Garden is "[if going state is true]You slip [noun] into the enclosure of the rose garden. [end if]The rest of the park, and the world, seems muted and quiet."

Instead of smelling the rose garden: try smelling the roses. Instead of smelling the roses, say "The smell tickles the back of your throat and makes you want to cough."

Instead of listening to the rose garden:
    say "You can't hear anything at all."

The Open Lawn is north of the Rose Garden, west of the Lower Terrace, and southwest of the Middle Terrace. The Croquet Ground is north of the Open Lawn, west of the Middle Terrace, southwest of the Upper Terrace, and northwest of the Lower Terrace.

A discarded champagne cork is in the Open Lawn. It stands 0 feet 2 inches.

A stone bench is an enterable supporter in the Croquet Ground. It stands 3 feet 8 inches."There is a stone bench here -- a sort of stone sofa, really, with nymphs

```
disporting themselves on the arms and back." The description of the bench is "It
used to be a Roman sarcophagus -- hence the nymphs -- but someone has
thoughtfully recarved it as lawn furniture."

The half-size Bentley is a vehicle in the Gravel Circle."A sort of child's-toy
version of a Bentley is parked [if something parkable is in the location]beside
[the tallest parkable thing in the location][otherwise]close at hand[end if]." The
description of the half-size Bentley is "Of beautiful and unambiguously luxurious
lines, but sized down to hold only one or (at a stretch) two people, and powered
by electricity." The half-size Bentley stands 3 feet 6 inches.

Definition: a thing is parkable if it is not a person and it is not the Bentley.

Instead of touching the obelisk, say "Though it is black stone in sunlight, the
obelisk is very cold to the touch."

Test me with "look east / look toward obelisk / s / s / e / sw / ne / n / n / w / n / n /
examine obelisk / touch obelisk / read inscription".
```

---

368 **Example Apples** *RB*

Prompting the player on how to disambiguate otherwise similar objects.

Inform by default detects whether two objects can be disambiguated by any vocabulary available to the player. If so, it asks a question; if not, it picks one of the identical objects at random.

Generally this produces good behavior. Occasionally, though, two objects have some distinguishing characteristic that doesn't appear in the object name. For instance, suppose we've created a class of apples that can be told apart depending on whether they've been bitten or not:

```
An apple is a kind of thing. Consumption is a kind of value. The consumptions
are pristine and bitten. An apple has a consumption. The description of an apple
is "It is [consumption]."

Understand the consumption property as describing an apple.
```

The player can meaningfully type

```
>EAT BITTEN APPLE
```

or

```
>EAT PRISTINE APPLE
```

but if he types

```
>EAT APPLE
```

Inform will, annoyingly, ask

```
Which do you mean, an apple or the apple?
```

This gives the player no indication of why Inform is making a distinction. So here we add a special "printing the name" rule to get around that situation:

```
"Apples"

Orchard is a room.

An apple is a kind of thing. Consumption is a kind of value. The consumptions
are pristine and bitten. An apple has a consumption. The description of an apple
is "It is [consumption]."

Understand the consumption property as describing an apple.

Before printing the name of an apple while asking which do you mean: say "
[consumption] ". Before printing the plural name of an apple while asking which
do you mean: say "[consumption] ".

The player carries three apples.

Instead of eating a pristine apple (called the fruit):
    say "You take a satisfying bite.";
    now the fruit is bitten.

Instead of eating a bitten apple (called the fruit):
    say "You consume the apple entirely.";
    now the fruit is nowhere.
```

Inform will also separate the bitten from the pristine apples in inventory listings and room descriptions, even though it's not clear why; we can improve on that behavior thus:

```
Before listing contents: group apples together.

Rule for grouping together an apple (called target):
    let source be the holder of the target;
    say "[number of apples held by the source in words] apple[s], some bitten".

Before printing the plural name of an apple (called target):
    let source be the holder of the target;
    if every apple held by the source is bitten, say "bitten ";
    if every apple held by the source is pristine, say "pristine ".

Test me with "i / eat apple / i / eat apple / pristine / i / eat apple / pristine / i".
```

---

369 **Example** **Originals** *RB*

Allowing the player to create models of anything in the game world;

parsing the name "model [thing]" or even just "[thing]" to refer to these newly-created models; asking "which do you mean, the model [thing] or the actual [thing]" when there is ambiguity.

We rely here on the understanding-by-relations rules we've already learned, but there is an additional trick: we want to make sure that if the player types "original" or "actual", this word will not be taken to refer to the thing modeled:

```
"Originals"

A model is a kind of thing. 100 models are in the model-repository.

Appearance relates one thing to various models. The verb to be shown by
means the appearance relation.

Indication relates a model (called X) to a thing (called Y) when Y is shown by X
and Y is suitable.

Understand "actual" or "original" as "[actual]". Understand "[actual]" as
something when the item described is not a model.

Definition: a thing is suitable:
    if the player's command includes "[actual]", no;
    yes.

Understand "[something related by indication]" as a model.

After printing the name of a model (called target): say " [random thing shown by
the target]"
```

Now our duplication command -- for the sake of simplicity, we'll suppose that in this scenario the player is duplicating objects by magic rather than creating them out of physical materials or supplies:

```
Understand "duplicate [something]" as duplicating. Duplicating is an action
applying to one visible thing.

The duplicating action has an object called the selected model.

Setting action variables for duplicating:
    let N be a random model in the model-repository;
    now the selected model is N.

Check duplicating:
    if the selected model is nothing, say "You're out of power." instead.

Carry out duplicating:
    now the noun is shown by the selected model;
    move the selected model to the player.
```

```
Report duplicating:
    say "You concentrate and manifest [a selected model]."
```

Now, the challenge is that we want to print the word "actual" before printing the name of an object, but only during disambiguation questions and only when we are not printing the name of the object as part of a model-name! (If we are not careful about the latter point, we will get responses such as "Which do you mean, the model actual deer or the actual deer?" which of course defeats the whole purpose.

The way around this is to remember that activities stack: we're printing the name of the deer while printing the name of a model that involves the deer. So if we set a flag while printing the name of a model, we can control the way the deer's name prints during the transaction. (We could use our ...while clause to specify while not printing the name of a model, except that we're already using it for "while asking which do you mean", and these do not stack.) So:

```
The virtual-context is a truth state that varies. The virtual-context is false.

Before printing the name of a model:
    now virtual-context is true.

After printing the name of a model:
    now virtual-context is false.

Before printing the name of something (called target) while asking which do you
mean:
    if the target is not a model and virtual-context is false:
        say "actual ".

Forest is a room. It contains a deer and a daisy. The deer is an animal.

Test me with "duplicate deer / x model deer / x deer model / drop deer / x deer /
actual / x deer / model".
```

370

Example **Walls and Noses**

RB

Responding to "EXAMINE WALL" with "In which direction?", and to "EXAMINE NOSE" with "Whose nose do you mean, Frederica's, Betty's, Wilma's or your own?"

Suppose we want our game to respond to "EXAMINE WALL" with "In which direction?", and to "EXAMINE NOSE" with "Whose nose do you mean, Frederica's, Betty's, Wilma's or your own?"

For the case of EXAMINE WALL, we need a way to determine whether every item being disambiguated is a direction. We'll start by making a "matched" adjective which will identify items being disambiguated:

```
"Walls and Noses"
```

```
Eight-Walled Chamber is a room. "A perfectly octagonal room whose walls are
tinted in various hues."

Understand "wall" as a direction.

Definition: a direction is matched if it fits the parse list.
Definition: a room is matched if it fits the parse list.
Definition: a thing is matched if it fits the parse list.

Rule for asking which do you mean when everything matched is direction:
    say "In which direction?"
```

Checking the parse list requires a bit of behind-the-scenes work with Inform 6. Fortunately, you don't have to understand this entirely in order to use the rest of the example:

```
To decide whether (N - an object) fits the parse list:
    (- (FindInParseList({N})) -)

Include (-
[ FindInParseList obj i k marker;
    marker = 0;
    for (i=1 : i<=number_of_classes : i++) {
    while (((match_classes-->marker) ~= i) && ((match_classes-->marker) ~= -i))
marker++;
    k = match_list-->marker;
    if (k==obj) rtrue;
    }
    rfalse;
];
-)
```

Now that we've defined our "matched" adjective, we can use it for other purposes as well -- even generating our own lists. Our second challenge was to respond to EXAMINE NOSE with "Whose nose do you mean, Frederica's, Betty's, Wilma's or your own?"

Here we need to change the way the question is worded (not "which do you mean" but "whose nose do you mean"). We also have to the names of the noses as they're printed in this particular context, so that they don't repeat the word "nose" over and over. And -- as a point of good English style -- we also want "your own" nose always to be last on the list.

For this purpose we may want to use the built-in "Complex Listing" extension, which allows us to print specially ordered lists. So:

```
Include Complex Listing by Emily Short.

Wilma, Betty, and Frederica are women in the Eight-Walled Chamber.
Understand "lady" or "woman" as a woman. A nose is a kind of thing. A nose is
part of every person.

Rule for asking which do you mean when everything matched is a nose:
    prepare a list of matched things;
```

```
    if your nose is an output listed in the Table of Scored Listing:
        choose row with an output of your nose in the Table of Scored Listing;
        now the assigned score entry is -1;
    say "Whose nose do you mean, [the prepared list delimited in disjunctive
style]?"

Rule for printing the name of a nose (called target) while asking which do you
mean :
    if everything matched is a nose:
        if the target is part of a person (called owner):
            if the owner is the player, say "your own";
            otherwise say "[the owner][apostrophe]s";
    otherwise:
        make no decision.

Understand "own" or "mine" as your nose.

Test me with "x wall / north / x nose / mine".
```

371

Example **Latin Lessons**

RB

Supplying missing nouns and second nouns for other characters besides the player.

If we're defining actions for other characters to follow, we may want to include them in our "rule for supplying a missing noun". We can do this if we write our "while..." clause to apply to any actor, as follows:

```
"Latin Lessons"

The Latin Studio is a room. Rick is a man in the Studio.

A dance-name is a kind of thing. Argentine tango, samba, merengue, cha-cha,
street salsa are dance-names.

Dancing is an action applying to one visible thing. Understand "dance [any
dance-name]" as dancing. Understand "dance" as dancing.

Rule for supplying a missing noun while an actor dancing:
    now the noun is street salsa.

Report someone dancing:
    say "[The actor] dances a few steps of [the noun] for you."

Report dancing:
    say "You dance a few steps of [the noun]."

Persuasion rule for asking someone to try dancing: persuasion succeeds.

Test me with "dance / dance samba / rick, dance / rick, dance merengue".
```

372 **Example Minimal Movement** *RB*

Supplying a default direction for "go", so that "leave", "go", etc., are always interpreted as "out".

Sometimes it would be nice to respond a little more sensitively to a vague command such as "leave" -- converting it, perhaps, to a "go out" command.

```
"Minimal Movement"

The Doll-like House is a room. The Postage-Stamp-Sized Garden is outside
from the House.

Rule for supplying a missing noun while going:
    now noun is outside.
```

This particular situation is very slightly complicated by the existing rules about vague movement, but fortunately we can easily turn those off.

```
The block vaguely going rule is not listed in the for supplying a missing noun
rules.

Test me with "go".
```

373 **Example Cloves** *RB*

Accepting adverbs anywhere in a command, registering what the player typed but then cutting them out before interpreting the command.

It has sometimes been suggested that IF should allow for the player to use adverbs, so that doing something "carefully" will have a different effect from doing it "quickly". There are several inherent challenges here: it's a good idea to make very sure the player knows all his adverb options, and the list of possibilities should probably not be too long.

Another trick is that adverbs complicate understanding commands, because they can occur anywhere: one might type >GO WEST CAREFULLY or >CAREFULLY GO WEST, and ideally the game should understand both. After reading a command is the best point to do this sort of thing, because we can find adverbs, interpret them, and remove them from the command stream. So:

```
"Cloves"

Manner is a kind of value. The manners are insouciantly, sheepishly, and
defiantly.
```

Now we have, automatically, a value called manner understood to be used whenever parsing manners, and we can use this even during the "after reading a command"

stage, so:

```
After reading a command:
    if the player's command includes "[manner]":
        cut the matched text;
    otherwise:
        say "But how, my dear boy, how? You simply can't do something without a
pose. Thus far you have mastered doing things [list of manners].";
        reject the player's command.

When play begins:
    now the left hand status line is "Behaving [manner understood]";
    now the right hand status line is "[location]";
    now the manner understood is insouciantly.

The Poseur Club is a room. "Lady Mary is laid out on a sofa, her wrists
bandaged importantly[if the manner understood is insouciantly] -- and she looks
all the more depressed by your indifference to her state[end if]; Salvatore is at
the gaming table, clutching his hair with both hands[if the manner understood is
defiantly] -- though he looks up long enough to snarl in response to that
expression of yours[end if]; Frackenbush is muttering lines from another of his
works in progress, as though poetry has nearly made him mad[if the manner
understood is sheepishly]. But he spares you a reassuring smile. He's not a bad
fellow, Frackenbush[end if].

The usual people, in short."

Instead of doing something other than waiting or looking:
    say "Dear. No. That would smack of effort."

Instead of waiting when the manner understood is sheepishly:
    say "You scuff your foot against the ground for a moment, and allow a seemly
blush to creep over your cheek. It's quite effective, you are sure, though you
can't look up and see how it is going."

Instead of waiting when the manner understood is insouciantly:
    say "Thrusting your hands into your pockets, you whistle a jaunty tune.

'Do shut up,' says a Melancholy Poseur from over by the window."

Instead of waiting when the manner understood is defiantly:
    say "You raise your chin and give a pointed glance around the room as though
to say that you are waiting for someone; you are unembarrassed about waiting
for her; you have by no means been stood up; and the first person to comment
will receive a poke in the eye."

Before looking when the manner understood is sheepishly:
    say "You gaze up from under your brows..."

Before looking when the manner understood is defiantly:
    say "You cast a withering gaze over the room."

Before looking when the manner understood is insouciantly:
    if turn count > 1,
```

```
        say "You turn an eye to your surroundings, looking faintly-- just faintly--
    amused."

    Test me with "wait / wait insouciantly / sheepishly look / defiantly look / look
    insouciantly".
```

The qualification about turn count is to prevent this before message from occurring when the player first looks around the room (automatically) at the start of play.

Note that to test this example, one must type INSOUCIANTLY TEST ME, and not simply TEST ME: a poseur's work is never done.

---

374 **Example** **Fragment of a Greek Tragedy** *RB*

Responding to the player's input based on keywords only, and overriding the original parser entirely.

Apologies to the shade of A. E. Housman.

```
"Fragment of a Greek Tragedy"

Understand "restart/restore/save/quit" as "[meta-command]".

After reading a command:
    if the player's command matches "[meta-command]", make no decision;
    say line break;
    repeat through Table of Current Topics:
        if the player's command includes topic entry:
            say "CHORUS: [reply entry][paragraph break]";
            follow the advance time rule;
            rule succeeds;
    say "[italic type] Pause.[roman type][line break]";
    follow the advance time rule;
    rule succeeds.

Table of Current Topics

    topic                                reply
    "journey/trip/travel/came/arrived"   "Sailing on horseback, or with feet for oars?"
    "horseback/legs/feet/oars"           "Beneath a shining or a rainy Zeus?"
    "shining/rainy/weather/zeus"         "Mud's sister, not herself, adorns thy boots."
```

This would be a bit bare if we didn't provide the player with some sort of context at the outset, so let's put some remarks before the first command prompt:

```
Before reading a command while the turn count is 1:
    say "CHORUS: O suitably-attired-in-leather-boots
    Head of a traveller, wherefore seeking whom
    Whence by what way how purposed art thou come
    To this well-nightingaled vicinity?
    My object in inquiring is to know.
    But if you happen to be deaf and dumb
```

```
And do not understand a word I say,
Then wave your hand, to signify as much."
```

This "turn count" condition is why it was useful to follow the advance time rule in "after reading a command": the game (or drama, if you like) will continue to count moves elapsed even though the rest of Inform's command parsing and world model is being ignored. In a longer and more ambitious implementation of this idea, we might want to allow scenes to govern the behavior and responses of the Chorus.

And then to give the whole exchange a play's format:

```
The Stage is a room.

The room description heading rule is not listed in the carry out looking rules.

When play begins:
    now the command prompt is "YOU: ";
    now left hand status line is "Fragment of a Greek Tragedy";
    now right hand status line is "A. E. Housman".
```

(Because this example manipulates commands outside of the normal parser, the mechanism for TEST will not work here. Try typing commands such as: TELL CHORUS ABOUT JOURNEY / TELL CHORUS ABOUT FEET / TELL CHORUS ABOUT SHROPSHIRE / TELL CHORUS ABOUT ZEUS)

---

375 **Example** **North by Northwest** *RB*

Creating additional compass directions between those that already exist (for instance, NNW) -- and dealing with an awkwardness that arises when the player tries to type "north-northwest". The example demonstrates a way around the nine-character limit on parsed words.

Suppose we wanted to add intermediate compass directions such as north-northwest to our game. Because of the limitations of the index map, we won't be able to view these connections on the world map, but we can certainly create them, and use them in route-finding, just like other directions.

Here's how we'd set up such a thing:

```
"North by Northwest"

Section 1 - Procedure

The north-northwest is a direction. North-northwest has opposite south-
southeast. Understand "n-nw" or "nnw" as north-northwest.

The north-northeast is a direction. North-northeast has opposite south-
southwest. Understand "n-ne" or "nne" as north-northeast.
```

```
The south-southwest is a direction. South-southwest has opposite north-
northeast. Understand "s-sw" or "ssw" as north-northwest.

The south-southeast is a direction. South-southeast has opposite north-
northwest. Understand "s-se" or "sse" as south-southeast.

The west-northwest is a direction. West-northwest has opposite east-southeast.
Understand "w-nw" or "wnw" as west-northwest.

The east-northeast is a direction. East-northeast has opposite west-southwest.
Understand "e-ne" or "ene" as east-northeast.

The west-southwest is a direction. West-southwest has opposite east-northeast.
Understand "w-sw" or "wsw" as west-northwest.

The east-southeast is a direction. East-southeast has opposite west-northwest.
Understand "e-se" or "ese" as east-southeast.
```

A complication arises because we reach the 9-character limit: Inform truncates the names of objects to nine characters before trying to understand them. To make matters worse, the hyphen (and other punctuation marks) count as two letters. So both north-northwest and north-northeast will get truncated to "north-no", and be indistinguishable when the player types them.

When we are compiling for Glulx, the limit is easily changed with a single line, setting the constant called DICT_WORD_SIZE. For instance, if we wanted to raise the limit to 15, we would simply write "Use DICT_WORD_SIZE of 15."

If we're compiling to the Z-machine, however, we'll have to resort to some manipulation of the player's command. The general solution is that when the player's name for an object is going to have to be longer than we can correctly read, we can substitute an unambiguous abbreviation for the thing the player typed. In this case, it will be simplest and most efficient always to condense the player's direction names to single letters, thus:

```
After reading a command:
    let N be "[the player's command]";
    replace the text "north" in N with "n";
    replace the text "east" in N with "e";
    replace the text "south" in N with "s";
    replace the text "west" in N with "w";
    change the text of the player's command to N.
```

For more on the use of text, see the Advanced Text chapter.

```
Section 2 - Scenario

The Empty Field is north-northwest of the Deserted Road.

A crop-dusting plane is a backdrop. It is not scenery. It is in the Deserted Road
and Empty Field. The initial appearance of the crop-dusting plane is "[one of]In
the distance[or]Approaching faster and faster[or]Flying ominously low and
```

```
directly towards you[or]Immediately overhead[or]Circling around for another
approach[cycling] is a standard crop-dusting plane."

After looking:
    say "From here you can run to [the list of adjacent rooms]."

Rule for printing the name of a room (called the target) which is not the location
while looking:
    let chosen direction be the best route from the location to the target;
    say "[chosen direction]".

Test me with "sse / north-northwest".
```

In practice, this is going to be overkill for almost all games: most players already find eight compass directions plus up and down to be enough (or more than enough) to keep track of. But the option exists, in case there is a compelling reason to use it.

(Note also that we are allowed to use multi-word direction names, so we could have called the directions "north by northwest", "north by northeast", and so on. This example deliberately takes the hard way in order to show how to resolve the nine-character problem.)

---

376  **Example Complimentary Peanuts** *RB*

A character who responds to keywords in the player's instructions and remarks, even if there are other words included.

The "reading a command" activity is not the only point at which we can interact with snippets, as it happens; it is merely the most useful. "The player's command" can be consulted at other points, however, as in this example of your somewhat deaf (or distracted, or simply cussed) Aunt:

```
"Complimentary Peanuts"

Instead of asking Aunt Martha to try doing something:
    repeat through Table of Aunt Martha's Commentary:
        if player's command includes topic entry:
            say "[commentary entry][paragraph break]";
            rule succeeds;
    say "'Hmmf,' says Aunt Martha."
```

The topic understood is also a snippet, so that whenever one has been generated, we can treat it in the same way as "the player's command":

```
Asking someone about something is speech.
Telling someone about something is speech.
Answering someone that something is speech.
Asking someone for something is speech.

Instead of speech when the noun is Aunt Martha:
    repeat through Table of Aunt Martha's commentary:
```

```
        if the topic understood includes topic entry:
            say "[commentary entry][paragraph break]";
            rule succeeds;
    say "'Hmmf,' says Aunt Martha."
```

This is superior to checking "the player's command" because we do not want ASK MARTHA ABOUT FRENCH FRIES to trigger the "Martha" keyword, only the "french fries" keywords.

```
The Empyrean Shuttle Bay is a room. "From here you have an excellent view of the colony world, which looks... well, it looks discouragingly orange. But terraforming is in progress."

Aunt Martha is a woman in the Empyrean Shuttle Bay. A gleaming shuttle and a stack of rations are in the Shuttle Bay. The shuttle is a vehicle. "Your shuttle awaits."

Table of Aunt Martha's Commentary

topic                commentary
"shuttle"            "'Shuttles! I hate shuttles,' Aunt Martha grumbles. 'Give me an airplane! AIRPLANE.'"
"airplane/airport"   "'Those were the days,' Aunt Martha agrees, plainly reliving the days when she wore a blue-and-white uniform and passed out packets of salted pretzels."
"rations"            "'Do you think there are any peanuts in there?' she asks in a wistful tone."

Test me with "martha, get in the shuttle / martha, for pity's sake, do you see an airplane around here? / martha, pass me the rations".
```

This means that Martha will respond to keywords regardless of the setting in which they occur. For instance:

```
>martha, get in the shuttle
"Shuttles! I hate shuttles," Aunt Martha grumbles. "Give me an airplane! AIRPLANE."

>martha, for pity's sake, do you see an airplane around here?
"Those were the days," Aunt Martha agrees, plainly reliving the days when she wore a blue-and-white uniform and passed out packets of salted peanuts.

>martha, pass me the rations
"Do you think there are any peanuts in there?" she asks in a wistful tone.
```

This is not the stuff of which Loebner-winning chatbots are made, admittedly, but it is occasionally a useful alternative to stricter modes of command-parsing.

377

Example **The Big Sainsbury's**

RB

Making implicit takes add a minute to the clock, just as though the player had typed TAKE THING explicitly.

Implicit takes are a convenience to players; in general, we would like to avoid asking players to type any more obvious commands than strictly necessary, while allowing

the computer to guess as much as it safely can.

Occasionally, though, we have designed a timed puzzle in which the player has a limited number of moves in which to accomplish his objectives. In that case, the implicit take complicates matters, because it means that a player who types

```
>EAT GATEAU
(first taking the gateau...)
```

gets away with a spare move compared to the precise but naïf dupe who types

```
>TAKE GATEAU
>EAT GATEAU
```

...and really, that doesn't seem quite fair. The way to fix this problem is to fill in the extra minute on the clock during the implicit take; and that is indeed what we do in the following example.

```
"The Big Sainsbury's"

Sainsbury's is a room.

The crispy duck and the Guinness steak pie are edible things in Sainsbury's.

Rule for implicitly taking something:
   follow the advance time rule;
   continue the activity.

When play begins:
   now the right hand status line is "[time of day]".

Test me with "take crispy duck / eat crispy duck / eat steak pie".
```

378

Example **Pizza Prince**

RB

Providing a pizza buffet from which the player can take as many pieces as he wants.

Suppose we want the player to have a pizza buffet from which he can take a number of slices. But we don't want to actually put the slices there in front of him, because "you can see 17 slices of pizza here" is not the descriptive effect we want, and because we want to pretend, at least, that the pizza supply is nearly infinite. In fact, we're going to replenish the supply by allowing eaten slices to return to the buffet table (safer in IF than in real life).

To do this, we create one object to stand in for the pizza supply, but whenever the player tries to take it, we give him a different "pizza slice" object instead. Thus:

```
"Pizza Prince"
```

```
The Pizza Prince is a room.

The buffet table is a supporter in Pizza Prince.

The pizza selection is a thing on the buffet table. Understand "slice" as the pizza
selection. The description is "They are all cheese-only, and all luke-warm."

Rule for writing a paragraph about the buffet table:
    say "On [the buffet table] is [a pizza selection]. [description of the pizza
selection][line break]".
```

Now we introduce our actual pizza slices, which are retained in a container out of play until they're needed:

```
A pizza slice is a kind of thing. 10 pizza slices are in Pizza Limbo. A pizza slice is
always edible. [After a fashion, anyway.]
```

In this example we've set that supply to be artificially small, to make it easier to test what happens when the player reaches the limit; but we could provide many more slices to start with in Pizza Limbo, and the aim in practice would be to pick a number high enough (such as 50 or 100) that the average player will get bored of TAKE PIZZA long before he reaches the limit.

The main thing to be aware of is that objects consume memory in the game file, so creating a large number of pizza slices might bulk the game out. This is more of a concern if we're compiling for the Z-machine than if we're compiling for Glulx.

Whenever the player tries to take the selection, we want him to wind up holding an individual slice instead; but of course we need to check and make sure that he hasn't exhausted the pizza slice supply.

```
Instead of taking the pizza selection:
    let chosen slice be a random pizza slice in Pizza Limbo;
    if chosen slice is nothing: [That is, there were no slices remaining]
        say "[manager refusal]";
    otherwise:
        move the chosen slice to the player;
        say "Taken (gingerly)."

To say manager refusal:
    say "[one of]'Hey!' barks a hitherto-unseen manager from behind you. 'It's an
'all you can eat' buffet, not an 'all you can stuff down your pants' buffet.'[or]You
are conscious of a disapproving huff from the manager, so you refrain.[stopping]"
```

That's fine for the case where the player is taking a new slice of pizza explicitly, but we need to handle it a little differently if the taking action is generated in response to EAT PIZZA. In that case, we need to take the slice and also change the identity of the noun, because after the implicit take action happens, the game will test whether the player is holding the noun before attempting to eat it. So we need to refocus its attention:

```
Rule for implicitly taking the pizza selection:
    let chosen slice be a random pizza slice in Pizza Limbo;
```

```
    if chosen slice is nothing: [That is, there were no slices remaining]
        say "[manager refusal]";
    otherwise:
        move the chosen slice to the player;
        say "(helping yourself from the selection)";
        now the noun is the chosen slice.
```

And finally, a bit of touch-up:

```
Rule for clarifying the parser's choice of the pizza selection while taking:
    say "(from the magnificent selection before you)[line break]"
```

For tidiness, we should probably also return the consumed pizza slices to Pizza Limbo so that they can be re-used later:

```
After eating a pizza slice:
    move the noun to Pizza Limbo;
    continue the action.
```

```
Test me with "i / get pizza / g / i / get pizza / drop pizza / look / get pizza / g / look / eat pizza / g / g / g / g / get pizza / g / g / g / g / g / g / g / g / g / g / g / i / eat pizza / take pizza / g".
```

---

379  **Example Lollipop Guild** *RB*

Overriding the rules to allow the player to show something to another character without first taking it.

As mentioned in this section, the "implicitly taking" activity does not allow us to skip an implicit take entirely. In order to do this, we need to borrow from the chapter on Rulebooks and tell Inform that one of the rules normally built in to the Standard Rules does nothing in certain circumstances:

```
"Lollipop Guild"

The carrying requirements rule does nothing when showing something to the guardian.
The can't show what you haven't got rule does nothing when showing something to the guardian.
The block showing rule does nothing.

Candyland is a room. "A fizzing, popping wonderland of sugary delights. A path tiled with butterscotch sweets leads to the horizon."

The butterscotch path is scenery in Candyland.

The player carries a basket. In the basket are a licorice gumdrop and a can of tuna. The gumdrop is edible. The description of the gumdrop is "Covered all over with grains of sugar." The can of tuna is edible. The description of the can of tuna is "A rare import in this place."
```

```
The giant lollipop is a fixed in place edible thing in Candyland. "Growing right next to the path, on a trunk of white paper, is a giant lollipop colored green and red and white." The description of the lollipop is "If you were very blind, like Aunt Myrtle, you might mistake it for a young sapling just planted: the lollipop is just that leafy shade of green, with swirls of white and red that might be branches or flowers."

The guardian is a man in Candyland. "Right beside you is a guardian in a mint-colored uniform." The description of the guardian is "A killjoy wielding a gigantic toothbrush." The guardian carries a gigantic toothbrush. The description of the toothbrush is "Bristles as long as your hand. Firm bristles, too, not those soft ones. The guardian doesn't care about your tender gums."

A thing can be sweet. The butterscotch path, the lollipop, and the gumdrop are sweet.

Carry out showing a sweet thing to the guardian:
    say "The guardian shrieks! You don't understand its language, but from its ululations you understand the idea of decay. There may have been a bit in there about a root canal." instead.

Carry out showing something to the guardian:
    say "The guardian nods approvingly at the unsweetened [noun]." instead.

Report eating a sweet thing in the presence of the guardian:
    say "The guardian looks mournful, but unholsters his tube of paste and begins applying it to the toothbrush, as though to say that he really did not want to have to do this...";
    end the story saying "Everything goes minty" instead.

Report eating something:
    say "You consume [the noun] with gusto." instead.

Test me with "x guardian / x toothbrush / show gumdrop to guardian / show path to guardian / show tuna to guardian / look / eat gumdrop".
```

Note that because we only deactivate the carrying requirements rule for showing purposes, the player still takes the gumdrop before eating it.

380

Example WXPQ

RB

Creating a more sensible parser error than "that noun did not make sense in this context".

The parser error "That noun did not make sense in this context" arises instead of "You can't see any such thing" when the player uses a command that could apply to any item in the game -- that is, a command such as

```
Understand "go to [any room]" as going directly to.
Understand "talk about [any subject]" as discussing.
```

...and so on. The idea here is that "You can't see any such thing" isn't a sensible rejoinder when the player doesn't really need to be able to see the object.

Nonetheless, "That noun did not make sense..." is itself a fairly dry and uninformative response, and we may want to override it to something more appropriate for the specific kind of context in which it might appear. For instance:

```
"WXPQ"

WXPQ Studio is a room. "After about 2 AM, no one is listening anyway, so you can more or less make up whatever you like to fill the airwaves."

John F Kennedy, Elvis, Ralph Nader, Tony Blair, and single-origin chocolate are things.

Understand "talk about [any thing]" or "discuss [any thing]" as discussing. Discussing is an action applying to one visible thing.

Carry out discussing:
    say "You babble for a while about your [one of]interest in[or]hatred of[or]passionate devotion to[or]conspiracy theory concerning[or]mother's secret love affair with[as decreasingly likely outcomes] [the noun]."

Rule for printing a parser error when the latest parser error is the noun did not make sense in that context error:
    say "For once, you're at a loss for anything to say."

Test me with "discuss Elvis / discuss Kennedy / discuss chocolate / discuss narratology vs ludology debate".
```

Note that this solution works as simply as it does because we only have one command in the game that can apply to an "[any]" token. If we had several, we'd need to distinguish between the parser error attached to "discuss" and the parser error attached to "go to" (for instance). In that case, we might instead write something like

```
Rule for printing a parser error when the latest parser error is the noun did not make sense in that context error:
if the player's command includes "go":
    say "There's no such place you know how to get to.";
otherwise:
    say "For once, you're at a loss for anything to say."
```

---

381  **Example Xot** RB

Storing an invalid command to be repeated as text later in the game.

In Hitchhiker's Guide to the Galaxy, any erroneous command the player types can return to haunt him later in the game. We could do the same, if we liked, by storing the player's command whenever we print a parser error.

```
"Xot"
```

```
Humiliation Chamber is a room. "A grim, grey-walled room. Cameras watch you
from every angle; convex mirrors reflect your actions; and up near the ceiling,
where you can't disable it, is a loudspeaker."

The last error is a text that varies. The last error is "xot".

Before printing a parser error:
    now the last error is the player's command.

Every turn when a random chance of 1 in 2 succeeds:
    say "Over the loudspeaker comes some distorted nonsense. If you listen
carefully, it sounds as though some fool is saying '[last error], [last error], [last
error]!'"

Test me with "wiggle / z / z / z / z / z / z".
```

---

382 **Example Bikini Atoll** *RB*

Delaying the banner for later.

```
"Bikini Atoll" by Edward Teller

The Hut and the Tropical Beach are rooms.

The conch shell is in the Hut. After taking the shell for the first time: say "As you
gather the oddly-warm conch shell into your arms, you experience a sudden
flash of deja-vu...[banner text]"; move the player to the Tropical Beach.

Rule for printing the banner text when the player is not carrying the shell: do
nothing.

Test me with "look / examine shell / get shell / look".
```

(By tradition, and as a courtesy to all the people who have worked on Inform, authors ensure that the banner is printed some time near the beginning of each game played. So please only defer it, rather than suppress it altogether.)

---

383 **Example Battle of Ridgefield** *RB*

Completely replacing the endgame text and stopping the game without giving the player a chance to restart or restore.

Occasionally, a piece of IF is sufficiently serious that it feels bathetic to offer the player the usual restore-restart-undo-quit options at the end. The following would replace "*** You have died ***" with a centered epitaph, then quit the game when the player hits a key.

This example relies on a standard extension to avoid any fancy programming:

```
"Battle of Ridgefield"

Include Basic Screen Effects by Emily Short.

Ridgefield is a room.

Instead of doing something when the turn count is greater than 1: say "Alas, you
no longer have the strength."; end the story.

Rule for printing the player's obituary:
    say paragraph break;
    center "In defense of American Independence";
    center "at the Battle of Ridgefield, April 27, 1777,";
    center "died Eight Patriots who were laid in this ground,";
    center "Companioned by Sixteen British Soldiers,";
    center "Living, their enemies,";
    center "Dying, their guests";
    say paragraph break;
    wait for any key;
    stop game abruptly;
    rule succeeds.
```

384

Example **Finality**

RB

Not mentioning UNDO in the final set of options.

By default, Inform reminds the player that he has the option of typing UNDO after a story-ending action. This is generally good practice, especially for the sake of novice players who might not be aware of this possibility otherwise, and might be frustrated by a loss they could easily step back from.

Just occasionally, though, we may decide that the player does not deserve any such notification:

```
"Finality"

Cliff Edge is a room. "This narrow strip overlooks a gorge many hundreds of feet
deep, at whose bottom is a river of molten lava. The walls of the gorge are lined
with poison-tipped spikes. Furthermore, the birds that inhabit this valley spit balls
of fire. Good thing you're safe up here."
```

The Table of Final Question Options determines what options are to be given to the player after the story ends. We can change what is mentioned there by altering the entries. (The example Jamaica 1688 explains this table in more detail, and demonstrates some other things that we might do with it.)

```
When play begins:
    choose row with a final response rule of immediately undo rule in the Table of
Final Question Options;
    blank out the final question wording entry.
```

```
Instead of jumping:
    say "If you insist.";
    end the story.
```

And if we decided that we didn't want the player to be able to undo the command at all, we should add the use option

```
Use undo prevention.

Test me with "jump".
```

385

Example Jamaica 1688

RB

Adding a feature to the final question after victory, so that the player can choose to reveal notes about items in the game.

The options offered to the player at the end of the game are listed in the Table of Final Question Options, which means that we can add to them simply by continuing the table; what's more, the table gives us the opportunity to create a "final response rule", a rule that the game should follow in order to parse the player's input at this point.

So, for instance, if we wanted the player to be allowed to ask for notes about any of the rooms, characters, or objects in a historical game:

```
"Jamaica 1688"

Use scoring.

Section 1 - Procedure

Table of Final Question Options (continued)
```

final question wording	only if victorious	topic	final response rule	final response activity
"REVEAL the inspiration for something or somewhere"	true	"reveal [any thing]"	investigate something rule	--
--	true	"reveal [any room]"	investigate something rule	--

```
This is the investigate something rule:
    repeat through the Table of Footnotey Stuff:
        if the player's command matches the topic entry:
            say "[revelation entry][paragraph break]";
            rule succeeds;
    say "I'm afraid I have no revelation to vouchsafe there."

Section 2 - Scenario

The Upper Deck is a room. Lucius is a man in the Upper Deck.

The maximum score is 501.
```

When play begins: now the score is 501; end the story finally.

Table of Footnotey Stuff

topic	revelation
"reveal [Lucius]"	"Lucius is based on a historical buccaneer who sailed with William Dampier. The original did carry a Greek New Testament, from which he read aloud when the men were stranded in the jungles near Panama."
"reveal [Upper Deck]"	"The Callisto is a simplified and tidied representation of a pirate sloop ca. 1688."

---

386 ★★★ **Example Xerxes** RB

Offering the player a menu of things to read after winning the game.

Building a menu is moderately tedious, so we will rely on the standard menu extensions provided. Thus:

"Xerxes"

Include Basic Screen Effects by Emily Short. Include Menus by Emily Short.

Table of Amusing Matter

title	subtable	description	toggle
"Cult Revisions"	--	"Did you try... [paragraph break] banning the worship of Seth? [line break] of Dionysus? [line break] assigning all your priests to Re? [line break] assigning male priests to Cybele? [line break] assigning married priestesses to Hestia? [line break] identifying one god as another (e.g., Isis and Hecate)? [line break] identifying a mortal as a god (e.g., Alexander as Helios-Apollo)?"	--
"Military Revisions"	--	"Did you try... [paragraph break] allying a Greek city-state with the Persians? (try >MEDIZE) [line break] playing Athens as a land-based power?"	--

Rule for amusing a victorious player:
 now the current menu is the Table of Amusing Matter;
 now the current menu title is "Things to Try";
 carry out the displaying activity;
 clear the screen.

Omitting about a half million words from this rigorous and educational but nonetheless enthralling simulation of centuries of history, culture, and religion, we will skip directly to:

Athens is a room.

Use scoring.

Every turn:
 if the score is greater than 10000, end the story finally.

When play begins: now the score is 10001.

Test me with "z".

387

Example **Blankness**

RB

Emptying the status line during the first screen of the game.

Occasionally we want to print something as our first screen and then pause the game. By default, Inform will print a rather odd status line, with "You" on the left side and "0" on the right. This is because the left hand status line is set to display the location, but (because we're not done with the when-play-begins rules) the player has not yet even been moved to a room.

We can tidy this up in the "starting the virtual machine" activity, by temporarily changing the status line content. We will not provide game-pausing code here, because that is easily done by extension; so:

```
"Blankness"

Include Basic Screen Effects by Emily Short.

When play begins:
    say "take me home";
    wait for any key;
    say " yeah";
    wait for any key;
    say " yeah";
    pause the game;
    now the left hand status line is "[location]";
    now the right hand status line is "[turn count]".

Before starting the virtual machine:
    now the left hand status line is "";
    now the right hand status line is "".

Paradise City is a room. The description of Paradise City is "The grass is green
and the girls are pretty."
```

Quite a modest effect, but occasionally useful.

Chapter 19: Rulebooks

Contents of *Writing with Inform*
Chapter 18: Activities
Chapter 20: Advanced Text
Indexes of the examples

§19.1. On rules

When we open the casing and look inside the machinery of Inform, what we see are rules and rulebooks. We seldom need to know how this machinery works, but every once in a while we want to replace components, or even install new mechanisms of our own. And as we shall see, creating new rulebooks can be a neat way to tackle complicated simulations full of exceptions and special cases.

So far we have seen many rules, and the term "rulebook" has frequently but vaguely been used. Here is a summary of the rulebooks seen so far:

```
before
instead
after
check taking, carry out taking, report taking
and three similar rulebooks for each of the 90 or so actions
persuasion
unsuccessful attempt
reaching inside
reaching outside
visibility
does the player mean

when play begins
when play ends
every turn
when Confrontation Scene begins
when Confrontation Scene ends
and two similar rulebooks for each scene we create, if any

before printing the name of
for printing the name of
```

```
after printing the name of
and three similar rulebooks for each of the 20 or so activities
```

Which makes around 340 rulebooks before we even start to write. All the same, not everything in Inform belongs to a rulebook - timed events, for example, are rules which normally live outside of rulebooks; and other constructions, such as newly-created phrases, or definitions, may look vaguely like rules, but they aren't. So the following are not rulebooks:

```
At 11:10 PM: ...
To dislodge the shelf: ...
Definition: ...
```


Start of Chapter 19: Rulebooks


Back to Chapter 18: Activities: §18.40. Starting the virtual machine


Onward to §19.2. Named rules and rulebooks

---

§19.2. Named rules and rulebooks

Most of the rules built into Inform have names. For instance, a rule called "the advance time rule" is the one which increments the number of turns and advances the clock, values which are usually not visible, but are ticking away behind the scenes.

A rulebook is a list of rules to be followed in sequence until one of them makes a decision. For instance, when actions get to the "instead" stage, each "instead" rule is tried until one of them chooses to do something. If the source text contains the rules

```
Instead of taking something: say "You have no particular need just now."
Instead of taking a fish: say "It's all slimy."
```

and a command to TAKE something is tried, then only one of these rules will have any effect. The "instead" rulebook contains:

```
Rule (1) to be applied if the action matches "taking a fish"
Rule (2) to be applied if the action matches "taking something"
```

Inside their rulebook, the rules are not listed in the order of definition in the source text. Rule (1) comes before rule (2) because it applies in more specific circumstances. This is the main idea: a rulebook gathers together rules about making some decision, or taking some action, and sorts them in order to give the more specific rules first choice about whether they want to intervene.

Whereas only some rules are named (the two "instead" rules above have no name, for instance), every rulebook has a name. For convenience, the following forms of rule and rulebook name are synonymous:

```
advance time = the advance time rule
```

```
the instead rules = instead rulebook = instead
```

The names of built-in rules have been chosen as descriptively as possible: the "can't go through closed doors rule", for instance. Names for rules tend to be verbose, but this is a situation where clarity is very much better than brevity.

Start of Chapter 19: Rulebooks

Back to §19.1. On rules

Onward to §19.3. New rules

Example 388: **Nine AM Appointment** A WAIT [number] MINUTES command which advances through an arbitrary number of turns.

Example 389: **Delayed Gratification** A WAIT UNTIL [time] command which advances until the game clock reaches the correct hour.

---

§19.3. New rules

Stretching a point seasonally, we might write:

```
Every turn, say "The summer breeze shakes the apple-blossom."
```

This rule is nameless. It needs no name because it will never need to be referred to: by identifying it as an every turn rule we have already said enough to lodge it in the "every turn" rulebook. In fact, though, it is easy to create a named rule:

```
This is the blossom shaking rule: say "The summer breeze shakes the apple-blossom."
```

The name of a rule must always end with the word "rule", for clarity's sake. (The phrasing "This is the ... rule" is used because "The ... rule" would be open to misinterpretation.)

Previously we had a rule which had no name, but belonged to a rulebook: now we have the opposite, because although the "blossom shaking rule" has a name, it has not been placed in any rulebook. That means it will probably never be applied, unless we give specific instructions for that.

Alternatively, it is possible to both name and place a rule in a single sentence:

```
Every turn (this is the alternative blossom rule): say "The summer breeze shakes the
apple-blossom."
```

Now the "alternative blossom rule" is a named rule in the "every turn" rulebook.

Start of Chapter 19: Rulebooks

Back to §19.2. Named rules and rulebooks

Onward to §19.4. Listing rules explicitly

Example 390: ★ **The Crane's Leg 2** A description text generated based on the propensities of the player-character, following different rulebooks for different characters.

Example 391: ★ **Stone** A soup to which the player can add ingredients, which will have different effects when the player eats.

Example 392: ★★ **Bribery** A GIVE command that gets rid of Inform's default refusal message in favor of something a bit more sophisticated.

---

§19.4. Listing rules explicitly

If rules can manage perfectly well without, why bother to have names for rules?

The answer is that although Inform contains an elaborate mechanism for placing rules into the correct rulebook at the correct position, and this happens automatically, Inform will sometimes get it wrong. It will use a rule we do not want, or place them in an order which does not suit us. To put this right, we can give explicit instructions which take precedence over Inform's normal practice. This is done with the "to list" verb, as in the following examples.

1. The simplest usage is to place a named rule, which currently has no home, in any rulebook of our choice. (This looks redundant, but just occasionally we want the same rule to appear in two different rulebooks.)

```
The blossom rule is listed in the every turn rules.
```

A rule can appear in more than one rulebook, but within any single rulebook it can only appear once.

2. We can also specify that the rule needs to appear before, or after, some other named rule in the same rulebook:

```
The collapsing bridge rule is listed before the moving doorways rule in the instead rules.
```

Instead of being placed in specificity order in the whole "instead" rulebook, the "collapsing bridge" rule would now be placed in specificity order only in the first half of the "instead" rulebook - the rules from the start up to (but not including) the "moving doorways" rule. To reiterate: that doesn't necessarily mean it will be immediately before the "moving doorways" rule; it will be placed according to Inform's usual sorting rules within that range.

"Listed" sentences are obeyed by Inform in sequence, so if later ones issue instructions contradicting earlier ones, it's the later ones which win out. Thus if we say "A is listed before B in X" and then "B is listed before A in X", the result is that B comes before A.

3. We can specify that a rule needs to appear first or last in a given rulebook:

```
The collapsing bridge rule is listed first in the instead rules.
```

Again, if we make several such instructions about the same rulebook then the most recent one wins: "A is listed first in X. B is listed first in X. C is listed first in X." causes rulebook X to begin C, B, A.

4. We can also substitute one rule for another:

```
My darkness rule is listed instead of the can't act in the dark rule in the visibility rules.
```

If rule A is listed instead of rule B in rulebook X, and A was already a rule in rulebook X, then A will move from its previous position to occupy the place where B was, and B will disappear. (In particular rule A will not be duplicated, which would break the principle that no rule occurs twice in the same rulebook.)

5. And we can strike down existing rules, either specifically or in all their applications:

```
The can't act in the dark rule is not listed in the visibility rules.
The can't remove from people rule is not listed in any rulebook.
```

This does not actually destroy the rules in question: they could still, for instance, be put into another rulebook, or even be applied explicitly, as we shall see. But unless we take deliberate action to the contrary, un-listing a rule amounts to abolishing it forever. This is a little drastic, and more subtle effects can be seen in the next section.

Start of Chapter 19: Rulebooks

Back to §19.3. New rules

Onward to §19.5. Changing the behaviour of rules

Example 393: **Saint Eligius** Adding a first look rule that comments on locations when we visit them for the first time, inserting text after objects are listed but before any "every turn" rules might occur.

Example 394: **Uptempo** Adjust time advancement so the game clock moves fifteen minutes each turn.

Example 395: **Verbosity 2** Making rooms give full descriptions each time we enter, even if we have visited before, and disallowing player use of BRIEF and SUPERBRIEF.

Example 396: **Slouching** A system of postures allowing the player and other characters to sit, stand, or lie down explicitly or implicitly on a variety of enterable supporters or containers, or in location.

Example 397: **Swigmore U.** Adding a new kind of supporter called a perch, where everything dropped lands on the floor.

---

§19.5. Changing the behaviour of rules

Here is another way to abolish an already-existing rule:

```
The print final score rule does nothing.
```

The rule continues to be listed in any rulebook it would normally be listed in: but now it doesn't do anything. More usefully, we can attach a condition:

```
The print final score rule does nothing if the score is 0.
```

or:

```
The print final score rule does nothing unless the score is 100.
```

We can also substitute a rule of our own:

```
This is the print fancy final score rule:
    say "Oh my, you scored a mammoth [score]!"

The print fancy final score rule substitutes for the print final score rule.
```

and once again a condition can be applied:

```
The print fancy final score rule substitutes for the print final score rule when the score is
greater than 100.
```

 Start of Chapter 19: Rulebooks

 Back to §19.4. Listing rules explicitly

 Onward to §19.6. Sorting and indexing of rules

 Example 398:  **Access All Areas** The Pointy Hat of Liminal Transgression allows its wearer to walk clean through closed doors.

---

§19.6. Sorting and indexing of rules

The Rules page of the index for a project offers a view of the rulebooks and their contents, with two major exceptions: built-in rules for specific actions are left to the Actions page, and any rules for scene endings or beginnings are left to the Scenes page.

As we have seen, we need to know the name of a rule before we can change its rulebook listing or alter its applicability. The Rules and Actions index pages show the names of the built-in rules, which are not worth memorising. (Typing can be saved by using the paste-into-source buttons, or by selecting a rule's name and copying and pasting it by hand.)

In the Rules index, each rulebook is named and then followed by a list of the rules within it, one on each line: if nothing follows, then the rulebook is currently empty. The rules are given in order, and icons are used which indicate which rules are more specific than which others. Hovering the mouse over such an icon should bring up a "tooltip" which explains Inform's reasoning.

As this suggests, Inform performs its automatic sorting using a precise collection of Laws (the term "rules" would be ambiguous here, so we call these guidelines Laws instead), and the tooltip shows which Law was applied. It is bad style to write source text which absolutely depends on detailed points of these Laws, but they are documented at the end of this chapter for those who do wish to see the full details.

Start of Chapter 19: Rulebooks
Back to §19.5. Changing the behaviour of rules
Onward to §19.7. The preamble of a rule

---

§19.7. The preamble of a rule

In general, a rule looks like this:

```
preamble : list of one or more phrases divided by semicolons
```

though in a few common cases (where the preamble begins with Before, After, Instead of, Every turn, or When, and there is only one phrase in the list) the colon can be replaced with a comma. Three kinds of declaration are special, and these we can tell apart by the first word:

```
To ... - a new phrase: see the chapter on Phrases
At ... - something due to happen at a given time: see Time
Definition: ... - a new adjective: see Descriptions
```

All other declarations (that is, starting with any other word) create rules fit for going into rulebooks. The preamble can either just be a name, which is required to end with the word "rule", or it can give circumstances and have no name, or it can do both:

```
This is the ...name of rule...
...circumstances...
...circumstances... (this is the ...name of rule...)
```

The circumstances should be a sequence of the following ingredients, each of which is optional except the name of the rulebook:

```
first or last
followed by ...rulebook name...
followed by about or for or of or on or rule
followed by ...what to apply to...
followed by while or when ...condition...
followed by during ...a scene...
```

The word "first" or "last", if present, is significant: it tells Inform exactly where the new rule should be placed into its rulebook, and so overrides the normal practice of placing the rule according to how specific it is.

On the other hand, the use of any of the following:

```
for
of
rule about
rule for
rule on
```

is purely to make the text easier to read: Inform does not make any direct use of these words (except perhaps that it may help to avoid ambiguities by separating the rulebook name from what is being applied to). Thus in the rule

```
Instead of kissing Clark: ...
```

the word "instead" is the rulebook's name, while "of" is technically optional. "Instead rule about kissing Clark: ..." would work just as well.

In this whole list of possible ingredients, only the rulebook name is compulsory. We could define a rule called simply "Instead: ..." if we wanted - though its universal applicability would make it pretty disruptive, with every action stopped in its tracks.

Start of Chapter 19: Rulebooks

Back to §19.6. Sorting and indexing of rules

Onward to §19.8. New rulebooks

Example 399: **Backus-Naur form for rules** The full grammar Inform uses to parse rule definitions, in a standard computer-science notation.

---

§19.8. New rulebooks

Creating a new rulebook is also straightforward, as we see in the following modest example story:

```
"Appraisal"

The Passage is east of the Tomb. The green-eyed idol is in the Tomb. A Speak-Your-Progress machine is in the Passage.

Appraisal rules is a rulebook.

An appraisal rule: say "Click... whirr... the score is [the score in words] points."

An appraisal rule:
    if we have taken the idol, say "Most importantly of all, the idol has been found."

Instead of switching on the machine, follow the appraisal rules.
```

The creation of the rulebook is all very well, but without the final sentence it would never be used. The crucial new phrase here is:

follow (rule)

This phrase causes the rule to be obeyed immediately (rather than simply at predetermined times such as when a particular action is being tried, or at the end of every turn, and such). Example:

```
follow the advance time rule;
follow the appraisal rulebook;
```

Like "number" or "text", "rule" and "rulebook" are kinds of value built into Inform: "the blossom rule" is a value whose kind is "rule", whereas "the every turn rules" is a value whose kind is "rulebook". In fact, Inform considers a rulebook to be a special case of a rule, so that whenever a rule is required it is legal to name a rulebook instead, but not vice versa. The "follow" phrase here...

```
Instead of switching on the machine, follow the appraisal rules.
```

...expects to be applied to a value of kind "rule"; "the appraisal rules" is in fact a rulebook, but since that counts as a rule the phrase makes sense to Inform. To follow a rulebook means to run through all its rules in turn, stopping when one rule reaches an outcome; to follow a single rule means just that one, of course.

When created, a rulebook starts out with no rules in it - in this example, of course, we quickly defined a couple of rules to go into it. But it's often the case in Inform that a rulebook exists without ever being stocked up, especially if the rulebook is for some obscure purpose never needed. The built-in adjectives "empty" and "non-empty", applied to a rulebook, test whether any rule is present or not.

Start of Chapter 19: Rulebooks

Back to §19.7. The preamble of a rule

Onward to §19.9. Basis of a rulebook

Example 400: ★★ **Solitude** Novice mode that prefaces every prompt with a list of possible commands the player could try, and highlights every important word used, to alert players to interactive items in the scenery.

Example 401: ★★★ **In Fire or in Flood** A BURN command; flammable objects which light other items in their vicinity and can burn for different periods of time; the possibility of having parts or contents of a flaming item which survive being burnt.

Example 402: ★★★★ **Patient Zero** People who wander around the map performing various errands, and in the process spread a disease which only the player can eradicate.

---

§19.9. Basis of a rulebook

Every rulebook works on a value supplied to it, though it doesn't always look that way. The kind of the value is called its "basis"; for example, if a rulebook works on a number, it's called a "number based rulebook". Most of the rulebooks seen up to now have been action based rulebooks:

```
Instead of eating the cake: ...
```

"Instead" is an action based rulebook, and the action it works on is the one currently being processed. Besides before, after and instead, other action based rulebooks include the check, carry out, and report rules; general rulebooks such as every turn rules, the visibility rules, the

turn sequence rules; and rules specially for dealing with the actions of other characters, such as the persuasion and unsuccessful attempt rules. But we have also seen object based rulebooks:

```
Rule for reaching inside the flask: ...
```

"Reaching inside" is an object based rulebook, and here we're giving it a rule which applies if the object is the flask. Inform would reject something like:

```
Rule for reaching inside 100: ...
```

because 100 has the wrong kind to fit - it's a number, not an object. There are many object based rulebooks, because most activities built-in to Inform act on objects. For example, the "printing the name of" activity has three rulebooks attached to it: before printing the name of, for printing the name of, after printing the name of. All of these are object based rulebooks.

Finally, we've also seen scene based rulebooks (which is how rules like "when a recurring scene ends" worked, in the Scenes chapter).

If a rulebook is declared like so:

```
Marvellous reasoning is a rulebook.
```

then it is an action based rulebook. If we want something different, we must write something like this:

```
Grading is a number based rulebook.
Grading 5: say "It's five. What can I say?" instead.
Grading an odd number (called N): say "There's something odd about [N]." instead.
Grading a number (called N): say "Just [N]." instead.

When play begins:
repeat with N running from 1 to 10:
    say "Grading [N]: ";
    follow the grading rulebook for N.
```

which produces:

```
Grading 1: There's something odd about 1.
Grading 2: Just 2.
Grading 3: There's something odd about 3.
Grading 4: Just 4.
Grading 5: It's five. What can I say?
Grading 6: Just 6.
Grading 7: There's something odd about 7.
Grading 8: Just 8.
Grading 9: There's something odd about 9.
Grading 10: Just 10.
```

Here we needed a variation on "follow" which supplies the value to apply to:

follow (values based rule producing values) **for** (value)

This phrase causes the rule to be obeyed immediately (rather than simply at predetermined times such as when a particular action is being tried, or at the end of every turn, and such), and applies it to the value given, which must be of a matching kind. Example:

```
follow the reaching inside rulebook for the electrified cage;
```

And here is an example based on objects:

```
The flotation rules are an object based rulebook.
A flotation rule for the cork: rule succeeds.
A flotation rule for an inflated thing: rule succeeds.
A flotation rule: rule fails.
```

And we might use the flotation rules in a circumstance like this:

```
After inserting something into the well:
    follow the flotation rules for the noun;
    if the rule succeeded:
        say "[The noun] bobs on the surface.";
    otherwise:
        now the noun is nowhere;
        say "[The noun] sinks out of sight."
```

Start of Chapter 19: Rulebooks

Back to §19.8. New rulebooks

Onward to §19.10. Rulebook variables

Example 403: **Flotation** Objects that can sink or float in a well, depending on their own properties and the state of the surrounding environment.

---

§19.10. Rulebook variables

When a rulebook is intended to perform some complicated task or calculation, it is sometimes useful for earlier rules to be able to leave information which will help later ones.

For instance, suppose we want a rulebook which is intended to print out the player's current aptitude. We will suppose that this is a number from 0 upwards: the higher, the apter. The player gets bonus aptitude marks for achievements, but marks deducted for accidents, and so on. Moreover, we want to design this system so that it's easy to add further rules. The natural solution is to have a number which varies (or 'variable') acting as the running aptitude total: it should start at 0 and be altered up or down by subsequent rules. First, we should make the rulebook, and then add a variable:

```
Aptitude is a rulebook. The aptitude rulebook has a number called the aptitude mark.
```

The new value 'aptitude mark' is shared by the rules of the rulebook: nobody else can see it. It is created at the start of the rulebook being followed, and destroyed at the end. (If the rulebook should be followed a second time inside of itself, a new copy is created which does not disturb the old one.) So, in this case, 'aptitude mark' is started as 0 (since it is a number) each time the aptitude rules run. We can then write whatever rules we please to modify it:

```
An aptitude rule:
    if in darkness:
        decrease the aptitude mark by 3.

An aptitude rule:
    if we have taken the idol:
        increase the aptitude mark by 10.
```

And we had better do something with the result:

```
The last aptitude rule: say "Your aptitude rating is [aptitude mark]."
```

A rulebook can have any number of variables like this. They behave much like "let" values except that they last for a whole rulebook, not an individual rule or To phrase definition. (Well, strictly speaking they are accessible not just to the rules which belong to the rulebook, but also to any rules which previously belonged to the rulebook but were kicked out by means of an explicit rule-listing sentence. This is good because otherwise they will suddenly cause problem messages when unlisted.)


Start of Chapter 19: Rulebooks


Back to §19.9. Basis of a rulebook


Onward to §19.11. Success and failure

---

§19.11. Success and failure

Though we have blurred over this point so far, each rule must ordinarily end with one of three outcomes: success, failure and neither ("no outcome").

When a rulebook is followed, what happens is that each of its rules is followed in turn **until one of them ends in success or failure** - if ever: it is possible that each rule is tried and each ends with no outcome, so that the rulebook simply runs out of rules to try.

For some rulebooks, these are not useful ideas: "every turn" rules, for instance, by default never produce an outcome, which is why the "every turn" rulebook normally runs through all its rules at the end of each turn. (Use of the phrases below can change that, so it's best not to use them in "every turn" rules.) But for other rulebooks, such as "check taking", it's important that a rule which fails will stop the whole rulebook. For instance, we might find that the "can't take yourself rule" produces no outcome (because we aren't trying to do that), and then likewise the "can't take other people rule" (ditto) but that the "can't take component parts rule" prints up a complaint, and fails the action: the rulebook stops, and never goes on to (for instance) the "can't take scenery rule". This is good, because an impossible action

often fails for several reasons at once, and we only want to print up one objection, not a whole list.

To follow the working of this mechanism, we need to be able to predict the outcome of any given rule. Sometimes this is easy to spot. For instance, in a rule which works on actions:

```
continue the action; means "end this rule with no outcome"
stop the action; means "end this rule in failure"
... instead; means "end this rule in failure"
```

("Success" and "failure" are technical terms here: they do not mean that the player has or hasn't got what he wanted.) This is why the rule:

```
Before taking something: say "The sentry won't let you!" instead.
```

ends in failure, and therefore stops the "before" rulebook. Another easy-to-spot case is when a rule makes use of the explicit phrases:

rule succeeds

This causes the current rule to end immediately, with its outcome considered to be a success. The rulebook being worked through also ends, and is also a success.

rule fails

This causes the current rule to end immediately, with its outcome considered to be a failure. The rulebook being worked through also ends, and is also a failure.

make no decision

This causes the current rule to end immediately, but with no outcome. That means the rulebook being worked through will continue to run on, beginning with the next rule.

But what happens if a rule simply doesn't say whether it succeeds, fails or has no outcome? In that case **it depends on the rulebook**. For almost all rulebooks, a rule which doesn't make a choice has no outcome, as in the following example:

```
Before taking something: say "The sentry looks at you anxiously!"
```

This rule, if it takes effect, ends with no outcome - so the action continues. But other rulebooks have a different convention: the most important is "instead", where a rule making no explicit choice is deemed to end in failure. For instance:

```
Instead of taking something: say "The sentry prods you with his rifle!"
```

This rule, if it takes effect, ends in failure and therefore stops the action.

We call this the **default outcome** of a rulebook. The default outcome of "before" (and of almost all rulebooks, in fact) is no outcome; the default outcome of "instead" is failure; the default outcome of "after" is success. The few exceptional cases with default outcome success or failure are marked as such in the Rules index.

When we create a rulebook, it will default to "no outcome". But we can specify otherwise with sentences like so:

```
The cosmic analysis rules are a rulebook. The cosmic analysis rules have default failure.
```

Finally, note that the default outcome for a rulebook is really the default outcome for any rule in that rulebook: if no rules in the rulebook ever apply, for instance if there aren't any and the rulebook is empty, then the rulebook ends with no outcome at all.

We can test the latest outcome like so:

if rule succeeded:

This condition is true if the most recently followed rule or rulebook ended in success. Example:

```
follow the hypothetical clever rule;
if rule succeeded:
    ...
```

if rule failed:

This condition is true if the most recently followed rule or rulebook ended in failure. Example:

```
follow the hypothetical clever rule;
if rule failed:
    ...
```

Note that this is not the opposite of "rule succeeded", because there's a third possibility: that it ended with no outcome.

Start of Chapter 19: Rulebooks

Back to §19.10. Rulebook variables

Onward to §19.12. Named outcomes

Example 404: **Kyoto** Expanding the effects of the THROW something AT something command so that objects do make contact with one another.

---

§19.12. Named outcomes

We have seen that the terms "success" and "failure" can be misleading - after all, it might be a good thing for a particular rulebook to "fail". At any rate, these are vague terms, and we don't want to have to remember the conventions used by every rulebook. This is why certain rulebooks have explicitly named outcomes instead. For instance, the "visibility" rules are allowed to have the outcomes:

```
there is sufficient light;
there is insufficient light;
```

These look like phrases, but are in fact named outcomes which can only be used in visibility rules. (They would make no sense elsewhere, and Inform will not allow their use if they are clearly out of context.) Such named outcomes are listed in the Rules index.

There can be any number of named outcomes. For instance, the Standard Rules define:

```
The does the player mean rules are a rulebook. The does the player mean rules have
outcomes it is very likely, it is likely, it is possible, it is unlikely and it is very unlikely.
```

which makes five possible outcomes. Five outcomes seems to contradict the principle that there are only three possible outcomes for a rule: in fact, though, the five are counted as five different forms of "success", and any of them will cause a "does the player mean" rule to succeed. If we do not want this, we can instead specify explicitly how the named outcomes correspond to success, failure or "no outcome":

```
Visibility rules have outcomes there is sufficient light (failure) and there is insufficient
light (success).
```

Again, see the Rules index for examples.

The same named outcome can be used for more than one rulebook, and can have different meanings in the context of different rulebooks - "good news" could be defined as success in one rulebook and failure in another, for instance. (This means that rulebook creators need not worry about name clashes and is an important difference in behaviour between rulebook outcomes and kinds of value.) We can even name a specific named outcome as the default outcome for rules in this rulebook:

```
Audibility rules have outcomes high background noise (failure), low background noise
(success - the default) and absolute silence (success).
```

After a rulebook using named outcomes has run, we can test which outcome occurred by using the phrase:

outcome of the rulebook ... rulebook outcome

This phrase produces the (named) outcome of the phrase most recently followed. Example:

```
follow the audibility rules;
if the outcome of the rulebook is the absolute silence outcome:
    say "You could hear a pin drop in here."
```

Each named outcome is a value if followed by the word "outcome", which is how "absolute silence" has become "the absolute silence outcome". Named outcomes can be said, so we could use the text substitution "[outcome of the rulebook]", for instance. A final caveat: it is perfectly legal to create a named outcome which means "no outcome", but if so then this will never be "the outcome of the rulebook" because "no outcome" is not an outcome.

Start of Chapter 19: Rulebooks

Back to §19.11. Success and failure

Onward to §19.13. Rulebooks producing values

Example 405: **Being Peter** A set of rules determining the attitude a character will take when asked about certain topics.

---

§19.13. Rulebooks producing values

We have now seen two ways to write the outcome of a rule: as simple success or failure, with more or less explicit phrases like:

```
rule succeeds;
rule fails;
continue the action;
stop the action;
```

and by using a named outcome for the current rulebook as if it were a phrase, as in:

```
low background noise;
```

There is still a third way: we can stop a rule and at the same time produce a value. This isn't needed very often - none of the built-in rulebooks in the Standard Rules produces a value.

As we've seen, every rulebook has one kind of value as its basis, and it also has another kind of value for it to produce. If we call these K and L, then we have altogether four ways to write down the kind of a rulebook:

```
rulebook
K based rulebook
rulebook producing L
K based rulebook producing L
```

If we don't mention K, Inform assumes the rulebook is action based. If we don't mention L, Inform assumes L is "nothing", that is, Inform assumes no value is ever produced. Thus

```
Drum summons rules is a rulebook.
```

is equivalent to

```
Drum summons rules is an action based rulebook producing nothing.
```

But let's now look at a rulebook which does produce something.

```
The cat behavior rules is a rulebook producing an object.
```

This rulebook works out which thing the cat will destroy next. We might have rules like this one:

```
Cat behavior when Austin can see the ball of wool:
    rule succeeds with result the ball of wool.
```

The value is produced only when a rule succeeds, using this phrase:

> **rule succeeds with result** (value)
>
> This phrase can only be used in a rule which produces a value, and the value given must be of the right kind. It causes the current rule to finish immediately, to succeed, and to produce the value given.

How are we to use the cat behavior rulebook? If we write:

```
follow cat behavior
```

then the rulebook runs just as any other rulebook would, but the value produced is lost at the end, which defeats the point. Instead, we might write:

```
Every turn:
    let the destroyed object be the object produced by the cat behavior rules;
    if the destroyed object is not nothing:
        say "Austin pounces on [the destroyed object] in a flurry.";
        now the destroyed object is nowhere.
```

The key phrase here is

```
object produced by the cat behavior rules
```

which accesses the value this rulebook produces. In general, we write:

(name of kind) **produced by** (rule producing values) **... value**

This phrase is used to follow the named rule, and to collect the resulting value.

(name of kind) **produced by** (values based rule producing values) **for** (value) **... value**

This phrase is used to follow the named rule based on the value given, and to collect the resulting value.

Start of Chapter 19: Rulebooks

Back to §19.12. Named outcomes

Onward to §19.14. Abide by

Example 406: ★ **Feline Behavior** A cat which reacts to whatever items it has handy, returning the result of a rulebook for further processing.

Example 407: ★★★ **Tilt 2** A deck of cards with fully implemented individual cards; when the player has a full poker hand, the inventory listing describes the resulting hand accordingly.

---

§19.14. Abide by

It often happens that one rule needs to invoke another one. Most of the time, the best way to do this is with "follow":

```
follow the magical mystery tour rule;
```

More often, though, we want not only to invoke another rule, but also to be guided by its advice. For this, we use the otherwise identical phrase:

abide by (rule)

This phrase applies the given rule, and makes its result the result of the present rule. If the rule being abided by succeeds or fails then the original rule also stops, at once and without going on to any further instructions. Example:

```
The omnibus rule:
    abide by the first rule;
    abide by the second rule;
    abide by the third rule;
    abide by the fourth rule.
```

This duplicates the effect of a rulebook of four rules: the "omnibus rule" tries each in turn, and stops as soon as any of them stop.

abide by (values based rule producing values) **for** (value)

This phrase applies the given rule to the given value, and makes its result the result of the present rule. If the rule being abided by succeeds or fails then the original rule also stops, at once and without going on to any further instructions.

Abide might be used in examples like this one:

```
A thing can be fragile or robust.
This is the can't handle fragile things roughly rule: if the noun is fragile, say "[The noun] is too fragile for such rough handling." instead.
A check dropping rule: abide by the can't handle fragile things roughly rule. A check throwing it at rule: abide by the can't handle fragile things roughly rule.
```

Had we used "follow" instead of "abide by", then in the event of the player typing "drop angel" the text "The glass angel is too fragile for such rough handling" would be printed, which is correct - but then the action would continue as though no difficulty had occurred, which is definitely not correct.

Finally, we can "anonymously abide":

anonymously abide by (rule)

or:
anonymously abide by (values based rule producing values) **for** (value)

This phrase applies the given rule, and makes its result the result of the present rule. If the rule being abided by succeeds or fails then the original rule also stops, at once and without going on to any further instructions. However, the rule deemed to have decided the outcome is the one abided by, not the one doing the abiding.

This is only useful in complicated situations where one rulebook uses another which... and so on. Its effect is exactly the same as "abide", except that the rule deemed to have decided the outcome is the one abided by, not the one doing the abiding. It thus allows a rule or rulebook to act purely as a middle-man, never getting the blame or the credit for what happens. The rule which made the decision is often not very relevant anyway, but it's used as the source of the value "reason the action failed" (see the Advanced Actions chapter).

 Start of Chapter 19: Rulebooks

 Back to §19.13. Rulebooks producing values

 Onward to §19.15. Two rulebooks used internally

---

§19.15. Two rulebooks used internally

Rulebooks handle almost all of the important tasks which an Inform work of IF must carry out in order to keep play going. We have seen them used in clarifying the player's command, supplying missing ingredients, processing the action to see what should happen, responding, and so on: by this point in the documentation, it must look as if Inform uses rulebooks for everything.

This is nearly true. There is not actually a super-rulebook controlling everything. (Such a super-rulebook would need to repeat itself and never finish, something a rulebook is not allowed to do.) Instead, what happens during play looks like so:

1. Following the "when play begins" rulebook.
2. Repeating:
   2(a). Reading and parsing a command into an action;
   2(b). Following the "action processing" rulebook;
   2(c). Following the "turn sequence" rulebook.

until the story has finished.

3. Following the "when play ends" rulebook.

The command parser occasionally calls on the services of activity rulebooks to help it, but otherwise gets on with its job in ways that we do not "see" as Inform 7 users. The rest of what happens involves rulebooks, and in particular two important beneath-the-surface rulebooks: action processing and turn sequence.

The **action processing rules** are used whenever an action must be tried, by whoever tries it. This usually happens in response to player commands, but not always: it might happen because of a "try...", and it can certainly interrupt an existing action.

The **turn sequence rules** are used at the end of each turn, and include housekeeping as well as timekeeping. They consult the "every turn" rulebook, and advance the time of day, among other useful tasks.

In general, we should only modify the operation of these two crucial rulebooks as a last resort. Play can evidently fall to pieces if they cease to work normally.

Start of Chapter 19: Rulebooks

Back to §19.14. Abide by

Onward to §19.16. The Laws for Sorting Rulebooks

Example 408: **Electrified** Adding a rule before the basic accessibility rule that will prevent the player from touching electrified objects under the wrong circumstances.

Example 409: **Timeless** A set of actions which do not take any game time at all.

Example 410: **Endurance** Giving different actions a range of durations using a time allotment rulebook.

Example 411: **Escape from the Seraglio** Replacing the usual response to TAKE ALL so that instead of output such as "grapes: Taken. orange: Taken.", Inform produces variable responses in place of "grapes:".

---

§19.16. The Laws for Sorting Rulebooks

Large works created by Inform are heaped high with rules, most of them instead rules, but with a leavening of befores and afters as well. What will happen if these conflict with each other? For instance:

> Instead of opening a container, say "Your mother-in-law looks on with such evident disappointment that you withdraw your hand again."
>
> Instead of opening an open container, say "Your daughter tuts in theatrical exasperation at your, like, lameness."

Such clashes are resolved by sorting the rulebooks in order of specificity: thus your daughter gets in before your mother-in-law, because although both have rules hanging on the "opening" action, "an open container" is more specific than "a container". The full set of Laws used by Inform to sort rulebooks is quite elaborate. As we've seen, practical consequences can be investigated using the Rules index; and in most cases, the results are either natural (as above) or irrelevant (because the two rules being compared could not both activate at the same time anyway); but the full set of Laws is laid out below, for reference. It is probably a bad idea to write source text which absolutely relies on non-obvious rule sorting conventions, just the same, because this will make the source text harder to read and understand.

Sorting is done by comparing rules in pairs to decide which is more specific. We shall call these rules X and Y. The Laws are tried in sequence; the first Law to distinguish X and Y gets to decide which is more specific. If no Law is able to decide, X and Y go into the rulebook in order of their appearance in the source text - that is, whichever is defined first appears earlier in the rulebook and therefore takes priority.

Law I - Number of aspects constrained. For action-based rulebooks, rules are scored from 0 to 6 according to whether they constrain any of: (i) the exotic "going" clauses (pushing, by and through), (ii) the location of the action (in, from and to), (iii) the things directly involved (actor, noun, second noun, "nowhere" in the case of "going"), (iv) the presence of others (in the presence of...), (v) the time at which the action occurs (when, or "for the nth time" or "for the nth turn"), and/or (vi) the scene the action occurs in (during). For value based rulebooks, rules are scored from 0 to 3 according to whether they constrain: (i) the value parameter, (ii)

the scene in which the rulebook is followed (when, during), and/or (iii) any condition which must hold or activities going on at the same time (when/while). A higher score is more specific than a lower one.

Law II - When/while requirement. A rule with a when/while restriction beats one without.

Law III - Action requirement. A rule with a more specific action requirement beats one with a more general action requirement. (Or similarly, for value based rulebooks, a rule with a more specific parameter requirement beats a more general one.) Details are given below.

Law IV - Scene requirement. A rule with a scene restriction ("during") beats one without.

Details of Law III now follow:

Law III.1 - Object To Which Rule Applies. For value based rulebooks only: the more specific value requirement wins.

Law III.2.1 - Action/Where/Going In Exotic Ways. A more specific combination of "...pushing...", "... by ...", and "... through ..." clauses in a "going" action beats a less specific. (Placing conditions on all three of these clauses beats placing conditions on any two, which beats any one, which beats none at all.) In cases where X and Y each place, let's say, two such conditions, they are considered in the order "...pushing...", "...by..." and then "...through..." until one wins. (The idea here is that pushing something from room to room is rarer than travelling in a vehicle, which in turn is rarer than going through a door. The rarer action goes first, as more specific.)

Law III.2.2 - Action/Where/Room Where Action Takes Place. A more specific combination of conditions on the room in which the action starts, and in which it ends, beats a less specific. For all actions other than "going", there is no combination to be considered, and what we do is to look at the specificity of the "... in ..." clause. (So "Before looking in the Taj Mahal" beats "Before looking".)

For "going" actions, there are strictly speaking three possible room clauses: "... in ...", "... from ..." and "... to ...". However, "... in ..." and "... from ..." cannot both be present, so that in practice a "going" rule constraining two rooms beats a "going" rule constraining only one.

If both the room gone from (the "...in..." or "...from..." room, whichever is given) and the room gone to (the "... to..." room) are constrained, then the constraints are looked at in the order from-room followed by to-room, since an action which goes to room Z could start in many different places and thus is likely to be more general.

Giving a place as a specific room beats giving only the name of a region; if region R is entirely within region S, then a rule applying in R beats a rule applying in S. (Note that regions can only overlap if one is contained in the other, so this does not lead to ambiguity.)

Law III.2.3 - Action/Where/In The Presence Of. A more specific "...in the presence of..." clause beats a less specific one. (This is again a constraint on where the action can take place, but it's now a potentially a constraint which could be passed in many different places at different times, so it's the most likely to be achieved and therefore the last to be considered of the Laws on Where.)

Law III.3.1 - Action/What/Second Thing Acted On. A more specific constraint on the second noun beats a less specific. Thus "putting something in the wooden box" beats

"putting something in a container".

Law III.3.2 - Action/What/Thing Acted On. A more specific constraint on the first noun beats a less specific. Thus "taking a container which is on a supporter" beats "taking a container".

In the case of "going" actions, the first noun is a direction. The special constraint "going nowhere" (which means: a direction in which the actor's location has no map connection) is considered more general than any other constraint placed on the first noun, but more specific than having no constraint at all. Thus "Instead of going north" beats "Instead of going nowhere" which beats "Instead of going".

Law III.3.3 - Action/What/Actor Performing Action. A more specific constraint on the actor beats a less specific.

Law III.4.1 - Action/How/What Happens. A more specific set of actions beats a less specific. For instance, "taking" beats "taking or dropping" beats "doing something other than looking" beats "doing something". A named kind of action (such as "behaving badly") is more specific than "doing something", but considered less specific than any explicitly spelled out list of actions.

Law III.5.1 - Action/When/Duration. An action with a constraint on its history ("for the fifth time", say, or "for the fifth turn") beats one without. If both rules place constraints on history, then the one occurring on the smaller number of possible turns wins (thus "for the third to seventh time" - 5 possible turns of applicability - beats "for less than the tenth turn" - 9 possible turns).

Law III.5.2 - Action/When/Circumstances. A more specific condition under "...when..." beats a less specific one. These conditions could potentially be complex: Inform judges how specific they are by counting the clauses found in them. The more clauses, the more specific the condition, it is assumed.

Law III.6.1 - Action/Name/Is This Named. A rule with a name ("the apple blossom rule", say) beats a rule without.

Start of Chapter 19: Rulebooks

Back to §19.15. Two rulebooks used internally

Onward to Chapter 20: Advanced Text: §20.1. Changing texts

Examples from Chapter 19: Rulebooks

Start of this chapter

Chapter 20: Advanced Text

Indexes of the examples

388 **Example** **Nine AM Appointment** *RB*

A WAIT [number] MINUTES command which advances through an arbitrary number of turns.

If there's some reason the player needs to be at a specific place and time, we might want to allow him to wait a number of minutes at once.

```
"Nine AM Appointment"

Waiting more is an action applying to one number.

Understand "wait [a time period]" or "wait for [a time period]" or "wait for a/an [a time period]" or "wait a/an [a time period]" as waiting more.

Carry out waiting more:
    let the target time be the time of day plus the time understood;
    decrease the target time by one minute;
    while the time of day is not the target time:
        follow the turn sequence rules.
```

The one nuance here is that after our wait command occurs, the turn sequence rules will occur one more time. So we need to subtract one minute from the parsed time to make the turn end on the desired number of minutes.

```
Report waiting more:
    say "It is now [time of day + 1 minute]."
```

And if we want to ensure that the player doesn't (accidentally or intentionally) put the interpreter through a really long loop, we could put an upper limit on his patience:

```
Check waiting more:
    if the time understood is greater than one hour, say "You really haven't got that kind of patience." instead.

The Specialist's Office is a room. The secretary is a woman in the Office. Instead of asking the secretary about "[appointment]", say "'Hang on just five more minutes,' she says, in a distracted manner."

Understand "appointment" or "specialist" or "doctor" as "[appointment]".

At 9:45 AM: say "At [the time of day in words], secretary glances at you and gives a reassuring smile."

Test me with "ask secretary about appointment / wait five minutes / g / g / wait 61 minutes / wait for half an hour / wait for a quarter of an hour / wait for an hour".
```

---

389 **Example Delayed Gratification** RB

A WAIT UNTIL [time] command which advances until the game clock reaches the correct hour.

```
"Delayed Gratification"
```

```
Hanging around until is an action applying to one time.

Check hanging around until:
    if the time of day is the time understood, say "It is [time understood] now!"
instead;
    if the time of day is after the time understood, say "It is too late for that now."
instead.

Carry out hanging around until:
    while the time of day is before the time understood:
        follow the turn sequence rules.

Report hanging around until:
    say "You yawn until [time understood]."

Understand "wait until [time]" as hanging around until.

The Empty Field is a room. "It's an ordinary empty field. Nothing to see here at
all-- yet. Wait until 11:45 PM, though."

At 11:45 PM:
    say "Suddenly the air is filled with light and the sounds of an approaching
band. Over the crest of the hill comes a parade of singing, stomping, hooting
people: and not just people, but dogs, horses, elephants, giraffes... There are
banners, and candles, and a flag that glows eerie-green in the dark; there is a
float shaped like an enormous turtle, its shell covered with winking green lights;
there is an old man dressed as a skeleton, carried in a litter, his neck garlanded
with dried chiles. There are small girls throwing rose petals from a basket, and
grown women half-naked carrying the emblems of Bacchic revelry, and two little
boys each with a silver basin of clear water. All these go by in procession, and
you join on at the end.";
    end the story finally.

Test me with "look / z / z / wait until 11:45 PM".
```

390

Example **The Crane's Leg 2**

RB

A description text generated based on the propensities of the player-character, following different rulebooks for different characters.

Names of rules can be listed in tables. This is convenient if, for instance, we decide that we'd like to swap the rules we use for a specific purpose, as in this continuation of our earlier example of automated description:

```
"The Crane's Leg, Grown Longer"

Material is a kind of value. The materials are wood, glass, stone, cloth, paper,
clay, and metal. A thing has a material.

Color is a kind of value. The colors are red, orange, yellow, green, blue, indigo,
violet, black, brown, and white. A thing has a color. A thing is usually white.
```

A height is a kind of value. 3 feet 11 inches specifies a height. A thing has a height. Definition: a thing is tall if its height is 6 feet 0 inches or more. Definition: a thing is short if its height is 2 feet 0 inches or less.

Imitation relates various things to one thing (called the ideal). The verb to imitate means the imitation relation.

A table is a kind of supporter. A table is usually wood. The height of a table is usually 3 feet 8 inches. The ordinary table is a table. Every table imitates the ordinary table.

A rock is a kind of thing. A rock is usually stone. The ordinary rock is a rock. The height of a rock is usually 0 feet 3 inches. Every rock imitates the ordinary rock.

The description of a thing is usually "[comparison with ideal][run paragraph on]".

To say comparison with ideal:
    say "You observe [the noun]:[paragraph break]";
    choose row with character of the player in Table of Descriptive Reporting;
    follow instructions entry.

This is the comparative observation rule:
    let the sample be the ideal of the noun;
    if the sample is not a thing:
        say "Nothing special, really.";
        rule succeeds;
    if the material of the noun is not the material of the sample:
        if the height of the noun is not the height of the sample:
            if the noun is shorter than the sample, say "Unusually short at [height of the noun], and made of [material of the noun].";
            otherwise say "Unusually tall at [height of the noun], and made of [material of the noun].";
        otherwise:
            say "Distinct mostly in being made of [material of the noun].";
    otherwise:
        if the height of the noun is not the height of the sample:
            if the noun is shorter than the sample, say "Unusually short at [height of the noun].";
            otherwise say "Unusually tall at [height of the noun].";
        otherwise:
            say "In every respect [a sample]."

The Pleasure Garden is a room. "At the riverbank, a pleasing garden, having many curving paths and one straight."

The low table is a table in the Pleasure Garden. The height of the low table is 2 feet 3 inches. On the low table is a yellow metal rock called a gold nugget. A willow is in the Pleasure Garden. The height of the willow is 20 feet 2 inches.

Understand "possess [any person]" or "be [any person]" as possessing.

Possessing is an action applying to one thing. Carry out possessing: now the player is the noun; say "You swap bodies!"

The crane is a person in the Garden. The height of the crane is 4 feet 0 inches.

```
Table of Descriptive Reporting

    character instructions
    yourself   comparative observation rule
    crane      bird observation rule

This is the bird observation rule:
    if the noun is shorter than the player, say "Small, like a duck[if the color of the
noun is not white]; and [color of the noun][end if].";
    otherwise say "Supremely tall[if the color of the noun is not white] and [color of
the noun][end if]."

Test me with "examine table / examine nugget / examine willow / possess crane
/ examine table / examine nugget / examine willow".
```

391

Example Stone

RB

A soup to which the player can add ingredients, which will have different effects when the player eats.

A thing can have a rule as a property, if we like. Here we are going to allow the player to make a soup whose effects will depend on its ingredients. Each ingredient will have its own "food effect" rule, to be followed when the food is eaten.

Note that there are other, slightly less cumbersome ways to do the same thing -- we will see in the chapter on Rulebooks that we could make a "food effects rulebook" and then write a number of rules such as "food effects rule for carrots" or "food effects rule for the stone". Nonetheless, we demonstrate rules-as-properties here for the sake of thoroughness.

So:

```
"Stone"

A food is a kind of thing that is edible. A food has a rule called the food effect.

Carry out eating a food:
    if a food is part of the noun:
        repeat with item running through things which are part of the noun:
            if item is a food, follow the food effect of the item;
    follow the food effect of the noun.

Report eating a food:
    say "You eat [the noun]. [diagnosis of the player]";
    stop the action.

To say diagnosis of (victim - a person):
    if the victim is ill:
        say "You are ill.";
        rule succeeds;
    otherwise:
        say "You are healthy. ";
```

```
    if the victim is awake, say "You are wide awake. ";
    otherwise say "You are sleepy. ";
    if the victim is bright-eyed, say "Your eyesight is clear. ";
    otherwise say "Your eyesight is dim. ";
    if the victim is weak, say "You are weak. ";
    otherwise say "You are strong. ";
    if the victim is hungry, say "You are hungry.";
    otherwise say "You are well-fed."
```

And now to provide some particular foods:

```
Some carrots are a food. The food effect of carrots is the bright-eye rule. This is
the bright-eye rule: now the player is bright-eyed.

Some potatoes are a food. The food effect of the potatoes is the sleepiness rule.
This is the sleepiness rule: now the player is sleepy.

The broth is a food. The indefinite article of the broth is "some". The food effect
of broth is the filling rule. This is the filling rule: now the player is full.

The hambone is a food. The food effect of the hambone is the heartiness rule.
This is the heartiness rule: now the player is strong. Instead of eating the
hambone: say "You cannot just eat a bone!"

The poison ivy is a food. "Poison ivy grows nearby." The food effect of poison is
the illness rule. This is the illness rule: now the player is ill.

A person can be bright-eyed or blind. The player is blind.

A person can be well or ill. The player is well.

A person can be hungry or full. The player is full.

A person can be strong or weak. The player is weak.

A person can be awake or sleepy. The player is sleepy.

The broth is in the kettle. The kettle is on the fire. The fire is in the Clearing. The
Clearing is a room.

The player carries the hambone, the potatoes, and the carrots. The ivy is in the
clearing.

Instead of examining the broth:
    if something is part of the broth, say "In the broth, [a list of things that are part
of the broth] float[if exactly one thing is part of the broth]s[end if].";
    otherwise say "It is just a thin broth with no other ingredients."

Instead of inserting something into the broth: try inserting the noun into the
holder of the broth.

Instead of taking the broth: say "You cannot take the broth in your bare hands."
```

And the following is a relatively unimportant nicety:

```
To sink is a verb.

After inserting a food which is not the broth into a container which contains the
broth:
    now the noun is part of the broth;
    say "[The noun] [sink] into [the second noun], making the broth richer."

Test me with "x broth / eat hambone / put hambone in kettle / x broth / put
potatoes in broth / x broth / eat carrots / eat broth / put ivy in kettle / eat ivy".
```

392

Example **Bribery**

RB

A GIVE command that gets rid of Inform's default refusal message in favor of something a bit more sophisticated.

If we want to rewrite the functionality of a command that usually ends with a "block..." rule, we will have to begin by turning the blocking off.

```
"Bribery"

The block giving rule is not listed in the check giving it to rules.
```

As it happens, correct behavior is built into the GIVE command once "block giving" is turned off, so we do not have to write a replacement report or carry-out rule; the object will be transferred to the possession of the caterpillar. But we do want to adjust the action just a little so that our gift cheers up the recipient:

```
Carry out giving (this is the gratitude for gifts rule): improve the mood of the
second noun.

Mood is a kind of value. The moods are hostile, suspicious, indifferent, friendly,
and adoring. An animal has a mood. An animal is usually indifferent.

To improve the mood of (character - an animal):
    if the mood of character is less than friendly, now the mood of the character is
the mood after the mood of the character.
```

Now whenever we give something to an animal, the animal will be pleased about the present. Of course, we might also want to add a check rule to giving, to see whether the offering is something the recipient really wants:

```
Check giving (this is the polite refusal of unwanted objects rule):
    unless the noun interests the second noun:
        say "[The second noun] disdainfully refuses [the noun]." instead.

To decide whether (item - a thing) interests (character - a person):
    if the character has the item, no;
```

```
    if the item is edible, yes;
    no.

Instead of showing something to someone:
try giving the noun to the second noun.
```

There is already a perfectly workable report rule that will describe what happens when we give something to someone, but let's say we want to report on the recipient's changed mood, too:

```
After giving something to someone:
    say "You give [the noun] to [the second noun], who appears mollified and is
now merely [mood of the second noun]."
```

And the rest is all scenario:

```
The Leafy Branch is a room. "You stand on smooth bark dappled by sunlight.
The scent-trail runs forward to home.

The branch continues forward and backward from here, and a stem extends
forward-up."

Instead of going south in Leafy Branch, say "You must not back down! The scent
trail leads onward!"

The Very Hungry Caterpillar is a hostile animal in the Leafy Branch. "[The
Caterpillar] looks [mood]." Instead of examining the Caterpillar, say "[The
caterpillar] appears [mood]."

The player carries an edible thing called a peanut crumb. The carrying capacity
of the player is 1. After taking something, say "You lift [the noun], though it is
nearly your own size."

Instead of going north in the presence of a hostile caterpillar:
    say "[The Caterpillar] moves to block your exit, glaring down at you with all the
bristles on its skin extended to full size."

Instead of going north in the presence of a suspicious caterpillar:
    say "[The Caterpillar] moves to block your exit, though it might allow you past
if you offered further tribute."

The Leaf Face is above the branch. "The smooth and shiny surface of the leaf
extends forward from here, but you have lost the scent-trail. This is not the way
home." The pear fragment is an edible thing in Leaf Face. The dead aphid is a
thing in Leaf Face.

The Twig is north of Leafy Branch. "The scent-trail is weak but not entirely gone,
and you pursue it faithfully..."

After going to the Twig:
    say "The scent-trail is weak but not entirely gone, and you pursue it
faithfully...";
    end the story finally.
```

```
Understand "forward-up" as up. Understand "forward" as north. Understand
"backward" as south. Understand "backward-down" as down.

Test me with "forward / give crumb to caterpillar / forward / forward-up / get aphid
/ get fragment / down / give aphid to caterpillar / drop aphid / forward-up / get
fragment / down / give fragment / forward".
```

393

Example Saint Eligius

RB

Adding a first look rule that comments on locations when we visit them for the first time, inserting text after objects are listed but before any "every turn" rules might occur.

A not-infrequent desire in IF is to provide a few lines of comment when the player first enters a new room, after the objects are described but before anything else (such as an every turn rule) can fire. The cleanest, most systematic solution is to add a rule to the carry out looking rulebook, so:

```
"Saint Eligius"

The first look rule is listed after the room description paragraphs about objects
rule in the carry out looking rules. A room can be commented or uncommented.
A room is usually uncommented.

This is the first look rule:
    if the location is uncommented, carry out the gawking at activity with the
location.

Gawking at something is an activity.

Rule for gawking at the Diamond Market:
    say "Your throat closes and your eyes begin to sting. You have long disdained
pomp and luxury, and railed against the oppression that brings such wealth to
some men at the cost of the lives of others; you were not prepared for the
magnificence."

After gawking at a room (called the target): now the target is commented.
```

And now the scene itself:

```
The Cobbled Alley is a room. "The Alley has never been made into a proper
street: the buildings on either side are simply too important to tear down. For all
that, there isn't much sign of the magnificence nearby. The entrance you seek is
set below street level, four grimy steps down to a half-basement."

After going to Diamond Market:
    say "You descend the steps quickly and step into the small foyer, allowing
yourself to be searched for weapons, before going on to...";
    continue the action.
```

Diamond Market is down from Cobbled Alley. "The roof is vaulted and painted in allegorical images representing Plenty, the Riches of the Earth, and Saint Eligius, patron of goldsmiths and jewelers.

Under their watchful eye, dozens of men in sober black robes sit; and on the tables before them are rubies, emeralds, sapphires from oversea, but most of all diamonds, both raw and cut."

The burly guard is a man in Diamond Market. "A burly guard patrols quite close to you, but even he is more sumptuously dressed than the average burly guard, and his buttons shine."

Test me with "d / look".

---

394 **Example Uptempo** RB

Adjust time advancement so the game clock moves fifteen minutes each turn.

Suppose a game in which all actions take a very long time. Here's a simple implementation:

"Uptempo"

The fast time rule is listed instead of the advance time rule in the turn sequence rules.

This is the fast time rule:
    increment the turn count;
    increase the time of day by 15 minutes.

When play begins: now the right hand status line is "[time of day]".

The Temporal Hot Spot is a room.

Test me with "z / z".

This works fine as it stands, but we may run into some difficulty with it if we add scheduled events:

At 9:30 AM:
    say "Two turtles run by, almost too fast to see."

At 9:37 AM:
    say "A snail blitzes past."

At 9:42 AM:
    say "The grass grows."

At 9:50 AM:
    say "Several flowers burst open."

Time is counted forward after the schedule has already been consulted, so that only the 9:30 AM event happens between 9:30 and 9:45; the next two appear to occur between 9:45 and 10:00 AM, and the 9:50 AM event is not reported until the 10:00 AM to 10:15 wait. To get around this, we might schedule events only on the fifteen-minute mark when we want them to occur. Alternatively, we might try instead

```
"Uptempo"

The fast time rule is listed before the timed events rule in the turn sequence
rules.

The advance time rule is not listed in the turn sequence rules.

This is the fast time rule:
    increment the turn count;
    increase the time of day by 15 minutes.

When play begins: now the right hand status line is "[time of day]".

The Temporal Hot Spot is a room.

At 9:30 AM:
    say "Two turtles run by, almost too fast to see."

At 9:37 AM:
    say "A snail blitzes past."

At 9:42 AM:
    say "The grass grows."

At 9:50 AM:
    say "Several flowers burst open."

Test me with "z / z / z / z".
```

This time our revised time-advancing rule is listed just before the event scheduler, not just afterwards.

395

Example **Verbosity 2**

RB

Making rooms give full descriptions each time we enter, even if we have visited before, and disallowing player use of BRIEF and SUPERBRIEF.

Suppose that we want the player always to see full room descriptions, even if he tries to reset the defaults -- perhaps because there is vital information there which he will miss if he turns off full-length room descriptions.

To do this, we might want to remove the standard behavior of the three actions associated with BRIEF, SUPERBRIEF, and VERBOSE, replacing them with

explanatory messages about how the game behaves. We cannot use Instead to override these actions, because Instead rules do not apply to actions out of world. Instead, we will want to remove and replace the carry out rules.

We can do this easily by going to the Actions Index, looking up the detail panel for, say, "preferring abbreviated room descriptions", and click the "unlist" button to paste in the sentence that will remove this rule from the rulebook.

Let's remove all three of the carry out rules and substitute our own:

```
"Verbosity 2"

Section 1 - Procedure

The prefer unabbreviated room descriptions rule is not listed in the carry out
preferring unabbreviated room descriptions rulebook.

The prefer sometimes abbreviated room descriptions rule is not listed in the
carry out preferring sometimes abbreviated room descriptions rulebook.

The prefer abbreviated room descriptions rule is not listed in the carry out
preferring abbreviated room descriptions rulebook.

Carry out preferring unabbreviated room descriptions:
    say "[story title] always provides full-length descriptions for your reading
pleasure."

Carry out preferring sometimes abbreviated room descriptions:
    say "For your playing protection, [story title] provides only full-length room
descriptions."

Carry out preferring abbreviated room descriptions:
    try preferring sometimes abbreviated room descriptions instead.

The standard report preferring abbreviated room descriptions rule is not listed in
the report preferring abbreviated room descriptions rulebook.

The standard report preferring unabbreviated room descriptions rule is not listed
in the report preferring unabbreviated room descriptions rulebook.

The standard report preferring sometimes abbreviated room descriptions rule is
not listed in the report preferring sometimes abbreviated room descriptions
rulebook.

Use full-length room descriptions.

Section 2 - Scenario

The Wilkie Memorial Research Wing is a room. "The research wing was built
onto the science building in 1967, when the college's finances were good but its
aesthetic standards at a local minimum. A dull brown corridor recedes both north
and south; drab olive doors open onto the laboratories of individual faculty
```

members. The twitchy fluorescent lighting makes the whole thing flicker, as though it might wink out of existence at any moment.

The Men's Restroom is immediately west of this point."

The Men's Restroom is west of the Research Wing. "Well, yes, you really shouldn't be in here. But the nearest women's room is on the other side of the building, and at this hour you have the labs mostly to yourself. All the same, you try not to read any of the things scrawled over the urinals which might have been intended in confidence."

Test me with "west / east / brief / w / e / superbrief / w / e / verbose".

396

Example Slouching

RB

A system of postures allowing the player and other characters to sit, stand, or lie down explicitly or implicitly on a variety of enterable supporters or containers, or in location.

Suppose we want to let the player explicitly sit, stand, or lie down on different enterable objects. (Normally Inform treats all these actions as entering, but there may be cases where we want to distinguish between the player sitting on a chair and the player standing on it.)

Our implementation gives each kind of enterable a range of allowed postures, and one preferred posture. If the player enters the supporter or container without specifying a posture (as in ENTER CHAIR), he will be put in the preferred posture. If he explicitly says, e.g., STAND ON CHAIR, he will be put in the standing position as long as standing is a posture that suits the chair.

"Slouching"

Section 1 - Posture Rules

A posture is a kind of value. The postures are seated, standing, and reclining.

A person has a posture. The posture of a person is usually standing.

A supporter has a posture. A container has a posture.

Posture-permission relates various things to various postures. The verb to allow means the posture-permission relation.

Understand the commands "stand" and "sit" and "lie" as something new.

Understand "sit on/in [something]" as sitting on.
Understand "lie on/in [something]" as lying on.
Understand "stand on/in [something]" as standing up on.

```
Sitting on is an action applying to one thing.
Lying on is an action applying to one thing.
Standing up on is an action applying to one thing.

Carry out an actor sitting on:
    if the holder of the actor is not the noun, silently try the actor entering the
noun;
    if the holder of the actor is the noun:
        if the actor is not seated, try the actor taking position seated;
        otherwise follow the report taking position rules.

Carry out an actor lying on:
    if the holder of the actor is not the noun, silently try the actor entering the
noun;
    if the holder of the actor is the noun:
        if the actor is not reclining, try the actor taking position reclining;
        otherwise follow the report taking position rules.

Carry out an actor standing up on:
    if the holder of the actor is not the noun, silently try the actor entering the
noun;
    if the holder of the actor is the noun:
        if the actor is not standing, try the actor taking position standing;
        otherwise follow the report taking position rules.

Understand "lie down" as lying down.
Understand "sit down" or "sit" or "sit up" as sitting down.
Understand "stand" or "stand up" as standing up.

Lying down is an action applying to nothing.
Sitting down is an action applying to nothing.
Standing up is an action applying to nothing.

Taking position is an action applying to one posture.

Instead of an actor lying down:
    try the actor taking position reclining; rule succeeds.
Instead of an actor sitting down:
    try the actor taking position seated; rule succeeds.
Instead of an actor standing up:
    try the actor taking position standing; rule succeeds.

Check an actor taking position:
    if the holder of the actor is not a room and the holder of the actor does not
allow the posture understood:
        if the actor is the player:
            say "You can't take that position [in-on the holder of the actor].";
        otherwise if the actor is visible:
            say "[The actor] can't take that position.";
        stop the action.

Check an actor taking position:
    if the posture understood is the posture of the actor:
        if the actor is the player:
            say "You are already [the posture understood].";
```

```
        otherwise:
            if the actor is visible, say "[The actor] is already [the posture
understood].";
        stop the action.

Carry out an actor taking position:
    now the posture of the actor is the posture understood.

Report someone taking position (this is the position-report rule):
    say "[The actor] is now [the posture of the actor][if the holder of the actor is
not the location of the actor] [in-on the holder of the actor][end if]."

Report taking position:
    say "You are now [the posture of the player][if the holder of the player is not
the location] [in-on the holder of the player][end if]."

To say in-on (item - a thing):
    if the item is a container, say "in [the item]";
    otherwise say "on [the item]".

Carry out an actor exiting (this is the departure-posture rule):
    let N be the holder of the actor;
    if N is a container or N is a supporter,
        now the posture of the actor is the posture of N;
    otherwise now the posture of the actor is standing.

The departure-posture rule is listed after the standard exiting rule in the carry out
exiting rulebook.
The departure-posture rule is listed after the standard getting off rule in the carry
out getting off rulebook.

Carry out an actor entering something (this is the arrival-posture rule):
    if the noun is a container or the noun is a supporter,
        now the posture of the actor is the posture of the noun.

The arrival-posture rule is listed after the standard entering rule in the carry out
entering rulebook.

Check an actor going somewhere:
    if the actor is in a room and the actor is not standing:
        say "([if the actor is not the player][the actor] [end if]first standing up)
[command clarification break]";
        silently try the actor taking position standing;
        if the actor is not standing, stop the action.


Section 2 - Some Generic Kinds

A chair is a kind of supporter. A chair is always enterable. Every chair allows
seated and standing. A chair is usually seated.
A sofa is a kind of supporter. A sofa is always enterable. Every sofa allows
seated, standing and reclining. A sofa is usually seated.
A hammock is a kind of container. A hammock is always enterable. Every
hammock allows seated and reclining. A hammock is usually reclining.


Section 3 - The Scenario
```

```
The Resort is a room.

The banana hammock is a hammock in the Resort. The stone bench is a sofa in
the resort.

Clark is a man in the Resort. A persuasion rule: persuasion succeeds.

Rule for writing a paragraph about someone (called target):
    say "[The target] is [posture] [if the holder of the target is the
location]nearby[otherwise][in-on the holder of the target][end if]."

Rule for writing a paragraph about something which encloses an unmentioned
person (called target):
    carry out the writing a paragraph about activity with the target instead.

Test me with "sit on bench / stand on bench / get up / lie on hammock / sit up / g
/ clark, sit on bench / look / clark, lie down / g / look / clark, get up / look / clark,
lie down / look / enter bench".
```

397

Example **Swigmore U.**

RB

Adding a new kind of supporter called a perch, where everything dropped lands on the floor.

Inform's default assumption is that if a player on an enterable object drops something, the dropped article winds up beside him on the same supporter or in the same container. This makes lots of sense for a dais, say, or a king-sized bed. It's a little less sensible if the enterable supporter in question is a bar stool or the like. So suppose we want to add a new kind of supporter called a perch, where everything dropped lands on the floor.

There are actually several ways of implementing this, but one of them is to reach right into the drop action and replace the "standard dropping rule" with a different one of our own invention -- like this:

```
"Swigmore U."

Moe's Tavern is a room. The bar is an enterable supporter in Moe's. A drink is a
kind of thing. On the bar is a drink called a flaming Homer.

A perch is a kind of supporter. A perch is always enterable. The stool is a perch
in Moe's.

The player carries a dead field mouse and a tomacco fruit.

The sophisticated dropping rule is listed instead of the standard dropping rule in
the carry out dropping rulebook.

This is the sophisticated dropping rule:
    if the player is on a perch (called the awkward position):
        let place be the holder of the awkward position;
```

```
        move the noun to the place;
    otherwise:
        move the noun to the holder of the player.

Test me with "sit on stool / drop mouse / look / get up / look".
```

Now the carry-out behavior of the dropping action has been changed, but we haven't had to interfere in the checks or reporting at all. The rest of the action works just as it always did.

Of course, maybe we do want to change the way the action is reported, to make it clearer to the player where the dropped article wound up:

```
The sophisticated report dropping rule is listed instead of the standard report
dropping rule in the report dropping rulebook.

This is the sophisticated report dropping rule:
    say "You drop [the noun] on [if the holder of the noun is a room]the
ground[otherwise][the holder of the noun][end if]."
```

398

Example Access All Areas

RB

The Pointy Hat of Liminal Transgression allows its wearer to walk clean through closed doors.

If somebody tries to walk through a closed door, the "can't go through closed doors rule" usually stops them. This is a rule belonging to the "check going" rulebook. These names are fairly explanatory when written out, but hard to remember: fortunately we don't need to remember them, as the Index panel contains a full inventory of the check, carry out and report rules for every action, showing all of their names and the order in which they are checked. (We can also find out which rules are stopping an action by typing the testing command ACTIONS.)

Here we make the rule do nothing provided a condition holds:

```
"Access All Areas"

The extremely difficult door is north of the Standing Room and south of the
Room of Walking Upside Down. It is a locked door.

The player is carrying the Pointy Hat of Liminal Transgression. The hat is
wearable.

The can't go through closed doors rule does nothing when the Hat is worn.

Test me with "n / wear hat / n".
```

(The Pointy Hat may be useful in debugging a game, even if it never makes it into the final published work.)

399

Example **Backus-Naur form for rules**

RB

The full grammar Inform uses to parse rule definitions, in a standard computer-science notation.

Backus-Naur form, or BNF, is a standard notation used by computer scientists to specify more or less precisely what the valid programs are for a given programming language. It tends to provide a good description for a language such as C or Pascal, where contextual rules are limited, but the authors of Inform are doubtful that it is such a good tool for a natural-language system. For those who are interested, though, the following gives a formal specification for Inform's rules.

```
<rule> ::=
    Definition : A/an <kind> is <new adjectival name> if/unless <definition>
    | <preamble> : <phrases>
    | <preamble> , <phrase> (* only allowed for a few cases: see below)

<definition> ::=
    <condition>
    | its/his/her/their <value property name> is/are <value> or less/more
    | : <phrases>

<preamble> ::=
    To <phrase template>
    | To decide if/whether <phrase template>
    | To decide which/what <kind of value> is <phrase template>
    | This is the <rule name>
    | [[A] Rule for] <circumstances> [(this is the <rule name>)]

<circumstances> ::=
    At <time>
    | When <event name>
    | [<placement>] <rulebook reference> [while/when <condition>] [during
<scene name>]

<rulebook reference> ::=
    <rulebook name> [about/for/of/on/rule] [<action pattern>]
    | <object-based-rulebook name> [about/for/of/on/rule] [<description>]

<placement> ::=
    a/an
    | [the] first
    | [the] last

<phrases> ::=
    <phrase>
    | <phrases> ; <phrase>
```

The following examples show how Inform breaks down some typical rules using the system above:

```
<rule> = At 2:09 PM: increase the score by 2; say "Progress!"
    <preamble> = At 2:09 PM
```

```
        <circumstances> = At 2:09 PM
          At
          <time> = 2:09 PM
    :
    <phrases> = increase the score by 2; say "Progress!"
        <phrase> = increase the score by 2
        ;
        <phrase> = say "Progress"

<rule> = Instead of eating the ostrich during Formal Dinner (this is the cuisine
rule), say "It's greasy!"
    <preamble> = Instead of eating the ostrich during Formal Dinner (this is the
cuisine rule)
        <circumstances> = Instead of eating the ostrich during Formal Dinner
          <rulebook reference> = Instead of eating the ostrich
            <rulebook name> = Instead
            of
            <action pattern> = eating the ostrich
          during
          <scene name> = Formal Dinner
        (
        this
        is
        the
        <rule name> = cuisine rule
        )
    ,
    <phrases> = say "It's greasy!"
        <phrase> = say "It's greasy!"

<rule> = After printing the name of a container: say "!"
    <preamble> = After printing the name of a container
        <circumstances> = After printing the name of a container
          <rulebook reference> = After printing the name of a container
            <object-based-rulebook name> = After printing the name
            of
            <description> = a container
    :
    <phrases> = say "!"
        <phrase> = say "!"
```

(*) The colon dividing a rule preamble from its definition can be replaced by a comma only if the preamble begins with the words "Instead of", "Before", "After", "Every turn" or "When", and if the definition consists only of a single phrase.

400

Example **Solitude**

RB

Novice mode that prefaces every prompt with a list of possible commands the player could try, and highlights every important word used, to alert players to interactive items in the scenery.

Observation of novice IF players suggests that they often have a hard time figuring out how to get started, especially if they are encountering the game in a context

where they don't have time to settle in and read instructions. Here we provide some training wheels to help them learn to communicate.

This is divided into several parts. The first part is the system of rules for general guidance, which could be excerpted and used anywhere. The second part is a scenario using these rules.

```
"Solitude"

Part 1 - General Rules

When play begins:
    say "Have you played interactive fiction before? >";
    now novice mode is whether or not the player consents.
```

The rationale for asking the question this way, and not another, is that novices asked whether they would like instructions very often say no, even if they need them.

```
Novice mode is a truth state that varies. Novice mode is true.

Stopping novice mode is an action out of world.
Starting novice mode is an action out of world.

Understand "novice mode off" or "novice off" as stopping novice mode.
Understand "novice mode on" or "novice on" as starting novice mode.

Carry out stopping novice mode: now novice mode is false.
Carry out starting novice mode: now novice mode is true.

Report stopping novice mode: say "Novice mode is now off."
Report starting novice mode: say "Novice mode is now on."

Before reading a command when novice mode is true:
    say "[line break]Some options to try:[line break]";
    follow the novice suggestion rules.

The novice suggestion rules is a rulebook.

A novice suggestion rule (this is the suggestion that he look rule):
    if not looking and not going, say " [bold type]look[roman type]".

A novice suggestion rule (this is the suggestion that he check inventory rule):
    if the player carries something and we are not taking inventory, say " [bold
type]inventory[roman type] (I)".

A novice suggestion rule (this is the suggestion that he put things on rule):
    if the player carries something and a free-standing supporter is relevant, say "
[bold type]put[roman type] something [bold type]on[roman type] [the list of
relevant supporters]".

A novice suggestion rule (this is the suggestion that he take things rule):
    if a gettable thing is relevant, say " [bold type]take[roman type] [the list of
gettable relevant things]".
```

```
A novice suggestion rule (this is the suggestion that he examine things rule):
    if an unexamined thing is relevant, say " [bold type]examine[roman type] (X)
[the list of unexamined relevant things]".

A novice suggestion rule (this is the suggestion that he enter things rule):
    if a relevant thing is worth entering, say " [bold type]enter[roman type] [the list
of worth entering relevant things], or [bold type]get out[roman type]".

A novice suggestion rule (this is the suggestion that he open things rule):
    if an unlocked openable thing is relevant, say " [bold type]open[roman type] or
[bold type]close[roman type] [the list of unlocked openable relevant things]".

A novice suggestion rule (this is the suggestion that he lock things rule):
    if a closed lockable thing is relevant, say " [bold type]lock[roman type] or [bold
type]unlock[roman type] [the list of closed lockable relevant things]".

A novice suggestion rule (this is the suggestion that he eat things rule):
    if the player carries an edible relevant thing, say " [bold type]eat[roman type]
[the list of edible relevant things carried by the player]".

A novice suggestion rule (this is the suggestion that he wear things rule):
    if the player carries a wearable relevant thing, say " [bold type]wear[roman
type] [the list of wearable relevant things carried by the player]".

A novice suggestion rule (this is the suggestion that he turn things on rule):
    if a device is relevant, say " [bold type]turn on[roman type] or [bold type]turn
off[roman type] [the list of relevant devices]".

A novice suggestion rule (this is the suggestion that he go places rule):
    if a room is adjacent, say " [bold type]go[roman type][exit list][if in darkness] or
try other directions in the dark[otherwise]".

A novice suggestion rule (this is the suggestion that he enter doors rule):
    if an open door is relevant, say " [bold type]go through[roman type] [the list of
relevant open doors]".

A novice suggestion rule (this is the suggestion that he interact with people rule):
    if another person is relevant, say " [bold type]kiss[roman type] or [bold
type]wake[roman type] [the list of relevant other people][if the player carries
something], or [bold type]give[roman type] things [bold type]to[roman type]
someone[end if]".

A novice suggestion rule (this is the suggestion that he ask for help rule):
    say " [bold type]help[roman type] to see a more complete set of instructions".

A novice suggestion rule (this is the suggestion that he turn off help rule):
    say " [bold type]novice mode off[roman type] to turn off this guidance".

Last novice suggestion rule:
    say "[line break]".
```

The suggestion about asking for help is no good unless we provide some. This might take any of a number of forms, but for the sake of example we'll use an easy way out:

Include Basic Screen Effects by Emily Short. Include Menus by Emily Short. Include Basic Help Menu by Emily Short.

After taking inventory when novice mode is true: say "To get rid of any of these objects, [bold type]drop[roman type] it."

A thing can be examined or unexamined. Carry out examining something: now the noun is examined.

A thing can be seen or unseen. A thing is usually unseen.

Definition: a thing is relevant if it is seen and it is visible. Before printing the name of something (called the target): now the target is seen; if novice mode is true, say "[bold type]". After printing the name of something: say "[roman type]".

Definition: a supporter is worth entering:
    if the player carries it, no;
    if it is enterable, yes.

Definition: a container is worth entering:
    if the player carries it, no;
    if it is enterable and it is open, yes.

Definition: a person is other if it is not the player. Definition: a person is another if it is other.

Definition: a thing is free-standing if it is in a room.

To say exit list:
    let place be location;
    let count be 0;
    repeat with way running through directions:
        let place be the room way from the location;
        if place is a room:
            increment count;
            say "[if count is greater than 1] or[end if] [bold type][way][roman type]".

Definition: a thing is gettable:
    if it is scenery, no;
    if it is fixed in place, no;
    if it is a person, no;
    if the player is carrying it, no;
    if the player is wearing it, no;
    yes.

Part 2 - On the Ground

Antarctic Research Station is a room. "Though not always the most stimulating of environments, the station is far from your ex-wife and most of the things in the world that annoy you, namely the other 6+ billion people. There is a second room to the south." The station contains a radio. The radio is a device. It is fixed in place.

South of the Station is Sitting Room. The description of the Sitting Room is "Just big enough for a very [comfortable chair]." The Sitting Room contains an

enterable supporter called a comfortable chair. The chair is scenery. A monograph about penguins is in the Sitting Room.

Blistering Cold is a room. "It is white out here and very very very cold." The white door is a door. "[The white door] leads to [the other side of the white door]." It is west of the Blistering Cold and east of the Antarctic Research Station.

Test me with "i / x radio / x door / s / i / x chair / x monograph / sit in chair / get up / n / open door / enter door".

401

 **Example** **In Fire or in Flood**

RB

A BURN command; flammable objects which light other items in their vicinity and can burn for different periods of time; the possibility of having parts or contents of a flaming item which survive being burnt.

"In Fire or in Flood"

Part I - Rules for combustion

Heat is a kind of value. The heats are whole, damp, and flaming. A thing has a heat. A thing is usually whole.

A thing has a number called endurance. The endurance of a thing is usually 5. A thing has a number called turns of burning. A thing can be flammable or flame-retardant.

Before printing the name of something flaming:
    say "flaming ".

Before burning something when the player is not carrying something flaming:
    if a flaming portable thing (called the lighter) is touchable:
        say "(with [the lighter], which you first take)[command clarification break]";
        try taking the lighter.

Instead of burning something when the player is not carrying something flaming:
    say "You would first need a fire source."

Instead of burning something flame-retardant:
    say "[The noun] is not the sort of thing that catches fire."

Instead of burning something flammable when the player is carrying something flaming (called the flame source):
    say "You light [the noun] with [the flame source].";
    now the heat of the noun is flaming.

Instead of burning something when the player is in the noun:
    say "That seems dangerous given that you yourself are in [the noun]."

Instead of burning something when the player is on the noun:
    say "That seems dangerous given that you yourself are on [the noun]."

```
Instead of examining something:
    say "Hm, the [printed name] appears to be [heat]."

Before taking a flaming thing:
    let turns remaining be the endurance of the noun minus the turns of burning of
the noun;
    if turns remaining is less than two, say "There's no portion of [the noun]
sufficiently cool for you to pick up." instead.
```

But that's only a small part of the battle. The thing about fire is that it keeps on doing fiery things even when the player is otherwise occupied: destroying items that are on fire, and spreading to other things nearby. So we need a set of rules for the fire's behavior.

```
Every turn when something is flaming:
    follow the fire rules.

The fire rules is a rulebook.

A fire rule (this is the can't hold flaming objects rule):
    repeat with item running through flaming things:
        if the item is held by the player:
            let turns remaining be the endurance of the item minus the turns of
burning of the item;
            if turns remaining is less than two:
                say "[The item] becomes too hot to hold! ";
                try dropping the item;
                if the item is held by the player, say "This is certainly painful."

A fire rule (this is the flames spread rule):
    repeat with item running through flaming things:
        if the turns of burning of the item is one:
            spread the flames from the item.

A fire rule (this is the fire destroys things rule):
    now started printing is false;
    repeat with item running through flaming things:
        increment the turns of burning of the item;
        if the turns of burning of the item is greater than the endurance of the item,
destroy the item;
    if started printing is true, say "[paragraph break]";
    now started printing is false.
```

Because we've labelled all the fire rules, we could swap their order, or turn some of them off, while allowing the others run as usual. For instance, if there were a pair of fireproof gloves in the game, we might want to turn off the "can't hold flaming objects rule" whenever the player is wearing them.

This sort of flexibility is especially useful in the context of extensions. Someone writing an extension about burning would have no way of anticipating the need for a Fireproof Gauntlet of Thog, but the author would nonetheless be able to implement one easily.

```
Definition: a thing is vulnerable if it is flammable and it is whole.
```

The contact between things is a critical factor when it comes to fire, so we might add a couple of conditional relations do determine what is touching what.

```
Reliance relates a thing (called X) to a thing (called Y) when X is part of Y or X is in Y or X is on Y. The verb to be relying on means the reliance relation.

Contact relates a thing (called X) to a thing (called Y) when X is relying on Y or Y is relying on X. The verb to be joined to means the contact relation.
```

Having these at our disposal makes it much tidier to write what happens next:

```
To spread the flames from (item - a thing):
    now started printing is false;
    if the item is joined to a flammable whole thing (called the sacrifice):
        if the sacrifice is visible:
            now started printing is true;
            say "Flames engulf [the list of flammable whole things which are joined to the item].";
        now all the flammable whole things joined to the item are flaming.

Started printing is a truth state that varies. Started printing is false.

To destroy (item - a thing):
    let home be the holder of the item;
    if the item is part of something (called the superstructure), let home be the holder of the superstructure;
    if the item is visible:
        now started printing is true;
        say "[The item] burns away[if something is relying on the item], leaving [a list of things which are relying on the item] behind[end if]. ";
    if something is relying on the item,
        now all the things which are relying on the item are in the home;
    now the item is nowhere;
    now the item is damp;
    now every flaming thing which is part of the item is damp.

To destroy (item - a door):
    let home be the holder of the item;
    if item is visible:
        now started printing is true;
        say "[The item] burns away[if something flame-retardant is part of the item], leaving [a list of flame-retardant parts of the item] behind[end if]. ";
    if home is a room, now all of the flame-retardant parts of the item are in the home;
    now the item is damp;
    now the item is open;
    now the item is unopenable.

Before printing the name of a damp door:
    say "burnt-out frame of ".

Instead of opening or closing a damp door:
    say "[The noun] can no longer be opened or closed in any meaningful sense."
```

```
Instead of doing something other than examining or dropping to a flaming thing
when the turns of burning of the noun is greater than 1:
    say "Fire has too thoroughly engulfed [the noun] for that to be a good idea."

Instead of taking something when the noun is in a flaming thing (called the
receptacle):
    say "You don't quite dare reach into [the receptacle]."

Instead of touching something which is within a flaming thing (called the
receptacle):
    say "It seems a little risky since [the receptacle] is on fire."

Instead of turning something when the noun is contained in a flaming thing
(called the receptacle):
    say "It seems a little risky since [the receptacle] is on fire."

Instead of pushing or pulling something when the noun is inside a flaming thing
(called the receptacle):
    say "[The receptacle] deters you."

Before burning something which is in a container when the holder of the noun
contains the player:
    say "This could make things toasty for you..."
```

And that completes the rules which cover burning: things can catch fire, fire will spread, and gradually consume the world in flames. All of that was general and could be used in any setting, but we now provide a small game to show it off.

```
Part II - Escape from the Library of the Dead

The Library of the Dead is a room. "This room -- little, dank, stone -- is filling with
some miasma you do not quite dare breathe. It is imperative that you get out."

The desk is a flammable supporter in the Library. A drawer is part of the desk.
The drawer is a flammable closed container. It is openable, lockable, and locked.
The desk is scenery.

A box is in the Library. A metal hinge is part of the box. The hinge is flame-
retardant. The box is open, flammable, and openable. The shroud of Laertes is a
flammable thing in the box.

Instead of examining something when something is part of the noun:
    say "You note [the list of things which are part of the noun]."

The world's last manuscript of the Psychagogoi by Aeschylus is on the desk.
The manuscript is flammable. The manuscript has endurance 1.

The torch is a flammable flaming thing carried by the player. It has endurance
60. The asbestos sack is a flame-retardant player's holdall in the drawer.

The trapdoor is up of the Library and east of the Plaza. The trapdoor is a door. It
is flammable, closed, lockable, and locked. "A trapdoor in the ceiling is your only
hope of escape[if flaming]. Fortunately, it is rapidly burning through[end if]." The
trapdoor has endurance 15.
```

```
Instead of going through the closed trapdoor, say "[The trapdoor] is closed."
```

We can then add a special fire rule to handle the trapdoor, which will be called as part of the same sequence. Again, this would be most important if the fire rules were part of a standard extension, and the trapdoor fire rule the author's own addition.

```
A fire rule:
    if the trapdoor is flaming and a random chance of 1 in 3 succeeds:
        let the caught thing be a random flammable whole thing which can be
touched by the trapdoor;
        if the caught thing is a thing:
            say "A spark from [the trapdoor] catches [the caught thing]!";
            now the caught thing is flaming.

Instead of going to the Plaza:
    say "Out at last!";
    end the story finally.

Test me with "get manuscript / get shroud / light desk / look / g / open drawer /
look / g / g / g / get sack / put shroud in sack / put manuscript in sack / close
sack / light trapdoor / look / g / g / g / g / g / g / g / g / g / g / g / g / g / up".
```

402

Example **Patient Zero**

RB

People who wander around the map performing various errands, and in the process spread a disease which only the player can eradicate.

```
"Patient Zero"

Use the serial comma and no scoring.

Understand "about" as asking for information. Asking for information is an action
out of world.

Carry out asking for information: say "An implementation of the following creative
brief:

People wander around some small map, on errands. One, sad to tell, has
Gelato's Syndrome, a tragic condition turning one's skin the colour of a random
flavour of ice cream (raspberry ripple, neapolitan, etc.). When two people are in
the same room, there's a 1/3 chance that an infected person will infect a non-
infected one. The player can cure any single person: victory condition - to stamp
out the disease."

When play begins:
    say "Gelato's Syndrome. It's struck, and it's struck hard. In these sticky
summer months, there's no telling who will contract the disease next.";
    now the command prompt is "[if the destination of the player is not blank]
(heading to [destination of the player]) [end if]>".

Section 1 - Errands
```

```
The current actor is a person which varies. The current owner is a person which
varies.

Every turn:
    if player is active, follow the character movement rules.

Every turn:
    now the last person named is the player;
    now the last thing named is the player;
    now every person is active.

A person can be active or passive. The player is passive.

The character movement rules are a rulebook.

The first character movement rule:
    now group size is 1;
    now the last person named is the player;
    now the last thing named is the player;
    now the player is passive.

A character movement rule:
    repeat with mover running through innocent people:
        now the current actor is the mover;
        follow the shopper rules;
        now the current actor is passive;
    follow the movement reporting rule.

A character movement rule:
    repeat with next mover running through mercantile people:
        now the current owner is the next mover;
        follow the shopowner rules;
        now the current owner is passive;
    follow the infection rule.

To decide whether movement has not yet occurred:
    if the player is passive, no;
    yes.

Definition: a person is mercantile if it owns a room. Definition: a person is
innocent if it is not mercantile and it is not the player.

The shopowner rules is a rulebook.

A shopowner rule:
    let the shop be a random room owned by the current owner;
    if the shop is air-conditioned and an open door (called the escape) protects
the shop, try the current owner closing the escape instead.

Report someone closing a door when the person asked owns the location:
    say "[The person asked], muttering darkly about air-conditioning and
electricity, closes [the noun]." instead.

Report Vanessa closing the metal door when the metal door is visible:
    if Vanessa is visible, say "Vanessa watches serenely as the metal door slides
```

```
automatically back in place, sealing Cold Comfort." instead;
    otherwise say "The metal door slides heavily back into place." instead.

A shopowner rule:
    if the location of the current owner encloses a submitted artwork (called the
target):
        try the current owner filing the target.

Filing is an action applying to one thing.

Before someone filing something which is not carried by the person asked:
    try the person asked taking the noun instead.

Carry out someone filing:
    if the person asked does not carry the noun and the person asked is visible,
say "[The person asked] tries unsuccessfully to get [the noun]." instead;
    now the noun is nowhere.

Report someone filing:
    say "[The person asked] registers [the noun] and files it away."

The shopper rules is a rulebook.

A shopper rule:
    if the current actor carries something (called the problem), try the current actor
resolving the problem instead.

A shopper rule:
    if the current actor is not in the pool hall and the air conditioner is switched on:
        try the current actor approaching the pool hall;
    otherwise:
        let way be a random direction;
        try the current actor going the way.

Definition: a room is air-conditioned:
    if it is outdoors, no;
    if it is the Pool Hall and the air conditioner is switched off, no;
    if it is protected by a door, yes;
    no.

Protection relates a door (called X) to a room (called Y) when the front side of X
is Y or the back side of X is Y. The verb to protect means the protection relation.

Ownership relates one person to various rooms. The verb to own means the
ownership relation.

Resolving is an action applying to one thing.

An artwork is a kind of thing. Before printing the name of an artwork, say italic
type. After printing the name of an artwork, say roman type. An artwork can be
submitted or reserved.

A book is a kind of artwork.
```

```
Before someone resolving a book when the person asked is not in the Public
Library:
    try the person asked approaching the Public Library instead.

Carry out someone resolving a book:
    move the noun to the Public Library;
    now the noun is submitted.

Report someone resolving a book:
    say "[The person asked] turns in [the noun]."

Before listing contents: group books together.

Before grouping together books: say "books entitled ".

A stamped envelope is a kind of thing.

Before someone resolving a stamped envelope when the person asked is not in
the Post Office:
    try the person asked approaching the Post Office instead.

Carry out someone resolving a stamped envelope:
    now the noun is nowhere.

Report someone resolving a stamped envelope:
    say "[The person asked] slips [a noun] into the outgoing mail slot."

Instead of someone resolving a stamped envelope when the person asked
carries at least two stamped envelopes:
    if the person asked is visible, say "[The person asked] shoves into the mail
slot [a list of stamped envelopes carried by the person asked].";
    repeat with item running through stamped envelopes carried by the person
asked:
        now the item is nowhere.

A DVD is a kind of artwork.

Before someone resolving a DVD when the person asked is not in the Rental
Store:
    try the person asked approaching the Rental Store instead.

Carry out someone resolving a DVD:
    now the noun is submitted;
    move the noun to the Movie Rental Store.

Report someone resolving a DVD:
    say "[The person asked] returns [the noun]."

Instead of someone resolving a DVD when the person asked carries at least two
DVDs:
    if the person asked is visible, say "[The person asked] turns in [a list of DVDs
carried by the person asked].";
    now every DVD carried by the person asked is submitted;
```

now every DVD carried by the person asked is in the location of the person asked.

Before listing contents: group DVDs together.

Before grouping together DVDs: say "DVDs of ".

Approaching is an action applying to one thing.

Carry out someone approaching:
    let the way be the best route from the location of the person asked to the noun, using doors;
    if the way is a direction, try the person asked going the way;
    otherwise stop the action.

A coupon is a kind of thing.

Carry out someone resolving a coupon:
    try the person asked giving the noun to Vanessa.

The block giving rule is not listed in any rulebook.

Check giving something to someone (this is the block player giving rule):
    abide by the block giving rule.

Before someone resolving a coupon when the person asked is not in Cold Comfort:
    try the person asked approaching Cold Comfort instead.

After someone giving a coupon to Vanessa:
    let the reward be a random ice cream cone;
    let the new flavor be a random infection color;
    now the infection color of the reward is the new flavor;
    move the reward to the person asked;
    now the noun is nowhere;
    if Vanessa is visible, say "[The person asked] trades in [the noun] and receives [a reward] from Vanessa."

Infection color is a kind of value. The infection colors are french vanilla, whole-bean vanilla, mint, chocolate, dark chocolate, chocolate chip, chocolate fudge, mint chocolate chip, chocolate chocolate chip, triple chocolate, white chocolate, white chocolate chip, aztec cocoa-chili, raspberry ripple, neapolitan, rum raisin, dulce de leche, strawberry chunk, rocky road, blackberry sorbet, lemon sherbet, lime ice, caramel swirl, mango, saffron silk, and cookie dough cream.

To say list of flavors:
    let current color be french vanilla;
    while current color is not cookie dough cream:
        say "[current color], ";
        now current color is the infection color after the current color;
    say "and [current color]".

Understand "ask vanessa for [flavored ice cream]" as buying the flavor.
Understand "buy [flavored ice cream]" as buying the flavor.

Buying the flavor is an action applying to one infection color.

Check buying the flavor:
    unless the player can see Vanessa:
        say "It would help if you were in the presence of an ice cream salesperson." instead.

Carry out buying the flavor: say "'Do you have a coupon?' Vanessa demands. You admit you do not. 'No [infection color understood] for you!'"

Understand "ice cream" or "cream" or "ice" or "sherbet" or "sorbet" as "[ice cream]".

Understand "[infection color]" or "[infection color] [ice cream]" as "[flavored ice cream]".

An ice cream cone is a kind of thing. An ice cream cone is always edible. An ice cream cone has an infection color. An ice cream cone can be half-eaten or fresh. Understand the infection color property as referring to an ice cream cone.

Carry out someone resolving an ice cream cone:
    try the person asked eating the noun instead.

Instead of someone eating a fresh ice cream cone:
    now the noun is half-eaten;
    if the person asked is visible, say "[The person asked] licks [the noun]."

Report someone eating an ice cream cone:
    say "[The person asked] pops the end of [the noun] into [if the person asked is female]her[otherwise]his[end if] mouth and swallows." instead.

Before printing the name of an ice cream cone:
    say "[if half-eaten]half-eaten [end if][infection color] ".

Section 2 - Infection Rules

This is the infection rule:
    if an infected person (called typhoid mary) can see a clean person (called random bystander) and a random chance of 1 in 3 succeeds:
        try typhoid mary sneezing on the random bystander.

A person can be infected or clean. A person has an infection color.

Every turn:
    if the player is infected, say "You feel itchy."

Definition: a person is other if it is not the player. Definition: a person is another if it is other.

When play begins: now right hand status line is "Sick: [number of infected people]/[number of people]".

Every turn:
    if every person is infected, end the story saying "Everyone succumbs";

if every person is clean, end the story finally saying "The Syndrome is eradicated".

Understand "sneeze on [something]" as sneezing on. Sneezing on is an action applying to one thing.

Check sneezing on:
if the player is clean, say "You're not sickly." instead;
if the noun is the player, say "Ew." instead;
if the noun is not a person, say "[The noun] cannot be infected." instead.

Carry out sneezing on:
now the noun is infected;
now the infection color of the noun is a random infection color.

Carry out someone sneezing on:
now the noun is infected;
now the infection color of the noun is a random infection color.

Report sneezing on:
say "Unable to control yourself, you sneeze on [noun].".

Report someone sneezing on:
say "[The person asked] sneezes on [if the noun is the player]you[otherwise][noun][end if]!".

Understand "inject [someone] with [something]" as injecting it with. Understand "inject [someone] with [syringe]" as injecting it with. Understand "use [syringe] on [someone]" as injecting it with. Understand the commands "innoculate" and "vaccinate" as "inject".

Injecting it with is an action applying to two things.

Check injecting it with:
if the second noun is not the syringe, say "[The second noun] cannot inject anything." instead;
if the noun is clean:
if the noun is the player, say "You're not infected yet." instead;
say "[The noun] is not infected, and the syringe contains a cure, not a vaccine." instead.

Carry out injecting it with: now the noun is clean.

After injecting the player with something: say "You inject yourself, wincing at the sting. But the itching fades almost at once."

Report injecting it with: say "You inject [the noun], who is now cured (but could easily be reinfected)."

Section 3 - Geography

Include Locksmith by Emily Short.

```
Understand "go to/toward/into [any room]" as going toward. Understand "enter
[any room]" as going toward.

A person has a room called the destination.

Going toward is an action applying to one thing.

Check going toward:
    if the noun is the location, say "You're already in [the location]." instead.

Carry out going toward:
    now the destination of the player is the noun;
    let heading be the best route from the location to the noun, using even locked
doors;
    if heading is not a direction, say "You can't think how to get there from here."
instead;
    try going heading;
    if the location is the destination of the player, now the destination of the player
is blank.

Instead of waiting when the destination of the player is not blank:
    if the destination of the player is the location:
        now the destination of the player is blank;
    otherwise:
        try going toward destination of the player;
        if the location is the destination of the player, now the destination of the
player is blank.

Understand "stop" or "cease" as stopping. Stopping is an action applying to
nothing. Carry out stopping: now the destination of the player is blank. Report
stopping: say "You stop in your tracks."

After going to an air-conditioned room:
    say "You step into the mercifully air-conditioned surroundings of...";
    continue the action.

After going from an air-conditioned room:
    say "You emerge from the air-conditioning into heat like a wall...";
    continue the action.

Instead of listening to an air-conditioned room:
    say "The air-conditioning hums softly."

The Alfred Cralle Pool Hall is a room. "The town's most popular gathering-place,
the pool hall is decorated in honor of the inventor of the ice cream scoop." The
air conditioner is a device in the Pool Hall. "[if switched off]An air conditioner sits
in the corner, unhappily inert[otherwise]The air conditioner hums briskly[end if]."

The felt door is west of the Pool Hall. The felt door is a door. The felt door is
open. The felt door is lockable and unlocked. The key to the city unlocks the felt
door. The description of the felt door is "It has a prominent lock, designed for an
old-fashioned key."

After locking a door with something in the presence of an other person (called
audience):
```

say "[The audience] looks a little non-plussed when you lock [the noun], but shrugs."

Nancy Johnson Memorial Square is west of the felt door. The description of Nancy Johnson Memorial Square is "Waves of August heat rise from the pavement: more than once you've had the fancy that your shoes are simply going to stick. At the center of the square, rubbed to a brownish polish by many adoring hands, is the statue of Mrs. Nancy Johnson of New Jersey."

The statue is scenery in Memorial Square. Understand "nancy" or "johnson" or "mrs" as the statue. The description of the statue is "Mrs. Johnson is pictured with a hand-cranked ice cream freezer tucked under one arm. Her other hand grips an ice cream scoop, ready to serve frozen dessert to the huddled masses." A hand-cranked ice cream freezer is part of the statue. The description is "The hand-cranked ice cream freezer was Mrs. Johnson's invention in 1846, though it was William Young who had the sense to patent the thing in 1848." The scoop is part of the statue. The description of the scoop is "An anachronism: Alfred Cralle would not invent the tool until 1897."

The Post Office is northwest of Nancy Johnson Memorial Square. "Service at the post office is on the slow side since everything went automated." The slot is scenery in the post office. The slot is a container. Carry out inserting something into the slot: now the noun is nowhere. Report inserting something into the slot: say "[The noun] falls out of sight, and you know you will never see it again."

Hamwi Street is northeast of an iron gate. "A U-shaped street running from Main Street around to the Memorial Square, Hamwi Street was recently added by ambitious city planners. The small and straggly line of trees has yet to grow enough to provide perceptible shade, so the street is even hotter and more unforgiving than the other parts of town."

The iron gate is northeast of Nancy Johnson Memorial Square. The iron gate is a door. It is lockable and unlocked.

Before printing the name of the iron gate while not opening or closing or locking or unlocking:
    if the person asked is the player:
        if the gate is open, say "open ";
    otherwise if the gate is locked:
        say "locked ";
    otherwise if the gate is closed:
        say "closed ".

Cold Comfort Ice Cream is north of a metal door. The metal door is north of Hamwi Street. A poster is fixed in place in Cold Comfort. "A poster fills one wall with the blazing promise of treats to come." The description of the poster is "Coming soon! Thai ice creams! Durian, jackfruit, taro, and coconut flavors!"

The metal door is a door. "A frosty metallic door separates [the location] from [the other side of the metal door]." The metal door is lockable and unlocked. The key to the city unlocks the metal door.

Marciony Street is southeast of Nancy Johnson Memorial Square. "A semi-circular terrace, named somewhat fancifully after one claimant to the invention of the ice cream cone -- though Hamwi Street competes for the same honor. There

are wedges of cool shadow here and there thanks to the buildings, but for the most part the southern exposure keeps Marciony unpleasantly hot."

The Movie Rental Store is west of a glass door. The glass door is a door. It is west of Marciony Street. The glass door is lockable and unlocked. The key to the city unlocks the glass door.

Main Street is southeast of Hamwi Street. Main Street is northeast of some bronze gates.

The emergency box is in Main Street. The emergency box is fixed in place. "A fire-red box with a glass front faces the sidewalk, with 'In case of emergency, BREAK GLASS' lettered on it." The emergency box is closed and transparent. Understand "glass" as the box. Instead of attacking the closed emergency box: say "You hit the emergency box, which shatters open."; now the emergency box is open. Instead of attacking the open emergency box: say "The glass has already been thoroughly broken."

The syringe is in the emergency box. The description of the syringe is "It contains the cure for Gelato's Syndrome. You can inject anyone you like with it."

The bronze gates are northeast of Marciony Street. The bronze gates are a door. The bronze gates are lockable and unlocked. The description of the bronze gates is "Erected during the milk-taint revolution of 1937, they were designed to keep Main Street safe from the depredations of dairy-starved rioters."

The Public Library is east of Main Street. "Built in the 1920s during the height of the dairy boom, the public library has lush pink velvet seats, marble walls the color of fresh cream, and a motif of cherries carved around every doorframe. An incongruous sign hangs from the ceiling." The incongruous sign is scenery in the Public Library. The description of the incongruous sign is "Eating and drinking in the library is STRICTLY PROHIBITED."

Town Hall is southeast of Main Street. "Town Hall was built during the slow days of the ice-cream bust, and therefore it is as joyless and utilitarian as the Public Library is ridiculous. Unwilling to be reminded of their pain, the inhabitants steered clear of any decoration that might remotely be construed to resemble a scoop of anything: so there are no curves, only disciplined right angles." The key to the city is in Town Hall. It unlocks the iron gate. It unlocks the bronze gates. The description of the key to the city is "A skeleton key."

A room can be indoors or outdoors. The Post Office, the Alfred Cralle Pool Hall, the Store, Cold Comfort, Town Hall, and the Library are indoors.

Use full-length room descriptions.

After looking in an outdoors room:
    let started printing be false;
    now every proximate door is not mentioned;
    if an indoors room is adjacent:
        let started printing be true;
        say "From here you can head into [the list of adjacent indoors rooms][if a proximate door is not mentioned], or go through [the list of proximate doors which are not mentioned][end if]. [run paragraph on]";
    if an outdoors room is adjacent:

```
        say "You could[if started printing is true] also[end if] go ";
        let count be the number of adjacent outdoors rooms;
        let index be count;
        repeat with next room running through adjacent outdoors rooms:
            let way be the best route from the location to the next room;
            say "[way] to [the next room]";
            decrement index;
        make delimiter index of count, continuing;
    if a proximate door is not mentioned:
        let started printing be true;
        say "[if started printing is true]Also available[otherwise]Your available
exits[end if] [is-are the list of proximate doors which are not mentioned].";
    otherwise:
        if started printing is true, say paragraph break.

Definition: a door is proximate:
    if the front side of it is the location, yes;
    if the back side of it is the location, yes;
    no.

Before exiting when the player is in an indoors room:
    if the player can see a door (called nearest exit), try entering the nearest exit
instead;
    repeat with way running through directions:
        let next room be the room way from the location;
        if the next room is a room, try going way instead.

Blank is a room. The destination of the player is Blank. Blank contains 15 ice
cream cones.

Section 4 - Other Players

Vanessa is a woman in Cold Comfort. Vanessa owns Cold Comfort.

Francine is a woman in the Public Library. Francine carries a book called Phlox
for Phyllis. Francine carries a stamped envelope called a pink stamped
envelope.

Lewis is a man in the Alfred Cralle Pool Hall. Lewis carries 3 stamped
envelopes. Lewis carries a book called Idiot's Guide to Dating. Lewis carries a
book called How to Meet Women. Lewis carries a book called Seduction in
Three Easy Steps. Lewis carries a DVD called Sleepless in Seattle.

Gene is a man in Nancy Johnson Memorial Square. Gene carries a stamped
envelope. Gene carries a DVD called Casablanca. Gene carries a coupon.

Rhoda is a woman in Marciony Street. Rhoda carries a book called The
Marciony Street Murders. Rhoda carries a DVD called Unsolved Serial Killings
XVIII. Rhoda carries a stamped envelope called a squashy package.

Martin is a man in Main Street. Martin carries a DVD called The Lifecycle of the
South Sea Tortoise. Martin carries a coupon.

Antony is a man in Movie Rental. Antony carries a coupon. Antony carries a
stamped envelope called a postcard.
```

```
Shelby is a man in the Town Hall. Shelby carries a DVD called Conducting An
Orderly Meeting. Shelby carries 5 stamped envelopes. Shelby carries an ice
cream cone. Shelby carries a coupon.

Christopher is a man in the Library. Christopher owns the Library.

Linnea is a woman in the Alfred Cralle Pool Hall. Linnea owns the Alfred Cralle
Pool Hall.

Ned is a man in the Movie Rental Store. Ned owns the Movie Rental.

After printing the name of someone (called target) while listing contents: if the
target owns the location of the target, say " (the owner)".

The description of a person is usually "[The noun] [if the noun is clean]looks
healthy[otherwise]is the color of [infection color of the noun][end if]."

After examining another person who is carrying something: say "[if the noun is
female]She[otherwise]He[end if] is carrying [a list of things carried by the noun]."

When play begins: let Patient Zero be a random other person; now patient zero
is infected.
```

This is a light variation of a previous example, but we use it here because it is convenient:

```
Section 5 - Conversation

A person has a table name called conversation.

Instead of asking someone about something:
    let the source be the conversation of the noun;
    if topic understood is a topic listed in source:
        if there is a turn stamp entry:
            say "You have already heard that [summary entry].";
        otherwise:
            now turn stamp entry is the turn count;
            now the character entry is the noun;
            say "[reply entry][paragraph break]";
    otherwise:
        say "[The noun] stares at you blankly.".

Instead of telling someone about something:
    try asking the noun about it.

Understand "recap" or "recall" or "review" as recalling conversations.

Recalling conversations is an action applying to nothing.

Carry out recalling conversations:
    repeat with speaker running through other people:
        let source be the conversation of the speaker;
        sort source in turn stamp order;
        say "[The speaker] has so far told you: [line break]";
```

```
let index be 0;
repeat through source:
    if there is a turn stamp entry and the speaker is character entry:
        let index be 1;
        say " [summary entry][line break]";
if index is 0, say " absolutely nothing[line break]";
say line break.
```

The conversation of a person is usually Table of General ChitChat.

Table of General ChitChat

topic	reply	summary	turn stamp	character
"weather/heat/warmth"	"'It's appalling, isn't it? You'd think we didn't pay our taxes.'"	"that the weather is appalling"	a number	a person
"sun/sunlight"	"'Good thing the town mostly switched to solar power, har, har.'"	"that the town is mostly relying on solar power"		
"rain"	"'Nope, there isn't going to be rain for 132 days,' replies [the noun]."	"that rain is not expected for another 132 days"		
"snow/hail/ice"	"This hilarious sally is greeted with hoots of laughter only."	"that the concept of snow is downright laughable"		
"disease/sickness/illness/syndrome"	"You get a cold, fixed stare in response. 'That's not funny,' [the noun] replies finally."	"that discussing the disease is more or less taboo"		
"cold comfort"	"'If you haven't tried it, you should,' says [the noun]. 'Best ice cream in town, and that's saying something, you bet.'"	"that Cold Comfort has the best ice cream in town"		
"town/city/village"	"'Yeah, it's a mite odd,' allows [the noun]. 'Not to everyone's taste, like...' [the noun as pronoun] considers for a moment. 'Like ginger ice cream. Big pieces of crystallized ginger... not everyone likes that.'"	"that the town is a mite odd"		
"forecast/weatherman" or "weather forecast/man"	"'Oh, the weather man's gotten a lot more reliable since the gummint started making it for us,' says [the noun]. 'Now he just reads off the schedule on the air every morning. Pretty much takes the fun right out of the news, if you ask me.'"	"that the weather is all generated by schedule"		
"taxes/tax" or "weather tax"	"A snort. 'You'd think for the rates we pay we'd get something a little pleasanter, don't you?'"	"that the weather tax really ought to be paying for something nicer than what you get"		
"job/employment/work"	"'[if the noun owns a room (called the shop)]I own [the shop],' replies [the noun][otherwise]Work at the creamery, like most folk around here,' answers [the noun]."	"this and that about employment in town"		
"book/books/reading"	"'The Public Library has a good selection, excepting only the cookbook section,' says [the noun]. 'That got censored way back when-- well, way back.'"	"that the Public Library has a good collection, except for the cookbook section"		

The conversation of Vanessa is the Table of Vanessa Chatter.

Table of Vanessa Chatter

topic	reply	summary	turn	character

```
"ice cream" or "sorbet/sherbet/flavor/flavors/flavour/flavours/ice/ices"    "'The flavors are [list of flavors],' she responds promptly, without needing to draw breath."    "that the flavors are [list of flavors]"    stamp a number    a person
```

```
After reading a command:
    while player's command includes "the":
        cut the matched text.
```

This strips 'the' out of the command, so that ASK PERSON ABOUT THE RAIN will be understood as well as ASK PERSON ABOUT RAIN.

Now we try something a bit unusual. Inform on its own will report each action on its own line, so that each character who walks into or out of a room will be described in a separate paragraph. This is usually fine, but in a game with a lot of characters moving around simultaneously, it can become a bit overwhelming. Instead, we may want to condense these reports into a single line, such as "Ben and Jerry enter from the south". The following accomplishes that goal by replacing some of the reporting rules, storing the information in a table, and then reading the table back later, once all the character movement has been resolved and the reports can usefully be collated:

```
Section 6 - Movement Description

A person has some text called walk style. The walk style of a man is usually "stride". The walk style of a woman is usually "strut". The walk style of Gene is "[one of]wander[or]stroll[purely at random]". The walk style of Francine is "waddle". The walk style of Antony is "scamper". The walk style of Rhoda is "sashay".

Table of Visible Exits

character    second    third    heading chosen    total
a person    a person    a person    a direction    a number
with 10 blank rows.

Table of Visible Entrances

character    second    third    heading chosen    total
a person    a person    a person    a direction    a number
with 10 blank rows.

To clear (current table - a table name):
    repeat through current table:
        blank out the whole row.

To tidy departures of (current table - a table name):
    let next direction be up;
    repeat through current table:
        if heading chosen entry is next direction:
            let accomplice be character entry;
            choose row with heading chosen of next direction in the current table;
            if total entry is 1:
                now second entry is accomplice;
                now total entry is 2;
            if total entry is 2:
                unless the second entry is accomplice:
```

```
            now third entry is accomplice;
            now total entry is 3;
        choose row with character of accomplice in the current table;
        blank out the whole row;
    otherwise:
        let next direction be heading chosen entry.

A door has a person called last opener.

Report someone opening a door:
    now group size is 1;
    now the last opener of the noun is the person asked;
    if the person asked is visible, say "[The person asked] opens [the noun]. [run
paragraph on]" instead;
    otherwise say "[The noun] opens from the other side. [run paragraph on]"
instead.

Report someone going through a door (called route):
    if the person asked is not the last opener of the route, continue the action;
    if the person asked is the last person named, say "[The person asked as
pronoun]";
    otherwise say "[The person asked]";
    say " [if the person asked is in the location]comes[otherwise]goes[end if]
through[if the last thing named is not the route] [the route][end if]." instead.

The last thing named is a thing that varies. Before printing the name of
something (called target) which is not a person: now the last thing named is the
target.

Report someone going a direction:
    if the person asked is in the location,
        choose a blank row in the table of visible entrances;
    otherwise choose a blank row in the table of visible exits;
    now character entry is the person asked;
    now total entry is 1;
    if the person asked is in the location,
        now heading chosen entry is the opposite of the noun;
    otherwise now heading chosen entry is the noun;
    stop the action.

This is the movement reporting rule:
    sort the Table of Visible Entrances in heading chosen order;
    tidy departures of the table of visible entrances;
    sort the Table of Visible exits in heading chosen order;
    tidy departures of the table of visible exits;
    let total row count be the number of filled rows in the Table of Visible
Entrances plus the number of filled rows in the Table of Visible Exits;
    if total row count is 0, rule succeeds;
    generate descriptions from the Table of Visible Entrances;
    generate descriptions from the Table of Visible Exits;
    clear the Table of Visible Entrances; clear the Table of Visible Exits.

To generate descriptions from (current table - a table name):
    let count be the number of filled rows in the current table;
    if count is 0, rule succeeds;
    let index be count;
```

```
    repeat through the current table:
        let accomplice be character entry;
        if character entry is a person, now character entry is marked for listing;
        if there is a second entry and second entry is a person, now second entry is
marked for listing;
        if there is a third entry and third entry is a person, now third entry is marked
for listing;
        let target be the room the heading chosen entry from the location;
        if total entry is 3, say "[The character entry], [the second entry][optional
comma] and [the third entry] ";
        if total entry is 2, say "[The character entry] and [the second entry] ";
        if total entry is 1:
            if the character entry is the last person named, say "[The character entry
as pronoun] ";
            otherwise say "[The character entry] ";
        if total entry is 1, say "[walk style of the character entry]s ";
        otherwise say "walk[if total entry is 1]s[end if] ";
        if the character entry is in the location:
            if location is indoors and target is indoors, say "over from ";
            if location is outdoors and target is indoors, say "out of ";
            if location is indoors and target is outdoors, say "in from ";
            if location is outdoors and target is outdoors, say "over from ";
        otherwise:
            if location is indoors and target is indoors, say "over to ";
            if location is outdoors and target is indoors, say "into ";
            if location is indoors and target is outdoors, say "out [if a door is visible]
[the random visible door][end if] to ";
            if location is outdoors and target is outdoors, say "over to ";
        if target is outdoors, say "[the heading chosen entry]";
        otherwise say "[the target]";
        if the total entry is 1 and count is 1 and accomplice carries something, say
", carrying [a list of things carried by the accomplice]";
        decrement index;
        make delimiter index of count, continuing;
        now group size is total entry;
    if a marked for listing person is infected:
        [eliminate the case in which we have already seen this description because
we just typed LOOK and the patient was in the room at the time]
        if looking and a marked for listing person is not in the location:
            clear marked people;
            say paragraph break;
        otherwise:
            describe patients;
    otherwise:
        clear marked people;
        say paragraph break.

The last person named is a person that varies. Before printing the name of a
person (called target): now the last person named is the target. Group size is a
number that varies. Group size is 1.

To clear marked people:
    repeat with named party running through people:
        now the named party is not marked for listing.

Before listing nondescript items:
    if the number of people who are marked for listing is 0, make no decision;
```

```
    say "You can see [a list of people who are marked for listing] here. ";
    now group size is the number of people who are marked for listing;
    describe patients;
    now every marked for listing person is not marked for listing.

To describe patients:
    if every marked for listing person is infected and at least three people are
marked for listing:
        say "They are all sick as dogs, every one.";
        clear marked people;
        rule succeeds;
    otherwise:
        if the number of people who are marked for listing is greater than two and
the number of infected people who are marked for listing is greater than the
number of clean people who are marked for listing:
            say "Only [the list of clean people who are marked for listing] currently
remain[if the number of clean people who are marked for listing is 1]s[end if]
untainted.";
            clear marked people;
            rule succeeds;
    let count be the number of marked for listing other people who are infected;
    if count is 0:
        say paragraph break;
        make no decision;
    let index be count;
    repeat with patient running through marked for listing other people who are
infected:
        if index is count:
            if count is 1 and the patient is the last person named:
                say "[The patient as pronoun]";
            otherwise:
                say "[The patient]";
        otherwise:
            say "[the patient]";
        say " [looks as though dipped in for index] [infection color of the patient]";
        decrement index;
        make delimiter index of count;
    clear marked people.

To say (named character - a man) as pronoun:
    if group size is 1, say "He"; if group size is 2, say "The latter"; if group size is
greater than 2, say "The last".

To say (named character - a woman) as pronoun: if group size is 1, say "She"; if
group size is 2, say "The latter"; if group size is greater than 2, say "The last".

To say looks as though dipped in for (index - a number):
    let divider be the number of filled rows in the Table of Dipping Phrases;
    if index is greater than 4, let index be the remainder after dividing index by
divider;
    choose row index in the Table of Dipping Phrases;
    say dipping entry.

Table of Dipping Phrases
dipping
"looks as though dipped in"
```

```
"could have been rolling in"
"has a bad case of"
"suffers from"
"contracted a virulent"

A door is usually scenery.
```

The next part could be simpler, but for rigor we will write it in such a way that it will work whether or not the serial comma is in use. This requires some extra work.

```
To make delimiter (index - a number) of (count - a number), continuing or halting:
    if index is 0:
        if continuing, say ". [run paragraph on]";
        otherwise say ".";
    otherwise if index is 1:
        if count is 2, say " and ";
        otherwise say "[optional comma] and ";
    otherwise:
        say ", ".

To say optional comma:
    if the serial comma option is active:
        say ",".

Test me with "go to cold comfort / z / z / z / z / ask vanessa for french vanilla /
ask vanessa for chocolate / ask vanessa about flavors / ask vanessa for
chocolate chocolate chip".
```

Because so much of this game is randomized, it will not be possible to provide a test command that systematically solves it. A good strategy is to go to Main Street, get the syringe; go to the Town Hall and get the key; then visit the shops, inject everyone, and lock them in when they've all been injected. Then go to the Pool Hall, turn on the air conditioner, and wait for the remaining parties to show up.

This is also something that could get fairly slow if we added many more rooms and characters to it. In that case, we might want to select fast route-finding so that character movement won't take as long. This will cost memory, possibly forcing the game into Glulx format if it isn't already, but significantly reduce the run-time for large maps with numerous people moving each turn:

```
Use fast route-finding.
```

403

Example Flotation

RB

Objects that can sink or float in a well, depending on their own properties and the state of the surrounding environment.

Here we want a rulebook to determine whether objects float or sink, so we create an object-based rulebook for the purpose. The more specific rules here, pertaining to corks and to inflated things, will be consulted first; then, as a default, the general flotation rule.

We also want a switch that can turn flotation off at will. The rule about the big switch will be observed before the others because the when... clause makes it more specific than the other rules in the flotation rulebook.

If we wanted, we could also put these rules into a rulebook in an explicit order, overriding Inform's automatic sorting by specificity.

```
"Flotation"

The Pumping House is a room.

A well is a fixed in place container in the Pumping House.

Instead of examining the well:
    say "[if something is in the well]On the surface of the water you can see [a list
of things in the well][otherwise]There is nothing on the surface of the water, nor
can you see into the depths[end if]."

The well bottom is a container.

The cork, the rubber ring and a lead ingot are in the Pumping House.

A big switch is a fixed in place device in the Pumping House. "A big switch
labelled 'MAKE EVERYTHING SINK' is mounted on one wall[if switched on]. It
crackles with electricity[otherwise]. It is currently switched off and silent[end if]."

A thing can be inflated or uninflated. A thing is usually uninflated. Before printing
the name of an inflated thing: say "inflated ".

The rubber ring is inflated.

The flotation rules are an object-based rulebook.

A flotation rule for the cork: rule succeeds.
A flotation rule for an inflated thing: rule succeeds.
A flotation rule when the big switch is switched on: rule fails.

After inserting something into the well:
    follow the flotation rules for the noun;
    if the rule succeeded:
        say "[The noun] bobs on the surface.";
    otherwise:
        move the noun to the well bottom;
        say "[The noun] sinks out of sight."

A thing can be sinking, rising, or static. A thing is usually static.

Definition: a thing is wet:
    if it is in the well, yes;
    if it is in the well bottom, yes;
    no.
```

```
Every turn:
    now every thing is static;
    repeat with item running through wet things:
        follow the flotation rules for the item;
        if the rule failed and the item is in the well, now the item is sinking;
        if the rule succeeded and the item is in the well bottom, now the item is
rising;
    now every rising thing is in the well;
    now every sinking thing is in the well bottom;
    if something is rising, say "[The list of rising things] rise[if the number of rising
things is 1]s[end if] to the surface of the well.";
    if something is sinking, say "[The list of sinking things] sink[if the number of
sinking things is 1]s[end if] out of sight."
```

And finally a few description rules to make things look prettier:

```
Rule for writing a paragraph about the well when the well contains something:
    say "The chief feature of the room is a concrete-sided well in which there
float[if the number of things in the well is 1]s[end if] [a list of things in the well]."

Rule for writing a paragraph about the well:
    say "The chief feature of the room is a concrete-sided well full of water."
```

As we recall from the chapter on activities, "writing a paragraph about..." is an activity; activities are themselves structured as sets of object-based rulebooks. The activity "writing a paragraph about" uses three object-based rulebooks (before writing..., for writing..., after writing...). We could have made a flotation activity as well, but in general it is overkill to make an activity to make success/failure decisions. For that purpose an object-based rulebook is sufficient.

```
Test me with "get all / put cork in well / put ring in well / put ingot in well / x well /
get cork / get ring / switch switch on / put cork in well / put ring in well / x well /
switch switch off / switch switch on".
```

404

Example Kyoto

RB

Expanding the effects of the THROW something AT something command so that objects do make contact with one another.

Suppose we want to expand the function of the existing THROW SOMETHING AT command so that a thrown object actually does make contact most of the time. A glance at the Actions index tells us that the Throwing it at rulebook currently looks like this:

```
Throwing something at something (past tense thrown it at)
    "drop [something held] at/against/on/onto [something]"

        check an actor throwing something at implicitly remove thrown clothing rule
        check an actor throwing something at futile to throw things at inanimate objects rule
        check an actor throwing something at block throwing at rule
```

Some of those still look useful. We want to leave the "implicitly remove thrown clothing" rule, for instance -- no fair having the player throw a hat that's on his head. On the other hand, the "futile to throw things at inanimate objects rule" is going to have to go, because that would prevent us from ever being able to complete the throwing command. So let's get rid of that:

```
"Kyoto"

Part 1 - Throwing Rules

The futile to throw things at inanimate objects rule is not listed in the check
throwing it at rules.
```

That "block throwing at" rule also looks sinister: any "block..." rule in the standard actions library is there to print a message telling the player he can't do what he's asked to do.

But it's not enough to ignore it, the way we did the "futile" rule. Since we are only expanding the command to affect inanimate objects, let's replace the "block throwing at" rule with a different one which will only prevent the player throwing things at people:

```
The block throwing at people rule is listed instead of the block throwing at rule in
the check throwing it at rules.

This is the block throwing at people rule:
    if the second noun is a person, say "That might be construed as an attack."
instead.
```

Now we've changed the command so that some action can sometimes be carried out here -- but we don't have any rules for what happens. It's time to create some rules for our model world.

```
A thing can be hard or soft. A thing can be fragile or strong. Shape is a kind of
value. The shapes are round, flat, and linear. A thing has shape.
```

If we're actually going to allow throwing, we might want to add a couple of extra checks to the rulebook to make sure that this happens when it ought:

```
Check throwing it at (this is the block juggling rule):
    if the player is carrying the second noun, say "It would be difficult to throw at
something you are yourself holding." instead.

Check throwing it at (this is the avoid throwing things into themselves rule):
    if the second noun is within the noun, say "That would be a nice magic trick."
instead.
```

And then the rules for the action itself:

```
Carry out throwing it at (this is the check aerodynamics rule):
    if the noun is flat:
        move noun to location;
```

```
        say "[The noun], unwieldy, flutters to the ground.";
        rule succeeds.
```

That "rule succeeds" ends the action here, if the noun is flat. If not, Inform goes on to the next rule in the carry out throwing it at rulebook:

```
Carry out throwing it at (this is the contact rule):
    say "[The noun] hits [the second noun].[paragraph break]";
    if the second noun is fragile and the noun is hard:
        destroy the second noun.

Carry out throwing it at (this is the landing rule):
    let destination be the location;
    if the second noun is on a supporter (called endtable), let destination be the
endtable;
    if the second noun is a supporter, let destination be the second noun;
    move the noun to the destination;
    if the noun is fragile and the second noun is hard:
        destroy the noun;
        rule succeeds;
    say "[The noun] lands [if the destination is the location]nearby[otherwise]on
[the destination][end if]."
```

These rules are assuming some backup information, so let's provide that as well:

```
Reliance relates a thing (called X) to a thing (called Y) when X is part of Y or X is
in Y or X is on Y. The verb to be relying on means the reliance relation.

To destroy (item - a thing):
    let home be the holder of the item;
    if the item is part of something (called the superstructure), let home be the
holder of the superstructure;
    if the item is visible:
        say "[The item] breaks[if something is relying on the item], leaving [a list of
things which are relying on the item] behind[end if].";
    if something is relying on the item,
        now all the things which are relying on the item are in the home;
    now the item is nowhere.
```

Now suppose we'd like to add some further cases for what happens if the player breaks a fragile door this way:

```
To destroy (item - a door):
    now the item is open;
    now the item is unopenable;
    say "[The item] smashes."

Rule for printing the name of an unopenable open door while not throwing
something at something:
    say "open doorway".

Understand "door" or "doorway" as a door.
```

This works, except that objects will continue to "strike" open, unopenable doors, with the result that the player can smash the same door over and over. What we need is another rule, after the aerodynamics rule and before the contact rule, that tells Inform how to handle throwing things at open doors.

```
This is the flying through doorways rule:
    if the second noun is an open door:
        let the distant room be the other side of the second noun;
        move the noun to the distant room;
        say "[The noun] flies out of sight into [the distant room].";
        rule succeeds.
```

If the original rulebook is one we wrote ourselves, we could just add that rule in the proper spot in order. If we got it from an extension, though, we might need to put it in the right place explicitly:

```
The flying through doorways rule is listed before the contact rule in the carry out
throwing it at rules.
```

The magic of rulebooks is that they allow authors to amend each other's work (or the Standard Rules) with a fair amount of freedom. A well-written extension will give individual names to its rules, to allow subsequent authors to modify the function of the extension without too much trouble.

Now for an actual scenario with which to test this:

```
Part 2 - The Study

The sliding paper screen is a door. It is north of the Moss Garden and south of
the Study. The paper screen is fragile.

The player carries a netsuke and a shamisen. The description of the netsuke is
"A weight for the cord on which you wear your purse or your medicine box. This
particular one has the shape of a bullfrog, carved from green stone." The
netsuke is round, hard, and strong. Understand "green" or "stone" or "bullfrog"
as the netsuke.

The description of the shamisen is "An instrument you have only begun to learn
to play." The shamisen is linear, soft, and fragile. A neck is part of the shamisen.
The neck is linear, strong, and hard. A body is part of the shamisen. The body is
round, fragile, and soft. A string is part of the shamisen. The string is linear, soft,
and strong. The printed name of the body is "[if the body is not part of the
shamisen]shamisen [end if]body". The printed name of the neck is "[if the neck is
not part of the shamisen]shamisen [end if]neck". Understand "shamisen" as the
body when the body is not part of the shamisen. Understand "shamisen" as the
neck when the neck is not part of the shamisen.

The description of the Study is "A restful three-tatami room." The Study contains
a calligraphy box and a hanging scroll. The initial appearance of the hanging
scroll is "A handsome scroll depicts two women in kimonos crossing a bridge;
Mount Fuji is in the background." The calligraphy box contains a brush. The box
is openable and closed. The brush is hard, linear, and strong. The calligraphy
box is round, soft, and strong. The hanging scroll is flat, soft, and strong.
```

The description of the Moss Garden is "Earlier today, you arranged three leaves on the moss in imitation of autumn. They must not be disturbed." The leaves are scenery in the Moss Garden. Instead of throwing something at the leaves: say "You spent too long over their placement."

Test me with "test one / test two".

Test one with "open screen / throw netsuke at screen / n / get netsuke / close screen / get scroll / throw scroll at screen / throw netsuke at scroll / get netsuke / throw netsuke at shamisen / drop netsuke".

Test two with "throw shamisen at netsuke / get all / throw netsuke at screen / get netsuke / throw netsuke at door / s / get netsuke".

405

Example Being Peter

RB

A set of rules determining the attitude a character will take when asked about certain topics.

Let's say that we're implementing a particularly irrational and volatile character. Some of the time she remains composed; some of the time she reacts with unexpected vehemence for reasons only partly related to what was said.

Moreover, her responses are divided between successful and failing outcomes, where success indicates that she's not too upset and failure means that she is distraught; we use this to determine how the rest of the room reacts.

"Being Peter"

The Drawing Room is a room. "The company is assembled here for champagne. Most of it, anyway: Mary is on the phone to her babysitter, Roger is keeping her anxious company, and Carol doesn't drink. But everyone else."

Maggie is a woman in the Drawing Room.

The player wears a top hat.

Quizzing it about is an action applying to one thing and one visible thing. Understand "ask [someone] about [any thing]" as quizzing it about.

Instead of quizzing Maggie about something:
    follow the attitude rules;
    say "Everyone waits to see what the reaction will be: [outcome of the rulebook].";
    if rule succeeded, say "There is general relief.";
    otherwise say "Everyone is pointedly silent."

The attitude rules are a rulebook. The attitude rules have outcomes she stays calm (no outcome - default), she gets angry (failure), she has a stroke (failure), she is only mildly annoyed (success), and she is elated (success).

Here we want Inform to consult every appropriate attitude rule until it gets to some answer; if an attitude rule does not provide a result, the default 'no outcome' will mean that we go on to the next rule, and so on.

```
A subject is a kind of thing. income, love life, and children are subjects.

An attitude rule for quizzing Maggie about love life:
    she gets angry.

An attitude rule:
    if the player wears the top hat, she gets angry.
```

Now, as we saw, the 'no outcome' result will never be returned and printed as Maggie's reaction, precisely because it is "no outcome". Therefore, we provide a final attitude rule which will give her a default response to all statements:

```
The last attitude rule:
    she is only mildly annoyed.

Test me with "ask maggie about love / ask maggie about income / take off hat /
ask maggie about income".
```

There are plenty of contexts where we might want named outcomes for clarity but not want to print the results literally afterward.

406

Example **Feline Behavior**

RB

A cat which reacts to whatever items it has handy, returning the result of a rulebook for further processing.

Suppose we have a cat which is supposed to react to (and destroy) the most interesting thing in its environment. There are several ways we could approach this problem, but for the sake of demonstration, let's have it follow a rulebook to figure out which thing it most wants to interact with. We will then return the chosen object as "the object produced by the cat behavior rules".

```
"Feline Behavior"

The Kitchen is a room. The cat is an animal in the Kitchen. In the Kitchen is a
bowl, a ball of wool, a newspaper. The bowl contains a quantity of cream.

The cat is wearing a silver collar. The description of the cat is "It is wearing [a list
of things worn by the cat]."

The player carries a closed openable container called a bag. The bag contains
catnip.

The cat behavior rules is a rulebook producing an object.
```

```
A cat behavior rule when the cat can touch the catnip:
    say "The cat frolics with the catnip until nothing remains of it.";
    rule succeeds with result catnip.

A cat behavior rule when the cat can touch the cream:
    say "The cat laps up the cream.";
    rule succeeds with result cream.

A cat behavior rule when the cat can touch the ball of wool:
    say "The cat makes the ball of wool into a useless tangle which must be
discarded.";
    rule succeeds with result ball.

A cat behavior rule when the cat can touch the newspaper:
    say "The cat bats playfully at the newspaper until all the nasty boring articles
are destroyed.";
    rule succeeds with result newspaper.

A cat behavior rule:
    say "The cat looks miffed at the lack of ready entertainment, and glares at you
with yellow eyes as though wondering whether your pants leg is good for claw-
sharpening.";
    rule fails.

Every turn:
    let the destroyed object be the object produced by the cat behavior rules;
    if the destroyed object is not nothing:
        now the destroyed object is nowhere;
        say "[line break]Good thing you have no use for [the destroyed object]
yourself.[paragraph break]".

Test me with "z / z / open bag / z / z".
```

We include the if rule succeeded... condition here because nothing will be returned if the cat's search failed (as for instance in the result of the final rule).

Naturally, if we wanted we could equally well ask "if rule failed...".

407

Example Tilt 2

RB

A deck of cards with fully implemented individual cards; when the player has a full poker hand, the inventory listing describes the resulting hand accordingly.

In our previous implementations of playing cards, we've gotten as far as creating decks of individual cards that the player can draw and discard. But in a poker game, one doesn't just have a collection of cards: one has a hand of a specific kind.

Here we take on the job of writing an inventory listing for a poker hand that will reflect the real value of what the player has drawn. To do this, we create a rulebook to sort and assess the cards in the player's hand; its possible return values are limited to the kinds of poker hands that exist, from "high card" to "royal flush".

The first three sections, creating the deck of cards and the means to parse their names, are identical to those we've already seen in Tilt 1; new material begins at section 4.

For the purposes of demonstration, we're simulating something akin to five-card draw without wilds; stud or hold-em variations would add some other complexities.

```
"Tilt"

Section 1 - Cards

Suit is a kind of value. The suits are hearts, clubs, diamonds, and spades.
Understand "heart" as hearts. Understand "club" as clubs. Understand
"diamond" as diamonds. Understand "spade" as spades.

A card is a kind of thing. A card has a suit. A card has a number called rank.
Understand the suit property as describing a card. Understand the rank property
as describing a card.

52 cards are in the card repository.

To say (count - a number) as a card value:
    choose row count in the Table of Value Names;
    say "[term entry]".

Rule for printing the name of a card (called target):
    say "[rank of the target as a card value] of [suit of the target]"

Table of Value Names

    term      value topic
    "ace"     "1"   "ace/A/one"
    "deuce"   "2"   "deuce/two"
    "three"   "3"   "three"
    "four"    "4"   "four"
    "five"    "5"   "five"
    "six"     "6"   "six"
    "seven"   "7"   "seven"
    "eight"   "8"   "eight"
    "nine"    "9"   "nine"
    "ten"     "10"  "ten"
    "jack"    "11"  "jack/knave/J"
    "queen"   "12"  "queen/Q"
    "king"    "13"  "king/K"

After reading a command:
    if the player's command includes "of [suit]":
        while the player's command includes "of":
            cut the matched text;
    repeat through the Table of Value Names:
        while the player's command includes topic entry:
            replace the matched text with value entry.

When play begins:
    reconstitute deck.
```

```
To reconstitute deck:
    let current suit be hearts;
    now every card is in the card repository;
    while a card is in the card repository:
        repeat with current rank running from 1 to 13:
            let item be a random card in card repository;
            now rank of item is current rank;
            now suit of item is current suit;
            now item is in the deck of cards;
        now current suit is the suit after the current suit.
```

Section 2 - The Deck and the Discard Pile

The Empty Room is a room. "Nothing to see here."

The deck of cards is in the Empty Room. It is a closed unopenable container. The description is "A standard poker deck."

The discard pile is a closed unopenable container. The description is "Cards in this game are discarded face-down, so the discard pile is not very interesting to see. All you can observe is that it currently contains [if the number of cards which are in the discard pile is less than ten][the number of cards which are in the discard pile in words][otherwise]about [the rounded number of cards which are in the discard pile in words][end if] card[s]."

```
To decide what number is the rounded number of (described set - a description of objects):
    let N be the number of members of the described set;
    let R be N divided by 5;
    let total be R times 5;
    decide on total.
```

Rule for printing room description details of something: do nothing instead.

Section 3 - Drawing and Discarding Actions

Understand the commands "take" and "carry" and "hold" and "get" and "drop" and "throw" and "discard" as something new.

Understand "take [text]" or "get [text]" or "drop [text]" as a mistake ("Here, you only draw and discard. Nothing else matters at the moment.").

Understand "draw" or "draw card" or "draw a card" as drawing. Drawing is an action applying to nothing. The drawing action has an object called the card drawn.

```
Setting action variables for drawing:
    now the card drawn is a random card which is in the deck of cards.
```

```
Check drawing:
    if the card drawn is nothing, say "The deck is completely depleted." instead.
```

```
Check drawing:
    if the number of cards carried by the player is greater than four,
```

```
        say "This is a five-card game; you must discard something before drawing
anything further." instead.

Carry out drawing:
    move the card drawn to the player.

Report drawing:
    say "You draw [a card drawn]."

Understand "discard [card]" as discarding. Discarding is an action applying to
one thing.

Check discarding:
    if the player does not carry the noun, say "You can only discard cards from
your own hand." instead.

Carry out discarding:
    now the noun is in the discard pile;
    if the discard pile is not visible, move the discard pile to the location.

Report discarding:
    say "You toss [the noun] nonchalantly onto the discard pile."
```

New material begins here. We want to start by grouping cards together, but identifying poker hands only if the player holds a full five cards.

```
Section 4 - Assessing Hands

Before listing contents while taking inventory: group cards together.

Before grouping together cards:
    if the number of cards carried by the player is 5:
        say "[run paragraph on]";
        follow the hand-ranking rules;
        if the rule succeeded, say "[the outcome of the rulebook]";
        otherwise say "some random cards";
        if the outcome of the rulebook is pair outcome, say " of [rank of the first
thing held by the player as a card value]s";
    otherwise:
        say "[number of cards carried by the player in words] assorted cards";
    say " (".

Rule for grouping together cards:
    say "[list hand]".

To say list hand:
    let chosen card be the first thing held by the player;
    while chosen card is a card:
        say "[chosen card]";
        now chosen card is the next thing held after chosen card;
        if chosen card is a card, say ", ".

After grouping together cards:
    say ")".
```

The ranking of poker hands traditionally depends on three features: 1) whether all the cards are of the same suit (flush); 2) whether the cards constitute a numerical run of ranks (straight); and 3) how many cards or sets of cards are of matching rank (pairs, three of a kind, and four of a kind). Here we will start by assessing our hand to determine these qualities:

```
The hand-ranking rules is a rulebook. The hand-ranking rules have outcomes
royal flush, straight flush, four of a kind, full house, flush, straight, three of a kind,
two pairs, pair, high card.

The hand-ranking rulebook has a truth state called the flushness.
The hand-ranking rulebook has a truth state called the straightness.

The hand-ranking rulebook has a number called the pair count.
The hand-ranking rulebook has a number called the triple count.
The hand-ranking rulebook has a number called the quadruple count.
```

For convenience in identifying hand features, and for elegance when we print the hand-listing, we start by sorting the cards in the player's hand so that the high-ranked cards are listed first. It is rare that we want to concern ourselves with this, but as we saw in the section on "Looking at containment by hand" in the chapter on Change, Inform keeps an ordered list of the items inside any given container; so it does order the objects in the player's hand, and the ordering depends on which things were added to the hand most recently. By moving something to the player's hand again (even if it was already there), we change this ordering, and wind up with a sorted hand.

```
A card can be sorted or unsorted. A card is usually unsorted.

Definition: a card is high if its rank is 11 or more.
Definition: a card is low if its rank is 4 or less.

A hand-ranking rule (this is the initial sort rule):
    now every card is unsorted;
    while the player carries an unsorted card:
        let item be the lowest unsorted card held by the player;
        move item to the player;
        now the item is sorted;
    if sort-debugging is true, say "-- after initial sort: [list hand]".
```

This last printing instruction is there for diagnostic purposes: later we'll add a testing command to turn debugging on and off; when it's on, the game will print out its card list at various stages in sorting, to help us trouble-shoot any problems. In normal play, however, this will be off.

Next up, a check to see whether the player has a flush:

```
A hand-ranking rule (this is the finding flushness rule):
    let called suit be the suit of a random card carried by the player;
    if every card carried by the player is called suit, now flushness is true.
```

Now we check for straights; this is slightly complicated by the fact that an ace can be either the bottom of a low straight (lower than 2) or the top of a high straight (higher

than king), so we explicitly check both possibilities.

```
A hand-ranking rule (this is the finding straightness rule):
    now straightness is true;
    let N be the rank of the highest card which is carried by the player;
    repeat with current rank running from N - 4 to N:
        now the test rank is the current rank;
        unless the player carries a matching card:
            if the current rank is N - 4 and the current rank is 9 and the player carries
an ace card, do nothing; [this covers the case where an ace could be the top
card of the sequence]
            otherwise now straightness is false.
```

And finally, we need to identify any groups of cards of the same rank. We want to know how many groups there are and how large each group is (though in practice there can only be one group of three or four in a standard-sized poker hand). We also want to mark any grouped cards so that we can move them to the front of the player's hand when we take inventory.

```
A card can be quadrupled, tripled, paired or uncombined.

Test rank is a number that varies. Definition: a card is matching if its rank is the
test rank.
```

This definition is a convenience so that we don't have to write so many explicit loops in the following rule:

```
A hand-ranking rule (this is the counting multiples rule):
    now every card is uncombined;
    repeat with current rank running from 1 to 13:
        now test rank is current rank;
        let N be the number of matching cards held by the player;
        if N is 4:
            increment the quadruple count;
            now every matching card held by the player is quadrupled;
        if N is 3:
            increment the triple count;
            now every matching card held by the player is tripled;
        if N is 2:
            increment the pair count;
            now every matching card held by the player is paired.
```

Next we tweak our sorting to reflect the make-up of the hand. There are two reasons why this might differ from the straight highest-to-lowest sort we did earlier:

1) we want to list aces as high unless they are serving as the bottom of a low straight, in which case they should appear last;

2) we want combinations to appear at the front of the list, sorted from highest value to lowest value: larger combinations first, then smaller combinations, and combinations of equal size sorted by rank.

```
A hand-ranking rule (this is the move aces up unless there's a low straight rule):
    unless the straightness is true and the lowest card carried by the player is an
```

```
ace card and the rank of the highest card carried by the player is 5,
        now every ace card which is carried by the player is carried by the player;
    if sort-debugging is true, say "-- after ace movement rule: [list hand]".

A hand-ranking rule (this is the move pairs forward rule):
    while the player carries a paired card:
        let selection be the lowest paired card which is carried by the player;
        move the selection to the player;
        now the selection is uncombined;
    if sort-debugging is true, say "-- after pairs movement: [list hand]".

A hand-ranking rule (this is the raise ace pairs rule):
    if the player carries exactly two ace cards:
        repeat with item running through ace cards which are carried by the player:
            move item to the player;
    if sort-debugging is true, say "-- after paired-ace movement: [list hand]".

A hand-ranking rule (this is the move multiples forward rule):
    while the player carries a tripled card:
        let selection be the lowest tripled card which is carried by the player;
        move the selection to the player;
        now the selection is uncombined;
    while the player carries a quadrupled card:
        let selection be the lowest quadrupled card which is carried by the player;
        move the selection to the player;
        now the selection is uncombined;
    if sort-debugging is true, say "-- after multiples movement rule: [list hand]".

Definition: a card is ace if its rank is 1.
Definition: a card is king if its rank is 13.
```

Now, having determined the salient qualities of our hand, we run through rules in order from the highest kind of poker combination to the lowest. Because of the order of the source, Inform will choose whichever combination applies first.

```
A hand-ranking rule (this is the royal-flush rule):
    if flushness is true and straightness is true and the highest card carried by the
player is king and the lowest card carried by the player is ace, royal flush.

A hand-ranking rule (this is the straight-flushes rule):
    if flushness is true and straightness is true, straight flush.

A hand-ranking rule (this is the four-of-a-kind rule):
    if the quadruple count is 1, four of a kind.

A hand-ranking rule (this is the full-house rule):
    if the pair count is 1 and the triple count is 1, full house.

A hand-ranking rule (this is the flushes rule):
    if flushness is true, flush.

A hand-ranking rule (this is the straights rule):
    if straightness is true, straight.
```

```
A hand-ranking rule (this is the three-of-a-kind rule):
    if triple count is 1, three of a kind.

A hand-ranking rule (this is the two-pair rule):
    if the pair count is 2, two pairs.

A hand-ranking rule (this is the pair rule):
    if the pair count is 1, pair.

A hand-ranking rule (this is the default rule):
    high card.
```

And finally, we need to define our debugging variable here, even though we won't give the player the ability to turn it on and off except in the special testing section.

```
Sort-debugging is a truth state that varies.
```

For many examples, a test-me script is enough to prove that the example does what it ought. This example, though, is a bit more complicated, and hard to test randomly. The remainder of the source here shows how we might write a test to verify the desired behavior of our rulebook. Those who are only interested in the rulebook itself can stop reading at this point.

```
Section 5 - Testing hand identification - Not for release
```

For the sake of testing our rules, we provide an apparatus that will load the player's hand up with sample hands of each kind, then show the result to make sure that the hand is being correctly identified.

```
Understand "debug sorting" as debugging hand sorting. Debugging hand sorting
is an action out of world.

Carry out debugging hand sorting:
    if sort-debugging is false, now sort-debugging is true;
    otherwise now sort-debugging is false.

Report debugging hand sorting:
    say "Sort debugging is now [if sort-debugging is true]on[otherwise]off[end if]."

Test me with "draw / g / g / g / g / force hand / g / g / g / g / g / g / g / g / g /
g".
```

The somewhat rough-and-ready principle of this table is that we will overwrite the cards in the player's hand by resetting their ranks and suits; every five rows of the table represent a new poker hand for the game to attempt to sort and identify. These include one example of each of the major kinds of poker hand, plus a couple of variations involving aces which test the special sorting rules.

```
Table of Testing Hands

set suit    set rank
spades      1           [royal flush]
spades      13
```

```
spades    12
spades    11
spades    10
clubs     12        [straight flush]
clubs     11
clubs     10
clubs     9
clubs     8
diamonds  8         [four of a kind]
hearts    8
spades    8
clubs     8
clubs     3
clubs     1         [full house]
spades    1
hearts    10
spades    10
clubs     10
hearts    2         [flush]
hearts    5
hearts    7
hearts    11
hearts    12
hearts    1         [straight]
spades    13
diamonds  12
clubs     11
hearts    10
hearts    2         [three of a kind]
spades    2
clubs     2
clubs     4
spades    3
diamonds  6         [two pairs]
spades    6
clubs     7
diamonds  7
hearts    9
diamonds  6         [two pairs, ace high]
spades    6
clubs     1
diamonds  7
hearts    1
hearts    12        [pair]
spades    12
diamonds  10
spades    7
clubs     4
diamonds  13        [high]
hearts    11
spades    9
clubs     7
diamonds  5
hearts    1         [tricky sorting: low straight]
diamonds  2
spades    3
diamonds  4
diamonds  5
```

Understand "force hand" as forcing a hand. Forcing a hand is an action out of world.

Current marker is a number that varies.

Carry out forcing a hand:
    repeat with item running through cards which are carried by the player:
        increment current marker;
        if current marker is greater than the number of filled rows in the Table of

```
Testing Hands, now current marker is 1;
        choose row current marker in the Table of Testing Hands;
        now the suit of item is the set suit entry;
        now the rank of item is the set rank entry.

Report forcing a hand:
    try taking inventory.
```

408

Example Electrified

RB

Adding a rule before the basic accessibility rule that will prevent the player from touching electrified objects under the wrong circumstances.

Suppose we want to prevent the player from touching anything electrified -- not just as a response to TOUCH OBJECT, but at any time when the action would require contact with the object in question.

```
"Electrified"

A thing can be safe or electrified. A thing is usually safe.

The Open Field is a room. "At this end of the field is a wire fence separating farm
country from the government testing grounds beyond." The wire fence is an
electrified thing in Open Field. It is scenery. The description of the wire fence is
"Built into the fence is [a list of things which are part of the fence]." The scary
box is an electrified container. It is part of wire fence. In the scary box is an
alluring prize.

The player carries a flashlight, a grappling hook, a very thick rubber glove, and a
length of rope. The glove is wearable.

This is the electrocution-wisdom rule:
    if the player wears the very thick rubber glove, make no decision;
    if the action requires a touchable noun and the noun is electrified, say "You
fear touching [the noun]." instead;
    if the action requires a touchable second noun and the second noun is
electrified, say "You fear touching [the second noun]." instead.

The electrocution-wisdom rule is listed before the basic accessibility rule in the
action-processing rules.

Before touching the scary box:
    say "You can't help noticing a bright red sticker on the surface of the box."
[This rule will fire even if we are not wearing the glove, because Before rules
occur before basic accessibility.]

Instead of opening the scary box:
    say "The scary box seems to be super-glued shut." [This one won't, because
Instead rules occur after basic accessibility.]

Test me with "touch fence / touch box / open box / wear glove / open box".
```

409

Example Timeless

RB

A set of actions which do not take any game time at all.

In a game with tight timing, it is sometimes friendliest to the player to let him LOOK and EXAMINE as much as necessary without being penalized.

```
"Timeless"

Examining something is acting fast. Looking is acting fast.
```

Now we need a rule which, just at the right moment, stops the turn sequence rulebook in the cast of our new fast-acting actions:

```
The take visual actions out of world rule is listed before the every turn stage rule
in the turn sequence rules.

This is the take visual actions out of world rule: if acting fast, rule succeeds.
```

Thus the rest of the turn sequence rulebook is omitted for looking or examining: in effect, they become out-of-world actions like "saving the game". If we wanted to add, say, taking inventory to the list of instant activities, we would just need to define it as acting fast, too.

Now the scenario for testing:

```
When play begins:
    say "You are cornered by a pack of zombie wolves, armed only with a torch
and a pair of pinking shears. This may be your last moment on earth, unless you
can think fast!"

Cleft is a room. "You're backed into a cleft in the granite: behind you are only
steep, high faces of stone, and before you a narrow passage."

The plural of zombie wolf is zombie wolves. A zombie wolf is a kind of animal.
Four zombie wolves are in Cleft.

Rule for writing a paragraph about zombie wolves:
    say "The good news is that there isn't much space in which for the zombie
wolves to attack.";
    now every zombie wolf is mentioned.

A steep high face of stone is scenery in Cleft. Understand "rock" as the stone.
The description is "Now that you look more closely, there appear to be pitons
driven into the rock."

Some pitons are part of the stone. The description of the pitons is "It looks as
though someone else has made this ascent before."
```

```
Instead of climbing the stone, try going up. Instead of climbing the pitons, try
going up.

Above the Cleft is Clifftop.

Every turn when the location is Cleft:
    say "Alas, your time has run out. The alpha wolf springs--";
    end the story.

Every turn when the location is Clifftop:
    say "After a breathless climb, you emerge at last onto the open clifftop.";
    end the story finally.

Test me with "x me / x stone / x pitons / climb pitons".
```

---

410  **Example Endurance** *RB*

Giving different actions a range of durations using a time allotment rulebook.

Here we move to a systematic way of giving different durations to different actions, including even variations on the same act -- so that for instance climbing a steep hill might take several minutes more than other going actions. We do this by setting a number, "work duration", to represent the number of minutes consumed by a given action, and then consulting a rulebook to find out how long the past turn's action should take. By default, an action will take 1 minute.

We'll start by emulating the behavior of "Uptempo": each turn we'll set the clock forward most of the way, then check to see what has changed since the last turn, print any relevant events, and only then set the clock forward the final minute. The exception is when an action is set to take no time at all; in that case, we'll skip the rest of the turn sequence rules entirely.

```
"Endurance"

Work duration is a number that varies.

Every turn:
    now work duration is 0;
    increment the turn count;
    follow the time allotment rules;
    if work duration is 0, rule succeeds;
    increase the time of day by (work duration minutes - 1 minute).

The time allotment rules are a rulebook.

A time allotment rule for examining or looking:
    now work duration is 0;
    rule succeeds.
```

```
A time allotment rule for going:
    now work duration is 2;
    rule succeeds.

A time allotment rule for going up:
    now work duration is 5;
    rule succeeds.

A time allotment rule for waiting:
    now work duration is 10;
    rule succeeds.

The last time allotment rule:
    now work duration is 1.

When play begins: now the right hand status line is "[time of day]".

The Quai is a room. "An attractive park at the edge of the river Aude: here you
can wander among palm trees, and watch cyclists go by on the bike path; in the
water there are ducks. In the cafe to your north, patrons sip their pastis; and
above you is the medieval walled city and its castle."

The Cafe is north of the Quai. "A charming collection of umbrella-shaded tables,
from which one can watch the river and the walls of the city beyond. The noise of
traffic is only a minor distraction."

The City is above the Quai.

After going to the City:
    say "You struggle uphill for some distance...";
    continue the action.

At 9:15 AM:
    say "The bells ring out from Place Carnot."

Test me with "z / n / s / u".
```

411

Example **Escape from the Seraglio**

RB

Replacing the usual response to TAKE ALL so that instead of output such as "grapes: Taken. orange: Taken.", Inform produces variable responses in place of "grapes:".

```
"Escape from the Seraglio"

Section 1 - Special Announcement Rules

The number of takes this turn is a number that varies. Every turn: now the
number of takes this turn is 0.
```

The friskily announce items from multiple object lists rule is listed instead of the announce items from multiple object lists rule in the action-processing rules.

This is the friskily announce items from multiple object lists rule:
    if taking:
        if the current item from the multiple object list is not nothing:
            increment the number of takes this turn;
            say "[if number of takes this turn is 1]First [otherwise if the number of takes this turn is 2]And then [otherwise if the number of takes this turn is 3]And I suppose also [otherwise if the number of takes this turn is 7]And on we wearily go with [otherwise if the number of takes this turn is 9]Oh, and not forgetting [otherwise]And [end if][the current item from the multiple object list]: [run paragraph on]";
    otherwise:
        if the current item from the multiple object list is not nothing, say "[current item from the multiple object list]: [run paragraph on]".

Rule for deciding whether all includes the person asked: it does not.
Rule for deciding whether all includes a person when taking: it does not.

Section 2 - The Scenario

The Palm Chamber is a room. Sarissa is a woman in the Palm Chamber.

The Palm Chamber contains a bottle of ink, a quill pen, a tangerine, a bunch of grapes, a length of silken rope, some perfume, a cake of incense, a fitted leather bodice, a sapphire anklet, an illustrated novel, a whip, and a heavy iron key.

A persuasion rule for asking Sarissa to try taking the key:
    say "Sarissa nervously demurs, knowing that it is forbidden.";
    persuasion fails.

A persuasion rule: persuasion succeeds.

Test me with "take all / drop all / sarissa, take all".

Chapter 20: Advanced Text

 Contents of *Writing with Inform*

 Chapter 19: Rulebooks

 Chapter 21: Lists

 Indexes of the examples

§20.1. Changing texts

So far, we have dealt with text as something which comes in little packets: we have printed it out, read it in from the keyboard, and compared it with other text. But we have never tried to open the packets and get at the contents, letter by letter, or to make any alterations, or look for certain combinations of letters. These tricks are surprisingly seldom needed - a surprise, that is, given that everything Inform does is textual - but they are in fact open to us. For example:

```
if character number 1 in "[time of day]" is "1", ...
```

will be true at, for example, 11:30 PM and 1:22 AM, but not at 3:15 PM. What happens here is that Inform expands the time of day into a text, say "11:30 PM", then extracts the first character, say "1", and tests it.

Until 2012, Inform had two kinds of text - plain "text", and "indexed text" - but there's now only "text", which has all of the abilities of both.

 Start of Chapter 20: Advanced Text

 Back to Chapter 19: Rulebooks: §19.16. The Laws for Sorting Rulebooks

 Onward to §20.2. Memory limitations

---

§20.2. Memory limitations

Inform creates "story files" for very small virtual computers (capable of running on phones, for instance) where memory is tight. If we create a number variable and keep on adding 1 to it, the value simply gets bigger. But if we make some text and keep on adding a letter "x" to it, the text takes up more and more space, growing into longer and longer runs of "x"s until there is no more space to hold it.

The following warnings are rather like the tiny print about side-effects on medicine bottles: that is, we mostly ignore them, and if the drugs should kill us, well, at least we have the consolation of knowing we were warned. There are basically three limitations on text:

(1) An amount of memory has to be set aside for text (and other flexible-sized data), and Inform guesses the amount needed. Story files using the Glulx format (see the Settings panel) are able to increase this as necessary in play, so there's no problem if the guess was wrong. But Z-machine story files are stuck with whatever amount of memory was initially chosen.

That choice can be increased with a use option, like so:

```
Use dynamic memory allocation of at least 16384.
```

Inform raises its estimate of the amount needed to ensure that this amount is always at least its own guess, and also at least any amount declared like this. (And then it rounds up to the nearest power of 2, as it happens.) The default value of "dynamic memory allocation" is 8192. In practice, this use option isn't needed much, though, because any story needing large amounts of dynamic memory will likely be on Glulx in any case.

(2) Text has a maximum length. This maximum is normally 1000 characters, which ought to be plenty, but can be raised by sentences such as:

```
Use maximum text length of at least 2000.
```

What happens if this is broken, that is, if we try to use text overrunning this length? The Z-machine may simply crash, so if there is any chance that any single text may grow unpredictably large, Glulx should always be used. On Glulx, overrunning text is truncated safely, except that under Glulx 3.1.0 or better the story file will try to use dynamic memory allocation to expand the limit as needed to avoid truncation. (Testing shows that text is slow to manipulate once it grows beyond about 20,000 characters in length, but this is not really surprising.)

(3) Under the Z-machine, text may only contain characters from the so-called "ZSCII" character set - standard numbers, letters, punctuation marks and the commonest West European accented letters. Anything more exotic is likely to be flattened into a question mark "?". Under Glulx, any character can be used.

All of this makes the Z-machine sound very inferior, for text purposes. But note that Z can handle all of the examples in this chapter perfectly happily.


Start of Chapter 20: Advanced Text


Back to §20.1. Changing texts


Onward to §20.3. Characters, words, punctuated words, unpunctuated words, lines, paragraphs

---

§20.3. Characters, words, punctuated words, unpunctuated words, lines, paragraphs

Inform can get at the contents of text in a variety of ways. The lowest-level is by character - a character is a letter, digit, punctuation symbol, space or other letter-form. (We use the term "character" rather than "letter" because otherwise we would have to call "5" a letter, and so on.) Characters number upwards from 1: character number 1, to repeat that, starts the text. We can get the Nth character with:

character number (number) **in** (text) **... text**

This phrase produces the Nth character from the text, counting from 1. Characters include letters, digits, punctuation symbols, spaces or other letter-forms. Example:

```
character number 8 in "numberless projects of social reform"
```

produces "e". If the index is less than 1 or more than the length of the text, the result is an empty text, "".

The maximum character number varies with the current length of the text, and can be evaluated as:

number of characters in (text) **... number**

This phrase produces the number of characters from the text. Characters include letters, digits, punctuation symbols, spaces or other letter-forms. Examples:

```
number of characters in "War and Peace"
number of characters in ""
```

produce 13 and 0 respectively.

We can also use the adjective "empty":

```
if the description of the location is empty, ...
```

The empty text, "", is the only one with 0 characters.

We can also extract the contents by word, again numbered from 1. Thus:

word number (number) **in** (text) **... text**

This phrase produces the Nth word from the text, counting from 1. Words for this purpose are what's left after breaking the text up at punctuation or spacing (spaces, line breaks, paragraph breaks) and then removing that punctuation or spacing. Example:

```
word number 3 in "ice-hot, don't you think?"
```

produces "don't". If the index is less than 1 or more than the number of words in the text, the result is an empty text, "".

number of words in (text) **... number**

This phrase produces the number of words from the text. Words for this purpose are what's left after breaking the text up at punctuation or spacing (spaces, line breaks, paragraph breaks) and then removing that punctuation or spacing. Example:

```
number of words in "ice-hot, don't you think?"
```

produces 5.

Note that the contraction apostrophe in "don't" doesn't count as punctuation. Because this is not always quite what we want, Inform offers two variations:

punctuated word number (number) **in** (text) **... text**

This phrase produces the Nth word from the text, counting from 1. Words for this purpose are what's left after breaking the text up at punctuation or spacing (spaces, line breaks, paragraph breaks) and then removing the spacing, but leaving the punctuation as independent words. Example:

```
punctuated word number 2 in "ice-hot, don't you think?"
```

produces "-". The punctuated words here are "ice", "-", "hot", ",", "don't", "you", "think", "?". If two or more punctuation marks are adjacent, they are counted as different words, except for runs of dashes or periods: thus ",," has two punctuated words, but "--" and "..." have only one each. If the index is less than 1 or more than the number of punctuated words in the text, the result is an empty text, "".

number of punctuated words in (text) **... number**

This phrase produces the number of words from the text. Words for this purpose are what's left after breaking the text up at punctuation or spacing (spaces, line breaks, paragraph breaks) and then removing the spacing, but leaving the punctuation as independent words. Example:

```
number of punctuated words in "ice-hot, don't you think?"
```

produces 8; see if you can find them all.

unpunctuated word number (number) **in** (text) **... text**

This phrase produces the Nth word from the text, counting from 1. Words for this purpose are what's left after breaking the text up at spacing (spaces, line breaks, paragraph breaks) but including all punctuation as if it were part of the spelling of the words it joins to. Example:

```
unpunctuated word number 1 in "ice-hot, don't you think?"
```

produces "ice-hot,". The unpunctuated words in "ice-hot, don't you think?" are "ice-hot,", "don't", "you", "think?". If the index is less than 1 or more than the number of punctuated words in the text, the result is an empty text, "".

number of unpunctuated words in (text) **... number**

This phrase produces the number of words from the text. Words for this purpose are what's left after breaking the text up at spacing (spaces, line breaks, paragraph breaks) but including all punctuation as if it were part of the spelling of the words it joins to. Example:

```
number of unpunctuated words in "ice-hot, don't you think?"
```

produces just 4.

Finally, on the larger scale still, we also have:

line number (number) **in** (text) **... text**

This phrase produces the Nth line from the text, counting from 1. Unless explicit use is made of line-breaking, lines and paragraphs will be the same - it doesn't refer to lines as visible on screen, because we have no way of knowing what size screen the player might have.

number of lines in (text) **... number**

This phrase produces the number of lines in the text. Unless explicit use is made of line-breaking, lines and paragraphs will be the same - it doesn't refer to lines as visible on screen, because we have no way of knowing what size screen the player might have. Example: the number of lines in

```
"Sensational news just in![paragraph break]The Martians have invaded
Miranda.[line break](One of the moons of Uranus, that is.)"
```

is 3.

paragraph number (number) **in** (text) **... text**

This phrase produces the Nth paragraph from the text, counting from 1.

number of paragraphs in (text) **... number**

This phrase produces the number of paragraphs in the text. Example: the number of paragraphs in

> "Sensational news just in![paragraph break]The Martians have invaded Miranda.[line break](One of the moons of Uranus, that is.)"

is 2.

(Attempting to make large enough texts to have a serious paragraph count is slightly risky if there is not much memory to play with, as on the Z-machine. But the facilities do exist.)

Start of Chapter 20: Advanced Text

Back to §20.2. Memory limitations

Onward to §20.4. Upper and lower case letters

---

§20.4. Upper and lower case letters

In most European languages the same letters can appear in two forms: as capitals, like "X", mainly used to mark a name or the start of a sentence; or in their ordinary less prominent form, like "x". These forms are called upper and lower case because, historically, typesetters kept lead castings of letters in two wooden cases, one above the other on the workbench. Lower case letters were in the lower box closer to hand, being more often needed.

Human languages are complicated. Not every lower case letter has an upper case partner: ordinal markers in Hispanic languages don't, for instance, and the German "ß" is never used in upper case. Sometimes two different lower case letters have the same upper case form: "ς" and "σ", two versions of the Greek sigma, both capitalise to "Σ". Inform follows the international Unicode standard in coping with all this.

We can test whether text is in either case like so:

if (text) **is in lower case:**

This condition is true if every character in the text is a lower case letter. Examples: this is true for "wax", but false for "wax seal" or "eZ mOnEy".

if (text) **is in upper case:**

This condition is true if every character in the text is in upper case. Examples: this is true for "BEESWAX", but false for "ROOM 101".

We can change the casing of text using:

(text) **in lower case ... text**

This phrase produces a new version of the given text, but with all upper case letters reduced to lower case. Example: "a ticket to Tromsø via Østfold" becomes

"a ticket to tromsø via østfold"

(text) **in upper case ... text**

This phrase produces a new version of the given text, but with all upper case letters reduced to lower case. Example: "a ticket to Tromsø via Østfold" becomes

"A TICKET TO TROMSØ VIA ØSTFOLD"

(text) **in title case ... text**

This phrase produces a new version of the given text, but with casing of words changed to title casing: this capitalises the first letter of each word, and lowers the rest. Example: "a ticket to Tromsø via Østfold" becomes

"A Ticket To Tromsø Via Østfold"

(text) **in sentence case ... text**

This phrase produces a new version of the given text, but with casing of words changed to sentence casing: this capitalises the first letter of each sentence and reduces the rest to lower case. Example: "a ticket to Tromsø via Østfold" becomes

"A ticket to tromsø via østfold"

Accents are preserved in case changes. So (if we are using Glulx and have Unicode available) title case can turn Aristophanes' discomfortingly lower-case lines

```
ἐξ οὗ γὰρ ἡμᾶς προὔδοσαν μιλήσιοι,
οὐκ εἶδον οὐδ᾽ ὅλισβον ὀκτωδάκτυλον,
ὃς ἦν ἂν ἡμῖν σκυτίνη "πικουρία
```

by raising them proudly up like so:

```
Ἐξ Οὗ Γὰρ Ἡμᾶς Προὔδοσαν Μιλήσιοι,
Οὐκ Εἶδον Οὐδ᾽ Ὅλισβον Ὀκτωδάκτυλον,
Ὃς Ἦν Ἂν Ἡμῖν Σκυτίνη "Πικουρία.
```

Title and sentence casing can only be approximate if done by computer. Inform looks at the letters, but is blind to the words and sentences they make up. (Note the way sentence casing did not realise "Tromsø" and "Østfold" were proper nouns.) If asked to put the name "MCKAY" into title casing, Inform will opt for "Mckay", not recognising this as the Scottish patronymic surname "McKay". Given "baym dnieper", the title of David Bergelson's great Yiddish novel of 1932, it will opt for "BAYM DNIEPER": but properly speaking Yiddish does not have upper case lettering at all, though nowadays it is sometimes printed as if it did. And conventions are very variable about which words should be capitalised in titles: English publishers mostly agree that connectives, articles and prepositions should be in lower case, but in France almost anything goes, with Académie Française rules giving way to avant-garde book design. In short, we cannot rely on Inform's title casing to produce a result which a human reader will always think perfect.

This discussion has all been about how Inform prints, not about how it reads commands from the keyboard, because the latter is done case-insensitively. The virtual machines for which Inform creates programs normally flatten all command input to lower case, and in any case Understand comparison ignores casing. Thus

```
Understand "mckay" as the Highland Piper.
```

means that "examine McKay", "examine MCKAY", "examine mckay", and so forth are all equivalent. The text of the player's command probably doesn't preserve the original casing typed in any event.

One more caution, though it will affect hardly anyone. For projects using the Z-machine, only a restricted character set is available in texts: for more, we must use Glulx. A mad anomaly of ZSCII, the Z-machine character set, is that it contains the lower case letter "ÿ" but not its upper case form "Ÿ", so that

```
"ÿ" in upper case
```

produces "Ÿ" in Glulx but "ÿ" in the Z-machine. This will come as a blow to Queensrÿche fans, but in all other respects any result on the Z-machine should agree with its counterpart on Glulx.

Start of Chapter 20: Advanced Text

Back to §20.3. Characters, words, punctuated words, unpunctuated words, lines, paragraphs

Onward to §20.5. Matching and exactly matching

Example 412: ★ **Capital City** To arrange that the location information normally given on the left-hand side of the status line appears in block capitals.

Example 413: ★ **Rocket Man** Using case changes on any text produced by a "to say..." phrase.

---

§20.5. Matching and exactly matching

Up to now, we have only been able to judge two texts by seeing if they are equal, but we can now ask more subtle questions.

if (text) **matches the text** (text)**:**

This condition is true if the second text occurs anywhere inside the first. Examples:

```
if "[score]" matches the text "3", ...
```

tests whether the digit 3 occurs anywhere in the score, as printed out; and

```
if the printed name of the location matches the text "the", ...
```

tests to see whether "the" can be found anywhere in the current room's name. Note that the location "Smotheringly Hot Jungle" would pass this test - it's there if you look. On the other hand, "The Orangery" would not, because "The" does not match against "the". We can get around this in a variety of ways, one of which is to tell Inform to be insensitive to the case (upper or lower) of letters:

```
if the printed name of the location matches the text "the", case insensitively: ...
```

if (text) **exactly matches the text** (text)**:**

This condition is true if the second text matches the first, starting at the beginning and finishing at the end. This appears to be the same as testing if one is equal to the other, but that's not quite true: for example,

```
if "[score]" exactly matches the text "[best score]", ...
```

is true if the score and best score currently print out as the same text, which will be true if they are currently equal as numbers; but

```
if "[score]" is "[best score]", ...
```

is never true - these are different texts, even if they sometimes look the same.

In the next section we shall see that "matches" and "exactly matches" can do much more than the simple text matching demonstrated above.

We can also see how many times something matches:

number of times (text) **matches the text** (text) **... number**

This produces the number of times the second text occurs within the first. The matches are not allowed to overlap. Example:

```
number of times "pell-mell sally" matches the text "ll" = 3
number of times "xyzzy" matches the text "Z" = 0
number of times "xyzzy" matches the text "Z", case insensitively = 2
number of times "aaaaaaaa" matches the text "aaaa" = 2
```

There's no "number of times WHATEVER exactly matches the text FIND" phrase since this is by definition going to have to be 0 or 1.

Start of Chapter 20: Advanced Text

Back to §20.4. Upper and lower case letters

Onward to §20.6. Regular expression matching

---

§20.6. Regular expression matching

When playing around with text, we tend to get into longer and trickier wrangles of matching - we find that we want to look not for simple text like "gold", but for "gold" used only as a separate word, or for a date in YYYY-MM-DD format, or for a seemingly endless range of other possibilities. What we need is not just for Inform to provide a highly flexible matching program, but also a good notation in which to describe what we want.

Fortunately, such a notation already exists. This is the "regular expression" notation, named for a 1950s mathematical model by the logician Stephen Kleene, applied to computing in the late 60s by Ken Thompson, borrowed almost at once by the early Unix tools of the 70s, and developed further by Henry Spencer in the 80s and Philip Hazel in the 90s. The glue holding the Internet together - the Apache web-server, the scripting languages Perl and Python, and so forth - makes indispensable use of regular expressions.

As might be expected from the previous section, we simply have to describe the FIND text as "regular expression" rather than "text" and then the same facilities are available:

if (text) **matches the regular expression** (text)**:**

This condition is true if any contiguous part of the text can be matched against the given regular expression. Examples:

```
if "taramasalata" matches the regular expression "a.*l", ...
```

is true, since this looks for a part of "taramasalata" which begins with "a", continues with any number of characters, and finishes with "l"; so it matches "aramasal". (Not "asal", because it gets the makes the leftmost match it can.) The option "case insensitively" causes lower and upper case letters to be treated as equivalent.

if (text) **exactly matches the regular expression** (text)**:**

This condition is true if the whole text (starting from the beginning and finishing at the end) can be matched against the given regular expression. The option "case insensitively" causes lower and upper case letters to be treated as equivalent.

And once again:

number of times (text) **matches the regular expression** (text) **... number**

This produces the number of times that contiguous pieces of the text can be matched against the regular expression, without allowing them to overlap.

Since a regular expression can match quite a variety of possibilities (for instance "b\w+t" could match "boast", "boat", "bonnet" and so on), it's sometimes useful to find what the match actually was:

text matching regular expression ... text

This phrase is only meaningful immediately after a successful match of a regular expression against text, and it produces the text which matched. Example:

```
if "taramasalata" matches the regular expression "m.*l":
    say "[text matching regular expression].";
```

says "masal."

Perhaps fairly, perhaps not, regular expressions have a reputation for being inscrutable. The basic idea is that although alphanumeric characters (letters, numbers and spaces) mean just what they look like, punctuation characters are commands with sometimes dramatic effects. Thus:

```
if WHATEVER matches the regular expression "fish", ...
if WHATEVER matches the regular expression "f.*h", ...
```

behave very differently. The first is just like matching the text "fish", but the second matches on any sequence of characters starting with an "f" and ending with an "h". This is not at all obvious at first sight: reading regular expressions is a skill which must be learned, like reading a musical score. A really complex regular expression can look like a soup of punctuation and even an expert will blink for a few minutes before telling you what it does - but a beginner can pick up the basics very quickly. Newcomers might like to try out and become comfortable with the features a few at a time, reading down the following list.

1. Golden rule. Don't try to remember all the characters with weird effects. Instead, if you actually mean any symbol other than a letter, digit or space to be taken literally, place a backslash "\" in front of it. For instance, matching the regular expression

```
"\*A\* of the Galactic Patrol"
```

is the same as matching the text "*A* of the Galactic Patrol", because the asterisks are robbed of their normal powers. This includes backslash itself: "\\" means a literal backslash. (Don't backslash letters or digits - that turns out to have a meaning all its own, but anyway, there is never any need.)

2. Alternatives. The vertical stroke "|" - not a letter I or L, nor the digit 1 - divides alternatives. Thus

```
"the fish|fowl|crawling thing"
```

is the same as saying match "the fish", or "fowl", or "crawling thing".

3. Dividing with brackets. Round brackets "(" and ")" group parts of the expression together.

```
"the (fish|fowl|crawling thing) in question"
```

is the same as saying match "the fish in question", or "the fowl in question", or "the crawling thing in question". Note that the "|" ranges outwards only as far as the group it is in.

4. Any character. The period "." means any single character. So

```
"a...z"
```

matches on any sequence of five characters so long as the first is "a" and the last is "z".

5. Character alternatives. The angle brackets "<" and ">" are a more concise way of specifying alternatives for a single character. Thus

```
"b<aeiou>b"
```

matches on "bab", "beb", "bib", "bob" or "bub", but not "baob" or "beeb" - any single character within the angle brackets is accepted. Beginning the range with "^" means "any single character so long as it is not one of these": thus

```
"b<^aeiou>b"
```

matches on "blb" but not "bab", "beb", etc., nor on "blob" or "bb". Because long runs like this can be a little tiresome, we are also allowed to use "-" to indicate whole ranges. Thus

```
"b<a-z>b"
```

matches a "b", then any lower case English letter, then another "b".

In traditional regular expression language, square brackets rather than angle brackets are used for character ranges. In fact Inform does understand this notation if there are actual square brackets "[" and "]" in the pattern text, but in practice this would be tiresome to achieve, since Inform uses those to achieve text substitutions. So Inform allows "b<a-z>b" rather than making us type something like

```
"b[bracket]a-z[close bracket]b"
```

to create the text "b[a-z]b".

6. Popular character ranges. The range "<0-9>", matching any decimal digit, is needed so often that it has an abbreviation: "\d". Thus

```
"\d\d\d\d-\d\d-\d\d"
```

matches, say, "2006-12-03". Similarly, "\s" means "any spacing character" - a space, tab or line break. "\p" is a punctuation character, in the same sense used for word division in the previous section: it actually matches any of

```
. , ! ? - / " : ; ( ) [ ] { }
```

"\w" means "any character appearing in a word", and Inform defines it as anything not matching "\s" or "\p".

"\l" and "\u" match lower and upper case letters, respectively. These are much stronger than "<a-z>" and "<A-Z>", since they use the complete definition in the Unicode 4.0.0 standard, so that letter-forms from all languages are catered for: for example "δ" matches "\l" and "Δ" matches "\u".

The reverse of these is achieved by capitalising the letter. So "\D" means "anything not a digit", "\P" means "anything not punctuation", "\W" means "anything not a word character", "\L" means "anything not a lower case letter" and so on.

7. Positional restrictions. The notation "^" does not match anything, as such, but instead requires that we be positioned at the start of the text. Thus

```
"^fish"
```

matches only "fish" at the start of the text, not occurring anywhere later on. Similarly, "$" requires that the position be the end of the text. So

```
"fish$"
```

matches only if the last four characters are "fish". Matching "^fish$" is the same thing as what Inform calls exactly matching "fish".

Another useful notation is "\b", which matches a word boundary: that is, it matches no actual text, but requires the position to be a junction between a word character and a non-word character (a "\w" and a "\W") or vice versa. Thus

```
"\bfish\b"
```

matches "fish" in "some fish" and also "some fish, please!", but not in "shellfish". (The regular expression "\w*fish\b" catches all words ending in "fish", as we will see below.) As usual, the capitalised version "\B" negates this, and means "not at a word boundary".

8. Line break and tab. The notations "\n" and "\t" are used for a line break ("n" for "new line") and tab, respectively. Tabs normally do not occur in Inform strings, but can do when reading from files. It makes no sense to reverse these, so "\N" and "\T" produce errors.

9. Repetition. Placing a number in braces "{" and "}" after something says that it should be repeated that many times. Thus

```
"ax{25}"
```

matches only on "axxxxxxxxxxxxxxxxxxxxxxxxx". More usefully, perhaps, we can specify a range of the number of repetitions:

```
"ax{2,6}"
```

matches only on "axx", "axxx", "axxxx", "axxxxx", "axxxxxx". And we can leave the top end open: "ax{2,}" means "a" followed by at least two "x"s.

Note that the braces attach only to most recent thing - so "ax{2}" means "a" followed by two of "x" - but, as always, we can use grouping brackets to change that. So "(ax){2,}" matches "axax", "axaxax", "axaxaxax",...

(It's probably best not to use Inform to try to match the human genome against "<acgt>{3000000000}", but one of the most important practical uses of regular expression matching in science is in treating DNA as a string of nucleotides represented by the letters "a", "c", "g", "t", and looking for patterns.)

10. Popular repetitions. Three cases are so often needed that they have standard short forms:

"{0,1}", which means 0 or 1 repetition of something - in other words, doesn't so much repeat it as make it optional - is written "?". Thus "ax?y" matches only on "ay" or "axy".

"{0,}", which means 0 or more repetitions - in other words, any number at all - is written "*". Thus "ax*y" matches on "ay", "axy", "axxy", "axxxy", ... and the omnivorous ".*" - which means "anything, any number of times" - matches absolutely every text. (Perhaps unexpectedly, replacing ".*" in a text with "X" will produce "XX", not "X", because the ".*" first matches the text, then matches the empty gap at the end. To match the entire text just once, try "^.*$".)

"{1,}", which means 1 or more repetitions, is written "+". So "\d+" matches any run of digits, for instance.

11. Greedy vs lazy. Once we allow things to repeat an unknown number of times, we run into an ambiguity. Sure, "\d+" matches the text "16339b". But does it look only as far as the "1", then reason that it now has one or more digits in a row, and stop? Or does it run onward devouring digits until it can do so no longer, so matching the "16339" part? These two strategies are called "lazy" and "greedy" respectively.

Do we care? Well, the strategy used makes no difference to whether there is a match, but it does affect what part of the text is matched, and the number of matches there are. Unless we mark for it, all repetitions are greedy. Usually this is good, but it means that, for instance,

```
"-.+-"
```

applied to "-alpha- -beta- -gamma-" will match the whole text, because ".+" picks up all of "alpha- -beta- -gamma". To get around this, we can mark any of the repetition operators as lazy by adding a question mark "?". Thus:

```
"-.+?-"
```

applied to "-alpha- -beta- -gamma-" matches three times, producing "-alpha-" then "-beta-" then "-gamma-".

A logical but sometimes confusing consequence is that a doubled question mark "??" means "repeat 0 or 1 times, but prefer 0 matches to 1 if both are possibilities": whereas a single question mark "?", being greedy, means "repeat 0 or 1 times, but prefer 1 match to 0 if both are possibilities".

12. Numbered groups. We have already seen that round brackets are useful to clump together parts of the regular expression - to choose within them, or repeat them. In fact, Inform numbers these from 1 upwards as they are used from left to right, and we can subsequently refer back to their contents with the notation "\1", "\2", ... After a successful match, we can find the results of these subexpressions with:

> **text matching subexpression** (number) **... text**
>
> This phrase is only meaningful immediately after a successful match of a regular expression against text, and it produces the text which matched. The number must be from 1 to 9, and must correspond to one of the bracketed groups in the expression just matched. Example: after
>
> ```
> if "taramasalata" matches the regular expression "a(r.*l)a(.)":
> ```
>
> the "text matching regular expression" is "aramasalat", the "text matching subexpression 1" is "ramasal", and "text matching subexpression 2" is "t".

For instance:

"(\w)\w*\1"

matches any run of two or more word-characters, subject to the restriction that the last one has to be the same as the first - so it matches "xerox" but not "alphabet". When Inform matches this against "xerox", first it matches the initial "x" against the group "(\w)". It then matches "\w*" ("any number of word-characters") against "ero", so that the "*" runs up to 3 repetitions. It then matches "\1" against the final "x", because "\1" requires it to match against whatever last matched in sub-expression 1 - which was an "x".

Numbered groups allow wicked tricks in matching, it's true, but really come into their own when it comes to replacing - as we shall see.

13. Switching case sensitivity on and off. The special notations "(?i)" and "(?-i)" switch sensitivity to upper vs. lower case off and on, mid-expression. Thus "a(?i)bcd(?-i)e" matches "abcde", "aBcDe", etc., but not "Abcde" or "abcdE".

14. Groups with special meanings. This is the last of the special syntaxes: but it's a doozy. A round-bracketed group can be marked to behave in a special way by following the open bracket by a symbol with a special meaning. Groups like this have no number and are not counted as part of \1, \2, and so forth - they are intended not to gather up material but to have some effect of their own.

"(# ...)"

Is a comment, that is, causes the group to do nothing and match against anything.

"(?= ...)"

Is a lookahead: it is a form of positional requirement, like "\b" or "^", but one which requires that the text ahead of us matches whatever is in the brackets. (It doesn't consume that text - only checks to see that it's there.) For instance "\w+(?=;)" matches a word followed by a semicolon, but does not match the semicolon itself.

"(?! ...)"

Is the same but negated: it requires that the text ahead of us does not match the material given. For instance, "a+(?!z)" matches any run of "a"s not followed by a "z".

"(?<= ...)" and "(?<! ...)"

Are the same but looking behind (hence the "<"), not forward. These are restricted to cases where Inform can determine that the material to be matched has a definite known width. For instance, "(?<!shell)fish" matches any "fish" not occurring in "shellfish".

"(> ...)"

Is a possessive, that is, causes the material to be matched and, once matched, never lets go. No matter what subsequently turns out to be convenient, it will never change its match. For instance, "\d+8" matches against "768" because Inform realises that "\d+" cannot be allowed to eat the "8" if there is to be a match, and stops it. But "(>\d+)8" does not match against

"768" because now the "\d+", which initially eats "768", is possessive and refuses to give up the "8" once taken.

"(?(1)...)" and "(?(1)...|...)"

Are conditionals. These require us to match the material given if \1 has successfully matched already; in the second version, the material after the "|" must be matched if \1 has not successfully matched yet. And the same for 2, 3, ..., 9, of course.

Finally, conditionals can also use lookaheads or lookbehinds as their conditions. So for instance:

"(?(?=\d)\d\d\d\d|AY-\d\d\d\d)"

means if you start with a digit, match four digits; otherwise match "AY-" followed by four digits. There are easier ways to do this, of course, but the really juicy uses of conditionals are only borderline legible and make poor examples - perhaps this is telling us something.

Start of Chapter 20: Advanced Text

Back to §20.5. Matching and exactly matching

Onward to §20.7. Making new text with text substitutions

Example 414: **Alpha** Creating a beta-testing command that matches any line starting with punctuation.

Example 415: **About Inform's regular expression support** Some footnotes on Inform's regular expressions, and how they compare to those of other programming languages.

---

§20.7. Making new text with text substitutions

Substitutions are most often used just for printing, like so:

```
say "The clock reads [time of day].";
```

But they can also produce text which can be stored up or used in other ways. For example, defining

```
To decide what text is (T - text) doubled:
    decide on "[T][T]".
```

makes

```
let the Gerard Kenny reference be "NewYork" doubled;
```

set this temporary variable to "NewYorkNewYork".

There is, however, a subtlety here. A text with a substitution in it, like:

"The clock reads [time of day]."

is always waiting to be substituted, that is, to become something like:

```
"The clock reads 11:12 AM."
```

If all we do with text is to print it, there's nothing to worry about. But if we're storing it up, especially for multiple turns, there are ambiguities. For example, suppose we're changing the look of the black status line bar at the top of the text window:

```
now the left hand status line is "[time of day]";
```

Just copying "[time of day]" to the "left hand status line" variable doesn't make it substitute - which is just as well, or the top of the screen would perpetually show "9:00 AM".

On the other hand, looking back at the phrase example:

```
To decide what text is (T - text) doubled:
    decide on "[T][T]".
```

"[T][T]" is substituted immediately it's formed. That's also a good thing, because "T" loses its meaning the moment the phrase finishes, which would make "[T][T]" meaningless anywhere else.

What's going on here is this: Inform substitutes text immediately if it contains references to a temporary value such as "T", and otherwise only if it needs to access the contents. This is why "[time of day]" isn't substituted until we need to print it out (or, say, access the third character): "time of day" is a value which always exists, not a temporary one.

Using the adjectives "substituted" and "unsubstituted", it's always possible to test whether a given text is in either state, should this ever be useful. For example,

```
now the left hand status line is "[time of day]";
if the left hand status line is unsubstituted, say "Yes!";
```

will say "Yes!": the LHSL is like a bomb waiting to go off. Speaking of which:

```
The player is holding a temporal bomb.

When play begins:
    now the left hand status line is "Clock reads: [time of day]".

After dropping the temporal bomb:
    now the left hand status line is the substituted form of the left hand status line;
    say "Time itself is now broken. Well done."
```

This is making use of:

substituted form of (text) **... text**

This takes a text and makes substitution occur immediately. For example,

```
substituted form of "time of death, [time of day]"
```

produces something like "time of death, 9:15 AM" rather than "time of death, [time of day]". It's entirely legal to apply this to text which never had any substitutions in, so

```
substituted form of "balloon"
```

produces "balloon".

Note that there's no analogous phrase for "unsubstituted form of...", because once text has substituted, there's no way to go back.

Start of Chapter 20: Advanced Text

Back to §20.6. Regular expression matching

Onward to §20.8. Replacements

Example 416: ★ **Identity Theft** Allowing the player to enter a name to be used for the player character during the game.

Example 417: ★ **Mirror, Mirror** The sorcerer's mirror can, when held up high, form an impression of its surroundings which it then preserves.

Example 418: ★★ **The Cow Exonerated** Creating a class of matches that burn for a time and then go out, with elegant reporting when several matches go out at once.

---

§20.8. Replacements

Suppose V is a text which varies - perhaps a property of something, or a variable defined everywhere, or a temporary "let"-named value. How do we change its contents? The easiest way is simply to assign text to it. Thus:

```
let V be "It is now [the time of the day in words]."
```

And, for instance,

```
let V be "[V]!"
```

adds an exclamation mark at the end of V.

Otherwise, it is more useful (also a little faster) to modify V by changing its characters, words and so on. Thus:

replace character number (number) **in** (text) **with** (text)

This phrase acts on the named text by placing the given text in place of the Nth character, counting from 1. Example:

```
let V be "mope";
replace character number 3 in V with "lecul";
```

```
say V;
```

says "molecule".

replace word number (number) **in** (text) **with** (text)

This phrase acts on the named text by placing the given text in place of the Nth word, counting from 1, and dividing words at spacing or punctuation. Example:

```
let V be "Does the well run dry?";
replace word number 3 in V with "jogger";
say V;
```

says "Does the jogger run dry?".

replace punctuated word number (number) **in** (text) **with** (text)

This phrase acts on the named text by placing the given text in place of the Nth word, counting from 1, and dividing words at spacing, counting punctuation runs as words in their own right. Example:

```
let V be "Frankly, yes, I agree.";
replace punctuated word number 2 in V with ":";
say V;
```

says "Frankly: yes, I agree.".

replace unpunctuated word number (number) **in** (text) **with** (text)

This phrase acts on the named text by placing the given text in place of the Nth word, counting from 1, and dividing words at spacing, counting punctuation as part of a word just as if it were lettering. Example:

```
let V be "Frankly, yes, I agree.";
replace unpunctuated word number 2 in V with "of course";
say V;
```

says "Frankly, of course I agree.".

replace line number (number) **in** (text) **with** (text)

This phrase acts on the named text by placing the given text in place of the Nth line, counting from 1. Lines are divided by paragraph or line breaks.

replace paragraph number (number) **in** (text) **with** (text)

This phrase acts on the named text by placing the given text in place of the Nth paragraph, counting from 1.

Last, but not least, we can replace text wherever it occurs:

replace the text (text) **in** (text) **with** (text)

This phrase acts on the named text by searching and replacing, as many non-overlapping times as possible. Example:

```
replace the text "a" in V with "z"
```

changes every lower-case "a" to "z": the same thing done with the "case insensitively" option would change each "a" or "A" to "z".

All very well for letters, but it can be unfortunate to try

```
replace the text "Bob" in V with "Robert"
```

if V happens to contain, say "The Olympic Bobsleigh Team": it would become "The Olympic Robertsleigh Team". What we want, of course, is for Bob to become Robert only when it's a whole word. We can get that with:

replace the word (text) **in** (text) **with** (text)

This phrase acts on the named text by searching and replacing, as many non-overlapping times as possible, where the search text must occur as a whole word. Example:

```
replace the word "Bob" in V with "Robert"
```

changes "Bob got on the Bobsleigh" to "Robert got on the Bobsleigh".

replace the punctuated word (text) **in** (text) **with** (text)

This phrase acts on the named text by searching and replacing, as many non-overlapping times as possible, where the search text must occur as a whole word or

run of punctuation.

But these are all just special cases of the grand-daddy of all replacement phrases:

replace the regular expression (text) **in** (text) **with** (text)

This phrase acts on the named text by matching the regular expression and replacing anything which fits it, as many non-overlapping times as possible. Example:

```
replace the regular expression "\d+" in V with "..."
```

changes "The Battle of Waterloo, 1815, rivalled Trafalgar, 1805" to "The Battle of Waterloo, ..., rivalled Trafalgar, ...". The "case insensitively" causes lower and upper case letters to be treated as if the same letter.

When replacing a regular expression, the replacement text also has a few special meanings (though, thankfully, many fewer than for the expression itself). Once again "\n" and "\t" can be used for line break and tab characters, and "\\" must be used for an actual backslash. But, very usefully, "\1" to "\9" expand as the contents of groups numbered 1 to 9, and "\0" to the exact text matched. So:

```
replace the regular expression "\d+" in V with "roughly \0"
```

adds the word "roughly" in front of any run of digits in V, because \0 becomes in turn whichever run of digits matched. And

```
replace the regular expression "(\w+) (.*)" in V with "\2, \1"
```

performs the transformation "Frank Booth" to "Booth, Frank".

Finally, prefixing the number by "l" or "u" forces the text it represents into lower or upper case, respectively. For instance:

```
replace the regular expression "\b(\w)(\w*)" in X with "\u1\l2";
```

changes the casing of X to "title casing", where each individual word is capitalised. (This is a little slow on large texts, since so many matches and replacements are made: it's more efficient to use the official phrases for changing case.)

Start of Chapter 20: Advanced Text

Back to §20.7. Making new text with text substitutions

Onward to §20.9. Summary of regular expression notation

Example 419: ★ **Blackout** Filtering the names of rooms printed while in darkness.

Example 420: ★ **Fido** A dog the player can name and un-name at will.

Example 421: ★ **Igpay Atinlay** A pig Latin filter for the player's commands.

Example 422: ★★ **Mr. Burns' Repast** Letting the player guess types for an unidentifiable fish.

Example 423: ★★ **Northstar** Making Inform understand ASK JOSH TO TAKE INVENTORY as JOSH, TAKE INVENTORY. This requires us to use a regular expression on the player's command, replacing some of the content.

Example 424: ★★★ **Cave-troll** Determining that the command the player typed is invalid, editing it, and re-examining it to see whether it now reads correctly.

---

§20.9. Summary of regular expression notation

MATCHING

Positional restrictions

```
^  Matches (accepting no text) only at the start of the text
$  Matches (accepting no text) only at the end of the text
\b Word boundary: matches at either end of text or between a \w and a \W
\B Matches anywhere where \b does not match
```

Backslashed character classes

```
\char If char is other than a-z, A-Z, 0-9 or space, matches that literal char
\\     For example, this matches literal backslash "\"
\n     Matches literal line break character
\t     Matches literal tab character (but use this only with external files)


\d Matches any single digit
\l  Matches any lower case letter (by Unicode 4.0.0 definition)
\p Matches any single punctuation mark: . , ! ? - / " : ; ( ) [ ] { }
\s Matches any single spacing character (space, line break, tab)
\u Matches any upper case letter (by Unicode 4.0.0 definition)
\w Matches any single word character (neither \p nor \s)


\D  Matches any single non-digit
\L  Matches any non-lower-case-letter
\P  Matches any single non-punctuation-mark
\S  Matches any single non-spacing-character
\U  Matches any non-upper-case-letter
\W Matches any single non-word-character (i.e., matches either \p or \s)
```

Other character classes

```
.       Matches any single character
<...>  Character range: matches any single character inside
<^...> Negated character range: matches any single character not inside
```

Inside a character range

e-h Any character in the run "e" to "h" inclusive (and so on for other runs)
>... Starting with ">" means that a literal close angle bracket is included
\ Backslash has the same meaning as for backslashed character classes: see above

Structural

| Divides alternatives: "fish|fowl" matches either
(?i) Always matches: switches to case-insensitive matching from here on
(?-i) Always matches: switches to case-sensitive matching from here on

Repetitions

...? Matches "..." either 0 or 1 times, i.e., makes "..." optional
...* Matches "..." 0 or more times: e.g. "\s*" matches an optional run of space
...+ Matches "..." 1 or more times: e.g. "x+" matches any run of "x"s
...{6} Matches "..." exactly 6 times (similarly for other numbers, of course)
...{2,5} Matches "..." between 2 and 5 times
...{3,} Matches "..." 3 or more times
....? "?" after any repetition makes it "lazy", matching as few repeats as it can

Numbered subexpressions

(...) Groups part of the expression together: matches if the interior matches
\1 Matches the contents of the 1st subexpression reading left to right
\2 Matches the contents of the 2nd, and so on up to "\9" (but no further)

Unnumbered subexpressions

(# ...) Comment: always matches, and the contents are ignored
(?= ...) Lookahead: matches if the text ahead matches "...", but doesn't consume it
(?! ...) Negated lookahead: matches if lookahead fails
(?<= ...) Lookbehind: matches if the text behind matches "...", but doesn't consume it
(?<! ...) Negated lookbehind: matches if lookbehind fails
(> ...) Possessive: tries to match "..." and if it succeeds, never backtracks on this
(?(1)...) Conditional: if \1 has matched by now, require that "..." be matched
(?(1)...|...) Conditional: ditto, but if \1 has not matched, require the second part
(?(?=...)...|...) Conditional with lookahead as its condition for which to match
(?(?<=...)...|...) Conditional with lookbehind as its condition for which to match

IN REPLACEMENT TEXT

\char If char is other than a-z, A-Z, 0-9 or space, expands to that literal char
\\ In particular, "\\" expands to a literal backslash "\"
\n Expands to a line break character
\t Expands to a tab character (but use this only with external files)
\0 Expands to the full text matched
\1 Expands to whatever the 1st bracketed subexpression matched
\2 Expands to whatever the 2nd matched, and so on up to "\9" (but no further)
\l0 Expands to \0 converted to lower case (and so on for "\l1" to "\l9")
\u0 Expands to \0 converted to upper case (and so on for "\u1" to "\u9")


Start of Chapter 20: Advanced Text


Back to §20.8. Replacements


Onward to Chapter 21: Lists: §21.1. Lists and entries

Examples from Chapter 20: Advanced Text

Start of this chapter
Chapter 21: Lists
Indexes of the examples

412 **Example Capital City** *RB*

To arrange that the location information normally given on the left-hand side of the status line appears in block capitals.

Not much is needed for this. The only noteworthy point is that it doesn't work by changing the LHSL to "[the player's surroundings in upper case]": it cannot do this because "the player's surroundings" is not a value. Instead, "[the player's surroundings]" is a text substitution sometimes printing the name of a room, sometimes printing "Darkness", and so on. We must therefore load it into a text first, and then apply "...in upper case".

```
"Capital City"

Capital City is a room. East is Lower Caissons. South of Lower Caissons
is San Seriffe. East of San Seriffe is a dark room.

To say the player's capitalised surroundings:
    let the masthead be "[the player's surroundings]" in upper case;
    say the masthead.

When play begins:
    now the left hand status line is "[the player's capitalised surroundings]".

Test me with "e / s / e".
```

---

413 **Example Rocket Man** *RB*

Using case changes on any text produced by a "to say..." phrase.

We can now change the case of any text produced by a "to say..." phrase. This is often useful when we would like to make use of a standard say phrase in some new context. Say, for instance, that we would like to "[is-are the list...]" in a context that needs the first letter to be capitalized.

We could write a new say phrase, such as "to say is-are the list of (N - a description of objects) in sentence capitalization"; but there is an easier way, and that is to set a text variable to the output of the to say phrase, and then print that text in the case of our choice.

For example:

```
"Rocket Man"
```

```
Instead of going somewhere from the spaceport when the player carries
something:
    let N be "[is-are the list of things carried by the player] really suitable gear to
take to the moon?" in sentence case;
    say "[N][paragraph break]".

The Spaceport is a room. North of the Spaceport is the Rocket Launch Pad. The
player carries a stuffed bear, a chocolate cookie, and a book.

The description of the book is "It is entitled [italic type]Why Not To Take
[sentence cased inventory] To The Moon[roman type]."

To say sentence cased inventory:
    let N be "[a list of things carried by the player]" in title case;
    say "[N]".

Test me with "n / x book".
```

---

414 **Example Alpha** RB

Creating a beta-testing command that matches any line starting with punctuation.

Sometimes we want to let testers of a game insert their own comments during a transcript, without those comments wasting turns of the game or producing lengthy or inappropriate parser errors. Many testers have a habit of prefacing comments with a punctuation mark, so let's say that we'd like to catch any command that starts with any punctuation at all:

```
"Alpha"

When play begins:
    say "Hi, Larry! Thanks for testing my game!!"

Unimplemented Room is a room. "Room description goes here..."

The scary troll is a man in Unimplemented Room.

After reading a command (this is the ignore beta-comments rule):
    if the player's command matches the regular expression "^\p":
        say "(Noted.)";
        reject the player's command.

Test me with "x me / x troll / !this game is a bit dull so far / kiss troll / ? does this
troll do anything? / :yawn".
```

---

415 **Example About Inform's regular expression support** RB

Some footnotes on Inform's regular expressions, and how they compare

to those of other programming languages.

There is not really any unanimity about what regular expression language is. The unix tools sed and grep extend on Kleene's original grammar. Henry Spencer's regex library extended on this again, and was a foundation for Perl, but Perl once again went further. Philip Hazel's PCRE, despite the name Perl Compatible Regular Expressions, makes further extensions still, and so on.

Inform's regular expressions are modelled on those of Perl, as the best de facto standard we have, but a few omissions have been inevitable. Inform's regex matcher must occupy source code less than one hundredth the size of PCRE, and it has very little memory. Inform aims to behave exactly like Perl except as follows:

(i) Inform allows angle brackets as synonymous with square brackets, for reasons explained above. This means literal angle brackets have to be escaped as "\<" and "\>" in Inform regular expressions, which is unnecessary in Perl.

(ii) Inform only has single-line mode, not multiline mode: this removes need for the mode-switches "(?m)" and "(?s)" and the positional markers "\A" and "\Z". Multiline mode is idiosyncratic to Perl and is a messy compromise to do with holding long files of text as single strings, yet treating them as lists of lines at the same time: this would not be sensible for Inform. Similarly, because there is no ambiguity about how line breaks are represented in Inform strings (by a single "\n"), initial newline convention markers such as "(*ANYCRLF)" are unsupported.

(iii) The codes "\a", "\r", "\f", "\e", "\0" for alarm, carriage return, form feed, escape and the zero character are unsupported: none of these can occur in an Inform string.

(iv) Inform does not allow characters to be referred to by character code (whereas Perl allows "\036" for an octal character code, "\x7e" for a hexadecimal one, "\cD" for a control character). This is because we do not want the user to know whether text is internally stored as ZSCII or Unicode.

(v) Inform's character class "\p" (and its negation "\P") have no equivalent in Perl, and Inform's understanding of "\w" is different. Perl defines this as an upper or lower case English letter, underscore or digit, which is good for programming-language identifiers, but bad for natural language - for instance, "é" is not matched by "\w" in Perl, but unquestionably it appears in words. Inform therefore defines "\w" as the negation of "\s" union "\p".

(vi) Inform supports only single-digit grouping numbers "\1" to "\9", whereas Perl allows "\10", "\11", ...

(vii) POSIX named character ranges are not supported. These are only abbreviations in any case, and are not very useful. (Note that the POSIX range "[:punct:]", which is supposedly for punctuation, includes many things we do not want to think of that way - percentage signs, for instance - and so "\p" has a more natural-language-based definition.)

(viii) Character classes can be used inside ranges, so that "<\da-f>" is legal, but not as ends of contiguous runs, so that "<\d-X>" is not legal. (As reckless as this is, it is legal in Perl.)

(ix) For obvious reasons, escapes to Perl code using the notation "(?{...})" are unsupported, and so is the Perl iteration operator "\G".

(x) Perl's extended mode "(?x)", a lexical arrangement which allows expressions to be expanded out as little computer programs with comments, is unsupported. It would look awful syntax-coloured in the Inform interface and is not a style of coding to be encouraged.

Inform further does not support the Python extension of named subexpression groups, nor the Java extension of the possessive quantifier "++". There was only so much functionality we could squeeze in.

As verification of Inform's matching algorithm, we took the Perl 5 source code's notorious "re-test.txt" set of 961 test cases, removed the 316 using features unsupported by Inform (220 tested multiline mode, for instance), and ran the remaining 645 cases through Inform. It agrees with Perl on 643 of these: the two outstanding are -

(i) Perl is able to match "^(a\1?){4}$" against "aaaaaa" but Inform is not - Inform's backtracking is not as good when it comes to repetitions of groupings which are recursively defined. (Note that the optional "\1" match refers to the value of the bracketed expression which contains it, so that the interpretation is different on each repetition. Here to match we have to interpret "?" as 0, 0, 1, 0 repeats respectively as we work through the "{4}".)

(ii) Perl matches "((<a-c>)b*?\2)*" against "ababbbcbc" finding the match "ababb", whereas Inform finds the match "ababbbcbc". This is really a difference of opinion about whether the outer asterisk, which is greedy, should be allowed three matches rather than two if to do so requires the inner asterisk, which is not greedy, to eat more than it needs on one of those three matches.

Case (i) is a sacrifice to enable Inform's back-tracking to use less memory. Case (ii) simply seems unimportant.

416

Example Identity Theft

RB

Allowing the player to enter a name to be used for the player character during the game.

Let's say we want to allow the player to enter any name he likes for his character. Moreover, we want to reject very long names (which are likely to be mistakes anyway), and we want to extract the player's chosen first name from the rest.

```
"Identity Theft"

The player's forename is a text that varies. The player's full name is a text that varies.

When play begins:
    now the command prompt is "What is your name? > ".
```

```
To decide whether collecting names:
    if the command prompt is "What is your name? > ", yes;
    no.

After reading a command when collecting names:
    if the number of words in the player's command is greater than 5:
        say "[paragraph break]Who are you, a member of the British royal family?
No one has that many names. Let's try this again.";
        reject the player's command;
    now the player's full name is the player's command;
    now the player's forename is word number 1 in the player's command;
    now the command prompt is ">";
    say "Hi, [player's forename]![paragraph break]";
    say "[banner text]";
    move the player to the location;
    reject the player's command.
```

We also want to postpone the proper beginning of the game until we've gotten the name:

```
Instead of looking when collecting names: do nothing.

Rule for printing the banner text when collecting names: do nothing.

Rule for constructing the status line when collecting names: do nothing.

Your Bedroom is a room. The printed name of Your Bedroom is "[player's
forename]'s Bedroom".

The player carries a letter. The description of the letter is "Dear [player's full
name], [paragraph break]You have won the Norwegian Daily Lottery! ...".
```

If we are compiling for Glulx, this is enough to capture not only the player's name but also the capitalization he uses.

If we are compiling for the Z-machine, the player's input will unfortunately be reduced to lower case before we can inspect it. If we would like by default to capitalize the first letter of each word of the name, we might substitute the following after reading a command rule:

```
After reading a command when collecting names:
    if the number of words in the player's command is greater than 5:
        say "[paragraph break]Who are you, a member of the British royal family?
No one has that many names. Let's try this again.";
        reject the player's command;
    now the player's full name is the substituted form of "[the player's command in
title case]";
    now the player's forename is word number 1 in the player's full name;
    now the command prompt is ">";
    say "Hi, [player's forename]![paragraph break]";
    say "[banner text]";
    move the player to the location;
    reject the player's command.
```

417

Example **Mirror, Mirror**

RB

The sorcerer's mirror can, when held up high, form an impression of its surroundings which it then preserves.

"Mirror, Mirror"

The Sorcerer's Workshop is a room. "The sorcerer's den is a dusty, whispering place. A grandfather clock with skeletal hands reads [the time of day in words]. The floorboards are stained where that porridge just wouldn't come out."

The Apprentice's Pantry is east of the Workshop. "This is where the aproned apprentice traditionally makes the camomile tea, cleans out the jackdaw cages and furtively examines purloined artefacts."

When play begins: erase the mirror.

The player carries a magic mirror. The magic mirror has a text called the mirror vision.

To erase the mirror: now mirror vision of the mirror is "The mirror is polished clean, and has no impression upon it."

To say current room description: try looking.
To expose the mirror:
    say "The mirror shines momentarily with a dazzling light.[paragraph break]";
    now mirror vision of the mirror is the substituted form of "The hazy image in the mirror preserves a past sight:[line break][current room description]All is distorted and yet living, as though the past and present are coterminous in the mirror."

Understand "hold up [something preferably held]" or "hold [something preferably held] up" as holding aloft. Holding aloft is an action applying to one carried thing. Report holding aloft: say "You hold [the noun] aloft."

Instead of rubbing the mirror: erase the mirror; try examining the mirror. Instead of holding aloft the mirror: expose the mirror.

The description of the mirror is "[mirror vision of the mirror]".

Test me with "look / examine mirror / hold up mirror / z / look / x mirror / rub mirror / east / hold mirror up / west / x mirror".

418

Example **The Cow Exonerated**

RB

Creating a class of matches that burn for a time and then go out, with elegant reporting when several matches go out at once.

Here we create a class of matches that can be used to burn other objects. Our objectives are as follow:

Burned objects other than matches should be removed from play instantly (just as edible objects are instantly eaten). We could give everything its own burning duration, but that complicates matters and allows for fire to spread from one object to another; for an example of how to do that, see the example "In Fire or in Flood".

Matches should be described to show whether they are burning or extinguished, and when the parser chooses one of several identical matches, it should make very clear which match it has selected.

The game must sensibly select and, if necessary, automatically light new matches to carry out a >BURN THING command.

The matches must burn for a set number of turns before going out, never to be used again.

And finally, the part for which the text will be useful: when several matches go out in the same turn, we want the game to print

```
Four matches go out.
```

rather than

```
A match goes out.
A match goes out.
A match goes out.
A match goes out.
```

This last function appears down in Section 3, if we wish to skip ahead and look at it.

```
"The Cow Exonerated"

Section 1 - Simple Burning

Understand the commands "light" and "burn" as something new.

Understand "burn [something] with [strikable-match]" as burning it with.
Understand "burn [something] with [something preferably held]" as burning it
with. Burning it with is an action applying to one thing and one carried thing.

Understand the command "light" as "burn".

A thing can be flammable or impervious. A thing is usually impervious.

Check burning something with something (this is the burn only with flaming
matches rule):
    if the second noun is not a strikable-match, say "You can only light things with
matches." instead;
    if the second noun is not flaming, say "[The second noun] needs to be burning
first." instead.

Check burning something with something (this is the burn only flammable things
rule):
    if the noun is impervious, say "[The noun] cannot be burned." instead.
```

```
Check burning something with something (this is the burn only things not held
rule):
    say "[one of]It occurs to you to set down [the noun] before burning, just for
safety's sake. [or]Again, you decide to put down [the noun] prior to burning.
[or]You try setting down [the noun] as usual. [stopping][run paragraph on]";
    silently try the player dropping the noun;
    if the player encloses the noun, stop the action.

Carry out burning something with something (this is the simplistic burning rule):
    now the noun is nowhere.

Report burning something with something:
    say "You burn up [the noun]."

Rule for implicitly taking the second noun while burning something with
something which is not a strikable-match:
    say "You can only light things with matches.";
    stop the action.
```

Section 2 - Matches

The word "matches" is used by Inform to compare snippets of text (see "Reading a command" in the Activities chapter). This can sometimes cause awkwardness if we also have a kind called "match", so for the occasion we will give our matches a more specialized name, never visible to the player:

```
A strikable-match is a kind of thing. The plural of strikable-match is s-matches.

A strikable-match has a number called duration. The duration of a strikable-
match is usually 3.

Rule for printing the name of a strikable-match: say "match".
Rule for printing the plural name of a strikable-match: say "matches".

Understand "match" as a strikable-match. Understand "matches" as a strikable-
match.

Flame-state is a kind of value. The flame-states are burnt, flaming, and new.
Understand "burning" or "lit" as flaming. Understand "unused" as new.

A strikable-match has a flame-state. A strikable-match is usually new.
Understand the flame-state property as describing a strikable-match.

Before printing the name of a strikable-match while asking which do you mean:
    say "[flame-state] ".

Before printing the name of a strikable-match while taking inventory:
    say "[flame-state] ".
Before printing the plural name of a strikable-match while taking inventory:
    say "[flame-state] ".

Before printing the name of a strikable-match while clarifying the parser's choice
of something:
    if not taking inventory, say "[flame-state] ".
```

```
After printing the name of a strikable-match (called special-target) while clarifying
the parser's choice of something:
    if the player carries the special-target:
        say " you're carrying";
    otherwise if the special-target is in the location:
        say " on the ground";
    otherwise:
        say " [if the holder of the special-target is a container]in[otherwise]on[end if]
[the holder of the special-target]".

Understand "strike [something]" as attacking.

Understand "strike [strikable-match]" as striking. Striking is an action applying to
one carried thing.

Understand "burn [strikable-match]" as striking.

Does the player mean striking a new strikable-match:
    it is very likely.

Does the player mean striking a burnt strikable-match:
    it is unlikely.

Check striking a strikable-match (this is the strike only new matches rule):
    if the noun is burnt, say "[The noun] has already burnt down and cannot be
relit." instead;
    if the noun is flaming, say "[The noun] is already burning." instead.

Carry out striking a strikable-match (this is the standard striking rule):
    now the noun is flaming;
    now the noun is lit.

Report striking a strikable-match (this is the standard report striking rule):
    say "You light [the noun]."

Before burning something with a new strikable-match (this is the prior lighting
rule):
    say "(first [if the player does not carry the second noun]taking and [end
if]lighting [the second noun])[command clarification break]";
    silently try striking the second noun;
    if the second noun is not flaming, stop the action.

Rule for implicitly taking a strikable-match (called target) while striking:
    try silently taking the target.

Does the player mean burning something with a flaming strikable-match:
    it is very likely.

Does the player mean burning something with a new strikable-match:
    it is likely.

Does the player mean burning something with a burnt strikable-match:
    it is unlikely.
```

```
Instead of burning a burnt strikable-match with something:
    say "[The noun] is completely consumed and cannot be relit."

Section 3 - Putting the Matches Out

Every turn:
    let N be 0; [here we track how many matches are being put out during this
turn, so that we don't have to mention each match individually if several go out
during the same move]
    repeat with item running through flaming s-matches:
        decrement the duration of the item;
        if the duration of the item is 0:
            now the item is burnt;
            now the item is unlit;
            if the item is visible, increment N;
    if N is 1:
        say "[if the number of visible flaming s-matches is greater than 0]One of the
matches [otherwise if the number of burnt visible s-matches is greater than
1]Your last burning match [otherwise]The match [end if]goes out.";
    otherwise if N is greater than 1:
        let enumeration be "[N in words]";
        if N is the number of visible s-matches:
            if N is two, say "Both";
            otherwise say "All [enumeration]";
        otherwise:
            say "[enumeration in title case]";
        say " matches go out[if a visible strikable-match is flaming], leaving [number
of visible flaming s-matches in words] still lit[end if]."

Section 4 - Scenario

Old Chicago is a room.

The player carries a flammable thing called a log. Understand "wooden" and
"wood" as the log.

The player carries two s-matches. The matchbox is an open openable container.
It contains five s-matches. The player carries the matchbox.

When play begins:
    now every strikable-match carried by the player is flaming;
    now every strikable-match carried by the player is lit.

Test me with "i / burn match / i / i / burn log with match / burn matchbox with
match / i".
```

419

Example **Blackout**

Filtering the names of rooms printed while in darkness.

RB

In this example, we want the names of rooms to be asterisked out if the player wanders around without the benefit of a candle. We can do this by treating the room

names as text, then replacing every letter:

```
"Blackout"

Tiny Room is a dark room. Absurdly Long-Named Room is a dark room. It is
west of Tiny Room.

The Candle Factory is north of Tiny Room. It contains a beeswax candle. The
beeswax candle is lit.

Rule for printing the name of a dark room:
    let N be "[location]";
    replace the regular expression "\w" in N with "*";
    say "[N]".

Test me with "w / look / e / n / get candle / s / w".
```

Notice that the hyphen in the Absurdly Long-Named Room does not get replaced. We could replace even that, if we liked, with

```
    replace the regular expression "\S" in N with "*";
```

which would catch every character that is not a space.

---

420 **Example Fido** *RB*

A dog the player can name and un-name at will.

Suppose we'd like to have a dog which the player is allowed to name himself. We'd like to deal correctly with both

```
>name the dog fido
```

and

```
>name the dog "fido"
```

so we'll also need to strip quotation marks out of the command. We can do this as follows:

```
"Fido"

The Back Yard is a room.

A dog is an animal in Back Yard. The dog has some text called the nickname.
The nickname of the dog is "nothing". Understand the nickname property as
describing the dog.

Rule for printing the name of the dog when the nickname of the dog is not
"nothing":
```

```
    say "[nickname of the dog]"

Naming it with is an action applying to one thing and one topic. Understand
"name [something] [text]" as naming it with. Check naming it with: say "You can't
name that."

Instead of naming the dog with "nothing":
    now the nickname of the dog is "nothing";
    now the dog is improper-named;
    say "You revoke your choice of dog-name."

Instead of naming the dog with something:
    let N be "[the topic understood]";
    replace the text "'" in N with "";
    now the nickname of the dog is "[N]";
    now the dog is proper-named;
    say "The dog is now known as [nickname of the dog]."

Test me with "name the dog Fido / name the dog Lawrence / look / x lawrence /
name Lawrence nothing / look / x lawrence".
```

---

421 **Example** **Igpay Atinlay** RB

A pig Latin filter for the player's commands.

For the sake of argument, suppose we want to parrot back all the player's commands in pig Latin:

```
"Igpay Atinlay"

Armfay is a room.

After reading a command:
    let N be "[the player's command]";
    replace the regular expression "\b(<aeiou>+)(\w*)" in N with "\1\2ay";
    replace the regular expression "\b(<bcdfghjklmnpqrstvwxz>+)(\w*)" in N with
"\2\1ay";
    say "[N][paragraph break]";
    reject the player's command.

Test me with "nix on the stupid".
```

---

422 **Example** **Mr. Burns' Repast** RB

Letting the player guess types for an unidentifiable fish.

Suppose we have an unhappily mutated fish that the player can refer to by any of a number of species names, or any word followed by -fish. We want to reject these commands, but preserve a memory of what the player last tried to call the thing:

```
"Mr. Burns' Repast"

Wharf is a room.

There is an unknown fish in the Wharf. The unknown fish has some a text called
the supposed name. The description of the unknown fish is "The victim of heavy
mutagens, this thing is not really recognizable as any species you know.".

Fish variety is a kind of value. The fish varieties are salmon, albacore, mackerel.

Rule for printing the name of the unknown fish:
   if the supposed name of the unknown fish is "", say the printed name of the
unknown fish;
   otherwise say the supposed name of the unknown fish.

After reading a command:
   if the unknown fish is visible and player's command matches the regular
expression "\b\w+fish":
      let N be "[the player's command]";
      replace the regular expression ".*(?=\b\w+fish)" in N with "";
      now N is "[N](?)";
      now the supposed name of the unknown fish is N;
      respond with doubt;
      reject the player's command;
   otherwise if the unknown fish is visible and the player's command includes "
[fish variety]":
      now supposed name of the fish is "[fish variety understood](?)";
      respond with doubt;
      reject the player's command.

To respond with doubt:
   say "You're not [italic type]sure[roman type] you're seeing any such thing."

Test me with "get swordfish / look / touch monkfish / look / listen to tunafish / x
fish / x salmon / look".
```

423

Example Northstar

Making Inform understand ASK JOSH TO TAKE INVENTORY as JOSH, TAKE INVENTORY. This requires us to use a regular expression on the player's command, replacing some of the content.

RB

Most of the time, Inform understands commands to other characters when they take the form "JOSH, TAKE INVENTORY" or "JOAN, WEAR THE ARMOR". But novice players might also try commands of the form ASK JOSH TO TAKE INVENTORY or ORDER JOAN TO WEAR THE ARMOR.

The easiest way to make Inform understand such commands is to meddle directly with the player's command, changing it into the format that the game will understand, as here:

```
"Northstar"

The Northstar Cafe is a room. "The Northstar is crammed with its usual brunch
crowd, and you were lucky to get a table at all. You are now awaiting the arrival
of your ricotta pancakes."

Josh is a man in The Northstar Cafe. "Josh is on his way past your table." The
description of Josh is "He is a waiter here, but you also know him socially, so he
tends to be more chatty than the other waiters." A persuasion rule: persuasion
succeeds.

After reading a command:
    let N be "[the player's command]";
    replace the regular expression "\b(ask|tell|order) (.+?) to (.+)" in N with "\2, \3";
    change the text of the player's command to N.

Test me with "ask Josh to take inventory / tell Josh to take inventory / order Josh
to take inventory".
```

Note that we have to copy N back explicitly to replace the player's command.

---

424  **Example Cave-troll** RB

Determining that the command the player typed is invalid, editing it, and re-examining it to see whether it now reads correctly.

Novice players of interactive fiction, unfamiliar with its conventions, will often try to add extra phrases to a command that the game cannot properly parse: HIT DOOR WITH FIST, for instance, instead of HIT DOOR.

While we can deal with some of these instances by expanding our range of actions, at some point it becomes impossible to account for all the possible prepositional phrases that the player might want to tack on. So what do we do if we want to handle those appended bits of text sensibly?

We could go through and remove any piece of text containing "with ..." from the end of a player's command; the problem with that is that it overzealously lops off the ends of valid commands like UNLOCK DOOR WITH KEY, as well. So clearly we don't want to do this as part of the "After reading a command..." stage.

A better time to cut off the offending text is right before issuing a parser error. At that point, Inform has already determined that it definitely cannot parse the instruction as given, so we know that there's something wrong with it.

The next problem, though, is that after we've edited the player's text we want to feed the corrected version back to Inform and try once more to interpret it.

This is where we have a valid reason to write a new "rule for reading a command". We will tell Inform that when we have just corrected the player's input to something new, it should not ask for a new command (by printing a prompt and waiting for

another line of input); it should instead paste our stored corrected command back into "the player's command" and proceed as though that new text had just been typed.

Thanks to John Clemens for the specifics of the implementation.

```
"Cave-troll" by JDC

Section 1 - The Mechanism

The last command is a text that varies.

The parser error flag is a truth state that varies. The parser error flag is false.

Rule for printing a parser error when the latest parser error is the only
understood as far as error and the player's command matches the text "with":
    now the last command is the player's command;
    now the parser error flag is true;
    let n be "[the player's command]";
    replace the regular expression ".* with (.*)" in n with "with \1";
    say "(ignoring the unnecessary words '[n]')[line break]";
    replace the regular expression "with .*" in the last command with "".

Rule for reading a command when the parser error flag is true:
    now the parser error flag is false;
    change the text of the player's command to the last command.

Section 2 - The Scenario

The Cave is a room.

The troll is a man in the cave.

The player carries a sword.

The chest is a locked lockable container in the cave.

Test me with "attack troll with sword / unlock chest with sword / attack troll as a
test".
```

A caveat about using this method in a larger game: "parser error flag" will not automatically control the behavior of any rules we might have written for Before reading a command... or After reading a command..., so they may now fire at inappropriate times. It is a good idea to check for parser error flag in those rules as well.

Chapter 21: Lists


Contents of *Writing with Inform*


Chapter 20: Advanced Text


Chapter 22: Advanced Phrases


Indexes of the examples

§21.1. Lists and entries

Many sections in this book begin by introducing a new kind of value. Reading through in order, the possibilities mount up: numbers, times, texts, and so on. (See the Kinds page of the Index for a convenient list of the options.) This section is a little different: rather than showing a single new kind of value, it shows how to make a new kind out of any existing one.

If K is any kind of value, then "list of K" is also a kind of value. For instance, we could write:

```
let L be a list of numbers;
```

and this would create a new "let" variable, called L, whose kind of value is "list of numbers". On the other hand, we are not allowed to write:

```
let L be a list;
```

because "list" by itself is not a kind of value. (Inform always needs to know what kinds the values entered in a list are going to have.)

Lists are like flexible-length table columns, but that probably makes them sound more mysterious than they really are. A list is simply a sequence of values, called its "entries", numbered from 1 upwards. The number of entries is called its "length". If we try

```
let L be a list of numbers;
say "L has [the number of entries in L] entries.";
```

then we find

```
L has 0 entries.
```

This is because all lists start out empty when created: that is, they initially have 0 entries. Inform has two built-in adjectives "empty" and "non-empty" which can apply to lists, and they mean just what they ought to mean: a list is empty if its length is 0, and otherwise non-empty.

We can add entries very easily:

```
add 2 to L; add 3 to L; add 5 to L;
```

We can now, for instance, try saying the list:

```
say "L is now [L].";
```

with the result

```
L is now 2, 3 and 5.
```

Note that only numbers can be added to L: if we try

```
add "clock" to L;
```

Inform will produce a problem message, because L has kind "list of numbers", whereas "clock" is text. In this way, Inform ensures that a list always contains values of the same kind throughout. So it's not possible to construct a list whose entries are:

```
2, "fish", 4 and the Entire Game
```

Such a list would be very hazardous to deal with, in any case. If what we need is a combination of different kinds of values, tables are a better option.

Finally, note that since "list of numbers" is a kind of value in its own right, so is "list of lists of numbers", and so on - though such lists are trickier to deal with, they are sometimes handy.


Start of Chapter 21: Lists


Back to Chapter 20: Advanced Text: §20.9. Summary of regular expression notation


Onward to §21.2. Constant lists

---

§21.2. Constant lists

It is convenient to have a concise way to write down a constant list. Just as we could write "231", say, or "7:01 AM" to refer to particular number and time constants, so we can write list constants:

```
let L be {1, 2, 3, 4};
```

Inform recognises that "{1, 2, 3, 4}" is a list because of the braces, and looks at the entries inside, sees that they are numbers, and deduces that it is a constant whose kind of value is "list of numbers". L is then a temporary list variable and we can add to it, remove things, and so on as we please - {1, 2, 3, 4} is merely its initial value.

When constructing lists, it is worth noting that Inform requires spaces after the commas (which seems a little harsh, but is necessary because otherwise many sensible literal specifications for units would be impossible - anyway, the reason isn't important here). So

```
let L be {1,2,3,4};
```

would produce problem messages. But Inform does not require spaces round its braces.

We call this way of writing a list "brace notation". In mathematics, braces are usually used for sets, and properly speaking these are sequences not sets - so that "{1, 2, 3, 4}" is different from "{4, 3, 2, 1}" - but it is still a familiar notation. Similarly,

```
let L be {"apple", "pear", "loganberry"};
```

makes L a list of texts; and

```
The marshmallow, the firework and the stink bomb are in the Scout Hut. The list of prohibited items is a list of objects that varies. The list of prohibited items is {the firework, the stink bomb}.
```

makes a global variable ("list of prohibited items") with kind of value "list of objects", and whose initial value is to contain two things: the firework and the stink bomb. More exotically, if we need to make lists of lists:

```
let L be {{1, 2}, {6, 7, 8}};
```

gives L the kind of value "list of lists of numbers", with (initially) two entries: the list {1, 2} (a list of numbers), then the list {6, 7, 8} (ditto).

Constant lists are convenient, too, when a column in a table needs to contain lists:

```
The duck, the orange, the cider, the cinnamon and the orange are in the Kitchen.

Table of Requirements

recipe              ingredients
"duck à l'orange"   {the duck, the orange}
"spiced cider"      {the cider, the cinnamon, the orange}
```

A special word about the constant list "{ }". This means the list with no entries - the empty list. If we try to create a new "let" variable M with

```
let M be { };
```

then Inform will produce a problem message, because it cannot tell what sort of list M will be: a list of numbers, or texts, or times, or...? On the other hand, writing

```
let M be { };
```

is fine provided that M already exists, and then does the obvious thing - empties M. Similarly, a table column in which every entry is "{ }" produces a problem message unless the heading for that column spells out the kind of value stored within it: for instance, "ingredients (list of texts)".

All of this is a notation for constant lists only, not some sort of gluing-things-together operation. So this, for instance:

```
let L be {100, the turn count};
```

is not allowed, even though "the turn count" is a number: because it is a number that varies, the braces do not contain constants, and therefore this is not a list constant.


Start of Chapter 21: Lists


Back to §21.1. Lists and entries


Onward to §21.3. Saying lists of values

---

§21.3. Saying lists of values

Any list L can be said, provided that its contents can be said. For example:

```
let L1 be {2, 3, 5, 7, 11};
say L1;
```

produces the text "2, 3, 5, 7 and 11" - unless we have "Use serial comma." set, in which case a comma appears after the 7. We also have the option of using the more formal notation:

say "[(list of values) **in brace notation]"**

This text substitution produces the list in the form of "{", then a comma-separated list, and then "}", which looks less like an English sentence but more mathematical. Example:

```
"[list of people in brace notation]"
```

might produce "{ yourself, Mr Darcy, Flashman }".

If we say a list of lists, then the individual entry lists are always printed in brace notation: the ordinary sentence way would be incomprehensible.

Of course, the values in L1 are written out in number form because L1 is a list of numbers: we could alternatively try

```
let L2 be {the piano, the music stand};
say L2;
```

which produces "piano and music stand". Lists of objects can be said in two additional ways:

say "[(list of objects) **with definite articles]"**

This text substitution writes out the list in sentence form, adding the appropriate definite articles. Example:

```
let L be {the piano, the music stand};
say "[L with definite articles]";
```

says "the piano and the music stand".

say "[(list of objects) **with indefinite articles]"**

This text substitution writes out the list in sentence form, adding the appropriate indefinite articles. Example:

```
let L be {the piano, the music stand};
say "[L with definite articles]";
```

says "a piano and a music stand".

Start of Chapter 21: Lists

Back to §21.2. Constant lists

Onward to §21.4. Testing and iterating over lists

Example 425: **Oyster Wide Shut** Replacing Inform's default printing of properties such as "(closed)", "(open and providing light)", etc., with our own, more flexible variation.

---

§21.4. Testing and iterating over lists

If L is a list, we can interrogate it to see whether it does or does not contain (at least one instance of) any compatible value V:

if (value) **is listed in** (list of values)**:**

This condition is true if the given value, which must be of a compatible kind, is one of those in the list. For instance, if L is our list of the numbers 2, 3, 5, 7 and 11 then 5 is listed in it but 6 is not.

if (value) **is not listed in** (list of values)**:**

This condition is true if the given value, which must be of a compatible kind, is not one of those in the list.

We can also repeat running through a list (just as we can with table rows). Thus:

repeat with (a name not so far used) **running through** (list of values)**:**

This phrase causes the block of phrases following it to be repeated once for each item in the given list, storing that value in the named variable. (The variable exists only temporarily, within the repetition.) Example:

```
let L be {2, 3, 5, 7, 11, 13, 17, 19};
repeat with prime running through L:
    ...
```

If the list is empty, nothing happens: the "..." phrase(s) are never tried.

In the next sections, we shall see that it is possible to change, reorder and resize lists. But it's important never to change a list that's being repeated through. The following:

```
let L1 be {1, 2, 3, 4};
repeat with n running through L1:
    remove n from L1;
```

leaves L1 containing {2, 4}, since the removals from the list cause it to shuffle back even while we repeat through it - a bad, bad idea.


Start of Chapter 21: Lists


Back to §21.3. Saying lists of values


Onward to §21.5. Building lists

---

§21.5. Building lists

We have already seen "add... to...". This in fact comes in two forms:

add (value) **to** (list of values)

This phrase adds the given value to the end of the list. Example:

```
let L be {60, 168};
add 360 to L;
```

results in L being {60, 168, 360}. Note that the value is added even if it already occurs somewhere in L; this can be avoided with "if absent". So:

```
add 168 to L, if absent;
```

would do nothing - it is already there.

add (list of values) **to** (list of values)

This phrase adds the first list to the end of the second. Example:

```
let L be {2, 3, 5, 7};
add {11, 13, 17, 19} to L;
```

results in L being {2, 3, 5, 7, 11, 13, 17, 19}.

If we don't want to add new entries at the end, we can instead say where they should go:

add (value) **at entry** (number) **in** (list of values)

This phrase adds the given value so that it becomes the entry with that index number in the list. Example:

```
let L be {1, 2, 3, 4, 8, 24};
add 12 at entry 6 in L;
```

sets L to {1, 2, 3, 4, 8, 12, 24}. If there are N entries in L, then we can add at any of entries 1 up to N+1: adding at entry N+1 means adding at the end. The phrase option "if absent" makes the phrase do nothing if the value already exists anywhere in L.

add (list of values) **at entry** (number) **in** (list of values)

This phrase adds the first list to the second so that it begins at the given position. Example:

```
let L be {1, 2, 3, 4};
add {4, 8, 12} at entry 3 in L;
```

results in L being {1, 2, 4, 8, 12, 3, 4}.

A list is allowed to contain duplicates, and the order matters. For instance:

```
let L be {2, 2, 3};
```

makes L into "2, 2 and 3". This is a different list to the one made by:

```
let M be {2, 3, 2};
```

even though L and M have the same values, repeated the same number of times - for two lists to be equal, they must have the same kind of entry, the same number of entries, and the same entries in each position.

We can also strike out values:

remove (value) **from** (list of values)

This phrase removes every instance of the given value from the list. Example:

```
let L be {3, 1, 4, 1, 5, 9, 2, 6, 5, 3};
remove 1 from L;
```

results in L being {3, 4, 5, 9, 2, 6, 5, 3}. Ordinarily "remove 7 from L" would produce a run-time problem, since L does not contain the value 7, but using the "if present" option lets us off this: the phrase then does nothing if L does not contain the value to be removed.

remove (list of values) **from** (list of values)

This phrase removes every instance of any value in the first list from the second. Example:

```
let L be {3, 1, 4, 1, 5, 9, 2, 6, 5, 3};
remove {0, 2, 4, 6, 8} from L;
```

results in L being {3, 1, 1, 5, 9, 5, 3}. If both lists are large, this can be a slow process, and we might do better by sorting them and trying a more sophisticated method. But this is convenient for anything reasonable-sized.

Again, we can also remove from specific positions:

remove entry (number) **from** (list of values)

This phrase removes the entry at the given position, counting from 1 as the first entry. (Once it is removed, the other entries shuffle down.) Example:

```
let L be {3, 1, 4, 1, 5, 9, 2, 6, 5, 3};
remove entry 3 from L;
```

results in L being {3, 1, 1, 5, 9, 2, 6, 5, 3}.

remove entries (number) **to** (number) **from** (list of values)

This phrase removes the entries at the given range of positions, counting from 1 as the first entry. (Once they are removed, the other entries shuffle down.) Example:

```
let L be {3, 1, 4, 1, 5, 9, 2, 6, 5, 3};
remove entries 3 to 6 from L;
```

results in L being {3, 1, 2, 6, 5, 3}.

Start of Chapter 21: Lists

Back to §21.4. Testing and iterating over lists

Onward to §21.6. Lists of objects

Example 426: **Robo 1** A robot which watches and records the player's actions, then tries to repeat them back in the same order when he is switched into play-back mode.

---

§21.6. Lists of objects

Lists can be made of values of any kind (including other lists), but lists of objects are especially useful. We could always make these "by hand":

```
let L be {the pot plant, the foxglove};
```

But it is usually easier and clearer to use descriptions.

list of (description of values) **... value**

This phrase produces the list of all values matching the given description. Inform will issue a problem message if the result would be an infinite list, or one which is impractical to test: for instance "list of even numbers" is not feasible.

While that works nicely for many kinds of value ("list of recurring scenes", say), it's particularly useful for objects:

```
let L be the list of open containers;
add the list of open doors to L;
```

means that L now contains the open containers (if any) followed by the open doors (if any). Or, for example:

```
let L be the list of things;
remove the list of backdrops from L;
```

makes a list of all non-backdrops.

As mentioned above, lists of objects can be said in two additional ways:

```
"[L with definite articles]"
"[L with indefinite articles]"
```

And as mentioned below, they can be sorted in property value order:

```
sort L in P order;
sort L in reverse P order;
```

where P is any value property. In all other respects, lists of objects are no different to other lists.

 Start of Chapter 21: Lists

 Back to §21.5. Building lists

 Onward to §21.7. Lists of values matching a description

 Example 427:  **What Makes You Tick** Building a fishing pole from several component parts that the player might put together in any order.

 Example 428:  **Formicidae** Manipulating the order in which items are handled after TAKE ALL.

---

§21.7. Lists of values matching a description

The useful "list of ..." syntax can also be used to produce lists of the values matching a description, too. Thus:

```
let L be the list of non-recurring scenes;
let C be the list of colours;
```

There is little to say here except for the usual warning that some kinds of value have a range which is too large to make this possible. For instance, Inform could not sensibly represent:

```
let N be the list of even numbers;
```

It would just be too large to hold. In general, if we can repeat through, or find the number of, values matching a description, then we can also use "list of" to bring them all together. See the chart of kinds of value in the Kinds index for a project for which kinds of value allow this.

Start of Chapter 21: Lists
Back to §21.6. Lists of objects
Onward to §21.8. Sorting, reversing and rotating lists

§21.8. Sorting, reversing and rotating lists

Any list L can be reversed:

reverse (list of values)

This phrase puts the list in reverse order. The old entry 1 becomes the new last entry, and so on: reversing an empty list or a list containing only one entry leaves it unchanged. Example:

```
let L be {11, 12, 14, 15, 16, 17};
reverse L;
```

results in L being {17, 16, 15, 14, 12, 11}.

And any list can similarly be sorted:

sort (list of values)

This phrase puts the list into ascending order. Example:

```
let L be {6 PM, 11:13 AM, 4:21 PM, 9:01 AM};
sort L;
```

results in L being {9:01 AM, 11:13 AM, 4:21 PM, 6 PM}.

sort (list of values) **in reverse order**

This phrase puts the list into descending order. Example:

```
let L be {6 PM, 11:13 AM, 4:21 PM, 9:01 AM};
sort L in reverse order;
```

results in L being {6 PM, 4:21 PM, 11:13 AM, 9:01 AM}.

sort (list of values) **in random order**

This phrase puts the list into a uniformly random order, shuffling it as if it were a pack of cards. Example:

```
let L be {1, 2, 3, 4, 5, 6};
sort L in random order;
```

might result in L being {3, 1, 5, 6, 4, 2}. Or any of 719 other arrangements, including being left as it was.

Lists of objects can also be sorted in property value order. For instance,

sort (list of objects) **in** (property) **order**

This phrase puts the list into ascending order of the values of the given property for the items in the list; this is only allowed if all of those values do have the property in question. Example:

```
let L be the list of people;
sort L in carrying capacity order;
```

would arrange people with weaklings first, titans last.

sort (list of objects) **in reverse** (property) **order**

This phrase puts the list into descending order of the values of the given property for the items in the list; this is only allowed if all of those values do have the property in question. Example:

```
let L be the list of people;
sort L in reverse carrying capacity order;
```

would arrange people with titans first, weaklings last.

Rotating a list means moving all of its entries along by one place, and then moving the one on the end back to the start. For instance, if L is {1, 2, 3, 4}, then

rotate (list of values)

This phrase shuffles the entries of the list forwards (to the right) by one place, so that the 1st becomes 2nd, the 2nd becomes 3rd, and so on until the last, which becomes the new first entry. Example:

```
let L be { "cow", "heifer", "bullock" };
rotate L;
```

results in L being { "bullock", "cow", "heifer" }.

rotate (list of values) **backwards**

This phrase shuffles the entries of the list backwards (to the left) by one place, so that the 3rd becomes 2nd, the 2nd becomes 1st, and so on; the previous 1st entry becomes the new last entry. Example:

```
let L be { "cow", "heifer", "bullock" };
rotate L backwards;
```

results in L being { "heifer", "bullock", "cow"}. (This achieves the same effect as "reverse L; rotate L; reverse L;" but is a little faster, and a lot less effort to read.)


Start of Chapter 21: Lists


Back to §21.7. Lists of values matching a description


Onward to §21.9. Accessing entries in a list

---

§21.9. Accessing entries in a list

The length of a list can change as values are added or removed, and can in principle be any number from 0 upwards. A list with 0 entries is empty. We can find the length with:

number of entries in/of (list of values) **... number**

This phrase produces the number of positions in the list. Example:

```
the number of entries in {1, 1, 1, 3, 1}
```

is 5, even though there are only two genuinely different items in the list.

If the length is N then the entries are numbered from 1 (the front) to N (the back). These entries can be accessed directly by their numbers. For instance,

```
entry 2 of L
```

refers to the second entry of L: it can be used as a value, or changed, just as if it were a named variable. For instance, we could write:

```
now entry 7 of L is "Spain";
say "The rain in [entry 7 of L] stays mainly in the plain.";
```

which would (untruthfully) print "The rain in Spain stays mainly in the plain", but only if L had an entry 7 to make use of: if L were a list of 5 entries, say, then a run-time problem results. (And if L cannot hold text, a problem message means that we never get as far as run-time.) Because entries number from 1, this is always incorrect:

```
entry 0 of L
```

and if L is currently empty, then there is no entry which can be accessed, so that any use of "entry ... of L" would produce a run-time problem. There are programming languages in the world where accessing entry 100 in a 7-entry list automatically extends it to be 100 entries long: Inform is not one of them. But see the next section for how to change list lengths explicitly.


Start of Chapter 21: Lists


Back to §21.8. Sorting, reversing and rotating lists


Onward to §21.10. Lengthening or shortening a list


Example 429:  **Robo 2** A robot which watches and records the player's actions, then tries to repeat them back in the same order when he is switched into play-back mode.

---

§21.10. Lengthening or shortening a list

We can explicitly change the length of a list like so:

> **change** (list of values) **to have** (number) **entries**/entry
>
> This phrase alters the given list so that it now has exactly the number of entries given. Example:
>
> ```
> change L to have 21 entries;
> ```
>
> If L previously had more than 21 entries, they are thrown away (and lost forever); if L previously had fewer, then new entries are created, using the default value for whatever kind of value L holds. So extending a list of numbers will pad it out with 0s, but extending a list of texts will pad it out with the empty text "", and so on.

We can also write the equivalent phrases:

> **truncate** (list of values) **to** (number) **entries**/entry
>
> This phrase alters the given list so that it now has no more than the number of entries given. Example:
>
> ```
> truncate L to 8 entries;
> ```

shortens L to length 8 if it is currently longer than that, trimming entries from the end, but would (for instance) leave a list of length 3 unchanged. Note that

```
truncate L to 0 entries;
```

empties it to { }, the list with nothing in.

truncate (list of values) **to the first** (number) **entries**/entry

This phrase alters the given list so that it now consists only of the initial part of the list with the given length. Example:

```
truncate L to the first 4 entries;
```

turns {1, 3, 5, 7, 9, 11} to {1, 3, 5, 7}.

truncate (list of values) **to the last** (number) **entries**/entry

This phrase alters the given list so that it now consists only of the final part of the list with the given length. Example:

```
truncate L to the last 4 entries;
```

turns {1, 3, 5, 7, 9, 11} to {5, 7, 9, 11}.

But we don't have to truncate: we can also -

extend (list of values) **to** (number) **entries**/entry

This phrase pads out the list with default values as needed so that it now has at least the given length. (If the list is already at least that length, nothing is done.) Example:

```
extend L to 80 entries;
```

lengthens L to length 80 if it is currently shorter than that.

For example,

```
To check sorting (N - a number):
    let L be a list of numbers;
    extend L to N entries;
    repeat with X running from 1 to N:
```

```
        now entry X of L is X;
    say "L unrandomised is [L].";
    sort L in random order;
    say "L randomised is [L].";
    sort L;
    say "L in ascending order is [L]."
```

builds a list of N numbers (initially all 0), fills it with the numbers 1, 2, 3, ..., N, then randomly reorders them, then sorts them back again, recovering the original order. The text produced by "check sorting 10" depends partly on chance but might for instance be:

```
L unrandomised is 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10.
L randomised is 6, 2, 9, 3, 10, 1, 7, 4, 8 and 5.
L in ascending order is 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10.
```

As with text in the previous chapter, a project which needs really long lists should use the Glulx virtual machine - "check sorting 10000", for instance, would break the default memory environment on the Z-machine, which is very tight, but works fine (if not very rapidly) on Glulx.


Start of Chapter 21: Lists


Back to §21.9. Accessing entries in a list


Onward to §21.11. Variations: arrays, logs, queues, stacks, sets, sieves and rings


Example 430:  **Leopard-skin** A maze that the player can escape if he performs an exact sequence of actions.


Example 431:  **The Facts Were These** Creating a variant GIVE action that lets the player give multiple objects simultaneously with commands like GIVE ALL TO ATTENDANT or GIVE THREE DOLLARS TO ATTENDANT or GIVE PIE AND HAT TO ATTENDANT. The attendant accepts the gifts only if their total combined value matches some minimum amount.

---

§21.11. Variations: arrays, logs, queues, stacks, sets, sieves and rings

Lists are highly adaptable, and many other collection-like constructions can be made using them. This section introduces no new material, but simply suggests some of the variations which are possible.

1. The traditional computing term **array** means a list of values accessed by their entry numbers, often used in mathematical computations. The difference between an array and a list is mostly one of attitude, but usually arrays are fixed in length whereas lists can expand or contract.

2. A **log** is a list which records the most recently arrived values, but does not allow itself to grow indefinitely. In the following, which remembers the seven most recently taken items, new values arrive at the end while old ones eventually disappear from the front:

```
The most-recently-taken list is a list of objects that varies.
Carry out taking something (called the item):
    truncate the most-recently-taken list to the last 6 entries;
    add the item to the most-recently-taken list.
```

```
After taking:
    say "Taken. (So, your recent acquisitions: [most-recently-taken list].)"
```

Note that the most-recently-taken list begins play as the empty list, grows as the first few items are taken, but then stabilises at length 7 thereafter. If we need to remember recent history, but only *recent* history, then a log is better than a list which can grow indefinitely, because there is no risk of speed reduction or memory exhaustion in a very long story.

3. A **queue** is a list of values which are waiting for attention. New values join at the back, while those being dealt with are removed from the front (whereupon the whole queue moves up one). An empty queue means that nobody is waiting for attention: but there is, in principle, no upper limit to the size of a queue, as anyone who has tried to make a couchette reservation at Roma Termini will know.

Queues typically form when two independent processes are at work, but going at different or variable speeds. An empty queue looks just like any other list:

```
The queue is a list of objects that varies.
```

(Invariably people, in what follows, but we'll make it a "list of objects" to allow for other possibilities too.) Once we identify a "new customer", we can join him to the queue thus:

```
add the new customer to the queue;
```

The process of serving the customers needs to make sure there is actually somebody waiting in the queue before it does anything:

```
Every turn when the number of entries in the queue is not 0:
    let the next customer be entry 1 of the queue;
    say "[The next customer] is served and leaves.";
    remove entry 1 from the queue.
```

Of course queues can also be constructed which empty from other positions, rather than the front: or we could make what computer scientists sometimes call a **deque**, a "double-ended queue" where new values arrive at both ends.

4. A **stack** is like a queue except that values arrive at, and are removed from, the same end. Stacks are slightly faster if the active end is the back rather than the front, though this will only be noticeable if they grow quite large.

To put a value V onto a stack S (which is known as "pushing") is simple:

```
add V to S;
```

And to remove a value from the top of the stack (which is known as "pulling"):

```
let N be the number of entries in S;
let V be entry N of S;
remove entry N from S;
```

Note that the middle line, accessing entry N, will fail if N = 0, that is, if the stack is empty: Inform's list routines will produce a run-time problem message.

Stacks are useful if some long-term process is constantly being interrupted by newer and more urgent demands, but they can also be used in planning. If a character has a long-term goal, which needs various short-term goals to be achieved along the way, then a stack can represent the goals currently being pursued. The top of the stack represents what the character is trying to achieve now. If the character realises that it needs to achieve something else first, we put that new goal onto the top of the stack, and it becomes the new current goal. When the character completes a task, it can be removed, and we can go back to trying to finish whatever is now on top. When the stack is empty, the character has achieved the original goal.

5. Notoriously, **set** has 464 distinct meanings in the Oxford English Dictionary, making it the single most ambiguous word in the language. Here we mean not the home of a badger or the Egyptian god of the desert, but the mathematical sense: a collection of values (sometimes called "elements") without duplicates, and which is normally written in brace notation and in some natural order for the reader's convenience.

The trick here is to maintain the principle that, at all times, our list is sorted in order and contains no duplicates. To provide an example, we start with two sets of numbers:

```
let S be {2, 4, 8, 16, 32, 64};
let T be {2, 4, 6, 10};
```

Here we add an element to T:

```
add 8 to T, if absent; sort T;
```

The "if absent" clause ensures that no duplicate can occur, and by sorting T afterwards, we maintain the principle that a set must remain in order - so T is now {2, 4, 6, 8, 10}, not {2, 4, 6, 10, 8}. (Inform's sorting algorithm is fast on nearly-sorted lists, so frequent sorting is not as inefficient as it might look.)

We next take the union of T and S, that is, the set containing everything which is in either or both:

```
let U be S; add T to U, if absent; sort U;
```

This makes U = {2, 4, 6, 8, 10, 16, 32, 64}, and once again no duplicates occur and we preserve the sorting. The intersection of T and S, the set of elements in both of them, is a little trickier:

```
let I be T;
repeat with the element running through T:
    if the element is not listed in S, remove the element from I.
```

(Faster methods could be devised which exploit the sortedness of T and S, but are not worth it for shortish lists.) This produces I = {2, 4, 8}. Lastly, we can form the set difference, consisting of those elements which are in S but not in T:

```
let D be S; remove T from D, if present;
```

Here, as with intersection, since all we do is to strike out unwanted elements, the surviving ones remain in order and there is no need to sort when we are finished. This produces D = {16, 32, 64}.

6. A **sieve** is used to make a complicated choice where there are many constraints, by ruling out impossible cases to see what is left. The term derives from the kitchen utensil (for sieving fine grains of flour), but via the name of the "sieve of Eratosthenes", an ancient Greek method for determining the prime numbers.

Using a sieve is much like using a set, and the difference is mainly one of outlook - we are interested in what does not belong, rather than what does.

7. A **ring** is not so much a row of values, more a circle, with the last and first entries thought of as adjacent. One position is usually thought of as special, and is the place where new items are added: this may as well be entry 1. For instance, to add "new item" to the ring:

```
add the item at entry 1 in the ring;
```

To set "item" to the frontmost value and extract it from the ring:

```
let the item be entry 1 of the ring;
remove entry 1 from the ring;
```

And we can rotate the ring in either direction, making a different entry the new entry 1 and therefore the new frontmost value:

```
rotate the ring;
rotate the ring backwards;
```

A last note to conclude the chapter on lists. Lists, like almost all other values in Inform, can be passed to phrases as parameters. However, note that they are genuine values, not what some programming languages call "references" or "pointers". So the following:

```
To mess with (L - a list of numbers):
    add 7 to L, if absent.
```

does nothing, in practice. If given a list, it adds 7 to the list, but then throws it away again, so the longer list is never seen; it's exactly like

```
To mess with (N - a number):
    now N is 3.
```

which can never affect anything other than its own temporary value "N", which expires almost immediately in any case.

If we want a phrase which changes a list in a useful way and gives it back to us, we need a phrase which both takes in and gives back:

```
To decide which list of numbers is the extended (L - a list of numbers):
    add 7 to L, if absent;
    decide on L.
```

And then, for example -

```
the extended { 2, 4, 6 };
```

produces:

```
{ 2, 4, 6, 7 }
```

(This may seem surprising since Inform's built-in phrases to adjust lists, like "add ... to ...", don't work this way. But those are written using Inform 6 code; see the phrase definitions in the Standard Rules for more.)

Start of Chapter 21: Lists

Back to §21.10. Lengthening or shortening a list

Onward to Chapter 22: Advanced Phrases: §22.1. A review of kinds

Example 432: **Circle of Misery** Retrieving items from an airport luggage carousel is such fun, how can we resist simulating it, using a list as a ring buffer?

Example 433: **Eyes, Fingers, Toes** A safe with a multi-number combination, meant to be dialed over multiple turns, is implemented using a log of the last three numbers dialed. The log can then be compared to the safe's correct combination.

Example 434: **The Fibonacci Sequence** The modest Leonardo Fibonacci of Pisa will be only too happy to construct his sequence on request, using an array.

Example 435: **I Didn't Come All The Way From Great Portland Street** In this fiendishly difficult puzzle, which may perhaps owe some inspiration to a certain BBC Radio panel game (1967-), a list is used as a set of actions to help enforce the rule that the player must keep going for ten turns without hesitation, repetition, or deviating from the subject on the card.

Example 436: **Lugubrious Pete's Delicatessen** In this evocation of supermarket deli counter life, a list is used as a queue to keep track of who is waiting to be served.

Example 437: **Sieve of Eratosthenes** The haughty Eratosthenes of Cyrene will nevertheless consent to sieve prime numbers on request.

Example 438: **Your Mother Doesn't Work Here** Your hard-working mother uses a list as a stack: urgent tasks are added to the end of the list, interrupting longer-term plans.

Examples from Chapter 21: Lists

Start of this chapter

Chapter 22: Advanced Phrases

Indexes of the examples

425

Example **Oyster Wide Shut** *RB*

Replacing Inform's default printing of properties such as "(closed)", " (open and providing light)", etc., with our own, more flexible variation.

As we've seen in earlier examples such as "Equipment List", it is possible to vary the way Inform creates inventory listings in general (to create lists that look more like

paragraphs of prose, lists divided between what the player is wearing and what he isn't, and so on). We can also use activities to alter the printing of specific objects' names and contents, as with the "omit contents in listing" feature after printing the name of something.

We may find, however, that we would like a great deal more control over Inform's printing of inventory details, not just as a special effect for a few items, but throughout the game.

We start by turning off Inform's native property writer:

```
"Oyster Wide Shut"

Section 1 - Procedure

The print standard inventory rule is not listed in any rulebook.

Carry out taking inventory (this is the new print inventory rule):
    say "You are carrying: [line break]";
    list the contents of the player, with newlines, indented, including contents, with
extra indentation.
```

This is very much like the library's standard behavior, but with the exception that "giving inventory information" or even "giving brief inventory information" are omitted. Here's how we supplant it:

```
After printing the name of something (called target) while taking inventory:
    follow the property-aggregation rules for the target.
```

Now, our property-aggregation rulebook is going to look at a given object and decide on a list of features that should be mentioned in inventory. We'll start by producing something quite similar to Inform's default behavior:

```
The property-aggregation rules are an object-based rulebook.
The property-aggregation rulebook has a list of text called the tagline.

A first property-aggregation rule for an openable open thing (this is the mention
open openables rule):
    add "open" to the tagline.

A first property-aggregation rule for an openable closed thing (this is the mention
closed openables rule):
    add "closed" to the tagline.

A property-aggregation rule for a closed transparent container which contains
nothing (this is the mention empty transparent containers rule):
    add "empty" to the tagline.

A property-aggregation rule for an open container which contains nothing (this is
the mention empty open containers rule):
    add "empty" to the tagline.
```

```
A property-aggregation rule for a lit thing (this is the mention lit objects rule):
    add "providing light" to the tagline.

A property-aggregation rule for a thing worn by the player (this is the mention
worn objects rule):
    add "being worn" to the tagline.

The last property-aggregation rule (this is the print aggregated properties rule):
    if the number of entries in the tagline is greater than 0:
        say " ([tagline])";
        rule succeeds;
    rule fails.
```

Notice that we don't need to write any rules about how to print that list of text: because Inform is printing out a list, it will automatically insert commas, spaces, and the word "and" where appropriate; and it will automatically follow the "use serial comma" option, if we have it set.

Now we're free to meddle. Let's give the player a bunch of possessions that will be listed in interesting ways in inventory:

```
Section 2 - Scenario

The Curved Beach is a room. "White sand stretches away both northeast and
northwest, enclosing this attractive little bay. Gentle waves lap at the beach."

The player carries a glowing plastic sack. The glowing plastic sack is lit and
transparent and openable and open. It contains a rock. It is wearable.

The player wears a flashlight lanyard. The flashlight lanyard is a device.

Carry out switching on the lanyard: now the noun is lit.
Carry out switching off the lanyard: now the noun is unlit.

The player carries an oyster. The oyster contains a pearl. The oyster is
openable.
```

Now suppose that we don't want the oyster to say "closed" when it's closed. Instead, we'd like it to say "clamped shut". As this is the only property the oyster will ever have, we can simply override his whole property-aggregation rulebook:

```
A property-aggregation rule for the oyster:
    if the oyster is closed:
        say " (clamped shut)";
    rule succeeds.
```

That's fine for the oyster because "clamped shut" is the only property he'll ever have. What if we'd like instead just to revise the way the sack (and only the sack) gets described as providing light?

```
The sacklight rule is listed after the mention lit objects rule in the property-
aggregation rules.
```

```
A property-aggregation rule for the plastic sack (this is the sacklight rule):
    if "providing light" is listed in the tagline:
        remove "providing light" from the tagline;
        add "gently glowing" to the tagline.
```

Now the flashlight (and any other regular light sources we might add to the game) will be described as "providing light", but the sack will only be said to be gently glowing -- a bit more appropriate for its rather fainter gleam.

We might also wish to add a systematic feature across the board to include a new property in the inventory list? Let's say the player can enchant his possessions, and enchanted possessions should thereafter be listed accordingly:

```
A thing can be magical or non-magical. A thing is usually non-magical.

Understand "enchant [something]" as enchanting. Enchanting is an action
applying to one thing.

Carry out enchanting something:
    now the noun is magical.

Report enchanting something:
    say "Ding! You turn [the noun] magical."

A property-aggregation rule for a magical thing:
    add "enchanted" to the tagline.

Test me with "i / close oyster / i / turn on flashlight / i / take off flashlight / i / turn
off flashlight / i / close sack / i / open sack / i / take all from sack / i / close sack / i
/ wear sack / i / enchant sack / i / open sack / put all in sack / i / close sack / i".
```

Further variations are possible as well: if we used a "before printing the name..." rather than an "after printing the name..." rule, we could automatically generate lines like "an open and empty phosphorescent plastic sack", removing some of the artificiality of the parentheses.

Or we could add more logic to the rules about which properties are mentioned, so that some features of objects were mentioned in inventory only if the player was wearing the correct detection device, like so:

```
The player wears enchantment-detecting goggles.

A property-aggregation rule for a magical thing:
    if the player is wearing the goggles:
        add "enchanted" to the tagline.
```

426

Example **Robo 1**

RB

A robot which watches and records the player's actions, then tries to repeat them back in the same order when he is switched into play-back mode.

"Robo"

The Experimentation Chamber is a room. Robo is a man in the Experimentation Chamber. "Robo, your prototype tin companion, stands awkwardly beside you. In the middle of his chest is a red enamel button[if the red button is switched on], currently depressed[otherwise], currently un-depressed[end if]."

The red button is a device. It is part of Robo. Instead of pushing the red button: if the red button is switched off, try switching on the red button; otherwise try switching off the red button.

After switching on the red button:
    say "CLICK! Robo is now in play-back mode."

After switching off the red button:
    say "CLACK! Robo is now in observation mode."

Definition: Robo is watching if the red button is switched off.

The current instruction set is a list of stored actions that varies.

After doing something when Robo is watching and Robo can see the player:
    now the actor is Robo;
    add the current action to the current instruction set;
    now the actor is the player;
    say "Robo watches you [the current action][one of], his yellow eyes lamp-like and observant[or]. In his metal head, gears whirr[or], his brushed-copper handlebar moustaches twitching[or] impassively[at random].";
    continue the action.

Every turn when Robo is not watching:
    if the number of entries in the current instruction set is 0:
        say "Robo has run out of behavior and grinds to an (expectant) halt.";
        now the red button is switched off;
    otherwise:
        let the next task be entry 1 of the current instruction set;
        try the next task;
        remove entry 1 from the current instruction set.

The red block and the blue cylinder are things in the Experimentation Chamber. The counter is a supporter in the Experimentation Chamber. The counter is scenery.

Report Robo examining Robo:
    say "Robo examines each of his hands in turn, then each of his legs (bending over mostly double in the middle to do this)." instead.

Report Robo examining the player:
    say "Robo stares at you, unblinkingly, for several seconds together[if a random chance of 1 in 7 succeeds]. His left moustache-bar twitches infinitesimally upward[end if]." instead.

```
Report Robo taking the cylinder:
    say "[one of][Robo] needs several attempts to get his metal fingers around
[the cylinder] -- they are not designed for grasping small objects elegantly. But at
last he succeeds[or]Once again, Robo struggles a bit before picking up [the
cylinder][stopping]." instead.

Test me with "z / take cylinder / take block / put cylinder on counter / put block on
counter / x robo / x me / get block / drop block / press red button / z / z / z / z / z /
z / z / z / z / z".
```

427

Example What Makes You Tick

RB

Building a fishing pole from several component parts that the player might put together in any order.

Suppose we want to let the player build a fishing pole out of three parts: a hook, a string, and a stick.

There are several things we must account for here. One is that our combination verb should be insensitive to ordering: it shouldn't matter whether the player types COMBINE STICK WITH STRING or COMBINE STRING WITH STICK.

Second, we need to make sure that our implementation handles intervening stages of assembly gracefully. The player should be able to combine string and hook first, or string and stick first, and be able to complete the assembly in either case.

Our implementation here uses a table of lists to determine which combinations of inputs should produce which result object. Because we sort our lists before comparing them, we guarantee that the player's ordering doesn't matter: COMBINE STICK WITH STRING will have the same effect as COMBINE STRING WITH STICK.

What's more, our implementation could be expanded to account for many other assemblages, if we wanted object-building to be a running theme of puzzles in our game.

```
"What Makes You Tick"

Understand "combine [something] with [something]" as combining it with.
Combining it with is an action applying to two carried things. Understand the
command "connect" as "combine".

Understand the command "attach" as something new. Understand "attach
[something] to [something]" as combining it with.

The combining it with action has an object called the item built.

Setting action variables for combining something with something:
    let X be a list of objects;
    add the noun to X;
    add the second noun to X;
```

```
    sort X;
    repeat through the Table of Outcome Objects:
        let Y be the component list entry;
        sort Y;
        if X is Y:
            now the item built is the result entry.

Check combining it with:
    if the item built is nothing or the item built is not in limbo,
        say "You can't combine [the noun] and [the second noun] into anything
useful." instead.

Carry out combining it with:
    move the item built to the holder of the noun;
    now the noun is nowhere;
    now the second noun is nowhere.

Report combining it with:
    say "You now have [an item built]."

Limbo is a container. Limbo contains a hookless fishing pole, a hooked line, and
a complete fishing pole.

Streamside is a room. The player carries a stick, a wire hook, and a string.

Table of Outcome Objects

    component list                      result
    {stick, string}                     hookless fishing pole
    {wire hook, string}                 hooked line
    {hooked line, stick}                complete fishing pole
    {hookless fishing pole, wire hook}  complete fishing pole

Test me with "combine stick with string / i / combine pole with hook / i".
```

This kind of implementation makes sense if we don't intend the player to take the fishing pole apart again, or to refer to any of its component parts once it is built. For an alternate approach that does allow assembled objects to be taken apart again, see "Some Assembly Required".

---

428 **Example Formicidae** *RB*

Manipulating the order in which items are handled after TAKE ALL.

Suppose we have an item that produces an interesting result the first time the player lifts it -- a rock with dangerous ants revealed underneath. The effect of the surprise is a little weakened, though, if the player sees that response as the result of a TAKE ALL, when it might be printed like this:

```
>[3] get all
tent peg: Taken.
water flask: Taken.
```

```
trading permit: Taken.
innocent-looking rock: You reach for the rock and turn it over to reveal a thriving
colony of flesh-eating ants. Needless to say, you drop the rock and jump back
with a decidedly effeminate scream. They can probably hear you all the way
back in the base camp.
rusty nail: Taken.

[Your score has just gone down by two points.]
```

The calm response to "rusty nail" looks odd now, and the score change is disconnected from the event that caused it.

To manage this, we might institute a system so that interesting objects are handled last in their list, like so:

```
"Formicidae"

Use scoring.

Section 1 - Procedure

The magic rule is listed before the generate action rule in the turn sequence
rules.

A thing has a number called dramatic potential.

This is the magic rule:
    let L be the multiple object list;
    if the number of entries in L is greater than 1:
        sort L in dramatic potential order;
        alter the multiple object list to L.

Section 2 - Scenario

The Foothills is a room. "The land has become hilly; though the soil is still mostly
coarse yellow sand, clumps of grass are able to grow in the shadier places.
Deep wagon ruts running from the southwest towards the mountains in the
northeast show where generations of caravans have already passed."

The water flask, the tent peg, and the trading permit are things in Foothills.

The rock is a thing in Foothills. Before printing the name of the rock when the
rock is not handled: say "innocent-looking ". The dramatic potential of the rock is
10.

The rusty nail is a thing in Foothills.

The ant colony is a fixed in place thing. "A busy group of ants are crawling to
and fro in the unaccustomed sun." Rule for deciding whether all includes the ant
colony while taking: it does not.

Instead of taking the rock when the rock is handled:
    say "It might still have a stray ant or two on it."
```

```
After taking the rock:
    now the rock is handled;
    move ant colony to the location;
    move the rock to the location;
    say "You reach for the rock and turn it over to reveal a thriving colony of flesh-
eating ants. Needless to say, you drop the rock and jump back with a decidedly
effeminate scream. They can probably hear you all the way back in the base
camp.";
    decrease score by 2.

Test me with "get peg / drop peg / get all / get rock".
```

Note that while one could also manipulate the object list to add or remove items at this stage, there's a simpler way to control what Inform considers "ALL" to mean in commands: see the activity "Deciding whether all includes" in the activities chapter.

429

Example Robo 2

RB

A robot which watches and records the player's actions, then tries to repeat them back in the same order when he is switched into play-back mode.

We have seen how we can make a robot that watches the player, then plays back the same actions again. A slightly more adventurous implementation would be to let the player create a whole series of named scripts which the robot will run on command.

To do this, we'll need each program to have a command that sets it off (stored as text, since this is the best way to capture and preserve arbitrary text entered by the player) and then the script of actions that must result:

```
"Robo 2"

Use dynamic memory allocation of at least 16384.

Chapter 1 - Programming

Section 1 - The Programs Themselves

The hard drive is a container. A program is a kind of thing. 15 programs are in
hard drive. A program has some text called the starter command. A program has
a list of stored actions called the script. Understand the starter command
property as describing a program.

Rule for printing the name of a program (called the target) which is not blank:
    say "[starter command of the target in upper case]".

Definition: a program is blank if the number of entries in its script is 0.

The current instruction name is some text that varies. The current instruction set
is a list of stored actions that varies.
```

Now, we want to let Robo learn new programs; for this purpose, we'll emulate the code from our previous implementation, so that Robo watches what the player does and stores those actions in his script:

```
Section 2 - Learning New Programs

Understand "learn [text]" as learning. Learning is an action applying to one topic.

Check learning:
    say "You have already learned all you need to know. Robo, however, remains
to be trained." instead.

Check Robo learning:
    if Robo is watching, say "Robo is already recording '[the current instruction
name]'." instead.

Carry out Robo learning:
    truncate the current instruction set to 0 entries;
    now the current instruction name is the topic understood;
    now Robo is watching.

Report Robo learning:
    say "'Learning [the current instruction name in upper case],' Robo replies."

After doing something when Robo is watching and Robo can see the player:
    now the actor is Robo;
    add the current action to the current instruction set;
    now the actor is the player;
    say "Robo watches you [the current action][one of], his yellow eyes lamp-like
and observant[or]. In his metal head, gears whirr[or], his brushed-copper
handlebar moustaches twitching[or] impassively[at random].";
    continue the action.
```

Of course, we also need to be able to switch learning mode off, and store any script learned this way. We'll also use the same STOP command to make Robo terminate a program he's in the middle of running.

```
Section 3 - Returning to Standby Mode

Understand "stop" as stopping. Stopping is an action applying to nothing.

Check stopping:
    say "The command is useful only for Robo." instead.

Check Robo stopping:
    if Robo is standing by, stop the action.

Carry out Robo stopping when Robo is watching:
    let N be a random blank program;
    if N is a program:
        now the starter command of N is the current instruction name;
        now the script of N is the current instruction set;
        say "'Stored [current instruction name in upper case].'";
```

```
        otherwise:
            say "FAILURE: no program slots remaining."

    Carry out Robo stopping:
        now Robo is standing by.

    Report Robo stopping:
        say "Robo is now standing by."
```

Next, we need to be able to play these programs back again. We'll give Robo a "current program" to store which program he's currently working on, and a number called "stage" which will record where he is in the script. Our previous implementation simply had Robo erase entries from his script list as he performed them, but this time we would like Robo to be able to remember and rerun the same scripts over and over, so we need something a little more subtle.

```
    Section 4 - Running Learned Programs

    Understand "run [any program]" as running. Running is an action applying to one
    visible thing.

    Check running:
        say "Only Robo can perform Robo's programs." instead.

    Check Robo running:
        if Robo is not standing by, stop the action.

    Unsuccessful attempt by Robo running:
        say "'ERROR: Robo can launch new programs only when on standby.'"

    Carry out Robo running:
        now the current program of Robo is the noun;
        now the stage of Robo is 1;
        now Robo is replaying.

    Report Robo running:
        say "'Running [the starter command in upper case],' Robo confirms."

    Every turn when Robo is replaying:
        let the chosen script be the script of the current program of Robo;
        let maximum be the number of entries in the chosen script;
        let N be the stage of Robo;
        let the next step be entry N of the chosen script;
        try the next step;
        increment the stage of Robo;
        if the stage of Robo is greater than the maximum:
            say "Robo's program ends, and he reverts to stand-by mode.";
            now Robo is standing by;
            now the stage of Robo is 1.
```

For the player's sanity, we should also provide a way to find out which programs Robo has stored in memory and what they do, so we design two listing commands:

```
    Section 5 - Listing Learned Programs
```

```
Understand "list programs" as requesting program list. Requesting program list
is an action applying to nothing.

Check requesting program list:
    say "You will have to ask Robo to list programs." instead.

Carry out Robo requesting program list:
    say "'The available program[if more than one program is not blank]s[end if] [is-
are list of programs which are not blank].'".

Understand "describe [any program]" or "list [any program]" as requesting script
of. Requesting script of is an action applying to one visible thing.

Check requesting script of:
    say "You will have to ask Robo to give you the script." instead.

Carry out Robo requesting script of:
    say "The script of [noun] is: [script of the noun]."
```

And to complete the suite, in case the player runs into Robo's fifteen-program limit:

```
Section 6 - Deleting Learned Programs

Understand "delete [any program]" as deleting. Deleting is an action applying to
one visible thing. Understand the command "erase" as "delete".

Check deleting:
    say "You will have to instruct Robo to delete [the noun]." instead.

Check Robo deleting (this is the can't delete except in standby rule):
    if Robo is not standing by, stop the action.

Unsuccessful attempt by Robo deleting:
    say "'ERROR: programs may only be deleted while Robo is in stand-by
mode.'" instead.

Carry out Robo deleting:
    truncate the script of the noun to 0 entries;
    now the starter command of the noun is "".

Report Robo deleting:
    say "'Program deleted.'"
```

Now we use pretty much the same set-up as before to test Robo's abilities:

```
Chapter 2 - The Scenario

The Experimentation Chamber is a room.

Robo is a man in the Experimentation Chamber. "Robo, your prototype tin
companion, stands awkwardly beside you[if watching], watching[end if]." Robo
can be watching, replaying, or standing by. Robo is standing by. Robo has a
program called the current program. Robo has a number called the stage.
```

```
Persuasion rule: persuasion succeeds.

The red block and the blue cylinder are things in the Experimentation Chamber.
The counter is a supporter in the Experimentation Chamber. The counter is
scenery.

Report Robo examining Robo:
    say "Robo examines each of his hands in turn, then each of his legs (bending
over mostly double in the middle to do this)." instead.

Report Robo examining the player:
    say "Robo stares at you, unblinkingly, for several seconds together[if a
random chance of 1 in 7 succeeds]. His left moustache-bar twitches
infinitesimally upward[end if]." instead.

Report Robo taking the cylinder:
    say "[one of][Robo] needs several attempts to get his metal fingers around
[the cylinder] -- they are not designed for grasping small objects elegantly. But at
last he succeeds[or]Once again, Robo struggles a bit before picking up [the
cylinder][stopping]." instead.

Test me with "test chocolate / test vanilla".

Test chocolate with "learn chocolate / stop / list programs / Robo, learn chocolate
/ take red / put all on counter / Robo, stop / Robo, list programs / Robo, run
chocolate / z / Robo, run chocolate / Robo, stop / z".

Test vanilla with "Robo, learn vanilla / take all / i / drop all / x robo / x me / Robo,
stop / Robo, list programs / Robo, list vanilla / Robo, run vanilla / z / z / robo,
delete vanilla / robo, stop / robo, list vanilla / robo, delete vanilla / robo, list
programs".
```

We could also have written this so that Robo learns new scripts by accepting the player's instructions, so that

```
>ROBO, LEARN LIBRARY THEFT
>ROBO, TAKE BOOK
>ROBO, EAST
>ROBO, STOP
```

...would store the script 'library theft' with the actions taking the book and going east. The implementation there would have been mostly identical, except that instead of an "after doing something..." rule, we would have captured commands as we asked Robo to perform them, and added those to the command list in progress. The alternative code might look something like this:

```
Before Robo doing something other than stopping when Robo is watching:
    add the current action to the current instruction set;
    say "'CHECK: [current action] added to script,' says Robo." instead.

Unsuccessful attempt by Robo doing something when Robo is watching:
    say "He does not actually perform the action."
```

430

Example Leopard-skin

RB

A maze that the player can escape if he performs an exact sequence of actions.

Suppose (as in Infocom's Leather Goddesses of Phobos) that we have a maze that the player can escape only by performing the correct sequence of actions in the correct order. One way to do this would be to keep a list of the player's most recent actions, and see whether these match up with the combination we have established as the maze's solution.

For instance:

```
"Leopard-skin"

The Fur-Lined Maze is a room. "This seemingly endless sequence of rooms is
decorated in a tasteful selection of exotic furs and gilded fixtures."

Clapping is an action applying to nothing. Understand "clap" as clapping.
Kweepaing is an action applying to nothing. Understand "kweepa" as
kweepaing.

Carry out clapping:
    say "You clap."

Carry out kweepaing:
    say "You holler 'KWEEPA!' triumphantly."

The maze-sequence is a list of stored actions that varies.

When play begins:
    add jumping to the maze-sequence;
    add clapping to the maze-sequence;
    add kweepaing to the maze-sequence.

The attempted-sequence is a list of stored actions that varies.

Every turn when the player is in the Fur-Lined Maze:
    truncate the attempted-sequence to the last two entries;
    add the current action to the attempted-sequence;
    if the attempted-sequence is the maze-sequence:
        say "That does it! You are instantly transported from the maze!";
        end the story finally.

Test me with "hop / clap / clap / hop / kweepa / hop / clap / kweepa".
```

431

Example The Facts Were These

RB

Creating a variant GIVE action that lets the player give multiple objects simultaneously with commands like GIVE ALL TO ATTENDANT or GIVE THREE DOLLARS TO ATTENDANT or GIVE PIE AND HAT

TO ATTENDANT. The attendant accepts the gifts only if their total combined value matches some minimum amount.

Occasionally it happens that we want to process an action on multiple items differently than we would if the player had just typed each of the individual actions separately. In this example, the reason is that we can only successfully GIVE items when their combined value passes a certain threshold amount; otherwise the recipient will reject them.

This works as an implementation of money, if we give value only to cash objects (though several other implementations of cash are available, most of which are simpler and more efficient). We could also imagine a mechanic like this being used for a bargaining or auction game as well, given a society that deals in objects rather than credits.

In order to consider all the items in the gift at once, we create an action that applies to multiple objects, but will in fact test the whole object collection during the first pass and print a definitive answer to whether the action succeeded. All subsequent times the game consults the rulebook will be stopped at the very beginning. No further processing will occur or output be printed.

```
"The Facts Were These"

Section 1 - Procedure
```

We start by creating the idea that everything in the game has a monetary value:

```
A price is a kind of value. $10 specifies a price. A thing has a price.

Understand "give [things preferably held] to [someone]" as multiply-giving it to.
Understand "give [things] to [someone]" as multiply-giving it to. Multiply-giving it
to is an action applying to two things.
```

A subtlety here: we say "things preferably held" to prefer items that the player is holding (so if the player has two dollars in hand and a third lies on the ground, he will use just the two he has).

The second grammar line allows Inform to match things that aren't held if it can't make up the list from things that are. If all three dollars are on the ground, the player can pick them up before spending them.

We do not, however, make multiply-giving apply to a "carried" item, because that will generate implicit takes of those items in a way that will mess up our action reporting. Instead, we're going to build the implicit takes into the system in a different way, one that permits us to collate the reports more attractively and print a short, one-sentence list of anything that the player had to pick up.

```
A thing can be given or ungiven. A thing is usually ungiven.
```

This is for record-keeping purposes so that we can print an attractive list of what was given at the end of the turn.

```
First check multiply-giving it to:
    if already gave at the office is true:
        stop the action.

Already gave at the office is a truth state that varies.
```

"Already gave at the office" is the perhaps-excessively-named flag that keeps track of whether we've already done this action once.

```
Check multiply-giving something to the player:
        now already gave at the office is true;
        say "You can hardly bribe yourself.[paragraph break]" instead;
```

The following rule is longish because it processes the entire list at once, generating implicit takes if necessary (but processing those implicit takes silently according to its own special rule, so that the output can be managed attractively). We are also, at the same time, calculating the total value of the player's offer.

```
Check multiply-giving it to:
    let L be the multiple object list;
    let bribe-price be $0;
    repeat with item running through L:
        if the player does not carry the item:
            abide by the ungivability rules for the item;
            carry out the implicitly taking activity with the item;
            if the player does not carry the item:
                now already gave at the office is true;
                say "You can't include [the item] in your bribe, since you're not holding
[them]![paragraph break]" instead;
        increase bribe-price by the price of item;
    if the number of entries in the recently-collected list is greater than 0:
        repeat with item running through the recently-collected list:
            now item is marked for listing;
        say "You pick up [the list of marked for listing things] and make your offer.
[run paragraph on]";
        now everything is unmarked for listing;
    if the bribe-price is less than the price of the second noun:
        now already gave at the office is true;
        say "[The second noun] angrily rejects your piffling bribe.[paragraph break]"
instead.
```

The bit about making some items "marked for listing", above, rather than printing the list directly, is that using the "[the list of....]" syntax guarantees that Inform will respect grouping rules in writing its description. For instance, if the player has automatically taken all three dollars, the output will say "the three dollars" instead of "the dollar, the dollar, and the dollar."

```
Carry out multiply-giving it to:
    let L be the multiple object list;
    repeat with item running through L:
        now the second noun carries the item;
        now the item is given;
    now already gave at the office is true;
```

```
Report multiply-giving it to:
    say "[The second noun] rather shamefacedly tucks [the list of given things]
away into a pocket.[paragraph break]".
```

Now we create our own variation of implicitly taking in order to customize the output for the multiply-giving action. The "ungivability rules" should disallow any object that the player absolutely cannot take, because we want "carry out the implicitly taking activity" to succeed every time -- and therefore not print out any less-attractive results from implicit takes that don't succeed. Otherwise, the player's GIVE TREE AND DOG TO ATTENDANT might produce the reply "That's fixed in place" -- without specifying which object is fixed in place.

Because of the way this works, we will want to be careful: if we have any "instead of taking..." rules for special objects in the game, we should be sure to mirror those with an ungivability rule to print something more suitable in the case that the player tries taking that object as part of the multiple giving action.

```
The ungivability rules are an object-based rulebook.

An ungivability rule for a person:
    now already gave at the office is true;
    say "Slavery is illegal.[paragraph break]" instead.

An ungivability rule for something (called the item) which is enclosed by
someone who is not the player:
    now already gave at the office is true;
    say "[The item] [aren't] yours to give.[paragraph break]" instead.

An ungivability rule for something which encloses the player:
    now already gave at the office is true;
    say "You don't want to end up as part of the gift.[paragraph break]" instead;

An ungivability rule for something (called the item) which is part of something:
    now already gave at the office is true;
    say "[The item] [are] attached to [a random thing which incorporates the item]
[paragraph break]" instead.

An ungivability rule for something (called the item) which is scenery:
    now already gave at the office is true;
    say "[The item] [are] unremovable.[paragraph break]" instead.

An ungivability rule for something (called the item) which is fixed in place:
    now already gave at the office is true;
    say "[The item] [are] fixed in place.[paragraph break]" instead.

An ungivability rule for a direction (called the item):
    now already gave at the office is true;
    say "[The item] [are] not susceptible to giving.[paragraph break]" instead.

Rule for implicitly taking something (called target) while multiply-giving:
    silently try taking the target;
    if the player carries the target:
        add the target to the recently-collected list.
```

```
The recently-collected list is a list of objects that varies.
```

And since we don't want to list the individual objects separately:

```
The selectively announce items from multiple object lists rule is listed instead of
the announce items from multiple object lists rule in the action-processing rules.

This is the selectively announce items from multiple object lists rule:
    if multiply-giving:
        do nothing;
    otherwise:
        if the current item from the multiple object list is not nothing:
            say "[current item from the multiple object list]: [run paragraph on]".
```

And now, since this ought to work symmetrically if the player provides just one high-value item:

```
Check giving something to someone:
    if the price of the noun is less than the price of the second noun:
        say "[The second noun] angrily rejects your piffling bribe." instead.
```

As we've seen elsewhere, the giving action by default returns a refusal, but is also written to start working if we remove the blockage. So we do that here, and revise the report rule to match the report rule we have for multiple giving.

```
The block giving rule is not listed in any rulebook.

The new report giving rule is listed instead of the standard report giving rule in
the report giving it to rules.

This is the new report giving rule:
    say "[The second noun] rather shamefacedly tucks [the noun] away into a
pocket."
```

After each instance of the multiply-giving action, we need to clear the variables we used to track its state. We could do this in "Before reading a command", but that's unsafe because the player might type GIVE PIE AND CAP TO ATTENDANT. GIVE DOLLARS TO ATTENDANT. all on a single line, and we would like to be able to clear the variables between one action and the next. The correct place to attach this behavior is immediately before the generate action rule, thus:

```
The before-generation rule is listed before the generate action rule in the turn
sequence rules.

This is the before-generation rule:
    now every thing is ungiven;
    now already gave at the office is false;
    truncate the recently-collected list to 0 entries.

Section 2 - Scenario

The Morgue Office is a room. "This is not the Morgue itself; this is only its outer
office. The familiar room full of silver drawers and cold air lies beyond."
```

```
The Morgue Attendant is a man in the Morgue Office. "The Attendant has seen
you come through a number of times, and is becoming suspicious of your
abiding interest in dead people." The description is "The Morgue Attendant is
fifty-four years, six months, five days, and three minutes old." The price of the
Morgue Attendant is $3.

A dollar is a kind of thing. The player carries three dollars. The price of a dollar is
always $1.

The player carries a miniature rhubarb pie. The price of the miniature rhubarb
pie is $5.

The player carries a knitted cap. The price of the knitted cap is $2.

Test me with "test dollars / purloin three dollars / test multi-line / purloin three
dollars / purloin pie / purloin cap / test specificity / purloin three dollars / test
largesse / test mixed-gift".

Test multi-line with "give dollar and pie to attendant. give dollars and cap to
attendant".

Test dollars with "drop all / give dollar to Morgue Attendant / give dollars to
Morgue Attendant / get dollars / give dollars to morgue attendant / purloin three
dollars / drop dollars / give dollars to Morgue Attendant".

Test specificity with "give three dollars to Morgue Attendant".

Test largesse with "give pie to Morgue Attendant".

Test mixed-gift with "give dollar and cap to Morgue Attendant / get cap / give
dollar and cap to morgue attendant / give me and dollar to attendant".
```

PURLOIN, used in the tests here, is a special debugging command that allows the player to acquire objects that wouldn't otherwise be possible to take. It is only active in non-release versions of the story. For more about debugging commands, see the chapter on Testing and Debugging.

432

Example **Circle of Misery** *RB*

Retrieving items from an airport luggage carousel is such fun, how can we resist simulating it, using a list as a ring buffer?

The only point we need to be careful about is that the carousel is simulated twice over, in the following text: once in the built-in way that objects are inside other objects, so that the luggage items are objects contained in the carousel object; but then again by the "circle of misery" list, a ring buffer keeping track of what order things are in. We need to be careful that these two records do not fall out of synchrony: anything put into the carousel must be added to the list, anything taken out must be removed. (In fact we forbid things to be added, for simplicity's sake.)

```
"Circle Of Misery"
```

```
Luggage item is a kind of thing. The blue golf bag, the leopardskin suitcase, the
green rucksack and the Lufthansa shoulder-bag are luggage items.

Heathrow Baggage Claim is a room. The carousel is a container in Heathrow.
"The luggage carousel, a scaly rubbered ring, does for the roundabout what
Heathrow Airport does for the dream of flight: that is, turns the purest magic into
the essence of boredom, only with extra stress. [if the number of entries in the
circle of misery is 0]For once it stands idle. Perhaps it's broken.[otherwise]The
baggage approaching you now: [the circle of misery with indefinite articles]."

The circle of misery is a list of objects that varies.

When play begins:
    now all the luggage items are in the carousel;
    add the list of luggage items to the circle of misery.
```

The list "circle of misery" is our ring, in which entry 1 is considered to be the position of whichever bag is currently frontmost. And here it goes, round and round:

```
Every turn when the number of entries in the circle of misery is not 0:
    rotate the circle of misery;
    let the bag be entry 1 of the circle of misery;
    say "The carousel trundles on, bringing [a bag] to within reach."

After taking a luggage item (called the bag):
    remove the bag from the circle of misery, if present;
    say "Taken."

Before doing something with a luggage item (called the bag) which is in the
carousel:
    if the bag is not entry 1 of the circle of misery, say "[The bag] is maddeningly
out of range. You'll have to wait for it to come round." instead.

Instead of inserting something into the carousel, say "In recent years, the
authorities have tended to frown upon depositing bags in random places at
airports."

Test me with "get suitcase / get suitcase / get suitcase / get suitcase / look / get
golf bag / look / get golf bag".
```

433

Example **Eyes, Fingers, Toes**

RB

A safe with a multi-number combination, meant to be dialed over multiple turns, is implemented using a log of the last three numbers dialed. The log can then be compared to the safe's correct combination.

It is not difficult to implement a safe which can be set to a single number to open; but a more common scenario in the real world is for the safe to open on a sequence of numbers when they have been dialed in the right order.

For IF, this means that we have to keep running track of the last N digits the player has dialed, dropping the first digit and adding a new one to the end each time the player re-dials the safe. This is a perfect occasion for lists:

```
"Eyes, Fingers, Toes"

The Addams Wine Cellar is a room. It contains a closed lockable locked
container called a safe.

The safe has a list of numbers called the current combination.

The safe has a list of numbers called the true combination. The true combination
of the safe is {2, 10, 11}.

Understand "set [something] to [a number]" as setting it numerically to. Setting it
numerically to is an action applying to one thing and one number.

Instead of examining the safe:
    if the number of entries in the current combination of the safe is 0,
        say "You haven't dialed the safe to any combination yet.";
    otherwise say "You have dialed the safe to [the current combination of the
safe].".

Check setting something numerically to (this is the block setting numerically
rule):
    say "[The noun] cannot be set."

Instead of setting the safe numerically to the number understood:
    truncate the current combination of the safe to the last 2 entries;
    add the number understood to the current combination of the safe;
    if the safe is locked and the current combination of the safe is the true
combination of the safe:
        say "You dial [the number understood], and [the safe] gives a joyous
CLICK.";
        now the safe is unlocked;
    otherwise if safe is unlocked and the safe is closed and the current
combination of the safe is not the true combination of the safe:
        say "You spin the dial, and [the safe] snicks locked.";
        now the safe is locked;
    otherwise:
        say "You dial [the number understood] on the safe."

Test me with "x safe / set safe to 10 / x safe / set safe to 29 / x safe / set safe to
2 / x safe / set safe to 10 / x safe / set safe to 11 / open safe / set safe to 14 /
close safe / set safe to 15 / open safe".
```

434

Example The Fibonacci Sequence *RB*

The modest Leonardo Fibonacci of Pisa will be only too happy to construct his sequence on request, using an array.

Fibonacci (a posthumous nickname) spread Arabic mathematical learning across Europe in the 13th century, and it's curious that his name lives on only for a single sequence.

```
"The Fibonacci Sequence"

Pisa is a room. Leonardo Fibonacci is a man in Pisa. "The modest Italian
mathematician, Leonardo Fibonacci (1170-1250), beams at you."

Sequencing is an action applying to one number. Understand "sequence
[number]" as sequencing.

Instead of sequencing, say "You make a feeble attempt, sketching in the sand,
but it goes nowhere. Leonardo is sympathetic. 'Often goes wrong for me, too,
actually. I didn't even invent the thing - the ancient Indians knew about it first.'"

Persuasion rule for asking Leonardo to try sequencing: persuasion succeeds.

Report Leonardo sequencing:
    let N be the number understood;
    say "Leonardo scratches his head and makes self-deprecating remarks,
before coming up with [the first N terms of the Fibonacci sequence]."
```

An array need not be fixed in length, as the following example shows:

```
To decide what list of numbers is the first (F - a number) terms of the Fibonacci
sequence:
    let the Fibonacci sequence be {1, 1};
    let N be 3;
    while N < F:
        let the last term be entry (N - 1) of the Fibonacci sequence;
        let the penultimate term be entry (N - 2) of the Fibonacci sequence;
        let the next term be the last term plus the penultimate term;
        add the next term to the Fibonacci sequence;
        increment N;
    decide on the Fibonacci sequence.

Test me with "sequence 20 / leonardo, sequence 20".
```

The result of "the first 20 terms of the Fibonacci sequence" is "1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584 and 4181". This is a sequence which has a knack of turning up in odd places - it was found in the 1970s to be related to the rings of florets in a sunflower, for instance - and here it is in a book about interactive fiction.

435

Example I Didn't Come All The Way From Great Portland Street

RB

In this fiendishly difficult puzzle, which may perhaps owe some inspiration to a certain BBC Radio panel game (1967-), a list is used as

a set of actions to help enforce the rule that the player must keep going for ten turns without hesitation, repetition, or deviating from the subject on the card.

There is very little to this, in fact. The tricky rule to enforce is Repetition: the player is forbidden to repeat any previously tried action. We keep track of this by keeping a set of past actions, which for want of a better term is called the "tally". All we need to do is:

```
if the current action is listed in the tally, challenge for "Repetition of [the current action]!";
otherwise add the current action to the tally.
```

Note that the tally can never contain duplicates, and that when, at the end of the round, we print it out, we sort it first - this makes a more natural-looking sentence. (Sorting a list of actions uses the natural order for actions: compare the sequence on the Actions page of the Index.) The full text, then, is:

```
"I Didn't Come All The Way From Great Portland Street"

The Paris Theatre is a room. An instrument is a kind of thing. The violin, the tuba, the xylophone and the triangle are instruments. The violin is inside the case. The tuba, the xylophone, the radish, the case, the bust of Nicholas Parsons, the purple felt hat and the triangle are in the Paris Theatre.

The Round is a scene. The Round begins when play begins. The Round ends when the turn count is 10.

The tally is a list of stored actions that varies.

When the Round begins:
    say "'And the subject on the card is... musical instruments. Will you carry out for us something to do with that, please, for ten turns starting - now!'"

When the Round ends:
    sort the tally;
    say "Phweeep![paragraph break]'So, when the whistle goes ten turns are up, you get a point for acting when the whistle blows, and in that round you entertained us by [the tally], and you also get a bonus point for keeping going until the whistle went.'";
    end the story finally.

To challenge for (infraction - text):
    say "Bzzzzt! 'And [one of]Clement Freud[or]Derek Nimmo[or]Kenneth Williams[or]Peter Jones[at random] has challenged.'[paragraph break]'[infraction]'[paragraph break]'Well, as it's your first time playing the game, and the audience was enjoying your contribution so much, I will disallow the challenge, you have [10 minus the turn count] turn[s] left on musical instruments, starting... now!"

Before doing something:
    if the current action is listed in the tally, challenge for "Repetition of [the
```

```
current action]!" instead;
    otherwise add the current action to the tally;
    if waiting, challenge for "Hesitation!" instead;
    if not looking and not waiting and the noun is not an instrument and the
second noun is not an instrument, challenge for "Deviation!" instead.

Test me with "look / wait / examine bust / take tuba / get triangle / hit xylophone /
get tuba / examine tuba / get violin".
```

(The Paris Theatre in Lower Regent Street, London, was for many years the home of BBC radio panel games.)

436

Example Lugubrious Pete's Delicatessen *RB*

In this evocation of supermarket deli counter life, a list is used as a queue to keep track of who is waiting to be served.

First, to set the scene:

```
"Lugubrious Pete's Delicatessen"

The Supermarket is west of the Delicatessen Counter. Lugubrious Pete is in the
Delicatessen. "Lugubrious Pete, dolefully slicing meats and cheeses, serves at
the counter." Alice, Beth, Gemma, Delia, and Eliza are women in the
Supermarket.

The deli queue is a list of objects that varies.
```

Two processes compete here: one that fills the queue, the other which will empty it. The first process is the one which brings shoppers in to the counter, joining the back of the queue, which is where "add ... to ..." puts new entries by default:

```
Every turn when a woman is in the Supermarket and a random chance of 2 in 3
succeeds (this is the customer arriving rule):
    let the customer be a random woman in the Supermarket;
    now the customer is in the Delicatessen;
    if the player is in the Supermarket, say "[The customer] walks into the
Delicatessen.";
    if the player is in the Delicatessen, say "[The customer] comes in from the
Supermarket, and [if the number of entries in the deli queue is 0]can't believe her
luck. The counter is free![otherwise]resignedly queues behind [the deli queue].";
    add the customer to the deli queue.
```

The competing process is the one which serves shoppers and thus gets rid of them again: unfortunately, it is slower. But Pete is fair if inefficient, and serves the customers in strict order of arrival. Each served customer is removed from the front of the list, and the others therefore all move up a place.

```
Every turn when the number of entries in the deli queue is not 0 and a random
chance of 1 in 3 succeeds (this is the customer being served rule):
    let the customer be entry 1 of the deli queue;
```

```
    if the player is in the Delicatessen, say "Pete gives a droopy expression as he
serves [customer], who nevertheless brightens and leaves.";
    if the player is in the Supermarket, say "[customer] emerges cheerfully from
the Delicatessen Counter, and goes about her regular shopping.";
    now the customer is in the Supermarket;
    remove entry 1 from the deli queue.

Instead of waiting in the Delicatessen when the number of entries in the deli
queue is not 0, say "Time passes, for [deli queue] quite as much as for yourself."

Test me with "wait / wait / wait / east / wait / wait / wait / wait / wait".
```

That completes the example, but here is a variation to show that queues need not empty from the front. The Deli already looks a pretty sexist establishment, with the customers all being women, but it is about to get a whole lot worse:

```
Modesty is a kind of value. The modesties are positively prim, buttoned up,
modest, flirty, revealing and downright immodest. Every woman has a modesty.
Alice is positively prim. Beth is downright immodest. Gemma is modest. Delia is
flirty. Eliza is revealing.
```

We could then rewrite the service rule like so:

```
Every turn when the number of entries in the deli queue is not 0 and a random
chance of 1 in 3 succeeds (this is the customer being served rule):
    let Pete's preference be the deli queue;
    sort Pete's preference in reverse modesty order;
    let the customer be entry 1 of Pete's preference;
    let the first in line be entry 1 of the deli queue;
    if the player is in the Delicatessen, say "[if the customer is the first in line]Pete
gives a droopy expression as he serves [the customer], who nevertheless
brightens and leaves.[otherwise]Outrageously, Pete scans the queue, notices
[the customer] in her [modesty of the customer] clothes, and serves her next,
while [the first in line] glares at him.";
    if the player is in the Supermarket, say "[The customer] emerges cheerfully
from the Delicatessen Counter, and goes about her regular shopping.";
    now the customer is in the Supermarket;
    remove the customer from the deli queue.
```

It is now heartbreakingly difficult for Alice to obtain her sliced chorizo sausage.

437

Example **Sieve of Eratosthenes**

RB

The haughty Eratosthenes of Cyrene will nevertheless consent to sieve prime numbers on request.

In the words of Wikipedia: "Eratosthenes of Cyrene (Greek Ερατοσθένης; 276 BC-194 BC) was a Greek mathematician, poet, athlete, geographer and astronomer." In the words of Tom Lehrer: "It's people like that who make you realise how little you've achieved in life."

A prime number is a number greater than 1 which is not a multiple of anything, so we can find the primes by starting with all the numbers and sieving out all the multiples of 2, then all the multiples of 3, and so on. Here we make our sieve of the unacceptable numbers (the "composite" or non-prime ones) first, then form a list of all the numbers, then sieve out the composites: what are left must be the primes.

```
"Sieve of Eratosthenes"

Alexandria is a room. Eratosthenes is a man in Alexandria. "The haughty Greek
mathematician, Eratosthenes, glowers at you."

Sieving is an action applying to one number. Understand "sieve [number]" as
sieving.

Instead of sieving, say "You make a feeble attempt, sketching in the sand, but it
goes nowhere. Eratosthenes smirks. 'I expect your friends call you gamma,
then?'"

Persuasion rule for asking Eratosthenes to try sieving: persuasion succeeds.

Report Eratosthenes sieving:
    let N be the number understood;
    let the composites be a list of numbers;
    let I be 2;
    while I times I is at most N:
        if I is not listed in the composites:
            let J be I times 2;
            while J is at most N:
                add J to the composites, if absent;
                increase J by I;
        increment I;
    sort the composites;
    let the primes be a list of numbers;
    repeat with P running from 2 to N:
        add P to the primes;
    remove the composites from the primes;
    say "Eratosthenes sketches lines in the sand with the air of much practice.
'The primes up to [N] are [the primes]. The composites are [the composites].'"

Test me with "sieve 10 / eratosthenes, sieve 100".
```

While this could all be done more efficiently with an array, that's only because what we are sieving are numbers: sieving is a technique which can be used for non-numerical decisions, too.

438

Example Your Mother Doesn't Work Here

RB

Your hard-working mother uses a list as a stack: urgent tasks are added to the end of the list, interrupting longer-term plans.

Suppose the player's mother is supposed to be cleaning the living room, but she can be interrupted by the need to pick up things the player has dropped. New tasks are added to the end of her "current plan" list; every turn, she attempts to do whatever is the last entry on that list.

```
"Your Mother Doesn't Work Here"

A person has a list of stored actions called the current plan.

Every turn:
    repeat with culprit running through people who are not the player:
        if the number of entries in current plan of the culprit is greater than 0:
            let N be the number of entries in the current plan of the culprit;
            try entry N of the current plan of the culprit;
            remove entry N from the current plan of the culprit.

The Living Room is a room. It contains a somewhat muddy Persian rug. Your
mother is a woman in the Living Room.

West of the Living Room is the Kitchen.

Instead of your mother rubbing the rug:
    say "Your mother scrubs the stained areas of the rug, muttering to herself."

Instead of taking something:
    say "Nah, Mom'll get that."

Report your mother taking something:
    say "Your mother picks up [the noun][one of], sighing deeply[or], jaw tight[or],
with assorted comments on your manners[or]; to judge from her comments, she
is also indulging in a pleasant fantasy about Swiss boarding schools[stopping]."
instead.

When play begins:
    add mother going west to the current plan of mother;
    add mother rubbing the rug to the current plan of mother.

Every turn:
    if mother is not in the Living Room, end the story finally.

Carry out dropping something:
    add mother taking the noun to the current plan of mother.

The player carries some dirty socks, some dirty shoes, a dirty hat, a pair of dirty
trousers, and a backpack.

Test me with "drop socks / z / drop shoes / drop hat / drop all / z / z".
```

As goal-seeking goes, this is fairly rudimentary; "Boston Cream" provides an alternative (and slightly more sophisticated approach), but for really complex goal-seeking characters, it is probably best to turn to the character extensions designed for Inform.

Chapter 22: Advanced Phrases

 Contents of *Writing with Inform*

 Chapter 21: Lists

 Chapter 23: Figures, Sounds and Files

 Indexes of the examples

§22.1. A review of kinds

Most of the time, what's created in an Inform source text will have a name which can be used as a value - sometimes openly so, sometimes not. In this book, we haven't gone out of our way to make that point, because there was no real need to do so. It's possible to make heavy use of rulebooks and write large-scale Inform projects without ever needing to use a rulebook's name as a value in its own right, for example. But if we want to create sophisticated extensions to Inform, or to use modern techniques such as functional and generic programming, we need to be fluent in the language of kinds.

Inform's language of kinds has four ingredients: base kinds, constructions, kind variables and kinds of kinds.

1. Base kinds. Inform provides the following base kinds for values:

```
object, number, real number, time, truth state, text, snippet, Unicode character, action,
scene, table name, equation name, use option, action name, figure name, sound name,
external file, rulebook outcome, parser error
```

together with a few others, such as "response" and "verb", to do with linguistic features.

And Inform allows us to create new base kinds either by making more specialised kinds of object:

```
A geographical layout is a kind of object.
A marmoset is a kind of animal.
```

Or by making new enumerations or arithmetical kinds:

```
Distance is a kind of value. 10km specifies a distance.
Colour is a kind of value. Red, green and blue are colours.
```

2. Constructions. These are ways to make new kinds from existing ones. The construction most often used is "list of...". For any kind K, there is a kind called list of K. So the range of possible kinds in Inform is actually infinite, because:

```
number
list of numbers
list of lists of numbers
list of lists of lists of numbers
...
```

are all different from each other. Inform has nine constructions, as follows:

```
list of K
description of K
relation of K to L
K based rule producing L
K based rulebook producing L
activity on K
phrase K -> L
K valued property
K valued table column
```

Some of these have appeared in previous chapters, but in abbreviated form. For example, "rulebook" abbreviates "action based rulebook producing nothing", and "either/or property" is a synonym for "truth state valued property". The kinds of descriptions and phrases haven't been covered yet, but are coming up in the sections following.

These constructions can of course be combined:

```
phrase (relation of numbers to colours, number) -> list of colours
```

Brackets can be used to clarify matters:

```
phrase (phrase number -> (phrase number -> number)) -> nothing
```

Nothing will make that a simple idea, but it's unambiguous and can be puzzled out with practice.

3. Variables. In a way, that's everything: there are just base kinds and constructions on them, and those construct every possible kind in Inform. But the language we use to describe kinds is larger than that, because it allows us to describe many kinds at once, in the same way that Inform reads the word "something" as applying to many objects at once, not as a single object.

Kind variables will be covered later in the chapter, but the idea is that:

```
To hunt for (needle - value of kind K) in (haystack - list of Ks): ...
```

allows us to describe the kinds acceptable in a phrase so that a wide range of possibilities are allowed. The above matches both:

```
hunt for 4 in { 2, 3, 4, 5 };
hunt for "fish" in { "lemon sauce", "fish", "garden peas" };
```

The letter K in the definition stood for any kind; in the first use of "hunt" here, K turned out to be "number", and in the second it was "text". On the other hand Inform would reject:

```
hunt for 4 in { containment relation, support relation };
```

because there is no kind K which can make this match the definition.

There are potentially 26 kind variables, A to Z, though it's customary to use them in the order K, L, M, ..., and it's very rare to need more than two at a time.

4. Kinds of kind. Inform understands several names which look as if they are kinds, but actually aren't:

```
value, arithmetic value, enumerated value, sayable value
```

(Again, these are built in to Inform.) They are not kinds because they're just too loose and vague. Instead, they can be used in phrase definitions to match against multiple possibilities:

```
To announce (X - sayable value): say "I declare that [X] has arrived."
```

This makes "announce X" work for any value X which can be said. All the same, "sayable value" is not a kind. It could never be safe for this to be the kind of a variable, because Inform would never know what could be done with the contents (except that it could be printed out).

5. Secret inner workings. There isn't a fifth ingredient, but if there were, it would be a handful of names used in matching some of the core built-in phrases of Inform which have so-called inline I6 definitions. These are not intended for anyone else to use, and are just an internal convenience; they aren't publicly documented and might change without notice. They don't describe kinds at all, because they tell the matcher to look for something else instead. For instance, there's one called "condition", which means "match a condition rather than a value". They appear in red ink in the Phrasebook index.


Start of Chapter 22: Advanced Phrases


Back to Chapter 21: Lists: §21.11. Variations: arrays, logs, queues, stacks, sets, sieves and rings


Onward to §22.2. Descriptions as values

---

§22.2. Descriptions as values

In the chapter on Descriptions, we saw that a description can be any source text which describes one or more objects: it might be as simple as "the Corn Market", or as complicated as "open containers which are in dark rooms". More or less the only restriction is that it must be unambiguous as to what counts and what does not: "three containers" is ambiguous as a description because it does not say which three.

We've now seen several interesting tricks with descriptions. In fact, if D is a description, then

```
say "You gaze mournfully at [the list of D].";
let the tally be the number of D;
let the surprise prize be a random D;
```

```
repeat with item running through D:
    ...
```

are all standard things to do. These examples make it look as if it must be possible to define phrases which act on descriptions, and in fact it is, because a description can be a value in itself. For example,

```
even numbers
open containers which are in dark rooms
```

are values of kind "description of numbers" and "description of objects" respectively. In general, if K is any kind then "description of K" is also a kind. Here is how we might make use of that:

```
To enumerate (collection - a description of objects):
    repeat with the item running through the collection:
        say "-- [The item]."
```

This makes "enumerate lighted rooms" run off a list of lighted rooms in a textual format different from the standard one produced by "say the list of lighted rooms". Inside the definition, "collection" can be used wherever a description might be used: here, for instance, we use it as the range for the repeat loop. (That's only possible because the range is limited in size: Inform wouldn't have allowed us to range through, say, all texts.)

Purely as a convenience, we can also write "member of" or "members of" in this context. For instance, in the enumerate definition, it would have been just as good to write "...running through the members of the collection..." in the repeat. (Similarly, we could write "number of members of the collection" or "a random member of the collection", which looks grammatically tidier than writing "number of the collection" or "random of the collection" - though in fact both of these do work.)

Finally, it's sometimes useful in an abstract situation to test

if (value) **matches** (description of values)**:**

This condition is true if the value matches the description; the kinds must be compatible, or Inform will issue a problem message. There is no point using this for cases where the description is given explicitly:

```
if 4 matches even numbers, ...
```

because it is easier to write just:

```
if 4 is an even number, ...
```

So this condition is only useful when the description is stored in some variable, and its identity is not known.

Start of Chapter 22: Advanced Phrases
Back to §22.1. A review of kinds
Onward to §22.3. Phrases as values
Example 439: **Curare** A phrase that chooses and names the least-recently selected item from the collection given, allowing the text to cycle semi-randomly through a group of objects.

---

§22.3. Phrases as values

Given any two kinds K and L, the kind "phrase K -> L" is now a kind. (This is meant to look like a mathematical function arrow.) For example, the phrase defined by

```
To decide which number is the square of (N - a number): ...
```

has the kind "phrase number -> number". Brackets and commas are used if the phrase combines several values, so

```
To decide which text is (T - text) repeated (N - a number) times: ...
```

has the kind "phrase (text, number) -> text". The word "nothing" is used if there are no values in, or no value out - thus

```
To decide which number is the magic target: ...
```

has kind "phrase nothing -> number", and

```
To dig (eastward - length) by (northward - length): ...
```

has the kind "phrase (length, length) -> nothing".

But how are we to get at these values? The answer is that we need to give a phrase a name in order to do so. For example:

```
To decide what number is double (N - a number) (this is doubling):
    decide on N plus N.
```

This is the same syntax used to name rules, and the idea is the same. If we

try "showme doubling", the result is

```
phrase number -> number: doubling
```

The main thing we want to do with a phrase is to apply it. So:

```
showme doubling applied to 2;
```

produces

```
"doubling applied to 2" = number: 4
```

There are versions of "applied to" for phrases applied to 0 to 3 values:

(phrase nothing -> value) **applied ... value**

This phrase produces the result of applying the given phrase, which must be one which takes no values itself.

(phrase value -> value) **applied to** (value) **... value**

This phrase produces the result of applying the given phrase, which must be one which takes one value itself.

(phrase (value, value) -> value) **applied to** (value) **and** (value) **... value**

This phrase produces the result of applying the given phrase, which must be one which takes two values itself.

(phrase (value, value, value) -> value) **applied to** (value) **and** (value) **and** (value) **... value**

This phrase produces the result of applying the given phrase, which must be one which takes three values itself.

So for example:

```
F applied
F applied to V
F applied to V and W
F applied to V and W and X
```

For phrases which do not produce a value, we use "apply":

apply (phrase nothing -> nothing)

This phrase causes the given phrase to be applied. It must be one which takes no values itself.

apply (phrase value -> nothing) **to** (value)

This phrase causes the given phrase to be applied. It must be one which takes one value itself.

apply (phrase (value, value) -> nothing) **to** (value) **and** (value)

This phrase causes the given phrase to be applied. It must be one which takes two values itself.

apply (phrase (value, value, value) -> nothing) **to** (value) **and** (value) **and** (value)

This phrase causes the given phrase to be applied. It must be one which takes three values itself.

Thus:

```
apply F;
apply F to V;
apply F to V and W;
apply F to V and W and X;
```


Start of Chapter 22: Advanced Phrases


Back to §22.2. Descriptions as values


Onward to §22.4. Default values for phrase kinds

---

§22.4. Default values for phrase kinds

The default value for "phrase K -> nothing" is a phrase which does nothing. For example, if we write:

```
let S be a phrase numbers -> nothing;
```

then S is created holding the default phrase numbers -> nothing, and if we then try it out with:

```
apply S to 17;
```

we will indeed find that nothing happens.

The default value for "phrase K -> L" is a phrase which, no matter what value of K it applies to, always produces the default value of L. (It's a sort of equivalent of the zero function in mathematics - indeed that's exactly what it is, if L is "number".) So:

```
let Q be a phrase numbers -> times;
showme Q;
showme Q applied to 4;
showme Q applied to -7;
```

produces:

```
"q" = phrase number -> time: default value of phrase number -> time
"q applied to 4" = time: 9:00 am
"q applied to -7" = time: 9:00 am
```

Here Q is set to the default phrase because we didn't give it any other value; it has the name we might expect ("default value of phrase number -> time") and it works as advertised, producing 9:00 am no matter what number is fed in.

More ambitiously, and supposing that we have a kind called "colour" whose first possible value is "red":

```
let R be a phrase numbers -> (phrase numbers -> colours);
showme R;
showme R applied to 3;
showme (R applied to 3) applied to 2;
```

produces:

```
"r" = phrase number -> (phrase number -> colour): default value of phrase
number -> (phrase number -> colour)
"r applied to 3" = phrase number -> colour: default value of phrase number
-> colour
"( r applied to 3 ) applied to 2" = colour: red
```

Start of Chapter 22: Advanced Phrases

Back to §22.3. Phrases as values

Onward to §22.5. Map, filter and reduce

---

§22.5. Map, filter and reduce

When a mass of computations has to be done, the traditional approach is to work through them in a "repeat" loop. One modern alternative, growing in popularity, is to form a list of inputs; then apply the same computation to each input in turn to form a list of results (this is called "mapping"); throw out any bad or unwanted results ("filtering"); and then combine the surviving results into a single composite answer ("reducing", though some programming languages call this "folding" or "accumulation"; it's a much-reinvented idea).

Inform provides all three of these fundamental list-processing operations. There is no special term for a "map", because Inform treats it as another case of "applied to".

(phrase value -> value) **applied to** (list of values) **... value**

This phrase takes the list, applies the phrase to each entry in the list, and forms a new list of the result. Example:

```
To decide what number is double (N - a number) (this is doubling):
    decide on N plus N.
```

Then "doubling applied to 2" produces 4, by the simpler definition of "applied to", but also:

```
doubling applied to {2, 3, 4}
```

produces the list {4, 6, 8}.

More divertingly, suppose we define:

```
To decide what text is the longhand form of (N - a number)
    (this is spelling out):
    decide on "[N in words]".

To decide what text is the consonant form of (T - text)
    (this is txtng):
    replace the regular expression "<aeiou>" in T with "";
    decide on T.
```

Then we can write a chain of three maps in succession:

```
txtng applied to spelling out applied to doubling applied to {3, 8, 4, 19, 7}
```

to produce the value {"sx", "sxtn", "ght", "thrty-ght", "frtn"}.

Next, filtering. Here we make use of descriptions, in order to say what values will be allowed through the filter. So:

filter to (description of values) **of** (list of values) **... value**

This phrase produces a new list which is a thinner version of the one given, so that it contains only those values which match the description given. Example:

```
filter to even numbers of {3, 8, 4, 19, 7}
```

produces {8, 4}, with the values 3, 19, and 7 failing to make it through. A sufficiently fine filter may well thin out a list to a single entry, or even no entries at all, but the result is always a list.

To get the full effect of filtering, we probably need to define an adjective or two. For example:

```
Definition: a text (called T) is lengthy if the number of characters in it is greater than 6.
```

We can then write:

```
filter to lengthy texts of spelling out applied to {15, 2, 20, 29, -4}
```

which produces {"fifteen", "twenty-nine", "minus four"}.

Lastly, reduction. In order to combine a whole list of values, we need a phrase to combine any two. Here are some samples:

```
To decide what number is the larger of (N - number) and (M - number)
   (this is maximizing):
   if N > M, decide on N;
   decide on M.

To decide what text is the concatenation of (X - text) and (Y - text)
   (this is concatenation):
   decide on "[X][Y]".
```

And here are some sample reductions:

```
let X be the maximization reduction of {3, 8, 4, 19, 7};
let Y be the concatenation reduction of txtng applied to spelling out
   applied to doubling applied to {3, 8, 4, 19, 7};
```

sets X to 19, the highest of the values, and Y to the text "sxsxtnghtthrty-ghtfrtn". In each case a list has been reduced to a single value which somehow combines the contents.

(phrase (value, value) -> value) **reduction of** (list of values) **... value**

This phrase works through the list and accumulates the values in it, using the phrase supplied. Example: if we have

```
To decide what number is the sum of (N - number) and (M - number)
   (this is summing):
   decide on N + M.
```

then the summing reduction of {3, 8, 4, 19, 7} is the number 41, obtained by

```
(((3 + 8) + 4) + 19) + 7
```

so that the summing phrase has been used four times.

Is map/filter/reduce always a good idea? Devotees point out that almost any computation can be thought of in this way, and in systems where the work has to be distributed around multiple processors it can be a very powerful tool. (There are programming languages without loops where it's essentially the only tool.) At its best, it reads very elegantly: one assembles all of the tools needed - definitions of doubling, lengthy, spelling out, concatenation and so on - and then each actual task is expressed in a single line at the end.

On the other hand, there are also times when this is a needlessly complicated disguise for what could more easily be done with a "repeat" loop, and also more efficiently since assembling and dismantling lists in memory does take some overhead time. So these list operations are not a panacea, but it's good to have them available.

Start of Chapter 22: Advanced Phrases
Back to §22.4. Default values for phrase kinds
Onward to §22.6. Generic phrases

---

§22.6. Generic phrases

The following looks quite innocent:

```
To say (V - value) twice: say "[V]. [V], I say!"
```

It's clear at a glance what this is intended to do, but at a second glance things aren't so straightforward. "Value" is not itself a kind - it's too big and unspecific. For instance, if we were to allow a variable to be just "a value", we could freely set it to 12 one minute and to "dahlias" the next, and such a variable would be dangerous since we would never know what could safely be done with its contents. A phrase like this one is called "generic", because it's not so much a single, actual phrase as a recipe to make phrases. (Inform automatically works out which kinds we need the phrase for, and creates a version of the phrase for those kinds.)

So "value" is not a kind, but a kind of kind. Inform has several of these:

```
value, arithmetic value, enumerated value, sayable value
```

These act as ways to say "a value of any kind matching this can go here". For example, "value" is a way to say "any kind at all"; "arithmetic value" is any kind which arithmetic can be performed on (any kind with the little calculator icon in the Arithmetic part of the Kinds index); and so on. If we write:

```
To double (V - arithmetic value): say "[V times 2]."
```

the restriction to "arithmetic value" means that although "double 3", "double 6 kg", etc., would be matched, "double the Entire Game" would not - you can't perform arithmetic on scenes. Similarly, it would have been tidier to write:

```
To say (V - sayable value) twice: say "[V]. [V], I say!"
```

because then Inform will make it clearer why "say X twice" won't work if X is one of those rare values which it can't say (an activity, for instance).

The Kinds index shows which kinds match against which of these "kinds of kind". For instance, it shows that "time"

```
Matches: value, arithmetic value, sayable value
```

which means that time is something we can do arithmetic on, and can say.

 Start of Chapter 22: Advanced Phrases

 Back to §22.5. Map, filter and reduce

 Onward to §22.7. Kind variables

---

§22.7. Kind variables

The examples of generic phrases in the previous section were really only toy examples. Suppose we want a phrase which will take any arithmetic value and triple it. We could do something like this:

```
To triple (V - arithmetic value): say "[V times 3]."
```

But this only prints the answer. Suppose we want to be given the value back, instead: how can we write the phrase? The trouble is that, not knowing the kind of V, we can't say what kind will be produced. We need a way of saying "the same kind comes out as went in". Inform expresses that using kind variables:

```
To decide which K is triple (original - arithmetic value of kind K):
    decide on 3 times the original.
```

Here, K stands for any kind which matches "arithmetic value". Inform supports exactly 26 of these symbols, which are written A to Z, but it's customary to use K and L. (They can be written in the plural if we like: e.g., "list of Ks". But they must always use a capital letter: "list of k" is not allowed.)

Each symbol we use has to be declared in exactly one of the bracketed ingredients for the phrase - here, the declaration is "arithmetic value of kind K". That creates K and says that it has to be arithmetic; if we'd just said "value of kind K", it could have been anything. (Alternatively, we could use any of the kinds of kind in the previous section.)

For a more ambitious example, here is one way to define the mapping operation described earlier in the chapter:

```
To decide what list of L is (function - phrase K -> value of kind L)
    applied to (original list - list of values of kind K):
    let the result be a list of Ls;
    repeat with item running through the original list:
        let the mapped item be the function applied to the item;
        add the mapped item to the result;
    decide on the result.
```

Here we need two symbols to explain the complicated way that the values going in and out have to match up to each other. Note also the way that the temporary variable "result" is created:

```
let the result be a list of Ls;
```

Ordinarily, of course, "L" is not a kind. But within the body of a phrase definition, it means whatever kind L matched against.

When a symbol occurs several times in the same definition, subtle differences can arise according to which appearance is the declaration. These are not quite the same:

```
To hunt for (V - value of kind K) in (L - list of Ks): ...
To hunt for (V - K) in (L - list of values of kind K): ...
```

The difference arises - though very rarely - if V has some different kind compared to the list entries, but which can be used as if it were of that kind. For example,

```
hunt for the player's command in {"take all", "wait"};
```

Here V is a snippet, but L is a list of texts; and a snippet can be used in place of a text, but not vice versa. So this will match the second definition, because K is set to "text", but it won't match the first, where K is set to "snippet".


Start of Chapter 22: Advanced Phrases


Back to §22.6. Generic phrases


Onward to §22.8. Matching the names of kinds

---

§22.8. Matching the names of kinds

Sometimes a phrase needs to know what kind it's to work on, but isn't going to be given any particular value of it. For example:

```
To assay (name of kind of value K):
    repeat with item running through Ks:
        say "There's [item].";
    say "But the default is [default value of K].";
```

Note that there's no hyphen, and no name for the bracketed token - it only sets K. We can then have, say:

```
assay colours;
assay vehicles;
```

But "assay texts" would throw a problem message, because we can't repeat through all possible texts. For a different reason,

```
assay open doors;
```

would not be allowed - "open doors" is a description which applies to some doors and not others; it isn't a kind. It would make no sense to talk about "default value of open door", for example.

Start of Chapter 22: Advanced Phrases
Back to §22.7. Kind variables
Onward to §22.9. In what order?

---

§22.9. In what order?

Recall the definition:

```
To slam shut (box - an open container): say "With great panache, you slam shut [the box].".
```

Suppose we then try to "slam shut the wall safe" at a time during play when the wall safe is already closed. An error message will then be printed during play, since there must be a mistake in the design. The combination of checking both when Inform builds the story file and then continuously when the story file is played guarantees that, in all cases, a varying item such as "box" in the definition of "To slam shut (box - open container)" always satisfies the condition laid down.

Instead suppose we also have the following definition:

```
To slam shut (box - a container): say "You are unable to slam shut [the box], which is already closed.".
```

We now have two definitions of "slam shut". Sometimes the box it's applied to will be closed, in which case only the second definition fits, and will be the one used. But if the box is open, both definitions fit. Which will happen? The rule is:

1. A narrower condition takes precedence over a broader one;

2. If neither condition is narrower than the other, then whichever phrase was defined later in the source code is the one taking precedence;

3. Except that if the phrase is being used in the definition of phrase P, then P is always last in precedence order, so that recursion is always the very last possibility considered. This allows more specific or later definitions to make use of less specific or earlier ones in a natural way.

Rule 1 ensures that a definition involving "open container" takes priority over one which merely involves "container", for instance.

And therefore when the box is open, it's the more specific phrase to do with open containers which is invoked: so, with great panache, the box is slammed shut.

On the other hand, neither of these patterns is narrower than the other:

```
To describe (something - transparent): ...
To describe (something - container): ...
```

Some containers are transparent, some not; some transparent things are containers, some not. Rule 1 therefore does not apply, so it is the later of the two phrases which takes effect.

Start of Chapter 22: Advanced Phrases
Back to §22.8. Matching the names of kinds
Onward to §22.10. Ambiguities

---

§22.10. Ambiguities

Another possible ambiguity occurs when a phrase might match two lexically different possibilities using the same words.

```
say the dishcloth;
```

could be construed as a usage of either of these cases:

```
say the (something - a thing)
say (something - a thing)
```

These of course have different effects - one produces the name with a definite article, the other just the name, so the difference is important.

The rule here is that whichever possibility contains the most words, in this case "say the (...)", takes precedence, because it's assumed to be a more specific form of the less wordy version.

Start of Chapter 22: Advanced Phrases
Back to §22.9. In what order?
Onward to Chapter 23: Figures, Sounds and Files: §23.1. Beyond text

---

Examples from Chapter 22: Advanced Phrases

Start of this chapter
Chapter 23: Figures, Sounds and Files
Indexes of the examples

439 **Example Curare** RB

A phrase that chooses and names the least-recently selected item from the collection given, allowing the text to cycle semi-randomly through a group of objects.

```
"Curare"
```

```
A thing has a number called the last use. The last use of a thing is usually 0.

Definition: a thing is old if its last use is 12 or less.
```

The actual number chosen in this definition is pretty much irrelevant: the main thing is that we want to establish relative values. The lower the "last use" number of an item, the older that item should be understood to be, as we see here:

```
To decide which thing is cyclically random (collection - a description of objects):
    let choice be the oldest member of the collection;
    now the last use of the choice is the turn count;
    decide on choice.
```

This phrase will select, from the collection of objects passed to it, the one that has been mentioned least recently. This means that if we consult it repeatedly about the same collection, it will begin to cycle predictably; but if new items are added to the collection, it will mention these first before returning to the previous cycle. Now we can use this:

```
After taking inventory:
    say "You stare morosely at [the cyclically random thing carried by the player],
wondering what you're ever going to find to do with it."
```

We could have said "You stare morosely at [the oldest thing carried by the player]" here, but doing so would not have set the "last use" property correctly, so we would not get the cycling behavior that we're looking for.

```
The Evidence Room is a room. Some shelves are scenery supporters in the
Evidence Room. A box is a kind of container which is open and not openable.
On the shelves is a box. It contains a deformed bullet and a driver's license.

The player carries a steel fish hook, a Chinese passport, a tube of synthetic
curare, and an envelope full of Euros.

Test me with "i / i / i / i / i / i / get all from box / i / i / i".
```

Chapter 23: Figures, Sounds and Files

 Contents of *Writing with Inform*

 Chapter 22: Advanced Phrases

 Chapter 24: Testing and Debugging

 Indexes of the examples

§23.1. Beyond text

In this chapter, we explore a number of ways to go beyond the traditional text-only, one-story-file-only model for IF.

These relatively exotic features are more demanding of the interpreter which a player uses than a plain text story file would be. They can only be used if the project is being compiled to the Glulx story file format (see the Settings panel for the project), and even then, the player will need to have a good Glulx interpreter - one which is reasonably up to date and well-written, that is - to be sure of everything working as intended.

 Start of Chapter 23: Figures, Sounds and Files

 Back to Chapter 22: Advanced Phrases: §22.10. Ambiguities

 Onward to §23.2. How IF views pictures

---

§23.2. How IF views pictures

Looking around a bookshop, perhaps half of all the books published have illustrations. The proportion may be lower for novels, but if we count maps or other occasional diagrams, even the fiction section turns out to be surprisingly pictorial. Illustrations do not suit every book, but they are an option we would like to have available.

In the cultural history of IF, graphics in text adventures have sometimes been looked at with suspicion. Mostly this is because attempts in the 1980s were not very successful, because computer graphics were so poor then (by modern standards). It may be that some people also felt that the takeover of computer games by graphical interfaces was the death knell of IF.

But pictures are now rendered in superb quality by computers, and the death of IF turned out to be an exaggeration, so it is time to move on.

Whether to have illustrations ought to be an artistic choice, like whether to include a romantic sub-plot or how much of the back story is revealed. But there are practical considerations too. The most successful illustrated books are those whose pictures are well-chosen, have a sense of design to them, and above all are consistent. Consider how much worse off *Winnie the Pooh* would have been if a selection of random teddy-bear drawings had been used, instead of E. H. Shepherd's beautifully conceived world; or a cookery book in which the recipes are all photographed at different distances and light levels. IF writers may want to look for collaborators with a visual eye, just as most novelists do not draw their own illustrations.

Another consideration is that displaying images is more complicated for computers than displaying text. Not all devices can show pictures (consider handheld gadgets) and if they do, they may use different colour ranges or resolutions. So IF with pictures is always just a bit less portable than IF without, and because of that we must next look again at IF story file formats.


Start of Chapter 23: Figures, Sounds and Files


Back to §23.1. Beyond text


Onward to §23.3. Virtual machines and story file formats

---

§23.3. Virtual machines and story file formats

Back in Chapter 2, we saw that the Settings panel allows any given Inform project to be produced in either of two possible story file formats. Recall that story files are the released IF works: what the player sees. The source text, the Index, and so on are not part of this.

A story file is not like a word-processed document, or a photograph. There are many rival formats for these - for instance, an image on a web page might be in JPEG or PNG format, among many others - but basically they are simple things for the reader to look at, and see everything in one go. An IF story file is more complicated, because the "reader" reacts to it, types in to it, is surprised by it, never quite knows what might happen next.

A story file is in fact a computer program in its own right, but not a program like iTunes or Firefox which runs on a typical home or business computer. Instead it is a program for an imaginary computer, called a "virtual machine" or "VM". This has a design ideally suited to IF, and it would be the perfect IF player's computer if only it actually existed. Because it doesn't, the player instead runs an "interpreter" program like Windows Glulxe or Zoom or Spatterlight - and this one is a program like iTunes or Firefox - and the interpreter acts as a middle-man. It creates a software version of the virtual machine, and then runs the IF story file on that VM. This sounds slow and impractical, but in fact it works well, and is also much safer since programs on the VM are not allowed to touch the real computer - so they cannot at all easily contain viruses or other malware. (In theory a malicious story file might try to exploit a bug in one of the various VM implementations in use, just as malicious image files have been used to attack bugs in web browsers, but this has never in practice happened. Nothing can be absolutely safe, but a story file belongs in the "mostly harmless" category of

files - like images - rather than the "how far do you trust this person?" category - like programs.)

The different formats of story file are programs for different virtual machines. Just as Windows and Mac OS X offer basically similar services to the user but are very different in appearance and their workings, so the different VMs are quite different. Some can display pictures, others not.


Start of Chapter 23: Figures, Sounds and Files


Back to §23.2. How IF views pictures


Onward to §23.4. Gathering the figures

---

§23.4. Gathering the figures

Inform provides basic support for displaying pictures and leaves more exotic effects for Extensions to provide. But either way, for reasons explained in the previous section, **we can only have pictures if the Settings for the project are set to the Glulx story file format**.

Inform calls these pictures "figures", following the usual Inform analogy with books. We will think of our work of IF as being like a mostly textual book which in broken up with illustrations here and there - Figure 1, Figure 2, and so on. These might be used to mark each new chapter of the plot, or each new location: whatever the author would like. So the first thing we need to do is decide when pictures should appear.

The second thing to do is to get hold of the pictures we want to use. These might be photographs, or artwork, or diagrams: anything, really, but we will need them to be in either JPEG or PNG format. Inform does not itself try to be an image editor, or an artwork program - there are many such programs already which do these things much better than Inform could.

The pictures then need to be put in a special place where Inform can reach them. Suppose the Inform project is called Example.inform. Then we need to create a folder alongside it called "Example.materials", and create a further folder inside that called "Figures". The actual images go inside "Figures". So we might then have files like so:

```
Example.inform
Example.materials
    Figures
        Woodlands.png
        Blackberry.jpg
        Red Admiral Butterfly.png
```

The ".materials" folder for an Inform project will turn out to have many other uses in the chapter on Publishing, and will be explained further there.

Start of Chapter 23: Figures, Sounds and Files
Back to §23.3. Virtual machines and story file formats
Onward to §23.5. Declaring and previewing the figures

---

§23.5. Declaring and previewing the figures

Inside Inform, the source text for a project always tries to avoid talking about filenames - we need a better way to refer to the individual figures.

We do this by declaring each figure with a sentence like the following examples:

```
Figure of Woodlands is the file "Woodlands.png".
Figure 2 is the file "Red Admiral Butterfly.png".
```

Figure names can consist of any text provided that text starts with the word "Figure". So "Figure 3 - Woodlands", for instance, or even "Figure W" would have been just as good as "Figure of Woodlands". Books tend to number figures, but then, in a book the order in which they appear is known in advance - which might not be true in IF.

The file names must be exactly those used in the Figures folder. We need not declare every image kept there, but those we don't declare - remember Blackberry.jpg? - cannot be displayed.

We can preview the stock of figures by going to the table of figures in the Contents index for a project (once the project has been built, that is, so that its index is up to date). This preview shows thumbnail forms of the pictures, the names, the formats and the image sizes in pixels. A warning triangle is shown for any images in the wrong format, or which are missing from the Figures folder.

Start of Chapter 23: Figures, Sounds and Files
Back to §23.4. Gathering the figures
Onward to §23.6. Displaying the figures

---

§23.6. Displaying the figures

Inform's basic picture support simply allows figures to be shown at particular times. Once seen, they scroll away, just as text does once it has been printed. These pictures are really part of the stream of narrative. (If we would like icons or other images to be permanently present on screen, and divide the screen up in pictorial ways to achieve interesting layouts, we need to use special extensions to access Glulx's more exotic features.)

Displaying a picture is therefore like printing some text. Rather than

```
say "The woodlands stretch from here to the horizon.";
```

we would use:

> **display** (figure name)
>
> This phrase causes the figure to be displayed in a way visible to the player. If the option "one time only" is used, it will have no effect if the figure has been displayed before. Example:
>
> ```
> display the Figure of Woodlands;
> ```

Once again, note that the "display" phrase does nothing unless the Settings for the project are set to the Glulx story file format. When a Glulx work is released as a blorb (the default setting for the way releases occur: see the chapter on Publishing), all the images used are automatically included.


Start of Chapter 23: Figures, Sounds and Files


Back to §23.5. Declaring and previewing the figures


Onward to §23.7. Recorded sounds

---

§23.7. Recorded sounds

Inform also supports the playing back of recorded sounds, which might be anything from a three-second sound effect for a creaking door to an epic orchestral symphony. **Sound support is very newly added to the system and work is still in progress. In particular, sounds are not played by Inform for OS X (although it does produce valid blorbed Glulx story files), though they should be audible from within the Inform application for Windows.**

Once again, sound effects are supported by Inform 7 only on the Glulx platform, and even then we must be prepared for the fact that not all interpreters will be able to play them back. We must also bear in mind that a sound recording is a large pile of bits, and that adding any kind of sounds will greatly increase the size of the Blorb file for the released Glulx story file.

The sound files provided must have one of two formats: AIFF or Ogg Vorbis. **AIFF** is a traditional format in the recording industry, though it is more familiar to Mac OS X users than Windows users. It is uncompressed, giving what can be excellent audio quality, but at the cost of sometimes enormous file sizes - perhaps as much as 10 MB per minute, though this can be greatly reduced by lowering the sampling frequency, and halved again by dropping from stereo to mono.

Except for very short sound effects, we recommend using **Ogg Vorbis** instead. This is a compressed format whose file sizes will typically be more like 1 MB per minute. Inform uses Ogg Vorbis as the only format safe from licencing and patent disputes. (We would very much have liked to provide MP3 support, but this is no longer legally possible for free software.)

Support for Ogg Vorbis is not built in to either Windows or Mac OS X, and any sound recording you make will probably have to be made first to another format (perhaps AIFF or WAV), and then converted. See **www.vorbis.com** for encoding software which can convert from other sound formats to Vorbis.

Lastly, it must be remembered that recording industry bodies are very hostile to established copyright law covering fair use, parody, quotation of insubstantial passages, etc., when it comes to mixing or using commercially released music. They are well-resourced and highly litigious. If you use sound effects not originated by yourself, you do so at your own risk, even if what you do is perfectly legal on any reading of the statutes.


Start of Chapter 23: Figures, Sounds and Files


Back to §23.6. Displaying the figures


Onward to §23.8. Declaring and playing back sounds

---

§23.8. Declaring and playing back sounds

Sound effects are accommodated on the same basis as illustrations. The relevant media files need to be placed in a subfolder of the project's ".materials" folder, but this time called Sounds rather than Figures, so for instance:

```
Example.inform
Example.materials
    Figures
        Woodlands.png
        Blackberry.jpg
        Red Admiral Butterfly.png
    Sounds
        Rustling leaves.ogg
```

Again, these must be declared in the source text:

```
Sound of rustling leaves is the file "Rustling leaves.ogg".
```

And they can be played using a special phrase:

play (sound name)

This phrase causes the sound effect to be played. If the option "one time only" is used, it will have no effect if the sound effect has been played before. Example:

```
play the sound of rustling leaves;
```

Start of Chapter 23: Figures, Sounds and Files
Back to §23.7. Recorded sounds
Onward to §23.9. Providing accessibility text

---

§23.9. Providing accessibility text

It's conventional for web pages to provide "alt-text" for significant images displayed, so that partially sighted or blind users can get an idea of what is being shown. Inform allows figures to be given these short descriptions like so:

```
Figure 2 is the file "butterfly.jpg" ("A red admiral butterfly.").
```

As we'll see, the same can be done for the cover image:

```
Release along with cover art ("A cathedral at sunset.").
```

And also for sounds:

```
Fugue is the file "Bach.ogg" ("A church organ playing a Bach fugue.").
```

Start of Chapter 23: Figures, Sounds and Files
Back to §23.8. Declaring and playing back sounds
Onward to §23.10. Some technicalities about figures and sounds

---

§23.10. Some technicalities about figures and sounds

(i) Names for figures, such as "Figure of Woodlands", are values for a special kind of value called "figure name". This can in turn be used to define variables, properties and phrases:

```
The turn card image is a figure name that varies.

An Old Master is a kind of thing. An Old Master has a figure name called appearance.
Figure 1 is the file "Giaconda.jpg". The Mona Lisa is an Old Master. The appearance of
the Mona Lisa is Figure 1.

To place (F - a figure name) in the gallery: ...
```

(ii) Similarly, names for sound effects, such as "Sound of rustling leaves", are values for the kind of value "sound name".

(iii) In the released, blorbed-up Glulx file, figures and sound effects are internally given resource ID numbers which count upwards from 2 in order of their declaration. (Figure and sound numbers can thus be intermingled, if their declarations are.) Resource ID number 1 is reserved for the image of the cover art, if there is any. (See the chapter on Publishing.) To obtain these numbers, if we need them, we can use:

Glulx resource ID of (figure name) **... number**

This phrase produces the ID number used in the eventual Glulx file for the given figure.

Glulx resource ID of (sound name) **... number**

This phrase produces the ID number used in the eventual Glulx file for the given sound effect.

(iv) Glulx hackers may also like to know that whenever Inform 7 builds a project for Glulx, the Inform 6 code it generates always contains a full copy of John Cater's definitive header file "infglk.h".

Start of Chapter 23: Figures, Sounds and Files
Back to §23.9. Providing accessibility text
Onward to §23.11. Files

---

§23.11. Files

Once an Inform project is released, it is playable as a "story file", which is in effect a computer program for a specially IF-adapted design of computer. Story files run in what in computing is sometimes called a "sandbox", a kind of safe play area where it can be guaranteed that they cannot do any harm. This is good, because it means a story file can't be infected with viruses or other malware. If the project's Settings panel has the story file format set to the Z-machine, the story file is so thoroughly boxed in that it cannot even see the bigger computer beyond: it lives in a world of its own. But the Glulx format opens the door a crack, allowing the story file to read and write a small number of data files, which live in a single folder on the bigger computer's hard drive.

Why might we want this? Among the reasons are -

- to remember what has happened in previous attempts by the player;
- to store the player's preferences;
- in a two-part story, where each part is an independently released story file, to allow Part I to save some information about its ending which Part II could then pick up and make use of;
- to communicate with some external program, such as an Internet service.

Start of Chapter 23: Figures, Sounds and Files
Back to §23.10. Some technicalities about figures and sounds
Onward to §23.12. Declaring files

---

§23.12. Declaring files

Like figures and sounds, files must be declared before they can be used. For instance:

```
The File of Glaciers is called "ice".
```

This creates a new named constant "File of Glaciers" to refer to the file, throughout the source text. We use this name for it whether or not the actual disc file exists yet: it might be one that will only be created if something unusual happens in play, for instance.

Quoted filenames should contain only letters and digits, should be 23 characters or fewer, and should begin with a letter. (In particular they can contain no slashes or dots - no subfolders or extensions can be indicated.) The actual filename this translates to will vary from platform to platform, but "ice.glkdata" is typical, stored in some sensible folder.

Every file has an owner - not a person, but the project which normally writes to it. Inform assumes that the current project will be owning any file which it declares - the File of Glaciers, for instance. But we can optionally specify that it is owned by somebody else:

```
The file of Boundaries (owned by another project) is called "milnor".
The file of Spectral Sequences (owned by project "4122DDA8-A153-46BC-8F57-
42220F9D8795") is called "adams".
```

Inform uses ownership to make sure that we do not accidentally read in a file which has nothing to do with us, but merely happens to use the same name. Thus it is an error to read a file whose ownership does not agree with our declaration. Saying that a file is owned by "another project" allows us to read it whatever the owner is (so this can be used for files shared between multiple projects); specifying exactly where it needs to come from allows us to pass information from one project to another. Note that we identify projects using the IFID number - this can be found in the Contents index for a project, or by typing VERSION during play; see the chapter on Publishing for more about IFIDs.

Files are indexed in the Contents index, alongside figures and sound effects.

Two technicalities. First, constants such as "File of Glaciers" are of a kind of value called "external file" (compare "figure name" and "sound name"). Second, Inform's file-handling is provided for the Glulx machine, which in turn uses the Glk interface. This allows for either text or binary files. Inform's higher-level phrases to do with files, described in this chapter, all use text files, and all declared files are text files by default. But we can optionally add the keyword "binary" to declare a binary file, if needed:

```
The binary File of Glaciation Data is called "icedata".
```

Start of Chapter 23: Figures, Sounds and Files
Back to §23.11. Files
Onward to §23.13. Writing and reading tables to external files

---

§23.13. Writing and reading tables to external files

The main use for files is to store and retrieve data, and the most flexible form of data used by Inform is the Table, so facilities are provided which make it as easy as possible to write and read the contents of a table to files. If so, the file must contain just one single table: so to write multiple tables, we need to write multiple files, one for each.

To save the contents of a table to a file, we use the phrase:

write (external file) **from** (table name)

This phrase causes the entire contents of the given table to be written out to the given file. Note that files must have been declared, and must be referred to by their Inform names, not by textual filenames. Example:

```
write File of Glaciation Data from the Table of Antarctic Reserves
```

Any blank rows in the table are automatically moved to the bottom, and only the non-blank rows are written.

To load a file back into a table,

read (external file) **into** (table name)

This phrase causes the entire contents of the given table to be read in from the given file. Note that files must have been declared, and must be referred to by their Inform names, not by textual filenames. Example:

```
read File of Glaciation Data into the Table of Antarctic Reserves
```

Any rows left spare at the foot of the table are automatically blanked. On the other hand if the file is too large to fit into the table - with too many columns or too many rows - a run-time problem is produced.

We can check if a file already exists using:

if (external file) **exists:**

This condition is true if the file-system used by the player appears to contain a file with the right name. For example, if we declared:

```
The binary File of Glaciation Data is called "icedata".
```

and then tested

```
if the File of Glaciation Data exists, ...
```

then Inform would search for a file called "icedata". (The arrangements for where this might be stored, and its filename extension, vary from platform to platform.)

One unfortunate restriction must be kept in mind. Some of what is stored in tables is solid information whose meaning never changes: the number 342, for instance, means the same to everyone. But other information depends entirely on the current location of certain structures in memory - for instance, a rule is internally referred to by its memory location. This potentially changes each time Go or Replay is clicked, and so it is not safe to pass it from one copy to another, or from one project to another. The only tables which Inform allows us to write into files are those containing "safe" data: numbers, units, times of day and kinds of value with named alternatives. Scenes, rules or rulebooks, in particular, are not allowed.

Start of Chapter 23: Figures, Sounds and Files

Back to §23.12. Declaring files

Onward to §23.14. Writing, reading and appending text to files

Example 440: ★ **Alien Invasion Part 23** Keeping a preference file that could be loaded by any game in a series.

Example 441: ★★ **Labyrinth of Ghosts** Remembering the fates of all previous explorers of the labyrinth.

Example 442: ★★★ **Rubies** A scoreboard that keeps track of the ten highest-scoring players from one playthrough to the next, adding the player's name if he has done well enough.

---

§23.14. Writing, reading and appending text to files

Text can also be saved to a file, and again all file-handling is automatic:

write (text) **to** (external file)

This phrase makes the given text become the entire contents of the named file. Note that files must have been declared, and must be referred to by their Inform names, not by textual filenames. Example:

```
write "Jackdaws love my big sphinx of quartz." to the file of Abecedary
Wisdom;
```

append (text) **to** (external file)

This phrase adds the given text to the end of the current contents of the named file (creating it if it does not exist on disc). Note that files must have been declared, and must be referred to by their Inform names, not by textual filenames. Example:

```
append "Jinxed wizards pluck ivy from the big quilt." to the file of Abecedary Wisdom;
```

The quoted text can, of course, contain substitutions, so can be long and complex if need be.

Text from a file is printed back with the text substitution:

say "[text of (external file)**]"**

This text expands to the contents of the named file. Note that files must have been declared, and must be referred to by their Inform names, not by textual filenames. Example:

```
"[text of the File of Abecedary Wisdom]"
```

To copy one file to another, for instance,

```
write "[text of the file of Abecedary Wisdom]" to the file of Secondary Wisdom;
```

Start of Chapter 23: Figures, Sounds and Files

Back to §23.13. Writing and reading tables to external files

Onward to §23.15. Exchanging files with other programs

Example 443: ★ **The Fourth Body** Notebooks in which the player can record assorted notes throughout play.

Example 444: ★★ **The Fifth Body** An expansion on the notebook, allowing the player somewhat more room in which to type his recorded remark.

---

§23.15. Exchanging files with other programs

Provided we declare the files in the right way, it is easy for one project to read a file created by another project.

But if we want more rapid communication, between two projects which are each playing at the same time, we need to be more careful. What if project A tries to read the file at the same moment that project B is writing it?

To avoid this, we have a concept of files being "ready". A file is ready if it exists, and is completely written, and not in use elsewhere. We have already seen:

```
if the file of Invariants exists...
```

But now we want a stronger condition:

if ready to read (external file)**:**

This condition is true if the file exists and is marked as being ready to read; that is, it is not in a state where another program is currently writing it. Example:

```
if ready to read the file of Invariants, ...
```

A file cannot be ready to read if it does not exist, so this is a stronger condition. If A and B are attempting communication in real time, both running at once, then Project A should check that an external file owned by B is ready before it tries to read it. Files can also be marked as ready or not ready, in effect claiming them, thus:

mark (external file) **as ready to read**

This phrase marks that we have finished writing to the given file, so that any external program is welcome to read it now. Example:

```
mark the file of Invariants as ready to read;
```

mark (external file) **as not ready to read**

This phrase marks that we are about to start writing to the given file, so that any external program should wait until we're finished if it wants to read the file. Example:

```
mark the file of Invariants as not ready to read;
```

Possibilities really begin to open up when project A is our story file, but B is not another story file at all: it is some external program such as a Web service, say. (Of course this is harder to set up, since the player needs to have both A and B running at the same time, but for stories running on an Internet server this can all be made seamless.)

When Inform begins writing a table, or text, to a file, it initially marks the file as not ready: only when the table or text is completely written and the file about to close is the file marked as ready.

In order to write non-story-file programs as B, communicating with story files as A, we need to know the file format used by Inform. An Inform file is currently a Unix text file (with 10 as the line division character), encoded as ASCII Latin-1. (We would like to use Unicode at some point in the future, but the Glk and Glulx layers are still not fully converted to Unicode.) It opens with a single header line in the form:

```
* //IFID// leafname
```

The opening character is an asterisk if the file is currently ready, a hyphen if the file is currently not ready. The IFID between the slashes is the IFID number of the project which last wrote to the file. (Marking "ready" or "not ready" does not count as a write for this purpose.) If an external program wrote the file, it should call itself something which will not clash with any story file's IFID. The leafname is the filename text used inside the story file where the file was declared. For instance:

```
* //4122DDA8-A153-46BC-8F57-42220F9D8795// ice
```


Start of Chapter 23: Figures, Sounds and Files


Back to §23.14. Writing, reading and appending text to files


Onward to Chapter 24: Testing and Debugging: §24.1. Checking against the Index


Example 445:  **Flathead News Network** Using external files, together with a simple Unix script running in the background, to provide live news headlines inside a story file.

Examples from Chapter 23: Figures, Sounds and Files


Start of this chapter


Chapter 24: Testing and Debugging


Indexes of the examples

440

 **Example** **Alien Invasion Part 23**

RB

Keeping a preference file that could be loaded by any game in a series.

Suppose that we have a series of games each of which allows the player to select a puzzle difficulty level. When the player plays a new game in the series, we want him to start out by default with the same difficulty level he faced earlier on, so we store this information in a small preferences file, as follows:

```
"Alien Invasion Part 23"

A difficulty is a kind of value. The difficulties are easy, moderate, hard, and
fiendish.

Understand "use [difficulty] puzzles" as selecting difficulty. Selecting difficulty is
an action out of world, applying to one difficulty.
```

```
Carry out selecting difficulty:
    choose row 1 in the Table of Preference Settings;
    now challenge level entry is difficulty understood;
    say "Puzzles will be [challenge level entry] from now on."

The File of Preferences is called "prefs".

When play begins:
    if File of Preferences exists:
        read File of Preferences into the Table of Preference Settings;
        choose row 1 in the Table of Preference Settings;
        say "(The current puzzle difficulty is set to [challenge level entry].)"

Check quitting the game:
    write File of Preferences from the Table of Preference Settings.

Table of Preference Settings
challenge level
easy

The Sewer Junction is a room.
```

Our preference file is restricted to a single option here for simplicity's sake, but we could keep track of more information -- whether the player preferred verbose or brief room descriptions, screen configurations, and so on.

If we were disposed to be somewhat crueler, we could use a similar method to make the player finish each episode of the series in order to "unlock" the next. All we would need to do is store a numerical password in our preferences file when the player finished a given level; the next level would check, say, the Table of Completed Levels for that password, and refuse to play unless the right number were present.

441

Example Labyrinth of Ghosts

RB

Remembering the fates of all previous explorers of the labyrinth.

A tradition among Nethack-like computer games of the old school is that a player's death in a given place leaves a ghost behind to haunt subsequent players. Information about past lives is sometimes stored in a "bones file", and in this example we do exactly that, for a grievously unfair little dungeon.

To begin with, the labyrinth itself. We create a kind of value to remember possible means of death in these tunnels, and we assign a coordinate position in some grid to each location. (We do this because grid positions can safely be stored in tables saved out to external files, whereas room names cannot - they represent data which changes each time we amend the source.)

```
"Labyrinth of Ghosts"

Use scoring.
```

```
A demise is a kind of value. The demises are drowned, buried by a rockfall,
pierced by an arrow and slain. The latest demise is a demise that varies.

A grid location is a kind of value. (1,19) specifies a grid location. A room has a
grid location called coordinates.

The Gateway is a room. "For the foolhardy adventurer, the perilous labyrinth lies
north, east or south." The coordinates are (6,6). The Tomb is east of the
Gateway. The coordinates are (7,6). The Rockfall Cave is north of the Gateway.
"This partly fallen cave may perhaps extend further north." The coordinates are
(6,5). Instead of going north in the Rockfall Cave, have the player buried by a
rockfall. The Archery Canyon is south of the Gateway. "No telling why this
canyon is named after archery, but perhaps if you wait around you'll find out."
The coordinates are (6,7). Instead of waiting in the Archery Canyon, have the
player pierced by an arrow. The Rock Pool is east of the Tomb. The coordinates
are (8,6). The cold mountain pool is in the Rock Pool. The cold pool is fixed in
place. Instead of entering the cold mountain pool, have the player drowned.

Every turn when a random chance of 1 in 10 succeeds:
    say "A dwarf appears out of nowhere, and throws a nasty little knife.";
    have the player slain.
```

And as compensation for these hazards:

```
Some silver bars are in the Tomb. The emerald is in the Rock Pool. The platinum
pyramid is in the Canyon.

Table of Point Values
item score

    silver bars         3
    platinum pyramid    10
    emerald             4

Report taking an item listed in the Table of Point Values:
    increase the score by the score entry;
    blank out the whole row.
```

We are now ready for the actual undertaking. The Table of Ghostly Presences holds up to twenty death notices, and is initially blank. Deaths are sequentially numbered, and this number is stored in the sequence column.

```
Table of Ghostly Presences

    haunted position    score at death  turns at death  manner of death  sequence
    a grid location     a number        a number        a demise         a number
    with 19 blank rows.
```

As the story file starts up, we look to see if a ghosts file already exists. If one does, we load up the Table of Ghostly Presences with it: and if not, as will be the case the first time the player explores, we leave the table blank. We sort the table so that it has earlier deaths (lower sequence numbers) first.

```
The File of Ghosts is called "ghosts".
```

```
When play begins:
    if the File of Ghosts exists, read File of Ghosts into the Table of Ghostly
Presences;
    sort the Table of Ghostly Presences in sequence order.
```

How will ghosts manifest themselves? Because this is only a small example, we will simply tell the player that he senses something. If several ghosts are present in the same place, the most aggrieved (that is, the most recent) is sensed first...

```
After looking:
    repeat through the Table of Ghostly Presences in reverse sequence order:
        if the haunted position entry is the coordinates of the location, say "You
sense the ghostly presence of an adventurer, [manner of death entry] with a
score of [score at death entry] in [turns at death entry] turns."
```

(For instance, "You sense the ghostly presence of an adventurer, buried by a rockfall with a score of 10 in 5 turns.") That just leaves the rule for bumping off the player. When the Table is full, and there are already 20 ghosts, the one who died longest ago (with the lowest sequence count) is eliminated, and his row blanked out. (This will always be row 1 since we sorted the table in sequence order on reading it in.)

```
To have the player (sticky end - a demise):
    let the new sequence number be 0;
    repeat through the Table of Ghostly Presences:
        let S be the sequence entry;
        if S is greater than the new sequence number, let the new sequence
number be S;
    increment the new sequence number;
    if the number of blank rows in the Table of Ghostly Presences is 0:
        choose row 1 in the Table of Ghostly Presences;
        blank out the whole row;
    choose a blank row in the Table of Ghostly Presences;
    now the sequence entry is the new sequence number;
    now the manner of death entry is the sticky end;
    now the turns at death entry is the turn count;
    now the score at death entry is the score;
    now the haunted position entry is the coordinates of the location;
    write the File of Ghosts from the Table of Ghostly Presences;
    now the latest demise is the sticky end;
    end the story saying "You have been [latest demise]".
```

Strictly speaking we ought to worry that after 2,147,483,647 deaths, the sequence numbers would grow too large to store in a single value, and then the sequence of ghosts will be erratic. But it seems unlikely that anyone will play this example 2.1 billion times.

442

Example **Rubies**

RB

A scoreboard that keeps track of the ten highest-scoring players from one playthrough to the next, adding the player's name if he has done well enough.

The trick here is that we need a table with text in order to keep track of the players' names.

Part 1 largely replicates the source from "Identity Theft"; new material starts at Part 2.

```
"Rubies"

Use scoring.

Part 1 - Collecting Names

The player's forename is a text that varies. The player's full name is a text that varies.

When play begins:
    now the command prompt is "What is your name? > ".

To decide whether collecting names:
    if the command prompt is "What is your name? > ", yes;
    no.

After reading a command when collecting names:
    if the number of words in the player's command is greater than 5:
        say "[paragraph break]Who are you, a member of the British royal family?
No one has that many names. Let's try this again.";
        reject the player's command;
    now the player's full name is the player's command;
    now the player's forename is word number 1 in the player's command;
    now the command prompt is ">";
    say "Hi, [player's forename]!";
    say "[banner text]";
    move the player to the location;
    reject the player's command.

Instead of looking when collecting names: do nothing.

Rule for printing the banner text when collecting names: do nothing.

Rule for constructing the status line when collecting names: do nothing.

Part 2 - Adding the Leaderboard

File of Leaderboard is called "leaderboard".

When play begins:
    if the File of Leaderboard exists:
        read File of Leaderboard into the Table of Leaders;
        sort the Table of Leaders in reverse scored amount order.

When play ends:
    choose row 10 in the Table of Leaders; [we've sorted the table, so the lowest
```

```
score will be the one at the bottom]
    if the score is greater than scored amount entry:
        now name entry is the player's forename;
        now the scored amount entry is the score;
    show leaderboard;
    write the File of Leaderboard from the Table of Leaders.

To show leaderboard:
    sort the Table of Leaders in reverse scored amount order;
    say "Current leading scores: [paragraph break]";
    say fixed letter spacing;
    repeat through Table of Leaders:
        if scored amount entry is greater than 0:
            say " [name entry]";
            let N be 25 minus the number of characters in name entry; [here we want
to space out the scores so they make a neat column]
            if N is less than 1, now N is 1;
            say N spaces;
            say "[scored amount entry][line break]";
    say variable letter spacing.

To say (N - a number) spaces:
    repeat with index running from 1 to N:
        say " ".

Table of Leaders

    scored amount name
    0             "Smithee"
    0             "Smithee"
    0             "Smithee"
    0             "Smithee"
    0             "Smithee"
    0             "Smithee"
    0             "Smithee"
    0             "Smithee"
    0             "Smithee"
    0             "Smithee"
```

And now we introduce a scenario that allows different players to come up with different scores -- admittedly not a very interesting scenario, but it will do for now:

```
Part 3 - Scenario

Carry out taking something which is not handled:
    increment score.

The Big Treasure Chamber is a room. It contains a ruby, an emerald, a gold
tooth, an antique katana, and a silver coin.

Instead of waiting, end the story finally.

Test me with "get ruby / z".
```

Example The Fourth Body

Notebooks in which the player can record assorted notes throughout play.

Some mystery games supply the player with an in-game system for taking notes, in case he doesn't want to rely on scraps of paper next to the computer. One way of doing this is to write out all the player's notes and observations into a notebook file, whose contents can be retrieved during play (or, indeed, after it).

We'll first invent a general system for writing text into notebooks, by creating a new kind called jotter. Each individual jotter will have its own disc file, and there will be basically three things which can be done with jotters: erasing, reading and writing.

```
"The Fourth Body"

A jotter is a kind of thing. A jotter has an external file called the text file. A jotter can be fresh or used. A jotter is usually fresh. A jotter has a text called the heading.

The currently erased jotter is an object that varies.
To erase (pad - a jotter):
    now the currently erased jotter is the pad;
    write "[heading of the currently erased jotter][paragraph break]" to the text file of the pad;
    now the pad is fresh.

To write in (pad - a jotter):
    append "[the time of day]: [topic understood][line break]" to the text file of the pad;
    now the pad is used.

To read (pad - a jotter):
    say "You read: [paragraph break][text of the text file of the pad]".
```

This is all as might be expected, except perhaps for the business of the "currently erased jotter". Why copy "pad" into this - why not simply write "[heading of the pad]"? The answer is that "pad" is a temporary "let" value, and cannot be used inside other phrases, such as the "write ... to ..." phrase.

We want to erase any jotters when play begins, as otherwise text left over from any previous games will still be visible:

```
When play begins:
    repeat with pad running through jotters:
        erase the pad.
```

Now we need to create rules to allow the player to control reading, writing and erasing. Reading we will handle with the ordinary examining action, but we create new actions for writing and erasing. A nice little trick allows WRITE WHATEVER to default to writing WHATEVER in a notebook being carried.

```
Instead of examining a used jotter (called the pad):
    read the pad.

Instead of examining a fresh jotter (called the pad):
    say "There is nothing of note in [the pad]."

Understand "write [text] in [something preferably held]" as writing it in.
Understand "write [text]" as writing it in. Writing it in is an action applying to a
topic and one thing. Rule for supplying a missing second noun while writing: if a
jotter (called the pad) is carried, now the second noun is the pad; otherwise say
"You will have to specify what to write that it."

Check writing it in:
    if the second noun is not a jotter, say "It would be better to write in a
notebook." instead.

Carry out writing it in:
    write in the second noun.

Report writing it in:
    say "Under the current time, you write '[the topic understood]' into [the second
noun]."

Understand "erase [something preferably held]" as erasing. Erasing is an action
applying to one carried thing.

Check erasing:
    if the noun is not a jotter, say "It's hard to see how." instead.

Carry out erasing:
    erase the noun.

Report erasing:
    say "You scrub out all the entries in [the noun]."
```

That completes a general-purpose implementation of jotters, and we put it to use:

```
The player carries a jotter called your notebook. The file of Player's
Observations is called "notebook". The text file of your notebook is the file of
Player's Observations. The heading of your notebook is "Observations in the
Pottingham Green Case".

The Damp Hillside is a room. "It is just after dawn: among the trees there is
misty and pale blue light. [if Havers is in the location]The only saturated color in
view is the orange-and-yellow jacket of [Detective Havers]. She is trying
unsuccessfully to light a cigarette. [end if][paragraph break]The body itself is
further down, closer to the bottom of the ravine. It would be foolish to speculate
before seeing it, but the odds are that the corpse was rolled down after death.
The ground is not steep enough for the fall itself to be deadly."

Detective Havers is a woman in the Damp Hillside. The description is "She gives
you a weak smile when you look at her: you know she hasn't slept more than
three hours any of the last few nights." Havers is scenery.
```

```
Havers is carrying a jotter called Barbara's notebook. The file of Barbara's
Observations is called "barbara". The text file of Barbara's notebook is the file of
Barbara's Observations. The heading of Barbara's notebook is "I could murder a
cup of tea".

The time of day is 6:32 AM.

Instead of examining your notebook when your notebook is fresh:
    say "Your notebook is blank. Back in the office, of course, there are a stack of
others. But you brought a fresh notebook in a kind of weary hope. You're going
to pretend, just for now, that this body might be unrelated to the graphic string of
murders you're already investigating."
```

---

444  **Example The Fifth Body** *RB*

An expansion on the notebook, allowing the player somewhat more room in which to type his recorded remark.

The implementation here is much like that of the previous example, except that we allow the player to write his notebook input as a separate command, leading to an exchange such as

```
>write in my notebook
You open your notebook and prepare to write in it.

>>Am beginning to think that HT and BGG are in this together.
You finish writing and fold your notebook away.

>read my notebook
You read:

Wednesday morning

Am beginning to think that HT and BGG are in this together.
```

The opening is much as before:

```
"The Fifth Body"

A jotter is a kind of thing. A jotter has an external file called the text file. A jotter
can be fresh or used. A jotter is usually fresh. A jotter has a text called the
heading.

The currently erased jotter is an object that varies.

To erase (pad - a jotter):
    now the currently erased jotter is the pad;
    write "[heading of the currently erased jotter][paragraph break]" to the text file
of the pad;
    now the pad is fresh.
```

```
To write in (pad - a jotter):
    append "[the time of day]: [player's command][line break]" to the text file of the
pad;
    now the pad is used.

To read (pad - a jotter):
    say "You read: [paragraph break][text of the text file of the pad]".

When play begins:
    repeat with pad running through jotters:
        erase the pad.

Instead of examining a used jotter (called the pad):
    read the pad.

Instead of examining a fresh jotter (called the pad):
    say "There is nothing of note in [the pad]."

Target jotter is an object that varies. The target jotter is usually nothing.

Understand "write in [something preferably held]" as writing in. Writing in is an
action applying to one thing.

Check writing in:
    if the noun is not a jotter, say "It would be better to write in a notebook."
instead.

Carry out writing in:
    now the command prompt is ">>";
    now the target jotter is the noun.

Report writing in:
    say "You open [the noun] and prepare to write in it."
```

Now what happens is that the player, having typed WRITE IN NOTEBOOK, will be faced with a ">>" prompt instead of the usual ">": a sign that the input mode has changed.

The next code is to react to reading a command. Whatever the player types at the >> prompt when the target jotter is set will now be recorded in the notebook, though with a character limit of about 60-100 characters depending on how much upper-case and punctuation he uses. (There are ways to lift the character length restriction as well, but they would take us into deeper waters.)

```
After reading a command when target jotter is a jotter:
    now the command prompt is ">";
    write in target jotter;
    now target jotter is used;
    say "You finish writing and fold your notebook away.";
    now the target jotter is nothing;
    reject the player's command.

Understand "erase [something preferably held]" as erasing. Erasing is an action
applying to one carried thing.
```

```
Check erasing:
    if the noun is not a jotter, say "It's hard to see how." instead.

Carry out erasing:
    erase the noun.

Report erasing:
    say "You scrub out all the entries in [the noun]."

The player carries a jotter called your notebook. The file of Player's
Observations is called "notebook". The text file of your notebook is the file of
Player's Observations. The heading of your notebook is "Sunday Morning".

The Vestry is a room. "[Havers] hangs back by the door: the forensics expert is
not finished with a preliminary examination of the body. From here you can't see
much, except that the expert has peeled back and laid to one side a liturgical
vestment that someone at the church used to cover the corpse until the police
came. What was once a cream silk with festive Easter embroidery is now stained
with blood-colored handprints."

Detective Havers is a woman in the Vestry. The description is "She looks glumly
back. There's still a purple-ish bruise on her cheekbone from the disaster
Thursday afternoon." Havers is scenery.

Havers is carrying a jotter called Barbara's notebook. The file of Barbara's
Observations is called "barbara". The text file of Barbara's notebook is the file of
Barbara's Observations. The heading of Barbara's notebook is "Sun. AM".

The time of day is 9:11 AM.
```

445

Example **Flathead News Network**

RB

Using external files, together with a simple Unix script running in the background, to provide live news headlines inside a story file.

This example can only work if we have a separate program running in the background while the story file is being played, and as such it will only work if we set things up in a special way. Exactly how to do this will vary from platform to platform.

First, the source text for the Inform end of the communication line:

```
"Flathead News Network"

The file of RSS Requests is called "rssrequest".

The file of RSS Responses (owned by project "RSS-SCRIPT") is called
"rssreply".

To request (RSS feed address - text):
    mark the file of RSS Responses as not ready to read;
```

```
    write the RSS feed address to the file of RSS Requests.

Newsroom is a room. "This is the secret nerve-centre of FNN, the Flathead
News Network."

The BBC button is in the Newsroom. Instead of pushing the BBC button: say
"Bong!"; request
"newsrss.bbc.co.uk/rss/newsonline_uk_edition/front_page/rss.xml".

The NASA button is in the Newsroom. Instead of pushing the NASA button: say
"Bang!"; request "www.nasa.gov/rss/breaking_news.rss".

The WHO button is in the Newsroom. Instead of pushing the WHO button: say
"Bing!"; request "http://www.doctorwhonews.net".

A screen is in the Newsroom.

Instead of examining the screen:
    if ready to read the file of RSS Responses, say "From the screen you read:
[line break][text of the file of RSS Responses]";
    otherwise say "The screen remains blank for now."
```

As far as the story file is concerned, then, it sends a request down the communication line by writing the chosen RSS feed to the file named "rssrequest", and expects a reply to come back down the line by being written to the file "rssreply". However, the story file needs to expect that this might take some time. (Maybe forever, if there is no program responding, or if the Internet connection is not working.) The story file marks the "rssreply" file as not ready before it makes a request; if it subsequently finds that the file is now ready, that must mean that the other program has done the honours, and that all is well. In the mean time, "The screen remains blank for now."

Now for the RSS-SCRIPT program. The following provides a crude but workable program suitable for running as a Perl script on a system which provides the standard Internet fetching program "curl": Mac OS X, for instance. (If you have OS X, you can paste the following into a (Unix-format) text file called "rss-script.pl", place it in your home folder, open the Terminal utility, type "perl rss-script.pl", and then hide the Terminal window again.)

```
for (;;) { # repeat forever:
    system("sleep 1"); # wait 1 second
    open REQUEST, "rssrequest.glkdata" or next;
    # the request file has been detected:
    $header_line = <REQUEST>; # the header line
    $rss_feed = <REQUEST>; # the actual content - the RSS feed URL
    close REQUEST;
    if ($header_line =~ m/^\*/) { # if the request file is marked ready
        $rss = system("curl $rss_feed >rawrss.txt"); # download the RSS feed
        # read the RSS XML into a single Perl string:
        open RAWRSS, "rawrss.txt" or next;
        $raw = "";
        while ($nl = <RAWRSS>) {
            $raw = $raw." ".$nl;
        }
        close RAWRSS;
```

```
        # look for the title and description of the first item:
        if ($raw =~ m/\<item\>\<title.*?\>(.*?)\<\/title\>.*?\<description.*?\>(.*?)\
<\/description\>/) {
            # write the reply:
            open REPLY, ">rssreply.glkdata" or next;
            print REPLY "* //RSS-SCRIPT// rssreply\n", $1, "\n", $2, "\n";
            close REPLY;
            # request safely dealt with, so we can remove it:
            system("rm 'rssrequest.glkdata'");
        }
    }
}
```

Chapter 24: Testing and Debugging

 Contents of *Writing with Inform*

 Chapter 23: Figures, Sounds and Files

 Chapter 25: Releasing

 Indexes of the examples

§24.1. Checking against the Index

Testing a story -- and indeed writing a story so that it is easy to test consistently -- is an art in itself. We should expect that we'll do some preliminary testing, both by running test commands and by playing through the story ourselves, and that we'll then hand on the story to beta-testers who will tell us about faults in the play experience that we haven't been able to see.

Every time Inform builds a new story file, it assembles a vast amount of information about that world, in the form of the Index. Often a visit to the Index is all that's needed to explain a piece of undesired behavior.

Is travel not working as it should? Check the World index and see whether the map shows the rooms arranged the way you thought.

Are objects not showing the behavior you'd expect based on their kind? Check the Kinds index and make sure they've been defined as the kind of thing you expected. For instance, we might find that we've written

```
The red door is west of Foo and east of Bar.
```

but not

```
The red door is a door.
```

A human reader wouldn't make this mistake, but Inform hasn't actually registered the red door as belonging to the door kind, and consequently has treated it as a room instead. All we need to do is add the kind declaration. The Kinds index will make that obvious.

When an error appears in the Index, there is often a link back to the source text that defined that room or object. If not, there's often at least some information about what rule or phrase might be responsible for it.

Start of Chapter 24: Testing and Debugging
Back to Chapter 23: Figures, Sounds and Files: §23.15. Exchanging files with other programs
Onward to §24.2. Debugging features to use in source

---

§24.2. Debugging features to use in source

The TEST command is an extremely useful way of managing a story and continuing to verify that it does everything we want. We can create new test commands of the form

```
Test me with "up / kill captain eo".
Test eo with "zap eo" holding the ray gun.
Test dinner with "eat bread / eat soup / eat butter" in the Ship Cafeteria.
```

and we are free to have as many of these tests as we would like. Test commands can call other tests, as well, so we might have a test command such as

```
Test megatest with "test me / test eo".
```

A word of warning: if the first command in the test is "again", that will likely repeat the TEST command, sending Inform round in circles forever.

For complicated objects and commands, sometimes it's a good idea to develop the test commands at the same time that we're writing the source code itself. Each time we add a new rule or piece of behavior, we also add to that object's special test command something that will put that new feature to the test. This means that we can keep running the test command as we work and verify that everything is behaving as expected.

Sometimes we need to get a look at what is happening within the source itself. Many of the most annoying bugs come about because we're making some assumptions about what's true in the story world that differ from Inform's assumptions. When that happens, we may need to add something to the source to check that the variables are set to what we think, that certain parts of the source are being reached, and so on.

For instance, suppose we have a phrase like this:

```
To say key score:
    let count be the number of keys which are not carried by the player;
    if count is greater than 2 and the player is timid:
        say "You're still missing a lot of keys, bucko!"
```

Now, we expect this to print something, but perhaps it's not doing so when we had anticipated that it would. At some point when we think the count is greater than 2 and the player is timid, at least one of those things is not true. An easy way to check up on this is to add a showme line to the source, like so:

```
To say key score:
    let count be the number of keys which are not carried by the player;
    showme count;
```

```
if count is greater than 2 and the player is timid:
    say "You're still missing a lot of keys, bucko!"
```

and this will then check the relevant number and print it to screen when this phrase is called, like so

```
"count" = number: 1
```

In this case, it looks like the count is not high enough to trigger the text, so we can concentrate on working out why that might be. Maybe we didn't correctly define something as a key, for instance.


Start of Chapter 24: Testing and Debugging


Back to §24.1. Checking against the Index


Onward to §24.3. High-level debugging commands

---

§24.3. High-level debugging commands

If an object is not responding in the way we expect, it may be that we're wrong about where it is or about some of its current properties or relations. We can find our current location and the things around us by typing

```
>SHOWME
Boudoir - room
    four-poster bed - supporter
    yourself - person
    pillow
```

and similarly we can inquire about the status of a particular object during play by typing SHOWME and the object's name:

```
>SHOWME BAT
bat - thing
location: on the table in Locker Room
singular-named, improper-named; unlit, inedible, portable, patterned
printed name: "bat"
printed plural name: none
indefinite article: none
description: none
initial appearance: none
```

This will work even if we're not in the same location as the object we want shown.

Another common type of problem is one in which we type a command but Inform does not perform the action that we were expecting as a result. In some cases, this is because the command we're typing is actually triggering some other action. An easy way to check on this is to type ACTIONS before issuing the command that is behaving unsatisfactorily. Thus:

```
>ACTIONS
Actions listing on.

>JUMP
[jumping]
You jump on the spot.
[jumping - succeeded]
```

This tells us how Inform interpreted our input and whether the action was successful or failed for some reason. If the command is being understood as a different command than we expected, that may mean that we have made a mistake in our Understand instructions, and need to double-check these.

Sometimes, however, the action is being correctly understood, but the action rules that are firing are producing a result other than we'd like. If we want to see which rules are running, we can type

```
>RULES
Rules tracing now switched on. Type "rules off" to switch it off again, or "rules all" to
include even rules which do not apply.

>JUMP
[Rule "announce items from multiple object lists rule" applies.]
[Rule "set pronouns from items from multiple object lists rule" applies.]
[Rule "before stage rule" applies.]
[Rule "instead stage rule" applies.]
[Rule "investigate player's awareness before action rule" applies.]
[Rule "player aware of his own actions rule" applies.]
[Rule "check stage rule" applies.]
[Rule "carry out stage rule" applies.]
[Rule "after stage rule" applies.]
[Rule "investigate player's awareness after action rule" applies.]
[Rule "report stage rule" applies.]
[Rule "report jumping rule" applies.]
You jump on the spot.
[Rule "last specific action-processing rule" applies.]

[Rule "A first turn sequence rule" applies.]
[Rule "every turn stage rule" applies.]
[Rule "A last turn sequence rule" applies.]
[Rule "notify score changes rule" applies.]
>
```

As we can see, RULES produces a lot of output, much of which is probably irrelevant to whatever problem we're tracking down. Nonetheless, knowing exactly which rule is printing undesirable output is helpful, especially if that rule comes out of an extension or some other source that we did not write ourselves: this output has told us that the text we saw came from the "report jumping rule".

To find out more about what is going on in specific rules, we can also turn to the Index tab under Actions and click through to that specific action. From there we will be able to see which rules are included, what responses they're writing, and where they were defined in the source text.

SCENES lists which scenes are currently playing and which are complete. This is valuable if scene-triggered events are not happening when we expect them to.

RANDOM sets the random number generator to a predictable seed. If we include this in a test command, it will guarantee that the subsequent behavior of the story is consistent across multiple playthroughs, which is helpful if we're trying to test something to do with, say, randomly wandering non-player characters.

RELATIONS lists all the relations defined in the story, except for things like support and containment that are part of the world model and are so numerous that the output would be overwhelming.

RESPONSES lists all the named responses established by all the extensions currently included. This can be informative, or it can be a bit overwhelming. Except where responses have been changed at runtime, the same information is available in a different form in the Index on Actions. If we're interested in a particular single response, digging into the actions index is probably the easiest way to find it.

If, however, we want a rapid overview of all the responses provided by a given extension (perhaps an extension we are ourselves writing), the RESPONSES command can be a help.


Start of Chapter 24: Testing and Debugging


Back to §24.2. Debugging features to use in source


Onward to §24.4. Low-level debugging commands

---

§24.4. Low-level debugging commands

In addition to the commands designed around Inform 7's data model, there are several debugging commands that have persisted since the days of Inform 6. These are generally less useful or necessary with Inform 7's features, but there are still times when they can come in handy for a quick and dirty resolution of a problem during gameplay. They are as follow.

PURLOIN moves an object to your possession, no matter where it is on the map, like so:

```
>PURLOIN TABLE
[Purloined.]

>I
You are carrying:
a table
```

Note that purloin does not consider the usual rules about whether something can be taken. In this case, we've just moved the table to our inventory even though it is a fixed in place supporter that could not be taken in the normal course of events.

Because purloin works on things that are far away as well as things that are close, it has to do a lot of extra parsing work and may take a moment or two to complete if we try it in a very large story. It is generally more efficient to give the player the relevant object using a testing command, like this:

```
Test me with "drop table" holding the table.
```

Nonetheless, there are occasionally times when we're halfway into a 2000-move story and suddenly realize we implemented a vital object in the wrong room, making the story unsolvable. We could fix the bug, press replay and return to this story state fairly quickly, but if we don't feel like waiting even that long, PURLOIN will resolve the issue.

ABSTRACT is PURLOIN's less useful cousin, allowing the player to move an object from one place to a specified other place, as in

```
Bar
You can see a table here.

>ABSTRACT KEY TO TABLE
[Abstracted.]

>LOOK
Bar
You can see a table (on which is a key) here.
```

GONEAR transports the player instantly to the vicinity of the named object, so for instance

```
>GONEAR GRAIN
Fertile Plain
You can see some grain here.
```

As a debugging command, this isn't protected in the ways that commands usually are. It's possible to type GONEAR NORTH and produce a run-time error when Inform tries to move the player into the object that represents the compass. Again, except in cases where we're tracing a problem very deep in an already running story, it is usually more practical to write a test command to put the player in the correct situation, as in

```
Test me with "eat grain" in the Fertile Plain.
```

VERIFY checks that the story file is intact rather than damaged, but it is hard to think of an occasion when this would be likely to arise within the Inform application. The command is a holdover from a time when data transfer was much slower and more error-prone, and it was plausible to have a story file of just a few hundred KB corrupted during transmission.

TREE creates a list of object containment. It is similar to SHOWME, but less elegant and thorough.

SCOPE lists the objects that are currently in scope for the player, which is to say, things that could be referred to when we're typing a typical command. Thus:

```
Bar
You can see a table here.

>SCOPE
1: yourself (574631)
2: a table (574759)
```

The following numbers are object IDs for these objects, which can distinguish items with identical names. It is likely that the output of this will not be terribly interesting or different from checking SHOWME, except in cases where the author is deliberately changing the scope to be something other than "the set of things that are visible in the room with the player right now". This usually involves the Deciding the scope of something activity (see the chapter on Activities).

SHOWHEAP shows how many bytes are currently free. This is usually not helpful.

SHOWVERB (verbname) lists the Understand information associated with a particular verb. Similar information, in a vastly more palatable form, is available in Index / Actions / Commands, so the one time SHOWVERB becomes useful is when Inform is considering the understand lines in the wrong order and producing a result we didn't want: SHOWVERB will show us the order in which the lines are being assessed. The challenge will then be to add conditions to the Understand lines to move them into the correct order.

Finally, TRACE (and its more advanced stages TRACE 2, TRACE 3, TRACE 4, and TRACE 5) will reveal things, more things than we ever wanted to know, about the assumptions being made by the parser when it takes in a command. In practice this information is almost never useful to an Inform 7 author.

There is no guarantee that any of these commands will make life better or that they won't crash the story or put it into an unwinnable state. There is also no absolute guarantee that they won't be withdrawn entirely from future versions of Inform. Consider them as Old High Magic, and treat accordingly.


Start of Chapter 24: Testing and Debugging


Back to §24.3. High-level debugging commands


Onward to §24.5. Adding new testing verbs and Release for Testing

---

§24.5. Adding new testing verbs and Release for Testing

As we saw in Chapter 2, we can mark some of our source text so that it will not be included in a finished story. This means that we can add special testing commands available to the author but not available to our eventual players. This is a good way to add our own suite of testing verbs to a story beyond the "Test me with..." features already described.

Here are some types of testing verbs that can be useful to add:

Chapter jumps. We might create test commands that took us to a later stage of the story (perhaps doing more setup than "Test me..." alone can handle).

Status information. We might create a test command that would show us status information beyond what's covered in the Standard Rules. For instance, if we had a story that heavily modeled the moods of other characters and we wanted to be able to check those moods at any time, we might add a SHOWMOOD command that would tell us about a character's emotional state.

Puzzle satisfaction lists. Some simulation-rich stories offer puzzles that can be solved in a variety of ways: for instance, a sealed glass box that can be smashed with any object that has been marked with the properties "hard" and "heavy". Later, we might want to be able to check which in-story objects would work as a solution to this puzzle, so we might create a command like

```
Listing hammers is an action out of world applying to nothing.

Understand "list hammers" as listing hammers.

Carry out listing hammers:
    say "These things can break the glass: [line break]";
    repeat with item running through portable hard heavy things:
        say "[item][line break]";
```

so that we can review that there are enough objects available and that the list doesn't include anything it shouldn't. In a small story this kind of thing is pretty easy to keep track of in the author's head. Large stories can contain thousands of objects, however, at which point it becomes valuable to have an automated method of verification.

Just occasionally, we might also want to build a version of a story that will allow beta-testers access to the debugging commands. This is especially relevant for long stories: if we're testing a story with a lot of playtime and the testers have already thoroughly reviewed the first portion of the story, we might want to let them have access to testing commands that fast-forward to later sections.

To do this, we can use the "Release for Testing" feature. Release for testing builds a version of the story that *does* include testing commands and any sections labeled "Not for release".


Start of Chapter 24: Testing and Debugging


Back to §24.4. Low-level debugging commands


Onward to §24.6. Testing for thoroughness

---

§24.6. Testing for thoroughness

The presence of actual bugs or defects is not the only thing we want to consider when testing a story. We may also want to check whether we have built the story with a consistent amount of depth.

Are there descriptions for everything the player might look at? If we've implemented special verbs, do they have appropriate reactions for all the different objects? If most objects in a story about restaurant reviewing have a special response to being tasted, for instance, it might be disappointing for the player to encounter late-added objects that don't.

Checking implementation thoroughness can be a laborious process, but there are a few things we can do to automate it. For instance, we might add to a not-for-release section a rule that checks for certain properties:

```
When play begins (this is the run property checks at the start of play rule):
    repeat with item running through things:
        if description of the item is "":
            say "[item] has no description."
```

This will confront us with a reminder of what we still need to fill in every time we start up the story.

There are also some extensions that are designed to assist with this, notably the massive Object Response Tests by Juhana Leinonen. Object Response Tests allows us to try out a long list of commands against any object in the story, so that we can quickly identify ones with nonsensical replies.


Start of Chapter 24: Testing and Debugging


Back to §24.5. Adding new testing verbs and Release for Testing


Onward to §24.7. Commands for beta-testers

---

§24.7. Commands for beta-testers

Inform includes a command that is especially designed to help beta-testers report flaws: namely, TRANSCRIPT. A tester can type TRANSCRIPT (or just SCRIPT) at the beginning of the story in order to start generating a recording of everything that happens. She can then add her own annotations when something buggy or otherwise notable occurs (for instance by typing a standard symbol, such as *, followed by a note).

When she then sends us the completed transcript, we can look through for these symbols and note the problems the tester found in the context of the rest of the story's behavior. Having information about how she reached that position typically makes it much easier to reproduce the problem than if she gave only a general account of it.


Start of Chapter 24: Testing and Debugging


Back to §24.6. Testing for thoroughness


Onward to §24.8. Help from the user community

---

§24.8. Help from the user community

Sometimes we get really stuck on a problem and despite all our best efforts cannot figure out how to solve it.

Fortunately, Inform has a lively and helpful community of users who are often willing to assist other authors. The easiest way to reach these users is to make a post at the intfiction forum at

```
http://www.intfiction.org/forum
```

and in particular to post Inform-related problems under the topic Inform 6 and 7 Development. Where possible, it's a good idea to post the example source that is causing trouble, and to make it as short as possible so that prospective helpers will not have to read any more than necessary in order to pinpoint the problem.

The user community is also a good place to find beta-testers who can try out our work and give feedback.

 Start of Chapter 24: Testing and Debugging

 Back to §24.7. Commands for beta-testers

 Onward to Chapter 25: Releasing: §25.1. The finished product

---

Chapter 25: Releasing

Contents of *Writing with Inform*

Chapter 24: Testing and Debugging

Chapter 26: Publishing

Indexes of the examples

§25.1. The finished product

This chapter and the next are about what to do when we have a complete, finished work on our hands.

For almost all of the time when a new work of IF is being written, it lives inside the familiar two-panel spread of the Inform user interface. But that isn't how eventual players will experience it. They will want to play a "story file" in a standard format, and they will do so with a wide range of different interpreters on many different computers or websites, including some -- like mobile phones -- on which Inform itself will not run.

So how does a new work of IF reach players? The simple answer, covered in this chapter, is that clicking the Release button instead of Go causes Inform to output a stand-alone story file. But as we will see, Release can do much more than that: it can attach covers, include bibliographic data, make websites and much more. Releasing is the process of making all of the material we want to deliver to our eventual players.

But that is only the first step. What do we do with the material when we have it? Printing out a manuscript of a novel is not the same as publishing it. So the next chapter, on Publishing, completes the story.

Start of Chapter 25: Releasing

Back to Chapter 24: Testing and Debugging: §24.8. Help from the user community

Onward to §25.2. Bibliographic data

---

§25.2. Bibliographic data

Almost all printed books have a title page and a so-called "imprint" page, often its verso, which make up a description of the contents. The title page gives the name of the book and of the author, while an imprint page contains a variety of details about the edition, the printing, and so on. An ISBN number is allocated so that, from the number alone, any book seller or cataloguer can identify exactly this work. Sometimes other cataloguing information is added, such as the Library of Congress classification. This set of information is called "bibliographic data", and without it libraries and booksellers would be at a total loss.

IF has bibliographic data, too. Inform has a number of special named values to hold this - who wrote the work being created, what it is called, what headline it has, what genre it has and what its release number is, and so on.

These can be set as follows:

```
The story title is "Mansfield Perk".
The story author is "Janet Austen".
The story headline is "An Interactive Romance".
The story genre is "Romance".
The release number is 7.
The story description is "In Miss Austen's new interactive novella, Miss Henrietta Pollifax
is adopted by the tempestuous landowner Sir Tankerley Mordant, and must make a new
life for herself on the rugged moors."
The story creation year is 2005.
```

Most of these are self-explanatory. The "story creation year" is provided so that if we need to revise the work to fix some bugs a year later - by no means an uncommon occurrence - then we can make sure it is correctly identified as still being basically a 2005 work. (Just as a book which has had innumerable revised printings may say "First published 1988" on its imprint page.) The "story description" is a piece of text, analogous to the back cover blurb on a book: it might be two or three paragraphs long, so the example above is rather minimal, but it should not be epic in length.

As we have already seen, a convenient abbreviation provides that if the first sentence of the source text consists solely of text in quotation marks, then that is considered the title. Thus if the source begins:

```
"Mansfield Perk"
```

then that will be the "story title". Further, we can write

```
"Mansfield Perk" by Janet Austen
```

with the obvious effect: quotation marks around the author's name are optional here, for convenience, but note that we'd better have them in cases like:

```
"Three Men in a Boat" by "Jerome K. Jerome"
```

as otherwise the full stop after the K will end the sentence prematurely.

The text of these bibliographic descriptions cannot normally include text substitutions, since they are written into external descriptions of the story file as part of its "binding". Two exceptions are allowed, though: "[']" makes a literal apostrophe, and can be used if we need to override Inform's normal conventions to do with converting apostrophes at the ends of words to double-quotes. For instance:

```
"Summer of [']69" by Buzz Aldrin
```

The other exception is that the "[unicode ...]" text substitution works, so for example:

```
The story description is "This is a sentence[unicode 8212]with a parenthetical in
dashes[unicode 8212]because 8212 is the Unicode number for an em-dash. But for
example, 'pawn to [unicode black chess bishop]4' draws in a black chess bishop, so it
works with names, too."
```

If the bibliographic named values are not set by the source text, Inform will still need to say something. Unset text and number variables evaluate to "" and 0 respectively, but this would make for a very unhelpful record. So Inform uses the following table instead of any value which is unset:

```
Story title: Untitled
Story author: Anonymous
Story headline: An Interactive Fiction
Story genre: Fiction
Release number: 1
```


Start of Chapter 25: Releasing


Back to §25.1. The finished product


Onward to §25.3. Genres

---

§25.3. Genres

The "story genre" is not used in the banner at all, and exists purely to help librarians. If it is at all possible to do so, authors are asked to use one of the following standard categories:

```
Comedy, Erotica, Fairy Tale, Fantasy, Fiction, Historical, Horror, Mystery, Non-Fiction,
Other, Romance, Science Fiction, Surreal
```

These categories are based on those currently used by bookshops, but a few notes may be helpful. "Fiction" is intended for works whose essential purpose is literary, in a way which trumps any subject they happen to have: if Julian Barnes writes a mystery, for instance, a bookshop will shelve it with modern novels rather than in the detective stories section, whereas P. D. James's Adam Dalgliesh mysteries will end up filed with detective fiction even though she has appreciable claims to be an important novelist.

"Comedy" is used rather than "humour" to avoid the clash of spellings with "humor". This genre includes parodies.

"Non-Fiction" would be used for a work of IF which is essentially a presentation, perhaps in a novel interactive format, of true information. A meticulous simulation of the Great Exhibition of 1851, for instance, might qualify.

The distinction between "Surreal" and "Other" is that "Surreal" works contain at least some semblance of narrative, whereas "Other" is intended for works which "abuse" the format to present some entirely different sort of game - Tetris, say, or Minesweeper.


Start of Chapter 25: Releasing


Back to §25.2. Bibliographic data


Onward to §25.4. The Library Card

---

§25.4. The Library Card

Bibliographic data is useful for two reasons. Firstly, it enables the equivalent of a title page to be printed - traditionally called the "banner" - at the start of play; secondly, Inform uses it to generate the equivalent of a library card for the work, which can be used by other programs to help organise, sort and classify interactive fiction. If the card is given to any other program on any other machine (or an Internet-based archive) then, in principle, that system can know about our work of fiction without a human librarian having to get hold of a copy, play it and laboriously copy out the details.

The "library card" is not of course a physical card, but a small "metadata" file which could potentially be transmitted quickly across the Internet. It contains no personal data other than what you choose to put on it, using the sentences documented in this chapter: it does not, for instance, identify your computer or IP address. In any case Inform does not send it anywhere, but merely keeps an up-to-date copy within the project, and includes it when making a release copy of the work. You can always see (a representation of) the current library card for a project in the Contents index.

Authors are asked to play fair, in return, by writing sensible and useful bibliographic information for any work which is likely to circulate to other people; by being honest (writing under a pseudonym is fine, but not impersonating other people); and by conforming to standard practice.

The Settings panel of each project contains a tick-box called "Bind up into a Blorb archive on release", and by default this is ticked. "Blorb" is a nonsense word from a popular early 1980s work of IF called "Enchanter", where it was the name of a magic spell whose purpose was to "safely protect a small object as though in a strong box". In the late 1990s, the name was borrowed for a standard format for what might be called the wrapping and packaging of IF. A typical Blorb archive produced by Inform contains the "story file" - the actual program for the story - together with its library card and cover art.

Modern IF interpreters such as Zoom for Mac OS X and Unix, and Windows Frotz, can play blorb archives directly, and the authors of Inform hope to make this the normal practice in future. Still, some interpreters cannot read blorbs directly and have to be given the actual story file: so by unchecking the above tick-box, we can insist that Inform creates only that.

The disadvantage with this, of course, is that the library card (with all its bibliographic data) and any cover art is lost in the process.


Start of Chapter 25: Releasing


Back to §25.3. Genres


Onward to §25.5. The Treaty of Babel and the IFID

---

§25.5. The Treaty of Babel and the IFID

During March and April 2006, an agreement was reached between the IF archive and most of the different systems for creating IF - of which Inform is only one - called the Treaty of Babel. While these different systems create computer programs which are quite different internally, the Treaty provides for works of IF to come with bibliographic data which identifies them in a standard way.

Inform is fully compliant with the Treaty. In particular, each new project created by Inform is allocated a unique identification number called its IFID. The IFID is the equivalent for IF of the ISBN of a printed book. Inform copies it onto the "library card" for the benefit of Internet-based libraries which may eventually accession the work. Of course many projects start but never see the light of day, so many possible IFIDs are "wasted": but that hardly matters, as there are plenty more numbers in the world.

The important thing is that

The IFID number must be unique to this one work out of all the IF ever created

Inform will make sure this is true, **unless** we do something to break this ourselves. For instance, if we take an existing project, copy it as a file, then work divergently on the original and on the copy so that they become two radically different works, they will still each have the same ID. This is a bad thing: if we want to duplicate a project but then turn it into something new, the best way to do that is to create a new project, and to copy and paste the source from the old to the new.


Start of Chapter 25: Releasing


Back to §25.4. The Library Card


Onward to §25.6. The Release button and the Materials folder

---

§25.6. The Release button and the Materials folder

Inform's Release button does two things: it makes a stand-alone, public version of the current project - a "story file" - and it gathers up, or creates, whatever material we want to go with it.

The release version of the project can be played by anyone with an "interpreter" - they do not need the Inform application installed on their computers, and they will not be able to see the source text. Released versions differ slightly from the versions playable in the Story panel of

Inform, because debugging commands such as ACTIONS are not included with them. (As we've seen, also excluded is any material in the source text under a heading including the words "not for release".) In some cases, if we release along with an interpreter, we can even make the project playable from a web browser, so that the player doesn't need to install any software at all, not even a browser plugin.

The Release button also creates a ".materials" folder for the project, if one doesn't exist already. (On some platforms, the Inform user interface creates it automatically alongside the project.) Inform adopts the following convention:

> The files associated with the project "Whatever.inform" should all be kept in a subfolder called "Whatever.materials" in the same folder that contains the project.

For example, if we have a project filenamed Magician.inform which lives in a folder called "Works in Progress", then files might be arranged like so:

```
Works in Progress
    Magician.inform
    Magician.materials
        Collegio.pdf
        Mating Wyverns.mp3
```

Of course "Magician" might not actually be the title of the project - it might be an abbreviation, or a working title. The name of the .materials folder has to match the name of the .inform file, not the title.

Several advanced features of Inform make use of the materials folder, and the "Release" button is one of them. It creates a further subfolder called "Release" within the materials folder. This is where it will always place the story file it creates, together with anything released "along with" the story - Inform will not need to put up a dialogue box asking us where to save the story file, because there is already a natural place. For instance, after a successful click on Release, we might then see:

```
Works in Progress
    Magician.inform
    Magician.materials
        Collegio.pdf
        Mating Wyverns.mp3
        Release
            Collegio.pdf
            Magician.zblorb
            Mating Wyverns.mp3
```

where "Magician.zblorb" is the actual story file produced by Inform. Note that Inform has made copies of the files to be released with it - the idea is that the Release subfolder contains only what Inform makes, and everything in the Release subfolder can be thrown away at any time.

This is especially useful if we're releasing along with a website (see below), as then the Release subfolder will be exactly what needs to be uploaded to a server to be shown to the world. Equally, the Release subfolder is what can be zipped up and uploaded to archives or (if small enough) emailed out.

Start of Chapter 25: Releasing
Back to §25.5. The Treaty of Babel and the IFID
Onward to §25.7. The Joy of Feelies

---

§25.7. The Joy of Feelies

"Feelie" is a slang word, again going back to the early days of IF, for something tactile included with commercially sold copies of IF games. For instance, Infocom's "Wishbringer" was not just a diskette in a pretty box: the box also contained a map, a letter, an envelope, a magic stone (well, a stone) and a booklet. Most of this was purely for fun, and to flesh out background to the story, but there would usually be clues sneaked into the text or artwork as well.

Today's IF is usually not supplied in physical packaging, and not accompanied by physical objects. But authors do sometimes want to include extraneous matter, whether it's a simple read-me file of instructions or a multimedia extravaganza. Inform does not provide facilities to make artwork, movies, soundscapes, booklets, etc.: there are plenty of programs out there to do all of that already.

But Inform does help with the collation and packaging-together. For instance, by placing the following sentence in the source text:

```
Release along with a file of "Collegio magazine" called "Collegio.pdf" and a file of "The mating call of the green wyvern" called "Mating Wyverns.mp3".
```

...we tell Inform that we will also be providing two additional files. Note that in each case we supply a brief description and a filename. The filename should always have a standard file extension for a well-known and thoroughly standardised file format - ".pdf" and ".mp3" are pretty safe: so for instance are ".txt", ".png", ".jpg", ".html". The filename should not include punctuation marks other than the full stop dividing name from extension, and should not exceed 30 characters in length.

It is also possible to supply a feelie which is not a single file, but is a mini-website: that is, a collection of interlinked HTML (and perhaps other) files. The convention here would be:

```
Release along with a file of "Baltrazar's Guide to Magic" called "Guide".
```

The absence of a file extension on the filename "Guide" tells Inform that the feelie in question is a mini-website: it is expected to sit inside a folder called "Guide", with its home page being "Guide/index.html".

We have seen that Inform takes the story file, which is analogous to the pages of a book, and places it into a Blorb archive, analogous to the binding. These new additional files are not placed in the Blorb, because that would make the Blorb archive rather large (and would hide them from the player, which defeats the purpose). But references to them do appear in the Blorb, so that any interpreter playing the Blorb would be able to tell that there are supposed to be additional files available. Similarly, references are entered onto the library card.

Start of Chapter 25: Releasing
Back to §25.6. The Release button and the Materials folder
Onward to §25.8. Cover art

---

§25.8. Cover art

Accompanying files are not the only things which can be included in a "release along with" sentence: for instance, we could

```
Release along with cover art ("A stone gargoyle"), a file of "Collegio magazine" called
"Collegio.pdf" and a file of "The mating call of the green wyvern" called "Mating
Wyverns.mp3".
```

Cover art can not only be used to advertise a work of IF, it is also displayed to players by certain interpreters, such as Zoom or Spatterlight for OS X, or Windows Frotz for Windows. It is also used on the IFDB (ifdb.tads.org), and by browsing applications. If Zoom is installed, then on Mac OS X Leopard, the Finder shows cover art directly:

Cover art for a work should be prepared in either JPEG (".jpg") or PNG (".png") format, and we recommend that it should be square, like a music album cover. Programs which notice the cover art for a work of IF are likely to scale this up or down as convenient for their own display purposes, but it would be helpful to provide the original art at 960 by 960 resolution. (Remember, it is intended for use on screen, so providing art at magazine full-page print quality will add nothing and will only make the resulting release an annoyingly large download: even as it is, 960x960 will print out as a pretty respectable CD insert sleeve.) The cover art must not be smaller than 120 pixels in either dimension.

To provide cover art, we should create two files: Cover.jpg (or .png), and a reduction of the image to a smaller "thumbnail" version called Small Cover.jpg (or .png). These should be placed in the project's .materials folder. For instance, we might have:

```
Works in Progress
   Magician.inform
   Magician.materials
      Collegio.pdf
```

```
Cover.jpg
Mating Wyverns.mp3
Small Cover.jpg
```

(supposing that, as in the previous examples, "Collegio.pdf" and "Mating Wyverns.mp3" are the filenames of two feelies that accompany the release). "Small Cover.jpg" should be a reduction of the main cover art image "Cover.jpg" to exactly 120 by 120 pixels.

The text in brackets after the release instruction...

```
Release along with cover art ("A cathedral at sunset.").
```

...is provided for the benefit of blind or partially sighted users, and should be brief.


Start of Chapter 25: Releasing


Back to §25.7. The Joy of Feelies


Onward to §25.9. An introductory booklet and postcard

---

§25.9. An introductory booklet and postcard

When IF is aimed particularly at people who have never played IF before, there are certain conventions which it's a good idea to explain, or players will simply not know what to do. It can become a chore writing a clear set of instructions, and then there is the further nuisance of explaining about the need for an interpreter program to play the IF story file.

To alleviate this, Inform can "Release along with an introductory booklet", as for instance in this example:

```
Release along with cover art, the introductory booklet, a file of "Collegio magazine"
called "Collegio.pdf" and a file of "The mating call of the green wyvern" called "Mating
Wyverns.mp3".
```

The introductory booklet is a standard 8-page PDF file, written and designed by Emily Short, which contains all the basic information needed for a player to get started. It has been written to be as general-purpose as possible, in the hope of being useful for a range of widely different works of IF. There will certainly be works to which it would not be an appropriate supplement, and some authors will certainly prefer to write their own notes for players, but of course it is not compulsory. By making it available as a convenience, the authors of Inform do not intend to say that these are the "official" instructions or that others are not. It is simply intended as a time-saver.

As an alternative, or a supplement, it's also possible to:

```
Release along with an introductory postcard.
```

which supplies a standard postcard about IF (everything new players need to know, at one glance) written and designed by Andrew Plotkin and Lea Albaugh.

Start of Chapter 25: Releasing
Back to §25.8. Cover art
Onward to §25.10. A website

---

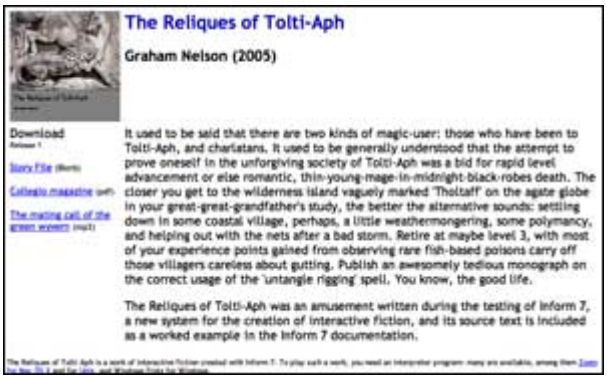
§25.10. A website

Much of the published IF of the last twenty years came with a brief text file describing what it was - a release note. Today it makes more sense to write this as a small web page, which can either be placed online, or simply distributed as part of the release.

Inform is able to manufacture such a website automatically. We request this by writing, for instance,

```
Release along with cover art, a website, a file of "Collegio magazine" called
"Collegio.pdf" and a file of "The mating call of the green wyvern" called "Mating
Wyverns.mp3".
```

where the list of ingredients now includes "a website". In fact, Inform makes only a single web page, called "index.html", which it places in the materials folder (as set up in the previous section): this then contains suitable links to all the other material, such as the cover art images, if they are also provided. For instance:



After a successful release now, then, we should see:

```
Works in Progress
    Magician.inform
    Magician.materials
        Collegio.pdf
        Cover.jpg
        Mating Wyverns.mp3
        Release
            Collegio.pdf
            Cover.jpg
            index.html
            Magician.zblorb
            Mating Wyverns.mp3
            Small Cover.jpg
        Small Cover.jpg
```

Start of Chapter 25: Releasing
Back to §25.9. An introductory booklet and postcard
Onward to §25.11. A playable web page

---

§25.11. A playable web page

Modern web browsers are now so powerful as computing environments that they almost amount to general-purpose computers in their own right. The websites made in the previous section were passive, and simply displayed information about a story file. But it's also possible to make a more active page - one which can play the story file, right inside the browser, for anybody who visits.

To make such a page, we must:

```
Release along with an interpreter.
```

This automatically releases along with a website as well, since we need the website in order to house the new page, which will be called "play.html". This page will be bundled up with a customised copy of a story file interpreter coded in Javascript - in effect, a program for a web browser to follow - and a suitably encoded version of the story file. The practical effect should be that anyone visiting the page with any modern browser can just play.

Inform ships with the "Parchment" and "Quixe" interpreters built in, one intended for each of the story file formats it can output - so by default Inform uses Parchment if the format (on the project's Settings panel) is set to Z-code, and Quixe if the format is Glulx. But Inform also supports the use of any other interpreter the author wants to try (including, for instance, later versions of Parchment or Quixe than the built-in ones). If we have access to an exotic Javascript-based interpreter called, let's say, "Urbzig", then we can install it by putting it into the "Templates" subfolder of the ".materials" folder for the project, and then ask for it to be used instead of "Parchment" like so:

```
Release along with the "Urbzig" interpreter.
```

Start of Chapter 25: Releasing
Back to §25.10. A website
Onward to §25.12. Using Inform with Vorple

---

§25.12. Using Inform with Vorple

"Vorple" is an innovative system by Juhana Leinonen for allowing web-based Inform stories to make use of web controls and other gadgets. Using Vorple, a story can in principle have an entirely different user interface, and can make much better use of CSS styling, interface to Javascript libraries, and so on.

At time of writing, "Vorple" is limited to Z-machine stories only, but it's hoped to bring it to Glulx stories too by 2015. Further information can be found at its home page:

```
http://vorple-if.com
```

In the mean time, users can get started with Vorple by downloading the Vorple extensions (available on the Public Library), and by using the release instruction:

```
Release along with the "Vorple" interpreter.
```

which is included with the current build of Inform.

Start of Chapter 25: Releasing

Back to §25.11. A playable web page

Onward to §25.13. Website templates

---

§25.13. Website templates

Web pages are very idiosyncratic things and Inform will almost certainly not produce exactly what we want. What it actually does is to take an existing "template" web page, and paste in the relevant information to make the final product. So by starting with a different template, we can end up with an entirely different-looking web page: like this one, for instance -

The template ordinarily used by Inform is called "Standard" and comes built in. (A second built-in template, "Classic", imitates the look used in 2005-08. The word "classic" here is to be understood in the sense of Classic Mac OS, the classic Doctor Who adventure "Time and the Rani", classic Mayan civilisation, and so forth - really pretty awful.)

Any other templates we must make ourselves, giving each one a different name, by convention a single word. In this section, we'll make a new one called "Platinum".

Suppose we write:

```
Release along with cover art, a "Platinum" website, a file of "Collegio magazine" called
"Collegio.pdf" and a file of "The mating call of the green wyvern" called "Mating
Wyverns.mp3".
```

This is identical to the previous version except for the "Platinum": note the quotation marks. When it needs to find a template, Inform searches the following places in sequence:

```
(a) the "Templates" subfolder of the project's own .materials folder, if this subfolder
should exist;
(b) the "Templates" folder in the user's own library - on Mac OS X, this is:
    ~/Library/Inform/Templates
or on Windows:
    My Documents\Inform\Templates
or on Linux:
    /Inform/Templates
(c) the built-in stock of templates, currently only "Standard" and "Classic".
```

What Inform looks for is a folder name matching that of the template - so in our case we need to provide a folder called "Platinum", and put it in either location (a) or (b).

The template folder is expected to contain some combination of the following files:

```
Platinum
    index.html
    source.html
    style.css
    (extras).txt
```

There are two HTML pages here, one for the main front page, the other for pages of displayed source text (if we release along with the source text - see later in the chapter). The CSS file defines styles of text - sizes, fonts, colours, and so on - and positions material on the page. The "(extras).txt" - which is optional, of course - allows additional HTML pages, images, movies and so on to be added.

If any of these is missing, Inform uses the one in "Standard" instead. In practice, this means the easiest way to create a new template is to supply just a new CSS file, which can change the colour, font, type size, and position of more or less everything in the site:

```
Platinum
    style.css
```

We probably want to start from the "Standard" version of "style.css" and edit in a few changes; the easiest way to get a clean copy of "Standard"'s CSS file to work on is to release the project with a "Standard" template, which causes this default "style.css" to appear in the "Release" subfolder of the project's .materials folder. (But it's wise to move the file out of "Release" before starting to edit it - files in "Release" are overwritten by Inform whenever a release is made.)

This is not the place to describe how CSS works. CSS is a more or less universal format today for describing how web pages should look - their style rather than their content. A dazzling variety of possibilities can be seen at the excellent:

```
www.csszengarden.com
```

but of course there are many, many other textbooks and websites which describe CSS.

Start of Chapter 25: Releasing
Back to §25.12. Using Inform with Vorple
Onward to §25.14. Advanced website templates

---

§25.14. Advanced website templates

The following describes how Inform uses the extras file and the two HTML pages in a template, and will only be needed if a new template has to make changes so radical that altering the CSS alone won't be enough.

The optional "(extras).txt" file - note brackets - is a text file which contains a list of named extras to throw in. For instance:

```
easter.html
egg.png
```

These named files need to be present in the template folder. Files with the extension ".html" go through the placeholder expansion process just like the index and source pages; all other files are copied verbatim.

HTML templates like "index.html" and "source.html" are fully valid HTML pages in their own right, though they have placeholder text where Inform will substitute the project's bibliographic data (see below). The "<head>" element should include a reference to "style.css", which of course will mean the CSS file given in the template (or the one from "Standard" if no CSS file is given) - for instance,

```
<link rel="stylesheet" href="style.css" type="text/css" media="all" />
```

When it turns the template into the final web page, what Inform does is to replace certain capitalised words in square brackets with the appropriate text:

```
[TITLE] becomes the story title
[AUTHOR] becomes the author's name
[YEAR] becomes the story creation year
[BLURB] becomes the story description
[RELEASE] becomes the release number
[COVER] becomes an image of the cover art (the small 120x120 cover image)
[DOWNLOAD] becomes the download link
[AUXILIARY] becomes the list of feelie-like files, if any
[IFID] becomes the IFID
[STORYFILE] becomes the "leafname" of the story file, e.g., "Bronze.gblorb"
[TEMPLATE] becomes the name of the template used to make the page
[SMALLCOVER] becomes the filename of the small cover image
[BIGCOVER] becomes the filename of the large cover image
[TIMESTAMP] and [DATESTAMP] become the time and date of releasing
```

Everything else is left alone. In source pages, five further placeholders are available:

```
[SOURCE] becomes the portion of the source text on this page
[SOURCELINKS] becomes the navigational links
```

```
[SOURCENOTES] becomes the footnote matter at the bottom of the source
[PAGENUMBER] and [PAGEEXTENT] are such that the text "page [PAGENUMBER] of
[PAGEEXTENT]" produces, e.g., "page 2 of 7"
```

Both [SOURCE] and [SOURCENOTES] must exist on the page, and [SOURCENOTES] must appear after [SOURCE] does in the file. (Of course the CSS in "style.css" might move the copy around on screen, but that's another matter.)


Start of Chapter 25: Releasing


Back to §25.13. Website templates


Onward to §25.15. Republishing existing works of IF

---

§25.15. Republishing existing works of IF

Some long-time users of Inform will have projects which were originally made using the very different Inform 6 language. Story files produced with Inform 6 do not have any of the extra touches in this chapter: in particular, they have no cover art and no bibliographic data, which makes them rather plain and anonymous to newer Treaty of Babel-equipped programs like Zoom, Spatterlight or Windows Frotz.

To help with this, today's Inform can republish an Inform 6 project by combining an Inform 7 source text which contains only release instructions and bibliographic data with an already-compiled Inform 6 story file. We do this by writing a short source text which contains:

```
Release along with an existing story file.
```

We then place the story file in the ".materials" folder. By default this will be called "Story.z8", but we can alternatively name it:

```
Release along with an existing story file called "Zork1_sg.z5".
```

The Settings panel must be switched to the Z-machine for this to work, since only Z-machine story files are supported this way, not Glulx. And we can now use the Release button to obtain the goods.

An existing story file can take advantage of all of the extra features - cover art, titling, website, feelies and so forth - earlier in this chapter, but not those - walkthrough, source text, map - which are still to come.

The following is a typical example of a source text used solely to bind up an old Inform 6-compiled story file:

```
"Curses" by Graham Nelson

The story genre is "Fantasy".

The story headline is "An Interactive Diversion".
```

```
The story creation year is 1993.

The release number is 16.

The story description is "It's become a matter of pride now not to give up. That tourist
map of Paris must be up here somewhere in all this clutter, even if it has been five years
since your last trip. And it's your own fault. It looks as if your great-grandfather was the
last person to tidy up these lofts..."

Release along with cover art and an existing story file.
```


Start of Chapter 25: Releasing


Back to §25.14. Advanced website templates


Onward to §25.16. Walkthrough solutions

---

§25.16. Walkthrough solutions

Since the earliest days of IF, players have distributed solutions to well-known stories, to help out other players at their wits' ends. The commonest format for these is a list of commands to type, sometimes with notes in the margin, and such a solution is called a "walkthrough", since it walks a player through the story.

Few authors publish solutions of their own works, but many supply their testers with solutions, especially towards the end of testing, or submit a solution as part of a competition entry. To help with this, Inform can generate such a walkthrough solution automatically:

```
Release along with a solution.
```

Inform will then place a file called "solution.txt" inside the "Release" folder. The solution might look like so (although probably much longer):

```
Solution to "Memoirs of India" by Graham Nelson

Choice:
INVENTORY -> go to branch (1)
EAST -> go to branch (2)

Branch (1)
DROP MANUSCRIPT
SOUTH

Branch (2)
INVENTORY ... Always a good idea
GIVE MANUSCRIPT TO THOMAS
```

Inform does not, of course, know how to solve IF all by itself, but derives the solution from the project's Skein. Since the Skein will have been used in testing the story, it will very likely contain a perfect solution - or several different ones, taking the story to a variety of possible endings. In the example above, there are two possible winning lines, which diverge right

from the first move. (There can be further divergences: for instance, if branch (2) splits, it will split into branches called (2.1), (2.2), (2.3) and so on.)

But the Skein will also contain plenty of unwanted diversions, so Inform does not rewrite the entire Skein as a solution. Instead, it looks for knots in the Skein which have been annotated. Any knot whose annotation begins "***" (three asterisks) is considered to be a final, winning move. (It is probably a good idea to lock such a knot once it has been annotated thus, too.) We can mark any number of knots "***" since, after all, we can declare any number of lines of play as possible solutions. Inform then constructs the solution out of all lines of play in the Skein which lead to "***" endings, and ignores other threads.

Annotations other than "***" in the Skein are turned automatically into comments in the solution text. For instance, the knot for the INVENTORY command in the second branch above was annotated "Always a good idea", and this was transcribed into the solution. (If an ending knot is annotated with, say, "*** Happy ending!" then the "***" marks it as an ending, and "Happy ending!" is added as an annotation to that ending.)

By default, the solution text is not linked from our webpage, on the assumption that we may want to generate a walkthrough but not immediately advertise it to players. If we wish to change this, we may write instead

```
Release along with a public solution.
```

The terms public and private may also be applied to other elements we are having Inform generate to include on our webpage: see also the notes on private source text, below.

Start of Chapter 25: Releasing
Back to §25.15. Republishing existing works of IF
Onward to §25.17. Releasing the source text

---

§25.17. Releasing the source text

Most authors will not want to publish the source text alongside the work itself, because this gives away all of its secrets. Inform provides the option mainly for the sake of the examples published on its own website, where making the source available is the whole point. But anyone is welcome to use the option, of course:

```
Release along with the source text.
```

If Inform is not also generating a website, this produces a plain text file called "source.txt" in the "Release" folder, and there is nothing more to be said.

However, if a website is also being released, the source is also converted to a suite of web pages which are linked to and from the home page. (Each heading with substantive content is placed on its own web page, with the opening page containing a contents list.)

Comments in the source are rendered in grey. As a special feature, any comment which begins with an asterisk is considered a footnote and is printed below the source text, with a link. Thus comments thus:

```
Hercules is a demigod.[* We're using Greek spellings so he ought to be Heracles, but players are so much more familiar with Hercules.]
```

will be printed more like so:

```
Hercules is a demigod.[1]
...
Note
[1]. We're using Greek spellings so he ought to be Heracles, but players are so much more familiar with Hercules.
```

Footnotes are automatically numbered from 1 on each source page.

By default, the source text is linked from our generated webpage, if we are releasing with a webpage. If we wish to change this, we may write instead

```
Release along with the private source text.
```

This will create a text file containing the source for our story, and place this file in our release folder, but not create a link so that the player can find it.

Finally, we can:

```
Release along with the library card.
```

which releases a stand-alone XML file in 'iFiction' format for the bibliographic data on the story file; this is the same data embedded in the blorb file itself, but having an external copy makes it easier to see what Inform has done, and some external programs can read iFiction data like this.


Start of Chapter 25: Releasing


Back to §25.16. Walkthrough solutions


Onward to §25.18. Improving the index map

---

§25.18. Improving the index map

As we have seen, "Release along with..." allows us to package up a work of IF with all manner of extra materials. But what are these to be? One popular option is to produce a map - sometimes partial, sometimes obfuscated - and supply that with the story: besides, there are some IF competitions where the rules require that the referee is supplied with a map even if the players are not, and failing that, it is sometimes nice to be able to print out a map of a work in progress.

The World map in the Index tab is heavily stylised and cartoonish, intended to be clicked on or moused over, and viewed in a browser: although it is, in fact, possible to print it, the results are not very good. Fortunately, the same underlying map mechanism can be used to output something more useful and very much more customisable, as we shall see.

The map-maker is one of the most complex parts of Inform, even though it actually contributes nothing to the final story file: the problem of how to draw up a "correct" map from the source text is by no means easy to solve. Inform tries, but it often gets things wrong. Its general practice is to place rooms on a square grid (actually a cubic lattice, as it works in three dimensions), but not all conceptual maps fit well onto this, and Inform often annoyingly puts a particular room in the "wrong" place. For instance, suppose Inform puts "Didcot" east of "Abingdon" and this makes the geometry look different to what we had in mind. We can correct with:

```
Index map with Didcot mapped southeast of Abingdon.
```

Note that this says nothing about exits from any room to any other room, and changes the final work of IF not at all: it simply helps Inform to draw the map index. (Instructions like this one are treated as being almost certainly true, but Inform does not quite always obey: it will never allow two rooms to be superimposed at the same grid position, no matter what we have asked in "Index map with..." instructions.) The same trick is useful if we have a situation like so:

```
Inside of Sweeping Sands is Beach Hut Interior.
```

"Beach Hut Interior" is a single room which does not connect to the rest of the map by any of the ten spatial directions, so Inform does not place it on the main map but instead moves it off out of the way in a map of its own. Given that it's just a single room, however, we might prefer to put into a convenient otherwise empty grid position like so:

```
Index map with Beach Hut Interior mapped west of Sweeping Sands.
```

Finally, note that this trick also ensures that the two locations are mapped on the same level vertically, and can be useful in cases where room A is both north of and above room B: Inform will want A to be higher up than B, but we can insist otherwise.


Start of Chapter 25: Releasing


Back to §25.17. Releasing the source text


Onward to §25.19. Producing an EPS format map

---

§25.19. Producing an EPS format map

The "Index map with..." instruction is a much more varied thing than hinted at in the previous section, and its general form is

```
Index map with [instruction] and [instruction] and ... and [instruction].
```

where the instructions can be of four different forms, as follows:

```
[room A] mapped [direction] of [room B]
EPS file
rubric [text] ... and some optional details ...
[setting] of [whatever] set to [value]
```

We have already seen the first of these instructions. The second is short and has a fixed wording:

```
EPS file
```

so can be invoked by typing "Index map with EPS file.", for instance. EPS stands for Encapsulated PostScript, which is a standard file format for line art. EPS files can be edited with sophisticated graphics programs such as Adobe Illustrator, and can be used as illustrations in many word-processors and page layout programs. They can also be converted to PDF by Mac OS X Preview, or used in Linux or Windows with the open-source Evince viewer. We need a line-art format because the map produced will never be exactly what we want: we are probably going to end up hacking it to change the fonts, add some drawings, tidy up the spacing and so on. A really large map will end up using quite a large "canvas", in EPS terms; it may be necessary to shrink it down in order to get it onto an A4 page, or to adjust whatever editing software is used to "custom paper size".

When the map-maker has been given the "EPS file" instruction, it writes an attempt to draw the current project's map in EPS format as a file in the project's ".materials" folder, with the filename "Inform Map.eps".

Note that Inform will over-write any existing file of this name: but that is intentional, because one usually ends up tweaking and rebuilding the project over and over to get the map just so, and it would be tiresome for Inform to produce endless copies "Inform Map 19.eps", etc.

(The reason the EPS file is not placed in the Release subfolder is that it is not going to releasable to the public as it stands: for one thing it will be too raw, and for another, EPS is not a format everyone can read. It is provided as raw materials.)


Start of Chapter 25: Releasing


Back to §25.18. Improving the index map


Onward to §25.20. Settings in the map-maker

---

§25.20. Settings in the map-maker

The map-maker has altogether 35 named settings, and tweaking these can affect the result in ways which vary from the subtle to the grotesque. An important point is that the map-maker deals separately with the three levels in its working: the big picture of the whole map; each of the vertical slices which contain sub-maps; and finally all of the individual rooms. For instance, we might have 67 rooms, arranged on 3 vertical levels, all shown on one big map: Inform will try to show these stacked above each other, with the highest level at the top of the map, then the middle level, then the bottom level.

Moreover, not only does the whole map have its 35 settings, but each level has its own independent collection of those 35 settings, and so does each individual room. So the actual number of variables in our example is 1+3+67 = 71 times 35, which is a lot. The convention is that setting the value of S (some setting, let's say) for something affects not only that thing, but also everything inside it, unless they have their own individual settings for S.

For example: one of the settings is called "room-size", and is the size of the little square boxes representing a room, measured in points. (One point is 1/72 of an inch, so 72 points equals 1 inch: it's a traditional printer's measure.) Suppose we write:

```
Index map with room-size set to 36
  and room-size of level 2 set to 28
  and room-size of the Hall of Kings set to 52.
```

The first instruction sets the value of "room-size" for the whole map (note the lack of an "of..."); the second for level 2 of the map, and the last for a single room only. The result is that the Hall of Kings is drawn as 52x52 point box, all rooms on level 2 are 28x28 (except the Hall of Kings, if it's on level 2), and all others are 36x36, half an inch square.

The setting instruction also allows three other useful forms. A setting "of the first room" applies to the room in which the story begins: we might for instance write

```
Index map with room-outline-thickness of the first room set to 2.
```

which gives this special room a bolder edge to it, since the default value is 1.

We can also apply settings not just to single rooms but to all rooms of a given kind:

```
A rivery room is a kind of room. Index map with room-colour of rivery rooms set to
"Navy" and room-name-colour of rivery rooms set to "White".
```

Lastly, we can apply settings to all rooms in a given region:

```
Northern Oxfordshire is a region. Hampton Poyle and Steeple Barton are in Northern
Oxfordshire. Index map with room-name-font of Northern Oxfordshire set to "Helvetica-
Oblique".
```

(Note that rooms and regions don't have their own individual sets of the 35 settings: what happens is just that instructions like the last one change more than one room at once.)

Start of Chapter 25: Releasing
Back to §25.19. Producing an EPS format map
Onward to §25.21. Table of map-maker settings

---

§25.21. Table of map-maker settings

Note that all map-maker settings have single word names, though many are hyphenated, and that "colour" is always given the English and Canadian spelling, not the American form "color".

font	font (named in double-quotes)
minimum-map-width	integer (measured in points: 72 = 1 inch)
title	text (in double-quotes)
title-size	integer (measured in points)
title-font	font (named in double-quotes)
title-colour	colour (named in double-quotes)
map-outline	on/off
border-size	integer (measured in points)
vertical-spacing	integer (measured in points)
monochrome	on/off
annotation-size	integer (measured in points)
annotation-length	integer (length to abbreviate down to)
annotation-font	font (named in double-quotes)
subtitle	text (in double-quotes)
subtitle-size	integer (measured in points)
subtitle-font	font (named in double-quotes)
subtitle-colour	colour (named in double-quotes)
grid-size	integer (measured in points)
route-stiffness	integer (Bezier spline curve scale factor)
route-thickness	integer (measured in points)
route-colour	colour (named in double-quotes)
room-offset	offset (in percentages of grid-size)
room-size	integer (measured in points)
room-colour	colour (named in double-quotes)
room-name	text (in double-quotes)
room-name-size	integer (measured in points)
room-name-font	font (named in double-quotes)
room-name-colour	colour (named in double-quotes)
room-name-length	integer (length to abbreviate down to)
room-name-offset	offset (in percentages of grid-size)
room-outline	on/off
room-outline-colour	colour (named in double-quotes)
room-outline-thickness	integer (measured in points)
room-shape	shape (named in double-quotes)

Start of Chapter 25: Releasing
Back to §25.20. Settings in the map-maker
Onward to §25.22. Kinds of value accepted by the map-maker

---

§25.22. Kinds of value accepted by the map-maker

Integer values are typed in the usual way: 3, -72, etc.

Text is in double-quotes: "Map of Lower Delta", etc.

Font names are in double-quotes: "Helvetica", etc. Note that Inform makes no effort to look for such fonts: if we give the name of a font we haven't got, the result will probably be that the map's EPS file will be displayed in various applications with Courier (which looks like bad typewriting) substituted. All fonts are by default equal to the global "font" setting (by default equal to "Helvetica"), so changing "font" for the whole map affects everything not explicitly specified as having a different font.

Shape names are in double-quotes with lower case. At present, the only legal shapes are "circle", "square" and "rectangle".

On/off values are written just thus: on, off. No quotation marks.

Offset values are actually pairs, and are written as two numbers (possibly negative numbers) joined by an ampersand, as in the example: "Index map with room-offset of Botley set to 10&-30." Note lack of spaces around the ampersand. This means that Botley's room is displaced from its correct grid position on the EPS map by 10% of the grid size eastwards, and 30% southwards. (The grid size is the distance between one grid position and the next: displacing Botley by -200&0 would move it two whole grid positions westwards.)

The **route-stiffness** setting is used when drawing routes between two rooms. These are drawn as Bezier curves, a standard way to make a smooth curve not only travel from A to B but also from pointing in a given direction at A to ending up pointing in a given direction at B. Thus a Bezier curve may turn a route round so that it leaves A pointing west, but curves around to enter B from the south. (Most routes involve leaving in one direction and arriving in the opposite direction, of course, and in those cases a Bezier curve is just a straight line.)

The stiffness factor for a given room measures how much the curves are allowed to warp around in order to force them to arrive at that room from exactly the right compass bearing. The default is 100. Raising to, say, 250 can force curved paths into freakish zig-zags: whereas lowering to 1, the minimum, may make the route arrive at completely the wrong bearing. (Formally speaking: at each end of the route, a "control point" for the Bezier curve is made by taking the centre point of the room, then adding the relevant compass bearing's vector, scaled up by the route-stiffness as a percentage of the grid size.)

Colour values are named and in double-quotes. These names are the same as those for the traditional set of web-page-safe colour chips, as follows:

"Alice Blue"
"Antique White"
"Aqua"
"Aquamarine"
"Azure"
"Beige"
"Bisque"
"Black"
"Blanched Almond"
"Blue"
"Blue Violet"
"Brown"
"Burly Wood"
"Cadet Blue"
"Chartreuse"
"Chocolate"
"Coral"

"Cornflower Blue"
"Cornsilk"
"Crimson"
"Cyan"
"Dark Blue"
"Dark Cyan"
"Dark Golden Rod"
"Dark Gray"
"Dark Green"
"Dark Khaki"
"Dark Magenta"
"Dark Olive Green"
"Dark Orange"
"Dark Orchid"
"Dark Red"
"Dark Salmon"
"Dark Sea Green"
"Dark Slate Blue"
"Dark Slate Gray"
"Dark Turquoise"
"Dark Violet"
"Deep Pink"
"Deep Sky Blue"
"Dim Gray"
"Dodger Blue"
"Feldspar"
"Fire Brick"
"Floral White"
"Forest Green"
"Fuchsia"
"Gainsboro"
"Ghost White"
"Gold"
"Golden Rod"
"Gray"
"Green"
"Green Yellow"
"Honey Dew"
"Hot Pink"
"Indian Red"
"Indigo"
"Ivory"
"Khaki"
"Lavender"
"Lavender Blush"
"Lawn Green"
"Lemon Chiffon"
"Light Blue"
"Light Coral"
"Light Cyan"
"Light Golden Rod Yellow"
"Light Grey"
"Light Green"
"Light Pink"
"Light Salmon"
"Light Sea Green"
"Light Sky Blue"
"Light Slate Blue"

"Light Slate Gray"
"Light Steel Blue"
"Light Yellow"
"Lime"
"Lime Green"
"Linen"
"Magenta"
"Maroon"
"Medium Aquamarine"
"Medium Blue"
"Medium Orchid"
"Medium Purple"
"Medium Sea Green"
"Medium Slate Blue"
"Medium Spring Green"
"Medium Turquoise"
"Medium Violet Red"
"Midnight Blue"
"Mint Cream"
"Misty Rose"
"Moccasin"
"Navajo White"
"Navy"
"Old Lace"
"Olive"
"Olive Drab"
"Orange"
"Orange Red"
"Orchid"
"Pale Golden Rod"
"Pale Green"
"Pale Turquoise"
"Pale Violet Red"
"Papaya Whip"
"Peach Puff"
"Peru"
"Pink"
"Plum"
"Powder Blue"
"Purple"
"Red"
"Rosy Brown"
"Royal Blue"
"Saddle Brown"
"Salmon"
"Sandy Brown"
"Sea Green"
"Sea Shell"
"Sienna"
"Silver"
"Sky Blue"
"Slate Blue"
"Slate Gray"
"Snow"
"Spring Green"
"Steel Blue"
"Tan"
"Teal"

"Thistle"
"Tomato"
"Turquoise"
"Violet"
"Violet Red"
"Wheat"
"White"
"White Smoke"
"Yellow"
"Yellow Green"

Start of Chapter 25: Releasing
Back to §25.21. Table of map-maker settings
Onward to §25.23. Titling and abbreviation

---

§25.23. Titling and abbreviation

The main title of the map is the value of "title" for the whole map, so for instance we might write:

```
Index map with title set to "Oxford and its Environs".
```

The subtitle settings apply to the subtitles used for each of the levels, so for instance

```
Index map with subtitle of level -1 set to "Tunnels and Sewers".
```

Names of individual rooms can be controlled with:

```
Index map with name of Radcliffe Camera set to "Library".
```

(By default, the name of a room is its name in the main IF project, of course.) The smallest writing on the map is normally that used to label unorthodox or unclear exits (in particular, those going from one layer to another): this is what the "annotation" size, font and colour are used for.

For most ways to set up the map, it's a practical necessity to abbreviate names of rooms, or they will spill out all over each other. Inform does this using the "room-name-length" setting. (The "annotation-name-length" is analogous.) For instance, if this setting is 5, then Inform will reduce the text of a name to at most 5 characters. It does this by successively throwing out spaces, lower case vowels, then other lower case letters, punctuation marks and finally upper case letters, always starting at the back of the name and working inwards: the process stops as soon as the name is short enough. For instance, "Reading" is abbreviated to "Redng", "Shangri-La" to "Shn-La" and "Cloud-Cuckoo-Land" to "C-C-L". The result can be a little comical, but is surprisingly unambiguous in practice. Abbreviation can effectively be abolished by raising the "room-name-length" to 128 (the highest permitted level), and note that the setting can be changed for individual rooms, so it is possible to have some room names abbreviated and others not, or in different degrees.

Start of Chapter 25: Releasing

Back to §25.22. Kinds of value accepted by the map-maker

Onward to §25.24. Rubrics

Example 446: **Baedeker** Creating a floorplan of the cathedral using the locations from previous examples.

Example 447: **Port Royal 5** Port Royal scenario given instructions for an EPS map.

Example 448: **Bay Leaves and Honey Wine** Creating a map of Greece using the locations from previous examples.

---

§25.24. Rubrics

Lastly, we can add our own arbitrary text to the map: perhaps to annotate points, perhaps just to add more heading matter (such as the author's name, or the date). Each individual line added - and only single lines can be added, not typeset paragraphs - is called a "rubric". (There can be up to 100 of these.) We can create a rubric like so:

```
Index map with rubric "Here Be Wyverns" size 16 font "Helvetica-Oblique" colour
"Thistle" at 150&0 from Cloud-Cuckoo-Land.
```

This gives rather more detailed information than is needed: "size 16" could have been omitted, giving us 12-point type by default, and similarly there is no need to specify a font unless it differs from the main "font" setting for the whole map; and the colour will be black if unspecified. The "at" position does need to be given, though. Note that it is relative to a given room on the map, and that the position specified is that of the centre-point of the text. (If we had written just "at 100&100", say, that would specify a position relative to the bottom left hand corner of the map.) So, for instance:

```
Index map with rubric "trapped door" size 8 at -60&-60 from Longwall.
```

would add a little 8-point-type safety tip for naive map-followers.

Inevitably, the settings in the map-maker will fail to get exactly the effect desired (though they will offer an excellent opportunity to waste entire days). But that is the whole point of producing output in EPS format: Inform aims not to produce final print-ready professional art, but to produce the raw material for making that final work of art. And if all that's required is a sketch-map, then Inform's output should be good enough quickly and without too much fuss.

Start of Chapter 25: Releasing

Back to §25.23. Titling and abbreviation

Onward to Chapter 26: Publishing: §26.1. Finding a readership

---

Examples from Chapter 25: Releasing

Start of this chapter
Chapter 26: Publishing
Indexes of the examples

446

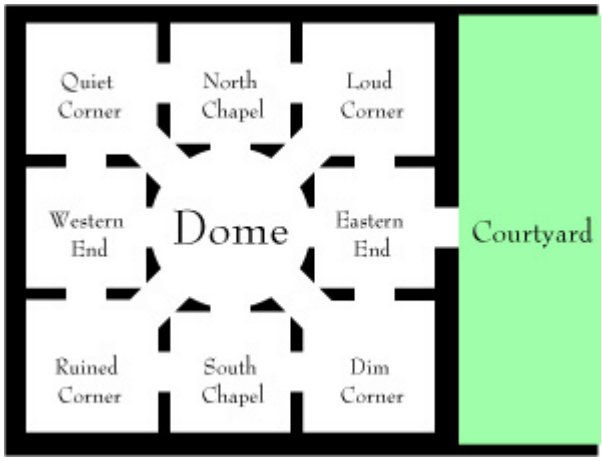
Example Baedeker RB

Creating a floorplan of the cathedral using the locations from previous examples.

If our map is largely or entirely set inside a single building, we might want to produce something that resembles a floorplan. It's possible to do this with a little tweaking, like so:

```
"Baedeker"

Dome is a room. North of Dome is North Chapel. South of the Dome is South
Chapel. West of the Dome is Western End. Quiet Corner is northwest of the
Dome, north of Western End, and west of North Chapel. Loud Corner is east of
North Chapel, northeast of Dome, and north of Eastern End. Eastern End is
north of Dim Corner and east of Dome. Dim Corner is southeast of Dome and
east of South Chapel. Ruined Corner is southwest of Dome, west of South
Chapel, and south of Western End.

The church door is east of Eastern End and west of the Courtyard. The church
door is a door.

Index map with
   room-shape set to "square" and
   room-size set to 60 and
   room-name-size set to 9 and
   room-name-length set to 13 and
   route-thickness set to 20 and
   room-outline set to off and
   map-outline set to off and
   route-colour set to "White" and
   room-colour set to "White" and
   room-shape of Dome set to "circle" and
   room-size of Dome set to 80 and
   EPS file.
```

Now we have a map made of white lines and boxes over a white background, which is not very exciting. If, however, we put a layer of black under this and slightly adjust the room shapes (using an image editor such as Adobe Illustrator), we can produce something that plausibly resembles a floorplan or museum map, like so:



447

Example Port Royal 5

RB

Port Royal scenario given instructions for an EPS map.

"1691"

Fort James is a room.

Thames Street End is south of Fort James.

Lime Street is south of Thames Street End. West of Thames Street End is north of Fisher's Row. The description of Fisher's Row is "A waterfront street that runs south towards Chocolata Hole, where the small craft are harboured. It also continues north around the tip of the peninsula from here, turning into the east-west Thames Street."

Water Lane is east of Thames Street End.

East of Water Lane is a room called Thames Street at the Wherry Bridge. Thames Street at the Wherry Bridge has the description "To the southwest is the fishmarket; directly across the street is the entrance to a private alley through a brick archway."

The Fishmarket is southwest of Thames Street at the Wherry Bridge.

The Private Alley is south of Thames Street at the Wherry Bridge.

Thames Street by the King's House is east of Thames Street at the Wherry Bridge.

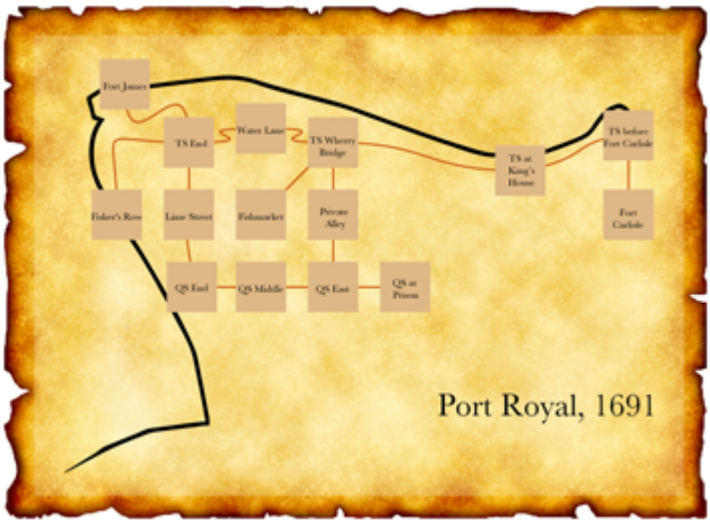
Thames Street before Fort Carlisle is east of Thames Street by the King's House.

South of Thames Street before Fort Carlisle is a room called Fort Carlisle. The description of Fort Carlisle is "Handsomely arrayed with cannons which you could fire at any moment -- though of course there are ships at dock which might be in the way."

```
Queen Street End is south of Lime Street.

Queen Street Middle is east of Queen Street End.

Queen Street East is east of Queen Street Middle and south of Private Alley.

Queen Street at the Prison is east of Queen Street East.

Index map with an EPS file and
    Fisher's Row mapped southwest of Thames Street End,
    room-size set to 50 and room-name-size set to 10,
    room-name-length set to 15,
    route-thickness set to 2,
    room-outline set to off,
    map-outline set to off,
    route-colour set to "Chocolate",
    room-colour set to "Burly Wood",
    title set to "Port Royal, 1691",
    font set to "Baskerville",
    room-offset of Thames Street by the King's House set to 160&-40,
    room-offset of Thames Street before Fort Carlisle set to 210&10,
    room-offset of Fort Carlisle set to 210&0,
    room-offset of Fort James set to -90&-20,
    room-offset of Water Lane set to 0&20,
    room-offset of Queen Street End set to 5&0.
```

This sentence has become a long catalogue of specifications. To break it down: we disambiguate the placement of Thames Street End, which otherwise is hard to locate because the directions to and from the room are not symmetrical. Then we apply some general rules about size, font, and color. Finally, we add instructions about offsetting the room locations of a few specific rooms.

This last part is a bit finicky and will not be necessary in many cases, but our goal this time is to create a map diagram that can be superimposed on the real coastal outline of Port Royal at the time. With a bit of editing, the result looks like this:



448

Example Bay Leaves and Honey Wine

RB

Creating a map of Greece using the locations from previous examples.

The map-maker can be used in quite versatile ways, in short; though the default is a schematic line-and-box affair, that is hardly the only option. While the EPS created is not always the result of our dreams, Inform usually can be made to do most of the hard and boring part, leaving the author to do only a bit of aesthetic touchup.

In many previous examples, we have sent hapless deities wandering around a map of Greece; we might like to chart that for ourselves, in a semi-realistic fashion. So:

```
"Bay Leaves and Honey Wine"

Corinth is a room. Athens is east of Corinth. Epidaurus is southeast of Corinth
and east of Mycenae. Mycenae is south of Corinth. Olympia is west of Mycenae.
Argos is south of Mycenae. Thebes is northwest of Athens. Pylos is south of
Olympia. Sparta is east of Pylos and south of Argos. Delphi is northwest of
Thebes.

Index map with an EPS file and
   room-size set to 8,
   map-outline set to off,
   room-name-offset set to 40&-40,
   room-outline set to off,
   room-colour set to "White",
   route-colour set to "White",
   room-name-colour set to "White",
   room-name-length set to 25,
   room-shape set to "circle".
```

This produces a line-and-dot map, where the names of rooms do not appear inside the city-circles, but rather (thanks to "room-name-offset") off to one side. We specify a long room-name-length because we want all the names of the cities spelled out in full; and we make all the elements white because we intend to place them over a black background layer.

We can then superimpose this on a vector map of Greece and tweak the exact positions of cities a little by hand (in Adobe Illustrator, as it happens, but other programs would also allow this level of editing). The result:


DELPHI
THEBES
ATHENS
CORINTH
OLYMPIA
MYCENAE
EPIDAURUS
ARGOS
PYLOS
SPARTA

Chapter 26: Publishing


Contents of *Writing with Inform*


Chapter 25: Releasing


Chapter 27: Extensions


Indexes of the examples

§26.1. Finding a readership

So the new work of IF is written, and tested, and has all its bibliographic data and a fancy cover illustration lined up. What next?

There is a thriving community of readers and writers of interactive fiction, and it is sometimes supported by grants from arts foundations and other cultural bodies: there's increasing attention from the academic world, and a general consensus has gradually grown that interactive fiction is a "valid" artistic medium for expression. Like poetry, it is something that a few people like a lot, and which most people can see the point of, even if they don't read it themselves. Over the last thirty years, a few authors have established durable reputations: they give occasional newspaper interviews, and have a very low-key kind of fame. There are competitions, and annual awards ceremonies. Newcomers are always welcome.

With some important exceptions, most works of interactive fiction have never been "published" in the sense of being issued for sale by a for-profit company. For the most part, IF has not been commercially valuable since about 1987. Successful authors of IF generally take the view that while they could, perhaps, make a very modest amount of money from sales, it would be a nuisance to collect and make no meaningful difference to their incomes; it would cut the number of readers, whereas one wants the satisfaction of being read; and besides, the whole culture of IF has always been characterised by giving and sharing. (Inform itself is free.)

Inform has nevertheless been used to produce commercial works (generally add-ons or bonuses to other games), and users are very welcome to sell works created by Inform with no royalty or requirement for rights clearance. It's also widely used in education, and as a prototyping tool for other kinds of story.

Start of Chapter 26: Publishing

Back to Chapter 25: Releasing: §25.24. Rubrics

Onward to §26.2. How a novel is published

---

§26.2. How a novel is published

Suppose that a (traditional) novel has been delivered to a publisher: the author has handed it over as a pile of twenty chapters of prose, and feels that it is finished. In fact there is much still to do:

(a) Editing. An editor works through the book, looking for problems in the plot, uneven passages, difficulties of tone and a hundred other nuanced points. The author generally then revises the book and submits again.

(b) Copy-editing. A copy-editor fixes punctuation errors, awkwardly worded sentences and other low-level problems.

(c) Bibliographic data is added.

(d) Printing. The text is given a clean, readable rendition, and no longer looks like a home-made typescript.

(e) Cover art is added. Even unillustrated novels have pictorial covers, and these images are often used to set the tone for the book - they set the reader's frame of mind, so something more is happening than mere marketing.

(f) A back cover blurb is added. This will also find its way into catalogues, onto book trade databases, appear on Amazon.com and so forth. Both a description and a lure, it gives a flavour of the work without actually being any part of it.

(g) Binding. Not only are the inside pages printed, but maps, plates of illustrations, free CDs, fold-out charts, etc., may be tipped in to the binding.

(h) Legal deposit. Copies are lodged with libraries of record, such as the British Library or the Library of Congress, to ensure that the work cannot be lost from cultural history. (In most countries, this is a legal obligation for publishers.)

(i) Shipping. Copies are sent out to bookshops.

(j) Publicity. The author and publisher combine to put out the word, circulate leaflets, put up posters and so forth.

(k) Reviews and awards. Reviews are published, usually stirring up interest in the book. These having been stellar, a few months later the author bashfully accepts a Pulitzer Prize, the Booker or some similar token of cultural esteem.


Start of Chapter 26: Publishing


Back to §26.1. Finding a readership


Onward to §26.3. How interactive fiction is published

---

§26.3. How interactive fiction is published

If we take the eleven novel-publishing stages of the previous section in order, we find that pretty well the same business goes on for works of IF.

(a) Editing. Working with a small number of trusted play-testers, and taking their responses seriously even when inconvenient, will almost always produce an immeasurably better work: not just better functionally, but better artistically, and more enjoyable. Play-testers can usually be recruited by placing an ad on **www.intfiction.org/forum**.

(b) Copy-editing. Play-testers will also pick up small stuff - spelling mistakes, and punctuation errors - but note that Inform for OS X will spell-check our source text on request.

(c) Bibliographic data is added.

(d) Printing. Clicking Inform's Release button is the equivalent the-die-is-cast moment.

(e) Cover art is added. As we saw in the previous chapter, Inform can add a cover image as part of the Release process, though it will not itself draw and design that image - like a printer, it expects to be supplied with the original.

(f) A back cover blurb is added. Inform does indeed allow us to compose such a piece of text and include it with the work's bibliographic data.

(g) Binding. The story file, which is akin to the inside pages of a book, is combined with its cover art, bibliographic data, and also with other non-textual materials provided by the author (booklets, sound samples, images, etc.). Inform does much of this automatically, producing a composite object called a "blorb".

(h) Legal deposit. The work is uploaded to the IF Archive (**www.if-archive.org**), whose librarians shelve it in the appropriate section. More on this later.

(i) Shipping. A work of IF is electronic rather than physical, so nothing is actually moved, but many authors like to put their works on their own websites as well as placing them in the Archive.

(j) Publicity. Authors often announce a new work on IFDB (**ifdb.tads.org**). Authors often also set up a personal web page about the work. Inform can generate such a web page automatically, as we saw in the chapter about releasing new works.

(k) Reviews and awards. The IF community has competitions and awards in abundance, and several websites gather reviews. It is usually safe to say that a well-written work will not go unnoticed if it is sensibly publicised.


Start of Chapter 26: Publishing


Back to §26.2. How a novel is published


Onward to §26.4. The IF Archive

---

§26.4. The IF Archive

Publishing an IF story consists of two steps. One is the technical task of making the story available to players - unless the plan is simply to email it to close friends, that means hosting

it somewhere on the Internet. The second is the promotional task of letting people know the work exists, and where to find it.

It is a community tradition that all serious work is uploaded to the **IF Archive**, which is IF's answer to a national library. This is a mirrored, stable collection of thousands of interactive fiction games and programming languages, manuals, fanzines, maps, walkthroughs, and other materials. As such, it's likely to stay around even if a personal website goes off-line; it's also the primary resource for people doing scholarship on interactive fiction (and there are a growing number of these).

The Archive is very much a library, for long-term archiving, rather than a book-store. The catalogue is sober and textual, and there are no visual shop-windows, or posters advertising new titles hot off the press. Newcomers sometimes need practice finding their way around. And the Archive hosts story files (and associated manuals, as appropriate) but not advertising for them - it does not provide web-hosting for authors to set up mini-sites.

Uploading a work to the IF Archive is not too difficult, and can be done in two ways. One way is to use the archive's web form at:

http://www.ifarchive.org/cgi-bin/upload.py

The other is to create a new page at the Interactive Fiction Database, at:

http://ifdb.tads.org/

It's then possible to upload the story file to the IF Archive from IFDB. This is easiest all round, since it allows both IFDB and IF Archive to be updated at once.

In either approach, an author chooses and uploads a file, and accompanies it with a name and email address (so that the archive maintainers can verify the legitimacy of the work). The "About this file" field is for a line or two explaining what the story is -- its full title and any critical information -- and is used in generating the archive index. This is normally much shorter than the "blurb" described earlier. There's also a field to suggest where in the archive the story should be stored, but this is optional and intended chiefly for people expert in how the archive is filed. The archive maintainers will file a new story file in the obvious directory for its format. For Inform works, that means other Z-Machine - "z-code" - or Glulx story files. The maintainers sometimes place the same story file in multiple places in the Archive, using links.

As with all large libraries, it takes the Archive a little while for new acquisitions to be processed. When this happens, one of the volunteer maintainers will email with the official URL from which anyone can now download the story file.

Committing a story to the Archive is meant to be permanent. While the maintainers will happily replace older versions of stories with new improved releases, they are less eager to remove stories entirely. If that doesn't seem appealing, or if we do not want our story to be treated as freeware with essentially unlimited distribution, the Archive may not be a good choice. But it is deeply valued by the IF community, and has saved many works which could otherwise easily have been lost forever. Many contributions important in the history of IF were made by people who are now not easy to trace, and whose websites are long gone. But their work lives on.

Start of Chapter 26: Publishing
Back to §26.3. How interactive fiction is published
Onward to §26.5. A Website of Its Own

---

§26.5. A Website of Its Own

While any good story file ought to go into the IF Archive, it's probably wise also to provide an easier-to-use home for the work, by putting up a web page and a copy of the story file on a private web host. That host should ideally be as stable as possible, so that the URL is likely to remain fixed for what might be a long period. Freeware stories have a long period of viability relative to commercial games, which means that players may still be hearing about and checking out a story years after its initial release. A stable address helps everyone with links, and makes it easier for Google to direct people.

Of course creating a web page involves a little design work, but tools are widely available which make this quite easy nowadays. And as we've seen, Inform can automatically generate web pages and whole small mini-sites to put all the information about a story file into a tidy format, even including the ability to play online.

Start of Chapter 26: Publishing
Back to §26.4. The IF Archive
Onward to §26.6. IFDB: The Interactive Fiction Database

---

§26.6. IFDB: The Interactive Fiction Database

Once the story file has a home online, and a URL (that is, a web address) at which it can be found, it needs to be registered with IFDB:

http://ifdb.tads.org/

the Interactive Fiction Database. Just as the IF Archive is a repository for stories themselves, IFDB is a database containing information about them - titles, authors, locations, solutions, reviews, recommendation lists and more.

The name IFDB echoes the Internet Movie Database (IMDB), but in some ways it is also like the iTunes Music Store. For one thing, it's a shop-window for what's new, with cover art to catch the eye. For another, some interpreters allow players to browse IFDB directly and launch new stories in a single click. This kind of integration is only likely to increase, so story files unregistered on IFDB are likely to be much less visible to players of the future. Promoting IF is all about pulling in impulse players -- people who are passingly interested, but might not try the story if there is any significant work involved is setting it up. This is what IFDB is all about.

IFDB is community-editable, like Wikipedia, though editors are required to create an account and log in first -- this is free, of course. A standard form is provided for creating a

new record (accessible by selecting the option to add a story listing). More or less the same information that appears on Inform's library card in the Contents index needs to be copied over: there's space for the author name, story title, genre, and so on. IFDB will also ask for an IFID, a code identifying the story uniquely. Inform generates one of these automatically for each project, and it, too, is on the Library Card. It can always be found by typing VERSION into the compiled story and looking at the line that says

Identification number: //[some letters and numbers]//

The part between the // marks is the IFID. If there's cover art, that can also be uploaded, and good cover art makes a big difference to shop-window-appeal.

The download link should give the most stable URL available. If you have not yet uploaded your story to the IF Archive, you may do so by selecting the "Upload it to the IF Archive" link instead of pressing the "Add a Link" button. The benefits of submitting your story to the IF Archive in this manner are two-fold. One, IFDB will fill in much of the information required by the IF Archive for you. Two, the link to your story will not appear until the IF Archive maintainers move it to its permanent home in the archive, at which point the download link will be automatically updated and presented on the story page.

If you choose to upload your story file to the IF Archive independent of IFDB, then once the story file is safely up at its permanent home on the IF Archive, that is an ideal address to quote here. Otherwise, the URL of the work's own website is best. (Note that the IFDB entry can always be edited later, if the URL moves.)

Commercial works which aren't available as free downloads can be registered on IFDB just the same, and this is almost certainly a good idea.


Start of Chapter 26: Publishing


Back to §26.5. A Website of Its Own


Onward to §26.7. Competitions

---

§26.7. Competitions

One very common way to get players for IF is to enter the story into an IF competition. The annual IF Competition is the most prestigious and has the widest field, but the Spring Thing, the One Room Game Competition and other events also catch people's attention. Entering a competition is a path of least effort for authors promoting their new work, because the competition organizer usually takes care of hosting and archiving submitted stories, promoting the competition as a whole, collecting votes, and encouraging players to post reviews. Different contests have different arrangements. The **ifwiki** usually posts a list of current and upcoming competitions, as well as lists of results for those recently past, on the front page:

http://www.ifwiki.org

Some competitions also have their own websites, at least at the relevant times of year.

All the same, there are many IF works that aren't cut out for competition release. Competitions tend to be best for short or medium-short works, because judges don't necessarily have time to play a lot of long stories at once, and sometimes this is a condition of entry.

It's also good for publicity to win one of the annual **XYZZY Awards**. All interactive fiction stories released in a given year are eligible, and authors do not need to do anything to enter. As with the Oscars, though, you can't plan to win: it happens or it doesn't.


Start of Chapter 26: Publishing


Back to §26.6. IFDB: The Interactive Fiction Database


Onward to §26.8. The Gaming Avant-Garde

---

§26.8. The Gaming Avant-Garde

The IF community is not the only potential audience for a work of interactive fiction. Some authors have successfully written and pitched IF to other groups -- audiences interested in a particular historical period, role-playing story universe, or web comic, for instance -- and those groups have to be reached through their own community forums and meeting places.

Even if we do write material mostly meant for the existing IF community, that doesn't mean the audience has to stop there. There are a number of independent gaming websites and blogs that feature IF reviews occasionally or regularly. These things change quickly, but at the time of writing, IF is featured with some regularity on:

> **JayIsGames** (www.jayisgames.com): a popular blog devoted to casual games, which occasionally profiles IF. JIG has also developed its own Flash interpreter for z-code games to allow the blog to host interactive fiction without requiring the casual audience to do any downloading. Experience suggests that JIG players are most interested in short IF -- serious or lighthearted, but humor goes over especially well -- and that they prefer very rigorously implemented work where a wide variety of player actions get customized replies. Small, polished gems do well here. JIG accepts story suggestions as well.
>
> **The Independent Gaming Source** (www.tigsource.com): occasionally publishes reviews or shorter announcements, and has even hosted an IF-writing competition. It may be worth drawing something to their attention.

This is hardly a complete list of gaming blogs with an interest in interactive fiction, just a collection of the most accessible ones, so it's worth doing a little research. One way is to pick a handful of works that coming closest to our own story in design and style, and search for reviews of those works. Where were they reviewed, and where were they well-received? Those venues might be good targets for our own production.

Finally, there is a constant round of competitions for independent games in general. Unlike IF community competitions, larger indie story comps usually don't require that a story be previously unreleased, only that it have no commercial funding. Some of these competitions offer substantial prizes in cash or computing equipment; some explicitly seek text-based games. Placing as a finalist in such a competition can mean having work displayed at a

gaming expo or hosted on a special wesbsite, and this will garner substantial outside press. The field of competitors will be large and serious, but that doesn't mean it's not worth giving it a try.

This may all be a bit overwhelming. All the publicity options can seem like more work than we want to handle while simultaneously finishing a substantial opus. In fact, we don't have to do it all at once. One sensible approach is to release to the IF community first, and then approach the external websites later with a second or third release that puts a final polish on the story in response to player feedback. That process is even more important for indie gaming competitions. Having a release that's been polished, together with an attractive website, cover art, and maybe even some quotes from IF reviewers, can make a story look much more like a legitimate contender among semi-pro works.


Start of Chapter 26: Publishing


Back to §26.7. Competitions


Onward to §26.9. The Digital Literature Community

---

§26.9. The Digital Literature Community

If our work is more a work of digital literature than a game, different venues are appropriate. PlayThisThing may still be interested; JayIsGames probably will not be. Instead, we might want to look at

GrandTextAuto (http://grandtextauto.soe.ucsc.edu): a group blog about new media and interactive story-telling, which sometimes posts announcements of interactive fiction, especially of the more literary kind.

The Electronic Literature Organization (http://eliterature.org/): an organization dedicated to preserving all kinds of new media literature. They list a number of events (such as readings open to the public, new media gallery exhibits, etc.) that an IF author could participate in, and they also have a directory of electronic literature, to which we can add our own works.

Digital Humanities Quarterly (http://www.digitalhumanities.org/dhq/): A scholarly publication, but one that looks at all aspects of digital literature, new media tie-ins for scholarship, etc. DHQ has run several articles about interactive fiction in the past, including Dennis Jerz's ground-breaking research on the writing of the first text adventure, Colossal Cave. DHQ might be open either to hosting a work or to publishing an article about it.

That still leaves out a category of IF -- namely, work written for a popular reading audience, work that isn't trying to be literary but also isn't primarily game-like. There are, at the moment, no ideal venues for promoting such work (that we're aware of), but it's certainly worth pursuing.

Start of Chapter 26: Publishing
Back to §26.8. The Gaming Avant-Garde
Onward to §26.10. A short concluding homily

---

§26.10. A short concluding homily

It's natural to want to make a huge splash with a story, but in the IF community, instant widespread adulation for any work is pretty uncommon.

For one thing, players tend to play when they get around to it... which may be weeks, months, or even years after the initial release. Reviews trickle rather than flooding in. Appreciation builds slowly. And sometimes works that placed unspectacularly in a competition, or seemed to be overlooked in the annual XYZZY Awards, gradually come to be regarded as classics because of some pioneering technique.

So it's wise (if difficult) not to judge a story's success entirely by its immediate feedback. Even after its debut, a story can often use a little care and attention if it's to reach all its potential fans -- whether that means building further releases, posting hint files or walkthroughs, developing new websites, or approaching outside reviewers.

Start of Chapter 26: Publishing
Back to §26.9. The Digital Literature Community
Onward to Chapter 27: Extensions: §27.1. The status of extensions

---

Chapter 27: Extensions

Contents of *Writing with Inform*
Chapter 26: Publishing
Indexes of the examples

§27.1. The status of extensions

The range of simulation offered by Inform's model world is intentionally limited to a core of basic essentials. We could argue at the margins, and the choice of what's in and what's out is partly traditional, but most people find the model reasonable as far as it goes.

Between 1993 and 2006, quite a range of "library extensions" for Inform 6 was written. Most of these extensions aimed to fill out the model by simulating other aspects of life, too: money, clothing, pourable liquids. None of these extensions was official and all of them were: it was a free-for-all, and in several cases different authors wrote rival extensions to model the same basic ideas. The development of Inform 7 was strongly influenced by this history and by the recognition that the base of rules and grammar inside a typical modern story are seldom written by a single author. They combine the standard Inform material with extensions by several third parties, together with anything specific to the story in question.

Inform 7 has a more organised idea of extensions, as we shall see. But anyone is free to write an extension on any terms or for any reason. Writers may wish to use the techniques in this chapter to develop private extensions of their own, used in several projects, or to share them with associates but not more widely.

But most writers of extensions do so to contribute to the Inform community, and for the satisfaction of solving a problem. Inform does not recognise anyone's approach to a particular need as "the official solution" - for instance, although the standard Inform

distribution includes a copy of Locksmith by Emily Short, that is not the "official" way to make automatically unlocking doors, and anyone is welcome to try a better one.

However, the Inform project does recognise some extensions as "public". Public extensions are the ones archived on the Inform website for the free use of all Inform writers. Those who wish to contribute an extension as a public one are obliged to follow a number of guidelines, which are mostly stylistic points intended to make the range of extensions easier to work with. Extension writers are asked to join in the spirit of these rules and help make the whole cooperative enterprise work harmoniously. Extensions which do play by these rules are also accepted into the Public Library, which makes them easy for all Inform users everywhere to find and obtain them.

Writers who wish to make their extensions public on the Inform website should also be clear that by doing so, they are donating their work to the community on the basis of the broadest form of Creative Commons license: that is, they retain copyright and the right to be identified as the author (and as we shall see they are automatically credited in any work of IF which uses their extension), but are giving unlimited permission to use, circulate and republish their extensions in any form, even as part of commercial works (should that arise). To publish a public extension is a public-spirited act, done for only the reward of a modest acknowledgement.

If the author of an extension has not made it public, or indicated in some other way that it is free to be used without the need for permission, then it would be both polite and prudent to check with the author before publishing something which incorporates his work.

 Start of Chapter 27: Extensions

 Back to Chapter 26: Publishing: §26.10. A short concluding homily

 Onward to §27.2. The Standard Rules

---

§27.2. The Standard Rules

When any source text is run through Inform, a secret first line is inserted, which reads:

```
Include the Standard Rules by Graham Nelson.
```

The "Standard Rules" file contains the definitions of the basic kinds, phrases, actions and grammar described in this documentation: for instance, it includes lines like

```
A container is a kind of thing.
```

...without which Inform would be lost. Although including the Standard Rules is compulsory, it is treated internally as if it were any other "extension".

What happens when an "Include" sentence is reached is that the sentence is replaced with the whole text of the file in question, often many paragraphs long.

If the file has already been included, then the sentence is simply ignored. This is so that we can have two extensions, each of which needs the other: if A says to include B, and B says to

include A, the result is that including one automatically includes the other, so we always get both which ever we ask for - not that there is a hideous infinite regress.

Start of Chapter 27: Extensions
Back to §27.1. The status of extensions
Onward to §27.3. Built-in, installed and project-specific extensions

---

§27.3. Built-in, installed and project-specific extensions

To recap: Inform builds projects from both the source text typed by the author and from Extensions; one of these, the Standard Rules, is always included; others are added as authors please. About 20 are "built-in" to Inform, meaning that they are stored inside the application and always available. Others must be "installed", and each Inform user will have a folder somewhere on his computer which contains these. Users typically obtain these from the Public Library feature in the application, but they can also download them directly from inform7.com and then use an Install Extension menu option in the application. Either way, the application then squirrels the file away, and it becomes available to any projects that that user may be working on.

It is also possible to have extensions available to just one project. These must be stored in the Extensions subfolder of the project's ".materials" folder, but otherwise are arranged the same as installed extensions - there's an outer folder for each author's name, and extensions are named with a ".i7x" extension within. For example:

```
Mourning Hypercritical.inform
Mourning Hypercritical.materials
    Extensions
        John Siracusa
            Fixing The Finder.i7x
```

When Inform needs to find an extension, it looks here first, then in the installed area, then in its built-in area. That means that we can make our own revised or hacked version of an extension, put it in the ".materials" area, and then have it take precedence over the installed or built-in one. We could even have our own private version of the Standard Rules here.

(This has a number of possible uses - for example, to provide a convenient test-bed when working on an experimental version of an extension.)

Start of Chapter 27: Extensions
Back to §27.2. The Standard Rules
Onward to §27.4. Authorship

---

§27.4. Authorship

Extensions are identified by author and by name, so that a given author can produce his or her own range of extensions, and need only ensure that these are named differently from each other. If John Smith and Mary Brown each want to write an extension called "Following People", there is no conflict.

The name of an extension, and of an author, should be written in Sentence Capitalisation: that is, upper case for the first letter in each word. (Inform uses this to minimise problems on machines where filenames are read with case sensitivity.) It is permitted for author names to include upper-case letters within words, as with the "G" in "Jesse McGrew". In general it is best to avoid accented or unusual letters in titles and author names, but the standard ISO Latin-1 characters should be allowed - for instance,

```
Étude Pour La Fênetre by Françoise Gauß begins here.
```

The author name must not start with "The", nor contain the words "by", "and" or "version", or contain punctuation, as in "John X. Doe"; the title similarly, except that "and" is permitted. Name and author's name must each be no more than 50 characters long, including any spaces between words.

Authors are asked to use real names rather than cryptic handles like "ifguy", and to use genteel, plausible pseudonyms like "Emily Short" rather than, say, "Drooling Zombie" or "Team Inform". Authors are also asked to use the same author's name for all their own extensions, and (it should go without saying) not to masquerade as anybody else.

Sometimes authorship is complicated. What if Mary Brown finds some Inform 6 code written by John Smith in the mid-90s, and puts an I7 gloss on it to make an I7 extension, but then Pierre Dupont translates it into French: who's the author of the result? The rule is that the person making the current, latest version is the author listed in the titling line, so we end up with

```
... by Pierre Dupont begins here.
```

But Mary and John deserve their credits too: see the next section for how to give them.


Start of Chapter 27: Extensions


Back to §27.3. Built-in, installed and project-specific extensions


Onward to §27.5. A simple example extension

---

§27.5. A simple example extension

Extensions are plain text files, and can be created with any text editor. (It is sometimes said that "there is no such thing as plain text", there being so many ways to represent exotic characters: so to be precise, an extension is a text file with the Unicode UTF-8 encoding, either with or without a BOM marker, using any of the possible forms of line-ending (Unix, Windows, Macintosh, or Unicode line divider). This is a detail which will only matter if the extension contains accented letters or other exotica.)

Extensions look very much like passages of Inform source, because except for a special introductory and concluding sentence, and one convention, that is all they are:

```
The Ducking Action by Graham Nelson begins here.

"An action for ducking one's head."

Ducking is an action applying to nothing. Report ducking: say "You duck!" Understand
"duck" as ducking.

The Ducking Action ends here.
```

Not a useful or interesting extension, but those few words add a whole new action and everything needed to make it work. It is Inform's ability to mix up rooms, things, kinds, grammar, phrases and rules, in more or less any order, which makes it possible for extensions to work.

The introductory sentence must be placed as the only content of line 1 of the file, which must not contain comments, and has to be written in exactly the correct form. Inform checks this very carefully when performing its census of installed extensions, on each translation of the text. (In case the extension's title is a plural, we are allowed to write "begin" and "end" instead of "begins" and "ends". For instance, the last line of the standard rules is "The Standard Rules end here.")

The "one convention" mentioned above is that if a double-quoted text is placed immediately after the beginning sentence (and with no intervening comments), then it is taken to be a short description of the extension's content called the "rubric". Hence the line:

```
"An action for ducking one's head."
```

Providing a rubric is helpful, because it enables Inform to give a meaningful listing even for an as-yet unused and unindexed extension, and because it helps the Inform website to produce better directories. Note the word "short": such text is likely to be truncated if it exceeds 500 characters.

A second double-quoted text can also, optionally, be added in yet a third special starting paragraph. This is to provide additional credits to people who have contributed to this or earlier versions. For instance:

```
The Ducking Action by Graham Nelson begins here.

"An action for ducking one's head."

"based on original Inform 6 code by Marc Canard"
```

Note the typical style here: it's a phrase rather than a sentence, and neither starts with an upper-case letter nor ends with a full stop. (The additional credit is then used in documentation and also in the VERSION text of any Inform story file using the extension.)

Start of Chapter 27: Extensions

Back to §27.4. Authorship

Onward to §27.6. Version numbering

Example 449: **Modern Conveniences** Exemplifying the kind of source we might use in writing extensions for kitchen and bathroom appliances.

---

§27.6. Version numbering

As we have seen, extensions are referred to by name and author, but they can also (optionally) be referred to by version. For instance:

```
Include version 2 of the Ducking Action by Graham Nelson.

Version 1/040426 of the Ducking Action by Graham Nelson begins here.
```

A version number must be a whole number 1, 2, ..., 999 (version 0 is not allowed, and nor are decimal points), optionally followed by a slash '/' and then a six-figure date, in the form YYMMDD: so 040426 means 26 April 2004.

A request to include version 2 of something implicitly means "version 2 or later", and similarly "version 2/040426" means version 2 with a date of 26 April 2004 or later, or else any version higher than 2. If Inform loads the extension but finds that its version is not good enough, an error is issued.

Where no version number is quoted, the rule is that an unspecified version predates any numbered version. This means that the line

```
Include the Ducking Action by Graham Nelson.
```

will be happy with any version of the extension at all, whether numbered or not; but

```
Include version 2 of the Ducking Action by Graham Nelson.
```

will only accept the extension if it has a version number attached (and with that number being 2 or higher).

During play of any story compiled by Inform 7, typing VERSION lists various serial numbers of the pieces of software used to make it. The list concludes with names, authors and version numbers of any extensions used. So every author whose work contributes to a story automatically gets a modest credit within it. The same list can be printed, at the discretion of the designer, using the textual substitution:

say "[the/-- list of extension credits]"

This text substitution expands to one or more lines of text crediting each of the extensions used by the current source text, along with their version numbers and authors. Extensions whose authors have chosen the "use authorial modesty" option are missed out. Example:

```
Standard Rules version 2/090402 by Graham Nelson
```

If we want our extension to go uncredited - perhaps if it is a low-level enabling sort of thing, for instance - we can place the following sentence inside the definition of the extension:

```
Use authorial modesty.
```

The same sentence placed in the body of a source text causes all extensions by the same author as the main source text to go uncredited. In other words, if Isaac Miggins writes a source text and includes, say, Unlikely Events by Isaac Miggins, then this extension will go uncredited in the VERSION command.

A complete list, undiluted by modesty, can always be obtained using:

say "[the/-- complete list of extension credits]"

This text substitution expands to one or more lines of text crediting each of the extensions used by the current source text, along with their version numbers and authors. Every extension is included, even those whose authors have opted for "use authorial modesty". Example:

```
Standard Rules version 2/090402 by Graham Nelson
Locksmith version 9 by Emily Short
```


Start of Chapter 27: Extensions

Back to §27.5. A simple example extension

Onward to §27.7. Extensions and story file formats

§27.7. Extensions and story file formats

Inform compiles to several different story file formats, and in each case uses only a small part of their abilities - especially when it comes to fancy tricks with the keyboard or screen. So people may well want to write extensions which provide access to some of these tricks (like "Basic Screen Effects", included in the standard Inform distribution, but more so). Unfortunately, these tricks are very likely to fail to compile - or fail to work - on some of the possible story file formats, so the resulting extension would probably go wrong (and mysteriously wrong) for users who have chosen a different format.

Inform therefore provides a way for extensions to declare the formats they are compatible with. All that is required is to add a proviso in brackets after the title is declared:

```
Version 2 of Basic Screen Effects (for Z-Machine version 8 only) by Emily Short begins
here.
```

Other examples might be "(for Glulx only)", or "(for Z-machine only)". If no such proviso is given, the extension is assumed to be compatible with every story file format.

Extensions are also able to include material which is only used on some story file formats and not others - in principle, this might allow the same facilities to be provided to the author whatever story file format is used, but to achieve these effects differently depending on the current Settings. The convention here is exactly like "not for release": if a heading or subheading in the source text contains a bracketed proviso, then the material under that heading (and under its dependent subheadings) will be ignored if the current story file format does not match. For example:

```
Section 2.3G (for Glulx only)

To reveal the explosion:
    [...the Glulx way...]

Section 2.3Z (for Z-machine only)

To reveal the explosion:
    [...the Z-machine way...]
```

would ensure that "reveal the explosion" works nicely whichever story file format is used.

Start of Chapter 27: Extensions

Back to §27.6. Version numbering

Onward to §27.8. Extensions can include other extensions

Example 450: **Tilt 3** Displaying the card suits from our deck of cards with red and black colored unicode symbols.

---

§27.8. Extensions can include other extensions

As was mentioned earlier, the same extension is sometimes requested several times. For instance, suppose the main source text asks to include version 2 of extension X, and also to include extension Y. Suppose further that Y contains a request to include version 4 of X. We now have two different requests for X. Inform works out the minimum version number needed to satisfy these requests (in this case 4) and gives an error if the extension it actually loads turns out to be earlier.

This is true even of the Standard Rules. Suppose that the Standard Rules had recently been republished in version 37, making the previously existing version 36 out of date, and that an extension is written which capitalizes on a new feature in v37. It will therefore not work if people try to use it with v36. All the extension needs to do is to say:

```
Include version 37 of the Standard Rules by Graham Nelson.
```

to guarantee that v37 or later will be used.

If an extension does include other extensions, it should do so in a paragraph immediately following the introductory sentence, so that anyone looking at the file can see these mutual dependencies at a glance.

 Start of Chapter 27: Extensions

 Back to §27.7. Extensions and story file formats

 Onward to §27.9. Extensions can interact with other extensions

---

§27.9. Extensions can interact with other extensions

When one extension is being used, it's probably only one among several. A really general-purpose extension might want to behave differently depending on which other extensions are also present. This can be achieved using headings which are "for use with" (or "without") other extensions. For instance:

```
Chapter 2a (for use with Locksmith by Emily Short)
```

specifies that everything under this heading (and its subheadings, if any) will be ignored unless the extension Locksmith by Emily Short is included. Conversely,

```
Chapter 2b (for use without Locksmith by Emily Short)
```

will be ignored unless it isn't included. This allows an extension to give two variations on the same material - one if Locksmith is present, the other if not.

Headings can also replace portions of extensions which have been included. For instance:

```
Section 6 - Hacked locking (in place of Section 1 - Regular locking in Locksmith by Emily Short)
```

places the source text under the new heading in the place of the old (which is thrown away). If there should be two or more headings of the same name in the given extension, the first is the one replaced; if two or more headings attempt to replace the same heading in the given extension, the final attempt in source text order is the one which succeeds; and finally, heading dependencies like the above are scanned in a top-down way. Thus, if we have:

```
Chapter 2a (for use with Locksmith by Emily Short)
...
Section 1 - Hacked marbles (in place of Section 4 in Marbles by Peter Wong)
...
```

and we don't include Locksmith, then the replacement of Section 4 of Marbles is not made, because Section 1 - Hacked marbles is subordinate to the Chapter 2a heading which we've told Inform to ignore.

If the name of the heading to replace contains the word "in", it's a good idea to use quotation marks for clarity:

Section - Hacked questions (in place of "Section 4 - Phrase used to ask questions in closed mode" in Questions by Michael Callaghan)

Start of Chapter 27: Extensions
Back to §27.8. Extensions can include other extensions
Onward to §27.10. Extensions in the Index

---

§27.10. Extensions in the Index

As soon as a project has successfully been translated, its Index is brought up to date: pages of the index record all the kinds and what they are for, all the phrases which can be used, and so on. Any kind or phrase created in an extension is automatically included. The extension's presence in the project is itself recorded - the Contents index for any project contains a brief list of all extensions used in that project, along with their authors and version numbers. For instance:

```
Standard Rules version 1/040731
Locksmith by Emily Short
Tupperware by Emily Short
```

The Kinds index aims to give the reader a brief note of what each kind is intended for. We can provide for this by writing a sentence like so:

```
The specification of player's holdall is "Represents a container which the player can carry around as a sort of rucksack, into which spare items are automatically stowed away."
```

There is no need to specify the properties which apply: that is all done automatically. "Specification" is a sort of pseudo-property used just for this: we can also give specifications to kinds of value and to actions, and these are similarly used in the Index pages.

Every extension has the right to its own set of headings and subheadings, independently of those used by the main source for the work or by any other extension which may be included. (So if the extension is divided into four sections and finishes on Section D, say, that doesn't mean that Section D will continue outside the extension as the main source of the story runs on.)

Extensions should, of course, be written so that they never produce Problem messages, so at first sight it appears that these headings will never be outwardly visible. In fact, though, Problems do occasionally turn up in extensions, usually when the user has made a mistake, or when two inconsistent extensions are used in the same project. But more importantly, the headings in an extension are used when indexing phrases (and also actions) to group similar phrases together. For instance, the Standard Rules contain the heading:

```
Section SR4/7 - Searching and sorting tables
```

The half-dozen phrases defined in this section of the Standard Rules are then indexed under the subheading "Searching and sorting tables": Inform looks for a hyphen in the heading and

then uses any text which follows the hyphen. (If there is no hyphen, the entire heading text is used.)

If an extension contains no headings, its phrases (or actions) are indexed simply as "Miscellaneous".

Finally, any phrase or variable defined immediately under a heading whose name ends in the word "unindexed" will be omitted from the Phrasebook or Contents index respectively. (That won't apply to definitions under subheadings of the heading.) This is intended so that technical apparatus used only inside the extensions can be concealed from the outside user's immediate view. Inform as it is presently constituted does not allow extensions to make fully private definitions, but this feature at least allows them to make unadvertised ones.


Start of Chapter 27: Extensions


Back to §27.9. Extensions can interact with other extensions


Onward to §27.11. Extension documentation

---

§27.11. Extension documentation

A basic mechanism for documenting extensions is built into Inform. For many extensions, this will probably do instead of a manual; for more complex ones, it should still prove a useful supplement to one.

As described in Chapter 2 above, whenever an extension is installed, its documentation is made available to the user. Such text should be written concisely, while giving examples wherever appropriate. Stylistically, it should ideally follow the model of the main Inform documentation: just as an extension expands the standard rules, so its documentation expands this manual. "We need..." is preferred to "You need...", and so on: we're all in this together.

In order to be recognised as documentation, this text should appear at the foot of the extension file, *after* the compulsory end sentence. The first paragraph must have exactly the following form, with a skipped line before and after:

```
---- DOCUMENTATION ----
```

For instance, the "Ducking Action" example might end:

```
...
The Ducking Action ends here.

---- DOCUMENTATION ----

This is a modest extension, with much to be modest about. It allows us to use a new
action for ducking, as in ducking the player's head (not as in ducking a witch). Ducking
will do nothing unless rules are added:

    Instead of ducking in the Shooting Gallery, say "Too late!"
```

```
...
```

We obtain indented code examples by beginning a line with a tab. A double indentation can be got with two tabs in a row, and so forth. (Beware: some text editors, or emailers, flatten tabs into a row of four or perhaps eight spaces each. Inform will not recognise such a line of spaces as a tab.)

Note that text in square brackets should be avoided in the documentation, because that's taken as being comment matter on the extension, and omitted.

Tables should be similarly indented, and should begin with the word "Table ...": the top line is taken to be the name of the table, and subsequent lines are tab-divided columns. Inform will automatically group this into a table, like so:

```
Table of Exemplariness

stellar object example
galaxy        "Andromeda Galaxy M31"
star          "Sirius"
planet        "Neptune"
moon          "Enceladus"
dwarf planet  "Ceres"
plutino       "38628 Huya"
cubewano      "Easterbunny"
```

(Footnote: Since the first appearance of this book, Easterbunny has been renamed Makemake, the creator god in the mythology of the people of Easter Island.)


Start of Chapter 27: Extensions


Back to §27.10. Extensions in the Index


Onward to §27.12. Examples and headings in extension documentation

---

§27.12. Examples and headings in extension documentation

Extensions with very large amounts of documentation can, if the author chooses, divide the material up using headings and/or subheadings. These must be written as paragraphs exactly like so:

```
Chapter: Avoiding Events

Section: Ducking examinations and tests
```

Inform will then typeset them to stand out, will number them automatically, and will add a table of contents at the top of the page. (For most extensions, the documentation will be short and sweet, and this would just be clutter: headings and subheadings are best used only where the text would otherwise be difficult to read.)

Any extension's documentation can contain Examples, just as the main Inform documentation does: these are automatically labelled A, B, C, ... rather than given numbers, to ensure that they do not clash with the numbering used in the built-in chapters. (The labels

may be helpful in writing an extension's documentation: we can write, for instance, a note such as "see Example C below".)

Examples must be given last in the documentation, and there can be up to 26 of them, though most extensions will need one example at the most, and some will have none at all. Each example must begin with a paragraph exactly like so:

> Example: ** We Must Perform a Quirkafleeg - Ducking to avoid arrows as one proceeds east across battlements.

Again, there must be a skipped line before and after. The row of asterisks must be *, **, *** or ****, just as in the main documentation, which we should follow on all points of style. The rest of the line contains the title, a hyphen, and then the description. The title should be given with Each Word except Prepositions and Similar Things Capitalized, while the description should look like a sentence, and end with a full stop.

The text of the example follows, of course, and continues until the end of the file, or the next "Example:" line, whichever comes first.

Each example should (normally) contain one single, complete, story, long enough to demonstrate the use of the extension and to have a little flavour to it, but not so long that the reader gets lost. It should have a title, which should match the name of the example (in the case above, "We Must Perform a Quirkafleeg"). It should conclude with a paragraph defining a test:

> Test me with "east / duck / east / jump / east / duck / east / rescue esmerelda".

The idea is that typing one single command, TEST ME, into the resulting story should show off what the extension does.

When an extension contains more than one example, they should be given in order of asterisk rating, that is, starting with the * examples, then the ** examples, and so on up.


Start of Chapter 27: Extensions


Back to §27.11. Extension documentation


Onward to §27.13. Implications

---

§27.13. Implications

Extensions often need to define new kinds or properties, which we want to make as helpful as possible for the user. In particular, we want them not to require additional work for the author just to obtain the effect which seems only natural.

For example, consider Inform's built-in "locked" property. If a door is locked, then it cannot be opened, which seems fair enough. But if the player tries to unlock the door, he might then find the following response:

> That doesn't seem to be something you can unlock.

Which does not seem right. In real life, almost all locked items have outwardly exposed locks which it is perfectly sensible to try to unlock, given a key. The problem is that our door has the "locked" property, but not the "lockable" one.

The Standard Rules solve this problem by including the following line:

```
Something locked is usually lockable.
```

This ensures that any door said by the author only to be "locked" will be "lockable" as well, and adds a small but worthwhile touch of realism.

Such a sentence is called an "implication", as it is in the form "Condition A implies Condition B". Note that the two conditions must consist of either/or properties with or without kinds attached. Thus:

```
A room in the Open Desert is usually lighted.
```

will not work because "a room in the Open Desert" is a more complicated grammatical construction than, say, "lighted" or "a lighted room": it contains a relative clause. Inform can only deal with simple implications.

Inform never overrides certainties with mere implications, and is cautious about allowing them to build overly long chains of argument. This is to prevent the following kind of difficulty:

```
An open door is usually closed. A closed door is usually open.
```

Implications work just the same for values which aren't objects, so:

```
Colour is a kind of value. The colours are red, green and blue.
A colour can be zesty or flat. A colour can be bright or dull.
Red and blue are bright. Blue is flat.

A bright colour is usually zesty.
```

results in red being zesty, but blue and green being flat; blue because the source text explicitly says so (which trumps the "usually"), and green because this isn't a bright colour, so the implication doesn't arise.

Implications have not been mentioned up to now since they are only really needed by extensions, but also because they can be tricky, with unforeseen consequences. We should handle them with care.


Start of Chapter 27: Extensions


Back to §27.12. Examples and headings in extension documentation


Onward to §27.14. Using Inform 6 within Inform 7

---

§27.14. Using Inform 6 within Inform 7

To return to what was said on the first page of this documentation: Inform 7 (or I7), the current version, is not at all like Inform 6 (or I6). Internally, however, I7 works by translating the source text into an I6 program, so that in a practical sense the current version of Inform actually contains its predecessor.

The final sections of this chapter show how I6 code can be mixed directly in with I7. The remaining pages will therefore make little or no sense to those who do not already use I6. But for those who do know I6 already, it would be all too easy to write highly hybridised code, constantly mixing I6 and I7. The authors of Inform hope that this will not happen: for almost all purposes, I7 is much more powerful than I6, and fails - when it has to fail - in a way more helpful to the user. Ideally, all I6 content would be confined to extensions (and this may be mandated in future releases of Inform), and even writers of extensions are asked to pare down their usage of I6 to the minimum necessary.

The methods for incorporating I6 code into I7 have been designed with this in mind, that is, to encourage people to use I6 in as self-contained a way as possible: in particular to isolate the relatively few functions which need to be written in I6, and to give them natural language expression.

Finally, anyone hacking with I7 for a while is likely to become curious about the Standard Rules file, and to look at the text on which the Inform world model is founded. The file is, of course, no secret, but it can be misleading. For one thing, it appears to have great freedom to set up the world model as it pleases, but in fact the I7 compiler may well crash unless certain things are done just so in the Standard Rules: they depend on each other.

Moreover, the Standard Rules use a number of syntaxes which are not documented in this chapter: these are constantly being altered, and it would not be safe to imitate them. Any I6-related syntax which is not documented in this chapter may be removed or changed in effect at any time without warning, for instance in an update of Inform to fix bugs.


Start of Chapter 27: Extensions


Back to §27.13. Implications


Onward to §27.15. Defining phrases in Inform 6

---

§27.15. Defining phrases in Inform 6

The phrases described in this documentation, such as "end the story", are all defined in the Standard Rules, and are for the most part defined not in terms of other I7 phrases but instead reduced to equivalents in I6. For instance:

```
To end the story (- deadflag=3; story_complete=false; -).
```

The notation "(-" and "-)" indicates that what comes in between is I6 code. The minus sign is supposed to be a mnemonic for the decrease from 7 to 6: later we shall use "(+" and "+)" to go back up the other way, from 6 to 7.

When a phrase is defined as containing only a single command, and that command is defined using I6 - as here - it is compiled in-line. This means that the phrase "end the story" will

always be translated as "deadflag=3; story_complete=false;", rather than being translated into a call to a suitable function whose only statement is "deadflag=3; story_complete=false;".

This is an easy case since the wording never varies. More typical examples would be:

```
To say (something - number): (- print {something}; -).
To sort (T - table name) in (TC - table column) order:
    (- TableSort({T}, {TC}, 1); -).
```

When the braced name of one of the variables in the phrase preamble appears, this is compiled to the corresponding I6 expression at the relevant position in the I6 code. So, for instance,

```
say the capacity of the basket
```

might be compiled to

```
print O17_basket.capacity;
```

because "{something}" is expanded to "capacity of the basket" (I7 code) and then translated to "O17_basket.capacity" (I6 code), which is then spliced into the original I6 definition "print {something};".

Braces "{" are of course significant in I6. A real brace can be obtained by making the character following it a space, and then I7 will not attempt to read it as a request for substitution.

It's also possible for the pair of characters "-)" to occur in I6 code, for example here:

```
for (i=3 : i>0 : i--)
```

and I7 will read the "-)" as terminating the I6; we can get around this with an extra space:

```
for (i=3 : i>0 : i-- )
```

Warning: Inform 6 uses a restricted character set, allowing use of most of the accented characters in ISO Latin-1 (those found in a set called ZSCII) but little beyond that. It's therefore hazardous to use any exotic Unicode characters in an inclusion.

 Start of Chapter 27: Extensions

 Back to §27.14. Using Inform 6 within Inform 7

 Onward to §27.16. Phrases to decide in Inform 6

 Example 451:  **Pink or Blue** Asking the player to select a gender to begin play.

---

§27.16. Phrases to decide in Inform 6

There are basically three forms of phrase in I7: phrases which do something, but produce no value or opinion as a result; phrases to decide whether or not something is true; and phrases to decide on a value. We have already seen examples of writing the first form in I6:

```
To say (something - number): (- print {something}; -).
```

Here the I6 form is required to be I6 routine code in void context, that is, it will normally be one or more statements each of which ends in a semicolon (unless there are braced code blocks present). In this case, we have just one I6 statement, ending in a semicolon.

An example of a phrase to decide whether something is true would be:

```
To decide whether in darkness: (- (location==thedark) -).
```

Here the I6 code providing the definition must be a valid I6 condition, and be in round brackets, but there is no semicolon.

Lastly, an example of a phrase to decide on a value:

```
To decide which number is the hours part of (t - time): (- ({t}/60) -).
```

Again, this is a value in I6 as well: no semicolon. It is probably safest to place the value in round brackets.


Start of Chapter 27: Extensions


Back to §27.15. Defining phrases in Inform 6


Onward to §27.17. Handling phrase options

---

§27.17. Handling phrase options

The Standard Rules use the Inform list-writer with the following definition, which shows how a much more complicated I6 routine can be given a natural-language expression.

```
To list the contents of (O - an object),
    with newlines,
    indented,
    giving inventory information,
    as a sentence,
    including contents,
    including all contents,
    tersely,
    giving brief inventory information,
    using the definite article,
    listing marked items only,
    prefacing with is/are,
    not listing concealed items,
    suppressing all articles
    and/or with extra indentation:
    (- I7WriteListFrom(child({O}), {phrase options}); -).
```

This can be used by, say:

```
list the contents of O, as a sentence, using the definite article
```

"{phrase options}" is a special substitution: it is a bitmap which assigns the given options one bit each, starting with the least significant bit for the first-mentioned option ("with newlines" above) and going up to the most significant bit for the last ("with extra indentation").


Start of Chapter 27: Extensions


Back to §27.16. Phrases to decide in Inform 6


Onward to §27.18. Making and testing use options

---

§27.18. Making and testing use options

Use options (see Chapter 2 above) manifest themselves in the I6 code generated by I7 as constants which are either defined, or not. For instance, the "use American dialect" option results in the constant DIALECT_US being defined, a constant which otherwise would not be. Some use options define the constant as a particular value, others simply define it (so that I6 gives this constant the value 0).

New use options can be created as in the following examples, which are found in the Standard Rules:

```
Use American dialect translates as (- Constant DIALECT_US; -).
Use full-length room descriptions translates as (- Constant I7_LOOKMODE = 2; -).
```

Most Inform users will not need to test whether a use option is currently set: after all, they will know whether or not their own story uses American dialect. But an extension does not know what use options apply in the story which is using it. An extension which needs to print a list, using its own formatting, might want to know whether "use serial comma" is set. Or it might want to speak differently in American dialect.

To test for American dialect, we should ideally not use I6 to look for the constant DIALECT_US using #ifdef: there is no guarantee that this constant will not be renamed at some point. Instead we can perform the test directly in I7:

```
if the American dialect option is active, ...
```

and similarly for all other named use options. The adjectives "active" and "inactive" have the obvious meanings for use options. This means it's possible to describe the current options like so:

```
say "We're currently using: [list of active use options].";
```

The result might be, say,

```
We're currently using: dynamic memory allocation option [8192], maximum text length
option [1024], maximum things understood at once option [100], American dialect option
and fast route-finding option.
```

This may be useful for testing purposes.

Use options can also allow the writer to raise certain maximum values. If we write an extension which needs some I6 array, say, and therefore has some limitation - for instance a footnotes presenter which can handle at most 100 footnotes before its array space runs out - it would obviously be cleaner to allow this maximum to be raised. We can set this up like so:

```
Use maximum presented footnotes of at least 100 translates as (- Constant
MAX_PRESENTED_FOOTNOTES = {N}; -).
```

With such a definition, the number given is the default value, and the I6 source is included whether or not anybody uses the option: the default value being given if nobody does. The text "{N}" is replaced with the value. So the above definition normally results in this being defined:

```
Constant MAX_PRESENTED_FOOTNOTES = 100;
```

but if the user writes

```
Use maximum presented footnotes of at least 350.
```

then instead the I6 inclusion becomes:

```
Constant MAX_PRESENTED_FOOTNOTES = 350;
```

The I6 constant MAX_PRESENTED_FOOTNOTES can then be used as the size of an array, for instance.


Start of Chapter 27: Extensions


Back to §27.17. Handling phrase options


Onward to §27.19. Longer extracts of Inform 6 code

---

§27.19. Longer extracts of Inform 6 code

Whole routines, object and class definitions (or any other directives) can be pasted in wholesale using sentences like so:

```
Include (-
[ MyInform6Routine a b; return a*b; ];
-).
```

Such inclusions are pasted into the final compiled code at the end of the file, after the I6 grammar has been declared.

In such extracts, we sometimes need to refer to objects, variables or values which can't be described using I6: or rather, which can be described, but we don't know how. To this end, any text in an inclusion written in "(+" and "+)" parentheses is treated as an I7 value, and compiled accordingly, with all type-checking waived for the occasion. For instance:

```
Include (-
Global my_global = (+ the tartan rucksack +);
-).
```

Here "the tartan rucksack" is translated into "O18_tartan_rucksack", or something similar: the I6 object created to represent the rucksack. Thus the actual line of code produced is

```
Global my_global = O18_tartan_rucksack;
```

The material between "(+" and "+)" is generally treated as a value, and thus compiles to the I6 form of that value. But it could also be a property name, which compiles to the I6 form in question, or a defined adjective, which compiles to the name of the routine to call which tests whether that adjective is true.

Three warnings. The material in "(-" and "-)" is called template code, and it is not quite treated as literal. That means certain characters cause Inform to react:

1. Beware of accidental "(+" usage - for instance,

```
Include (-
[ MyCleverLoop i; for (++i; i<10; i++) print i; ];
-).
```

looks reasonable, but contains "(+" and "+)". Spaces around the first "++" would have been enough to avoid this one; "+)" is only significant where it follows a "(+".

2. Beware of placing an "@" character in the first column, that is, immediately following a new line. (In template code this marks off paragraph divisions.) So for instance,

```
Include (-
[ Set_Stream ret;
@glk 67 ret;
];
-).
```

is tripped up by the Glulx assembly language opcode "@glk" because this occurs in column 1. Indenting it with a little space or a tab is enough to avoid the problem.

3. Be careful if you're creating an I6 variable holding initialised I7 text. For example,

```
Include (-
Global saved_optional_prompt = (+ "!!>" +);
-) after "Definitions.i6t".
```

looks as if it will work, but doesn't, for reference-counting reasons we needn't go into; instead you need

```
Include (-
Array sop_storage --> PACKED_TEXT_STORAGE "!!>";
Global saved_optional_prompt = sop_storage;
-) after "Definitions.i6t".
```

But it's far better to avoid initialising text variables from I6 entirely. The same problems arise with constant lists.

Start of Chapter 27: Extensions

Back to §27.18. Making and testing use options

Onward to §27.20. Primitive Inform 6 declarations of rules

Example 452: **Status line with centered text, the hard way** A status line which has only the name of the location, centered.

---

§27.20. Primitive Inform 6 declarations of rules

By writing a sentence like this:

```
The underground rule translates into I6 as "UNDERGROUND_R".
```

we create a new rule, the "underground rule", and also notify Inform that it will have no definition as I7 source text: instead, it will be provided as an I6 routine called "UNDERGROUND_R". We can define this with an Include like so:

```
Include (-
[ UNDERGROUND_R;
    if (real_location hasnt light) { RulebookSucceeds(); rtrue; }
    rfalse;
];
-).
```

The rule should return false if it wants to make no decision, but call either RulebookSucceeds or RulebookFails and return true if it does. These routines can optionally take an argument: which will be the return value from the rulebook.

Note that UNDERGROUND_R itself has no arguments. In the case of an action based rulebook, the I6 variables noun, second and actor can be referred to, while for a value based rulebook the parameter is stored in the I6 global variable parameter_object (which is not necessarily an object, in spite of the name).

We can put this rule into a rulebook in the same way that any named rule can be:

```
The underground rule is listed in the spot danger rules.
```

Start of Chapter 27: Extensions
Back to §27.19. Longer extracts of Inform 6 code
Onward to §27.21. Inform 6 objects and classes

---

§27.21. Inform 6 objects and classes

As might be expected, I7 compiles an I6 class for each kind, and an I6 object for each of its own objects. We can meddle with its compilation process here using a further refinement of Include. For instance, suppose we want the I6 class definition for things to come out containing a property like this:

```
Class K2_thing ...
  with marmalade_jar_size 6,
  ...
```

How to arrange this? One way is to create an ordinary I7 property, like so:

```
A thing has a number called marmalade jar size. The marmalade jar size of a thing is
usually 6. The marmalade jar size property translates into I6 as "marmalade_jar_size".
```

(Without that last sentence, the property won't get any familiar name.) But sometimes we need more, and want to actually write new material to go into the definition. This can be done like so:

```
Include (- with before [; Go: return 1; ], -) when defining a vehicle.
```

This glues in a new property to the class compiled to represent the I7 kind "vehicle". (See the DM4 for why. However, since the entire actions machinery is different in the I7 world, note that "after", "react_before" and "react_after" no longer have any effect, and nor does "before" for rooms.)

And similarly:

```
Include (- has my_funny_attribute, -) when defining the hot air balloon.
```

If we need a particular I7 object or kind to end up with a particular I6 name, we can write:

```
The whatsit object translates into I6 as "whatsit".
The thingummy kind translates into I6 as "thingummy_class".
```

Start of Chapter 27: Extensions
Back to §27.20. Primitive Inform 6 declarations of rules
Onward to §27.22. Inform 6 variables, properties, actions, and attributes

---

§27.22. Inform 6 variables, properties, actions, and attributes

I7's variables are usually compiled as entries in an array rather than as I6 variables. However, we can instead tell Inform to use an existing I6 variable (either one that we declare ourselves, or one in the I6 template layer). For example:

```
Room description style is a kind of value. The room description styles are Brief, Verbose and Superbrief.
The current room description style is a room description style that varies.
The current room description style variable translates into I6 as "lookmode".
```

This is a feature provided to help I7 source text to use variables internal to the I6 template code. It can, if really necessary, also be used to give I7 names to entirely new I6-level variables, created like so:

```
Include (- Global my_variable = 0; -) after "Definitions.i6t".
```

This style of hybrid coding is really not encouraged.

I7's properties are compiled sometimes as I6 properties, sometimes as I6 attributes, sometimes as bits in a bitmap somewhere. However, we can override I7 by telling it that one of its property names is equivalent to an already-existing I6 property or attribute: if so then I7 will use that name and will not compile any directive to create it. For example:

```
The switched on property translates into I6 as "on".
The initial appearance property translates into I6 as "initial".
```

We do not need to translate "switched off", the opposite to "switched on": I7 will now compile this to "~on".

Lastly, actions can also be translated (though it's usually better to translate their rules instead and invent new I7 actions covering them):

```
The unlocking it with action translates into I6 as "Unlock".
```

Start of Chapter 27: Extensions

Back to §27.21. Inform 6 objects and classes

Onward to §27.23. Inform 6 Understand tokens

---

§27.23. Inform 6 Understand tokens

The parser which deciphers the player's typed commands is written in I6, and many of the basic tokens of Understand grammar are implemented as "general parsing routines" (GPRs), the specification of which is described fully in the Inform 6 Designer's Manual. I7 translates much of the source text's Understand grammar into GPRs, and once again we can bypass this process and supply an Understand token directly as an I6 GPR. For example:

```
The Understand token squiggle translates into I6 as "SQUIGGLE_TOKEN".
```

We then have to include a routine of that name into I7's output using the "Include" instruction, on which more later.

This creates a token "[squiggle]"; so for instance if the source text contains:

```
Understand "copy [squiggle]" as ...
```

then Inform would parse the command COPY FIGURE EIGHT by calling the SQUIGGLE_TOKEN routine as a GPR with the word marker at 2, that is, at the word FIGURE.

As always, this should be done only where there seems no better way, or where speed is very important. For any fairly simple range of possibilities, it's better to use the techniques in the Understand chapter, or to use unit specifications.


Start of Chapter 27: Extensions


Back to §27.22. Inform 6 variables, properties, actions, and attributes


Onward to §27.24. Inform 6 adjectives

---

§27.24. Inform 6 adjectives

There are three ways to specify that an adjective is defined at the I6 level. For example:

```
Definition: a number is prime rather than composite if I6 routine
    "PRIMALITY_TEST" says so (it is greater than 1 and is divisible only by itself and 1).
```

Inform now actually tests if a number N is prime by calling PRIMALITY_TEST(N), and it assumes that we have also included such a routine in the output. The routine is expected to return true or false accordingly.

The text in brackets does nothing functional, but is the text used in the Lexicon dictionary part of the Phrasebook index for the user's benefit; it should be a brief definition. Extension authors are asked to provide these little definitions, so that their users won't be confused by blank lexicon entries.

The second way makes a more capable adjective, since it can not only be tested, but also made true or false using "now". For example:

```
Definition: a scene is crucial if I6 routine "SceneCrucial" makes it so
    (it is essential to winning).
```

The difference here is "makes it so", not "says so", and as this implies, the routine has more power. "SceneCrucial" is called with two arguments: SceneCrucial(S, -1) tests whether the scene is crucial or not and returns true or false; SceneCrucial(S, true) must make it true; and SceneCrucial(S, false) must make it false. Another useful difference is that if the kind of value is one which is stored in block form (e.g. for an adjective applying to text), the routine is given a pointer to the block, not a fresh copy.

A third way to define an adjective, which should be used only if speed is exceptionally important, is to provide a "schema" - a sort of I6 macro, like those provided by the C preprocessor. For example:

```
Definition: a rulebook is exciting if I6 condition
    "excitement_array-->(*1)==1" says so (it is really wild).
```

The escape "*1" is expanded to the value on which the adjective is being tested. (This is usually faster than calling a routine, but in case of side-effects, the "*1" should occur only once in the condition, just as with a C macro.) To repeat: if in doubt, use the I6 routine method above.


Start of Chapter 27: Extensions


Back to §27.23. Inform 6 Understand tokens


Onward to §27.25. Naming Unicode characters

---

§27.25. Naming Unicode characters

Inform allows the Unicode characters to be identified either with a decimal number or by name, but it has none of the character names built-in, and for efficiency reasons it only learns them when necessary.

Users normally teach these names to Inform by including one of the extensions "Unicode Character Names" or "Unicode Full Character Names", which consist of many hundreds of sentences like so:

```
anticlockwise open circle arrow translates into Unicode as 8634.
```

Nothing restricts this usage to those extensions.


Start of Chapter 27: Extensions


Back to §27.24. Inform 6 adjectives


Onward to §27.26. The template layer

---

§27.26. The template layer

When Go is clicked, Inform translates the I7 source text into I6 code, but the directly translated code could not survive on its own: it needs a large body of supporting code, also written in I6, to sustain it. (Just as a program like iTunes or Firefox cannot run on a bare machine, but needs an operating system already up and running to support it.) Until 2008, this supporting code was provided by the I6 library, that is, the standard distribution of useful I6 code supplied with distributions of Inform 6.

However, the supporting code is now generated from a collection of about 35 "segments" of I6 code which together make up "the template layer". The reason for the term "template" is that the segments are not quite directly copied into I7's output, but instead act as templates from which I7 generates code - the final output contains variations according to what the original source text needs.

Each segment has its own name, which looks like a leafname plus the ".i6t" filename extension (which stands for "I6 template"). Internally, a segment is itself divided up into named parts. For instance, the segment "Relations.i6t" contains a part called "Symmetric One To One Relations" which provides I6 routines for changing and testing such relations. There are more than 600 named parts across the template as a whole; it is quite a large program. An annotated, typeset version of the template - amounting to a roughly 500-page book - is available for download from the Inform website.

The most powerful use of "Include" allows code to included before, instead of or after any named part or segment in the template. For example:

```
Include (- ... -) before "Relations.i6t".
Include (- ... -) instead of "Relations.i6t".
Include (- ... -) after "Symmetric One To One Relations" in "Relations.i6t".
```

Multiple such inclusions can be made for the same segment or part. If so, all will take effect in the case of "before" or "after", but for "instead of" only the most recent one takes effect. Inclusions requested before, or after, a segment or part which has been replaced with "instead of" will take effect and appear before or after the code which appears instead of it.

The pre-2008 syntax

```
Include (- ... -) before the library.
```

has been withdrawn; the new syntax

```
Include (- ... -) after "Definitions.i6t".
```

should have the same effect.

Template files are not written in literal I6, but in a marked-up, annotated form of I6 which has special transcription commands embedded into it. These commands should absolutely not be used except in the built-in template files, with one exception:

```
{-segment:Flowers.i6t}
```

places the whole of the template file "Flowers.i6t" in this position. The built-in template does not of course contain "Flowers.i6t", but Inform allows the optional "I6T" subfolder of the ".materials" folder of a project to hold additional or replacement template files. Thus the project "Botanic Gardens.inform" might store:

```
Botanic Gardens.materials/I6T/Flowers.i6t
```

It could even contain:

```
Botanic Gardens.materials/I6T/Relations.i6t
```

in which case this would automatically be used instead of the built-in copy of "Relations.i6t", without any change to the I7 source text being needed. In this way, projects can (if they need to) use partly or entirely customised templates.

One application of this might allow for chunks of I6 code generated by external utilities - Perl scripts, lex and yacc, or other code generators - by compiling those to suitable template files in .materials/I6T and then using an inclusion like

```
Include (- {-segment:MyStuff.i6t} -).
```

in the I7 source text.

Template hacking, as it's called, is a last resort. If there is any way to achieve the same ends by writing ordinary I7 source text, then that will always be better. If it is possible to write "Include (- ... -)" without mentioning any segment or part, that's much to be preferred, because it has more chance of continuing to work into the future when the template might have been rewritten.


Start of Chapter 27: Extensions


Back to §27.25. Naming Unicode characters


Onward to §27.27. Translating the language of play

---

§27.27. Translating the language of play

The "language of play" is the natural language used to communicate with the player at run-time: this is normally English.

That means that it is difficult to write, say, Spanish-language IF using Inform 7, though heroic work by the Spanish IF community has overcome this. Inform 6 provided for translation by isolating its linguistic code in a part of the I6 library called the "language definition file", which was normally "English.h". Translations were gradually made to most major European languages, resulting in alternative language definition files called "French.h", "Italian.h" and so on. Full details on how to write a language definition file were given in the Translations chapter of the DM4, that is, the fourth edition of the Inform 6 Designer's Manual.

In I7 the system is different. We use the template, not a library. Instead of providing a language definition file such as "French.h", a translator should create an extension called something like "French Language by Jacques Mensonge". (The language should be named in English, so "French Language by ...", not "Langue français by ...") This extension should then contain broadly the same material as an I6 language definition file, but written in a mostly higher-level way. See the extension "English Language by Graham Nelson" supplied with I7, which is included automatically by default.

Start of Chapter 27: Extensions
Back to §27.26. The template layer
Onward to §27.28. Segmented substitutions

---

§27.28. Segmented substitutions

A "segmented" substitution is a syntax where text is placed between two or more different text substitutions. Examples include:

```
"This hotel is [if the player is female]just awful[otherwise]basic[end if]."
"Annie [one of]dances[or]sulks[or]hangs out at Remo's[at random]."
```

To create such syntaxes, it is not enough just to define how each expands into I6 code: for one thing we may need to know about the later terms in order to expand the earlier ones, which is normally impossible, and for another thing, the individual text substitutions mean nothing in isolation. For instance, Inform produces a problem if the following is tried:

```
"The hotel [at random] is on fire."
```

because "[at random]" is only legal when closing a "[one of] ..." construction. But if "[at random]" had been defined as just another text substitution, Inform would not have been able to detect such problems.

Inform therefore allows us to mark text substitutions as being any of three special kinds: beginning, in the middle of, or ending a segmented substitution. There can be any number of alternative forms for each of these three variants. The syntax policed is that

```
(a) Any usage must lie entirely within a single say or piece of text.
(b) It must begin with exactly one of the substitutions marked as "beginning".
(c) It can contain any number, including none, of the substitutions marked as
"continuing" (if there are any).
(d) It must end with exactly one of the substitutions marked as "ending".
```

A simple example:

```
To say emphasis on -- beginning say_emphasis_on: (- style underline; -).
To say emphasis off -- ending say_emphasis_on: (- style roman; -).
```

This creates "[emphasis on]" and "[emphasis off]" such that they can only be used as a pair. The keyword "say_emphasis_on", which must be a valid I6 identifier (and hence a single word), is never seen by the user: it is simply an ID token so that Inform can identify the construction to which these belong. (We recommend that anybody creating such constructions should choose an ID token which consists of the construction's name but with underscores in place of spaces: this means that the namespace for ID tokens will only clash if the primary definitions would have clashed in any case.)

Start of Chapter 27: Extensions
Back to §27.27. Translating the language of play
Onward to §27.29. Invocation labels, counters and storage
Example 453: **Chanel Version 1** Making paired italic and boldface tags like those used by HTML for web pages.

---

§27.29. Invocation labels, counters and storage

The process of expanding the I6 code which represents a phrase is called "invocation". As we have seen, when a phrase is defined using a single piece of I6 code, invocation consists of copying out that I6 code, except that tokens in braces "{thus}" are replaced:

```
To say (something - number): (- print {something}; -).
```

Ordinarily the only token names allowed are those matching up with names in the prototype, as here, but we have already seen one special syntax: "{phrase options}", which expands as a bitmap of the options chosen. And in fact the invocation language is larger still, as a skim through the Standard Rules will show. The notes below deliberately cover only some of its features: those which are likely to remain part of the permanent design of Inform, and which are adaptable to many uses. **Please do not use any of the undocumented invocation syntaxes: they change frequently, without notice or even mention in the change log.**

The first special syntaxes are textual tricks. {-delete} deletes the most recent character in the I6 expansion of the phrase so far. {-erase} erases the I6 expansion of the phrase so far. {-open-brace} and {-close-brace} produce literal "{" and "}" characters.

The following:

```
{-counter:NAME}
{-counter-up:NAME}
{-zero-counter:NAME}
{-counter-makes-array:NAME}
```

create (if one does not already exist) a counter called NAME. This is initially zero, and can be reset back to zero using "{-zero-counter:NAME}", which expands into no text. The token "{-counter:NAME}" expands into the current value of the counter, as a literal decimal number. The token "{-counter-up:NAME}" does the same, but then also increases it by one. Finally, the token "{-counter-makes-array:NAME}" expands to nothing, but tells Inform to create an "-->" array called "I7_ST_NAME" which includes entries from 0 up to the final value of the NAME counter.

This allows each instance in the source text of a given phrase to have both (i) a unique ID number for that invocation, and (ii) its own word of run-time storage, which can allow it to have a state preserved in between times when it is executed. For example:

```
To say once only -- beginning say_once_only:
    (- {-counter-makes-array:say_once_only}if (I7_ST_say_once_only-->{-
counter:say_once_only} == false) {-open-brace} I7_ST_say_once_only-->{-counter-
up:say_once_only} = true; -).
```

```
To say end once only -- ending say_once_only:
    (- {-close-brace} -).
```

To complete the tools available for defining a segmented substitution, we need a way for the definition of the head to know about the middle segments and the tail:

When invoking either the head or the tail, {-segment-count} expands to the literal decimal number of pieces of text in between the two, which is always one more than the number of middle segments, since the text comes in between the segments. When invoking any middle segment, {-segment-count} expands to the number of pieces of text so far -- thus it expands to 1 on the first middle segment invoked, 2 on the next, and so on.

Lastly {-final-segment-marker} expands to the I6 identifier which marks the end segment, or to I6_NULL if the end segment has no marker. The idea of markers is to enable the head's definition to know which of a number of choices has been used for the tail, supposing that this is a construction with a variety of legal endings. For example:

```
To say emphasise -- beginning say_emphasise:
    (- style {-final-segment-marker}; -).
To say with italics -- ending say_emphasise with marker underline:
    (- style roman; -).
To say with fixed space type -- ending say_emphasise with marker fixed:
    (- style roman; -).
```

The markers used for the tails here are "underline" and "fixed", and when the head is invoked, the marker for its tail is expanded into the argument of I6's "style" statement.

The examples above are all to do with segmented substitutions, which is where they are most useful, but most of the syntaxes above work equally well for ordinary "To..." phrase definitions.


Start of Chapter 27: Extensions


Back to §27.28. Segmented substitutions


Onward to §27.30. To say one of

---

§27.30. To say one of

Many of the invocation syntaxes described in the previous section are used in the definition by the Standard Rules of the "[one of] ... [or] ... [purely at random]" construction, so it makes a good example of how they can be used.

First, this is a segmented substitution with a single possible beginning ("[one of]"), a single possible middle ("[or]") but a choice of many possible endings. Almost everything is compiled by the invocation of the beginning:

```
To say one of -- beginning say_one_of (documented at phs_oneof): (-
    {-counter-makes-array:say_one_of}
    {-counter-makes-array:say_one_flag}
    if (I7_ST_say_one_flag-->{-counter:say_one_flag} == false) {
```

```
        I7_ST_say_one_of-->{-counter:say_one_of} = {-final-segment-marker}
(I7_ST_say_one_of-->{-counter:say_one_of},
{-segment-count});
        I7_ST_say_one_flag-->{-counter:say_one_flag} = true;
    }
    if (say__comp == false) I7_ST_say_one_flag-->{-counter:say_one_flag}{-counter-
up:say_one_flag} =
false;
    switch ((I7_ST_say_one_of-->{-counter:say_one_of}{-counter-up:say_one_of})%({-
segment-count}+1)-1)
{-open-brace}
        0: -).
To say or -- continuing say_one_of (documented at phs_or):
    (- @nop; {-segment-count}: -).
To say purely at random -- ending say_one_of with marker I7_SOO_PAR (documented
at phs_purelyrandom):
    (- {-close-brace} -).
```

The 3rd invocation of this (say) might compile the following:

```
I7_ST_say_one_of-->2 = I7_SOO_PAR(I7_ST_say_one_of-->2, 4);
switch((I7_ST_say_one_of-->2)%5 - 1) {
    0: ... first text ...
    1: ... second text ...
    2: ... third text ...
    3: ... fourth text ...
}
```

First, we notified Inform that it needs to allocate an array (I7_ST_say_one_of) providing storage associated with the counter "say_one_of". This we used to count off individual invocations of "[one of]", so that each would have its own word of storage - for the 3rd invocation, I7_ST_say_one_of-->2. We then call a state-changing routine, in this case I7_SOO_PAR, which is allowed to know the previous state and also the number of options available, and which returns the new state. The state is supposed to be the option chosen last time, but that means that there are not 4, but 5 possibilities: 0 for "there was no last time", then 1 to 4 for the possible outcomes. We reduce the state mod 5 to obtain the decision this time, and subtract 1 because it happens to be convenient to make the switch statement run from 0 to 3 rather than 1 to 4. (The reason we reduce the state mod 5 is to allow the state-changer to squirrel away secret information in the upper bits of the state, if it wants to. Note that subtracting one means that the switch value might be -1, which results in no text being printed: thus if the state-changer chooses 0, it can decide on none of the above.)

In this design, the marker attached to the choice of ending substitution is the name of the I6 state-changer: here is the I7_SOO_PAR routine.

```
[ I7_SOO_PAR oldval count; if (count <= 1) return count; return random(count); ];
```

As it happens, this ignores the old value: after all, it is meant to be purely at random, and nothing could be less pure than taking the last outcome into consideration when choosing the next.

Note that the counter say_one_of is advanced in invocation of the head. It might seem that the tidier design, somehow, would be to advance the counter in the invocation of the tails, but this is not a good idea. In general it is not safe to assume that the counter will have the

same value when the tail is invoked that it had when the head was invoked, because segmented say constructions can legally be nested in Inform strings. Because of this, it is best to deal with a counter entirely in a single invocation, either of the beginning or the ending.

Because "[one of] ... [or] ..." is such a useful construction - switching between alternative forms of text, which writers of IF very often do - the above implementation is intentionally left open for new endings to be added, and the examples below show how easily this can be done.

Start of Chapter 27: Extensions

Back to §27.29. Invocation labels, counters and storage

Example 454: **Blink** Making a "by atmosphere" token, allowing us to design our own text variations such as "[one of]normal[or]gloomy[or]scary[by atmosphere]".

Example 455: **Uncommon Ground** Making a "by viewpoint" token, allowing us to design our own text variations such as "[show to yourself]quaint[to Lolita]thrilling[to everyone else]squalid[end show]" depending on the identity of the player at the moment.

Examples from Chapter 27: Extensions

Start of this chapter

Indexes of the examples

449 **Example** **Modern Conveniences** *RB*

Exemplifying the kind of source we might use in writing extensions for kitchen and bathroom appliances.

Suppose we want to write an extension or other portable code which defines a "kitchen" kind of room and a "bathroom" kind of room. All kitchen rooms we create in the future will automatically contain standard kitchen appliances: fridge, freezer, sink with taps, counters, cabinets, and a stovetop with built-in oven. Similarly, all bathrooms will have sinks, baths, cabinets, and toilets, and respond to some standard interactions.

We would do this with a standard assembly:

```
"Modern Conveniences"

Section 1 - Kitchens

A kitchen is a kind of room.

A refrigerator is a kind of container. A refrigerator is usually closed and
openable. A refrigerator is usually fixed in place. A refrigerator is usually
scenery. Understand "fridge" as a refrigerator.
```

```
A freezer compartment is a kind of container. A freezer compartment is usually
closed and openable. A freezer compartment is part of every refrigerator.
```

Now: we're going to want many of the items in our kitchen to have switches, and to handle input sensibly regardless of whether the player types TURN ON STOVE or TURN ON STOVE SWITCH. (This is apparently a stove with only one burner.) For convenience, we'll define an "includes" relation:

```
Inclusion relates a thing (called X) to a thing (called Y) when Y is part of X. The
verb to include means the inclusion relation.

A stove is a kind of supporter. It is usually scenery.
An oven is a kind of container. An oven is usually openable and closed. One
oven is a part of every stove.
A switch is a kind of device. A switch is part of every stove. A switch is part of
every oven.
Understand "[something related by reversed incorporation] switch" as a switch.
```

What follows is fairly straightforward, but notice that we are somewhat obsessively naming every rule. This is much more important in extensions (where someone else may need to manipulate our code from within their own source) than it is when we are simply composing source text for ourselves.

```
Setting action variables for opening a stove (this is the stove-opening rule):
    let relevant oven be a random oven which is part of the noun;
    now the noun is the relevant oven.

Setting action variables for switching on something which includes a switch (this
is the redirecting switches for switching on rule):
    let relevant switch be a random switch which is part of the noun;
    now the noun is the relevant switch.

Setting action variables for switching off something which includes a switch (this
is the redirecting switches for switching off rule):
    let relevant switch be a random switch which is part of the noun;
    now the noun is the relevant switch.

Before printing the name of a switch (called target) (this is the switch
identification rule):
    say "[random thing which includes the target] ".

A sink is a kind of container. A sink is usually fixed in place and scenery. A tap is
a kind of switch. A tap is part of every sink. Understand "faucet" or "taps" as a
tap. Understand "[something related by reversed incorporation] tap/faucet/taps"
as a tap.

Instead of opening a tap, try switching on the noun. Instead of closing a tap, try
switching off the noun.

Report switching on a tap (this is the standard report switching taps on rule):
    say "You turn on [the noun]." instead. [since "switch on" sounds weird in this
context.]
```

```
Report switching off a tap (this is the standard report switching taps off rule):
    say "You turn off [the noun]." instead.

After examining something which includes a switched on tap (called relevant tap)
(this is the report flowing water rule):
    say "The water is flowing from [the relevant tap]."

A drain is a kind of container. A drain is part of every sink. Understand "plughole"
as the drain. Understand "[something related by reversed incorporation]
drain/plughole" as a drain.

Instead of inserting something into a drain (this is the no clogging drains rule),
say "Pointless."
```

This is probably about as far as we want to go in a generic simulation: it is tempting to code up water, drains down which the player can lose objects, sinks that get clogged and overflow, and so on; but the more we embellish in these ways, the more likely the end result would be disruptive to individual games. For right now what we're aiming for is something simple which provides the basic interactions a player might expect in this kind of room, but which does not have any significant implications for the surrounding world model.

A particularly conservative author might even want to make it turn out that the water has been shut off and nothing flows from the taps: in the extension documentation, we might want to include a line or two of example showing how this might be done by changing or removing the relevant rules of our extension.

```
A cabinet is a kind of container. A cabinet is usually openable and closed. It is
scenery.
    Understand "cupboard" or "cupboards" or "cabinets" as a cabinet.

A counter is a kind of supporter. It is scenery.
    Understand "countertop" as a counter.

A cabinet is in every kitchen.
A counter is in every kitchen.
A refrigerator is in every kitchen.
A sink is in every kitchen.
A stove is in every kitchen.

Section 2 - Bathrooms

A bathroom is a kind of room.

A toilet is a kind of supporter. Understand "john" as a toilet. A toilet is usually
fixed in place and enterable and scenery.

A bath is a kind of container. A bath is usually a fixed in place and enterable and
scenery. A tap is part of every bath. A drain is part of every bath. Understand
"bathtub" or "shower" as a bath.

A sink is in every bathroom.
A toilet is in every bathroom.
```

```
A bath is in every bathroom.
A cabinet is in every bathroom.
```

If we were feeling particularly ambitious and inclined toward interior decoration, we could add bath mats, mirrors, plungers, toilet brushes, overhead lighting, towel racks, scented candles, boxes of facial tissue, shampoo bottles, scrubbing loofahs, etc. ad nauseam; but we'll keep it relatively simple for now. Of course, if we have a toilet, we pretty much have to accept that the player will try to make use of it:

```
Understand "flush [toilet]" or "use [toilet]" as a mistake ("You have no need at the
moment.").

Understand "take shower" or "take bath" or "bathe" or "wash" as bathing.
Bathing is an action applying to nothing.

Check bathing (this is the restrict baths to bathrooms rule):
    if the location is not a bathroom, say "This isn't the place." instead.

Check bathing (this is the block bathing rule):
    say "You haven't time for a bath." instead.

Washing is an action applying to one thing. Understand "clean [something]" or
"wet [something]" or "wash [something]" as washing.

Instead of washing the player, try bathing.

Check washing (this is the restrict washing to the proximity of sinks rule):
    unless the player can touch a sink, say "This isn't the place." instead.

Check washing (this is the block washing rule):
    say "It doesn't seem worth the bother." instead.
```

Now we might put this to work in a short example.

One slight challenge lies in giving these assembled pieces separate descriptions. When we have an assembly that adds parts to objects, we can then talk about (for instance) "the stove's switch" elsewhere in the code. But items that have been assigned rooms are not named in the same way, so we cannot talk about "the Industrial Kitchen's stove" in our code as a way to assign it a description or special behavior. In quite a simple example, we could make the descriptions of the kind simply be the descriptions we want for the individual items:

```
Section 3 - An Example We Might Offer

Our Household Kitchen is a kitchen.

The Tiny Bathroom is a bathroom. It is west of Our Household Kitchen.

The description of a stove is "Scrupulously polished."

The description of a refrigerator is "It is baby blue and has the contours of a
50[']s chevy. One of these days it really will break down, but it's been serving
```

```
your family faithfully since your grandmother's honeymoon."

Test me with "x refrigerator / open fridge / x freezer / look in freezer / open
freezer / turn on stove / turn on oven / x oven switch / turn off oven switch / turn
off stove switch / turn on taps / x sink / w / x sink / turn on sink / take bath / use
toilet".
```

In a game that featured multiple bathrooms and kitchens, this wouldn't be enough; our author might give the stove kind (say) a description that checked its location, as in

```
The description of a stove is "[if in Industrial Kitchen]A massive stainless steel
stove-top with six burners[otherwise]Your standard four-burner item[end if]."
```

or create an

```
Instead of examining the stove in the Industrial Kitchen: ...
```

sort of rule for those objects he wanted to describe specially; or he might use a when play begins rule to initialize a few things:

```
When play begins:
    let N be a random stove in the Industrial Kitchen;
    move the boiling pot to N;
    change the description of N to...
```

Or we might even (if we anticipate lots of these kinds of amendments) want to rig up something more complex that finds the descriptions of appliances in a table, rather than relying on their individual description properties. This can all be done, but it is only interesting as long as it remains genuinely labor-saving: that is, as long as the convenience of having the assembly is greater than the annoyance of writing special rules to cover for the automation. In the end, the "kitchen" and "bathroom" room types are likely to be most useful for authors who want to include the standard props but not actually make them a critical part of the game; if stoves and sinks have more of a starring role in the production, authors may be better off coding them or at the very least placing them by hand, as in

```
The Industrial Kitchen is a room.

Thor is a stove in the Industrial Kitchen. It supports a boiling pot.
```

All these quirks are things that we (as the extension author) want to think out in advance: we should ideally warn authors about possible pitfalls in using our extension (if we can think of them) and point out ways of customizing the behavior (if there are interesting ways).

---

450 **Example Tilt 3** RB

Displaying the card suits from our deck of cards with red and black colored unicode symbols.

Books and articles about card-playing traditionally abbreviate card names into a simple two-symbol notation: a number or letter representing the card rank, followed by a symbol indicating the card suit. Suppose that we want to emulate this notation when taking inventory in our poker game.

The trick here is that colored output is done in different ways by the Z-Machine and by Glulx, so we'll need two different versions of the same section in order to produce this output. The relevant source is right at the beginning:

```
"Tilt"

Section 0 - Colored Output in Two Forms
```

For the suit symbols, we'll want the Unicode extension included with Inform:

```
Include Unicode Character Names by Graham Nelson.

Rule for printing the name of a card (called target) while grouping together:
    say "[rank of the target as abbreviated value][suit of the target as symbol]".

To say (current suit - a suit) as symbol:
    if current suit is diamonds, say "[red letters][unicode black diamond suit]
[default letters]";
    if current suit is spades, say "[unicode black spade suit]";
    if current suit is clubs, say "[unicode black club suit]";
    if current suit is hearts, say "[red letters][unicode black heart suit][default
letters]".

Section 0Z (for Z-machine only)
```

The Basic Screen Effects extension bundled with Inform includes mechanisms to change the text color, so for the Z-machine, we need only include this:

```
Include Basic Screen Effects by Emily Short.

Section 0G (for Glulx only)
```

Under Glulx, we need slightly more set-up: Glulx requires that we define special user font styles when we plan to make display changes. A fuller discussion of this (and of how to define new colors) appears in the documentation of "Glulx Text Effects", but an implementation sufficient to our purposes would be

```
Include Glulx Text Effects by Emily Short.

Table of User Styles (continued)

    style name      glulx color
    special-style-1 g-pure-red

Table of Common Color Values (continued)

    glulx color value assigned number
```

```
g-pure-red        16711680
```

```
To say red letters: say first custom style.

To say default letters: say roman type.
```

From here, the rest of the source is mostly as we've seen in previous examples:

```
Section 1 - Cards

Suit is a kind of value. The suits are hearts, clubs, diamonds, and spades.
Understand "heart" as hearts. Understand "club" as clubs. Understand
"diamond" as diamonds. Understand "spade" as spades.

A card is a kind of thing. A card has a suit. A card has a number called rank.
Understand the suit property as describing a card. Understand the rank property
as describing a card.

52 cards are in the card repository.

To say (count - a number) as a card value:
   choose row count in the Table of Value Names;
   say "[term entry]".

Rule for printing the name of a card (called target):
   say "[rank of the target as a card value] of [suit of the target]"

To say (count - a number) as abbreviated value:
   choose row count in the Table of Value Names;
   say "[abbrev entry]".

Table of Value Names

term      value abbrev topic
"ace"     "1"   "A"     "ace/A"
"deuce"   "2"   "2"     "deuce/two"
"three"   "3"   "3"     "three"
"four"    "4"   "4"     "four"
"five"    "5"   "5"     "five"
"six"     "6"   "6"     "six"
"seven"   "7"   "7"     "seven"
"eight"   "8"   "8"     "eight"
"nine"    "9"   "9"     "nine"
"ten"     "10"  "10"    "ten"
"jack"    "11"  "J"     "jack/knave/J"
"queen"   "12"  "Q"     "queen/Q"
"king"    "13"  "K"     "king/K"

After reading a command:
   if the player's command includes "of [suit]":
      while the player's command includes "of":
         cut the matched text;
   repeat through the Table of Value Names:
      while the player's command includes topic entry:
         replace the matched text with value entry.
```

```
When play begins:
    reconstitute deck.

To reconstitute deck:
    let current suit be hearts;
    now every card is in the card repository;
    while a card is in the card repository:
        repeat with current rank running from 1 to 13:
            let item be a random card in card repository;
            now rank of item is current rank;
            now suit of item is current suit;
            now item is in the deck of cards;
        now current suit is the suit after the current suit.
```

Section 2 - The Deck and the Discard Pile

```
The Empty Room is a room. "Nothing to see here."

The deck of cards is in the Empty Room. It is a closed unopenable container.
The description is "A standard poker deck."

The discard pile is a closed unopenable container. The description is "Cards in
this game are discarded face-down, so the discard pile is not very interesting to
see. All you can observe is that it currently contains [if the number of cards
which are in the discard pile is less than ten][the number of cards which are in
the discard pile in words][otherwise]about [the rounded number of cards which
are in the discard pile in words][end if] card[s]."

To decide what number is the rounded number of (described set - a description
of objects):
    let N be the number of members of the described set;
    let R be N divided by 5;
    let total be R times 5;
    decide on total.

Rule for printing room description details of something: do nothing instead.
```

Section 3 - Drawing and Discarding Actions

```
Understand the commands "take" and "carry" and "hold" and "get" and "drop"
and "throw" and "discard" as something new.

Understand "take [text]" or "get [text]" or "drop [text]" as a mistake ("Here, you
only draw and discard. Nothing else matters at the moment.").

Understand "draw" or "draw card" or "draw a card" as drawing. Drawing is an
action applying to nothing. The drawing action has an object called the card
drawn.

Setting action variables for drawing:
    now the card drawn is a random card which is in the deck of cards.

Check drawing:
    if the card drawn is nothing, say "The deck is completely depleted." instead.
```

```
Check drawing:
    if the number of cards carried by the player is greater than four,
        say "This is a five-card game; you must discard something before drawing
anything further." instead.

Carry out drawing:
    move the card drawn to the player.

Report drawing:
    say "You draw [a card drawn]."

Understand "discard [card]" as discarding. Discarding is an action applying to
one thing.

Check discarding:
    if the player does not carry the noun, say "You can only discard cards from
your own hand." instead.

Carry out discarding:
    now the noun is in the discard pile;
    if the discard pile is not visible, move the discard pile to the location.

Report discarding:
    say "You toss [the noun] nonchalantly onto the discard pile."

Section 4 - Assessing Hands

Before listing contents while taking inventory: group cards together.

Before grouping together cards:
    if the number of cards carried by the player is 5:
        say "[run paragraph on]";
        follow the hand-ranking rules;
        if the rule succeeded, say "[the outcome of the rulebook]";
        otherwise say "some random cards";
        if the outcome of the rulebook is pair outcome, say " of [rank of the first
thing held by the player as a card value]s";
    otherwise:
        say "[number of cards carried by the player in words] assorted cards";
    say " (".

Rule for grouping together cards:
    say "[list hand]".

To say list hand:
    let chosen card be the first thing held by the player;
    while chosen card is a card:
        say "[chosen card]";
        now chosen card is the next thing held after chosen card;
        if chosen card is a card, say "-".

After grouping together cards:
    say ")".
```

```
The hand-ranking rules is a rulebook. The hand-ranking rules have outcomes
royal flush, straight flush, four of a kind, full house, flush, straight, three of a kind,
two pairs, pair, high card.

The hand-ranking rulebook has a truth state called the flushness.
The hand-ranking rulebook has a truth state called the straightness.

The hand-ranking rulebook has a number called the pair count.
The hand-ranking rulebook has a number called the triple count.
The hand-ranking rulebook has a number called the quadruple count.

A card can be sorted or unsorted. A card is usually unsorted.

Definition: a card is high if its rank is 11 or more.
Definition: a card is low if its rank is 4 or less.

A hand-ranking rule (this is the initial sort rule):
    now every card is unsorted;
    while the player carries an unsorted card:
        let item be the lowest unsorted card held by the player;
        move item to the player;
        now the item is sorted;
    if sort-debugging is true, say "-- after initial sort: [list hand]".

A hand-ranking rule (this is the finding flushness rule):
    let called suit be the suit of a random card carried by the player;
    if every card carried by the player is called suit, now flushness is true.

A hand-ranking rule (this is the finding straightness rule):
    now straightness is true;
    let N be the rank of the highest card which is carried by the player;
    repeat with current rank running from N - 4 to N:
        now the test rank is the current rank;
        unless the player carries a matching card:
            if the current rank is N - 4 and the current rank is 9 and the player carries
an ace card, do nothing;
            otherwise now straightness is false.

A card can be quadrupled, tripled, paired or uncombined.

Test rank is a number that varies. Definition: a card is matching if its rank is the
test rank.

A hand-ranking rule (this is the counting multiples rule):
    now every card is uncombined;
    repeat with current rank running from 1 to 13:
        now test rank is current rank;
        let N be the number of matching cards held by the player;
        if N is 4:
            increment the quadruple count;
            now every matching card held by the player is quadrupled;
        if N is 3:
            increment the triple count;
            now every matching card held by the player is tripled;
        if N is 2:
```

```
            increment the pair count;
            now every matching card held by the player is paired.

A hand-ranking rule (this is the move aces up unless there's a low straight rule):
    unless the straightness is true and the lowest card carried by the player is an
ace card and the rank of the highest card carried by the player is 5,
        now every ace card which is carried by the player is carried by the player;
    if sort-debugging is true, say "-- after ace movement rule: [list hand]".

A hand-ranking rule (this is the move pairs forward rule):
    while the player carries a paired card:
        let selection be the lowest paired card which is carried by the player;
        move the selection to the player;
        now the selection is uncombined;
    if sort-debugging is true, say "-- after pairs movement: [list hand]".

A hand-ranking rule (this is the raise ace pairs rule):
    if the player carries exactly two ace cards:
        repeat with item running through ace cards which are carried by the player:
            move item to the player;
    if sort-debugging is true, say "-- after paired-ace movement: [list hand]".

A hand-ranking rule (this is the move multiples forward rule):
    while the player carries a tripled card:
        let selection be the lowest tripled card which is carried by the player;
        move the selection to the player;
        now the selection is uncombined;
    while the player carries a quadrupled card:
        let selection be the lowest quadrupled card which is carried by the player;
        move the selection to the player;
        now the selection is uncombined;
    if sort-debugging is true, say "-- after multiples movement rule: [list hand]".

Definition: a card is ace if its rank is 1.
Definition: a card is king if its rank is 13.

A hand-ranking rule (this is the royal-flush rule):
    if flushness is true and straightness is true and the highest card carried by the
player is king and the lowest card carried by the player is ace, royal flush.

A hand-ranking rule (this is the straight-flushes rule):
    if flushness is true and straightness is true, straight flush.

A hand-ranking rule (this is the four-of-a-kind rule):
    if the quadruple count is 1, four of a kind.

A hand-ranking rule (this is the full-house rule):
    if the pair count is 1 and the triple count is 1, full house.

A hand-ranking rule (this is the flushes rule):
    if flushness is true, flush.

A hand-ranking rule (this is the straights rule):
    if straightness is true, straight.
```

A hand-ranking rule (this is the three-of-a-kind rule):
    if triple count is 1, three of a kind.

A hand-ranking rule (this is the two-pair rule):
    if the pair count is 2, two pairs.

A hand-ranking rule (this is the pair rule):
    if the pair count is 1, pair.

A hand-ranking rule (this is the default rule):
    high card.

Sort-debugging is a truth state that varies.

Section 5 - Testing hand identification - Not for release

Understand "debug sorting" as debugging hand sorting. Debugging hand sorting is an action out of world.

Carry out debugging hand sorting:
    if sort-debugging is false, now sort-debugging is true;
    otherwise now sort-debugging is false.

Report debugging hand sorting:
    say "Sort debugging is now [if sort-debugging is true]on[otherwise]off[end if]."

Test me with "draw / g / g / g / g / force hand / g / g / g / g / g / g / g / g / g / g / g / g / g / g".

Table of Testing Hands

set suit	set rank	
spades	1	[royal flush]
spades	13	
spades	12	
spades	11	
spades	10	
clubs	12	[straight flush]
clubs	11	
clubs	10	
clubs	9	
clubs	8	
diamonds	8	[four of a kind]
hearts	8	
spades	8	
clubs	8	
clubs	3	
clubs	1	[full house]
spades	1	
hearts	10	
spades	10	
clubs	10	
hearts	2	[flush]
hearts	5	
hearts	7	
hearts	11	
hearts	12	
hearts	1	[straight]
spades	13	
diamonds	12	
clubs	11	

```
hearts      10
hearts      2       [three of a kind]
spades      2
clubs       2
clubs       4
spades      3
diamonds    6       [two pairs]
spades      6
clubs       7
diamonds    7
hearts      9
diamonds    6       [two pairs, ace high]
spades      6
clubs       1
diamonds    7
hearts      1
hearts      12      [pair]
spades      12
diamonds    10
spades      7
clubs       4
diamonds    13      [high]
hearts      11
spades      9
clubs       7
diamonds    5
hearts      1       [tricky sorting: low straight]
diamonds    2
spades      3
diamonds    4
diamonds    5
```

```
Understand "force hand" as forcing a hand. Forcing a hand is an action out of
world.

Current marker is a number that varies.

Carry out forcing a hand:
    repeat with item running through cards which are carried by the player:
        increment current marker;
        if current marker is greater than the number of filled rows in the Table of
Testing Hands, now current marker is 1;
        choose row current marker in the Table of Testing Hands;
        now the suit of item is the set suit entry;
        now the rank of item is the set rank entry.

Report forcing a hand:
    try taking inventory.
```

451

Example **Pink or Blue**

RB

Asking the player to select a gender to begin play.

Suppose we would like to allow the player to choose a gender for the main character. We'd also like this to happen at the beginning of the game and outside the main parsing sequence. "When play begins" seems like a good place to put this.

```
"Pink or Blue"
```

```
When play begins:
    say "Should your character be male or female? >";
    if men win, now the player is male;
    otherwise now the player is female;
    say paragraph break.
```

Now a piece of Inform 6 code handles the unusual input. It's not necessary to understand this to use it, and the code should work for any question you'd like to ask the player. The first three words in quotation marks ('male', 'M', 'man'...) correspond to positive feedback; the later three words correspond to negative feedback. So "to decide whether men win" will be true if the player types one of the first three, and false if he types one of the last three.

```
To decide whether men win:
    (- Question('male','M//','man','female','F//','woman') -)

Include (-

[ Question pos1 pos2 pos3 neg1 neg2 neg3 first_word_typed;
    while (true) {
        VM_ReadKeyboard(buffer, parse);
        wn = 1; first_word_typed = NextWordStopped();
        if (first_word_typed == pos1 or pos2 or pos3) rtrue;
        if (first_word_typed == neg1 or neg2 or neg3) rfalse;
        print "Please choose ", (address) pos1, " or ", (address) neg1, ". > ";
    }
];

-)

Instead of examining the player when the player is female:
    say "Congratulations, you are a girl!"

Instead of examining the player when the player is male:
    say "Congratulations, you are a boy!"

The Room of Self-Knowledge is a room. "Mirrors cover every available wall-
surface of this hexagonal chamber, allowing you to examine yourself from all
angles."
```

452

Example Status line with centered text, the hard way

RB

A status line which has only the name of the location, centered.

Making major changes to display features, such as the construction of the status line, sometimes requires that we rely on Inform 6 in bulk; here is how we might produce the Trinity-style status line, with the location centered at the top center of the screen.

```
"Corner of No and Where"

No is a room. Where is west of No.
```

```
Rule for constructing the status line:
    print the location in the center of the status line;
    rule succeeds.

To print the location in the center of the status line:
    (- PrintCenteredStatus(); -).

Include (-

Array printed_text --> 64;

[ PrintCenteredStatus i j;
    @set_cursor 1 0;
    i = 0->33;
    spaces(i);
    @output_stream 3 printed_text;
    print (name) location;
    @output_stream -3;
    j = (i - (printed_text-->0))/2;
    @set_cursor 1 j;
    print (name) location;
    spaces j-1;
];

-)

Test me with "w / e".
```

In fact, as we've already seen, many extra modifications to the display behavior are possible using Basic Screen Effects.

453

Example **Chanel Version 1**

RB

Making paired italic and boldface tags like those used by HTML for web pages.

HTML uses angled brackets to achieve effects, and places italicised text between and tags; and similarly boldface between and . We can mimic this very easily by setting each up as a segmented substitution:

```
"Chanel Version 1"

To say i -- beginning say_i -- running on: (- style underline; -).
To say /i -- ending say_i -- running on: (- style roman; -).
To say b -- beginning say_b -- running on: (- style bold; -).
To say /b -- ending say_b -- running on: (- style roman; -).

Place Vendôme is a room. "[i]Fashion fades, only style remains the same[/i]
([b]Coco Chanel[/b]). And this elegant drawing-room, once a milliner's shop, is a
case in point."
```

```
Instead of going nowhere, say "[i]Don't spend time beating on a wall, hoping to
transform it into a door.[/i] ([b]Coco Chanel[/b]) This one is a wall.".

Test me with "look / e".
```

We have had to use square instead of angle brackets, but then, "in order to be irreplaceable one must always be different" (Coco Chanel).

(Marking these as substitutions which run on prevents unexpected paragraph breaks if they should appear immediately after the end of a sentence.)

454

Example Blink

RB

Making a "by atmosphere" token, allowing us to design our own text variations such as "[one of]normal[or]gloomy[or]scary[by atmosphere]".

Suppose we are writing a game in which the mood of the piece changes, and we would like to have lots of descriptions that vary according to its current state. We might in that case want to create our own "by atmosphere" token, to control text variations, like this:

```
"Blink"

Atmosphere is a kind of value. The atmospheres are normal, melancholy, and
creepy.

The current atmosphere is an atmosphere that varies.

To say by atmosphere -- ending say_one_of with marker I7_SOO_ATM:
    (- {-close-brace} -).
```

Since we're operating within the untyped Inform 6, we can make use of the fact that kinds of value are (internally) just constants, enumerated in the same order in which they were originally defined. In other words, "normal" at the I6 level translates to 1, "melancholy" to 2, and "creepy" to 3; so we can return the value of the current atmosphere, and thereby select option 1, 2, or 3:

```
Include (-
[ I7_SOO_ATM oldval count;
    if (count < (+ current atmosphere +)) return count;
    return (+ current atmosphere +); ];
-)
```

And that concludes the hard part. Now to test that it works:

```
The Flat is a room. "A small [one of]but cozy[or]depressing[or]imprisoning[by
atmosphere] flat. Outside the window, the sun is [one of][or][or]apparently [by
atmosphere]shining and there is a brisk breeze through the leaves of the birch
trees. [one of]It would be quite nice weather for a walk[or]The rest of the world
```

```
apparently has no appreciation of what you suffer[or]It all looks deceptively
normal[by atmosphere]."

Instead of waiting when the current atmosphere is normal:
    say "Everything stretches wide and flat for just a moment, as though all the
world around you were painted on a thin rubber sheet that is being [italic
type]stretched[roman type]. Then it snaps back into place, leaving your ears
ringing. But that little glitch was enough to warn you. Someone is tampering with
space-time again. Someone very close by.";
    now the current atmosphere is creepy.

Test me with "look / z / look".
```

455

Example Uncommon Ground

RB

Making a "by viewpoint" token, allowing us to design our own text variations such as "[show to yourself]quaint[to Lolita]thrilling[to everyone else]squalid[end show]" depending on the identity of the player at the moment.

A slightly more challenging case than the "by atmosphere" example is one in which we want to create text variations depending on the identity of our player character.

What we want to do is build a switch statement in I6, one that looks something like

```
switch(player)
{
    yourself: print "quaint";
    Lolita: print "thrilling";
    default: print "squalid";
}
```

out of I7 that looks like this:

```
say "[show to yourself]quaint[to Lolita]thrilling[to everyone else]squalid[end
show]".

"Uncommon Ground"

The Mud Village is a room. "You stand at the center of a [show to
yourself]quaint[to Lolita]thrilling[to everyone else]squalid[end show] mud village."

Leforge is a man in the Mud Village. Lolita is a woman in the Mud Village.

Instead of waiting:
    if the player is Lolita, now the player is Leforge;
    if the player is yourself, now the player is Lolita;
    say "You jump bodies. Whoops!"

To say show to (N - a person) -- beginning say_seen_by:
    (-
```

```
        switch(player)
        {-open-brace}
        {N}:
    -).

To say to (N - a person) -- continuing say_seen_by:
    (-
        {N}:
    -).

To say to everyone else -- continuing say_seen_by:
    (-
        default:
    -)

To say end show -- ending say_seen_by:
    (-
        {-close-brace}
    -)

Test me with "look / z / look / z / look".
```
