32.5.2 Uuencoding

Another odd-man-out in the template alphabet is `u`, which packs an "uuencoded string". ("uu" is short for Unix-to-Unix.) Chances are that you won't ever need this encoding technique which was invented to overcome the shortcomings of old-fashioned transmission mediums that do not support other than simple ASCII data. The essential recipe is simple: Take three bytes, or 24 bits. Split them into 4 six-packs, adding a space (0x20) to each. Repeat until all of the data is blended. Fold groups of 4 bytes into lines no longer than 60 and garnish them in front with the original byte count (incremented by 0x20) and a `"\n"` at the end. - The `pack` chef will prepare this for you, a la minute, when you select pack code `u` on the menu:

```
my $uubuf = pack( 'u', $bindat );
```

A repeat count after `u` sets the number of bytes to put into an uuencoded line, which is the maximum of 45 by default, but could be set to some (smaller) integer multiple of three. `unpack` simply ignores the repeat count.

32.5.3 Doing Sums

An even stranger template code is `%`<*number*>. First, because it's used as a prefix to some other template code. Second, because it cannot be used in `pack` at all, and third, in `unpack`, doesn't return the data as defined by the template code it precedes. Instead it'll give you an integer of *number* bits that is computed from the data value by doing sums. For numeric unpack codes, no big feat is achieved:

```
my $buf = pack( 'iii', 100, 20, 3 );
print unpack( '%32i3', $buf ), "\n";  # prints 123
```

For string values, `%` returns the sum of the byte values saving you the trouble of a sum loop with `substr` and `ord`:

```
print unpack( '%32A*', "\x01\x10" ), "\n";  # prints 17
```

Although the `%` code is documented as returning a "checksum": don't put your trust in such values! Even when applied to a small number of bytes, they won't guarantee a noticeable Hamming distance.

In connection with `b` or `B`, `%` simply adds bits, and this can be put to good use to count set bits efficiently:

```
my $bitcount = unpack( '%32b*', $mask );
```

And an even parity bit can be determined like this:

```
my $evenparity = unpack( '%1b*', $mask );
```

32.5.4 Unicode

Unicode is a character set that can represent most characters in most of the world's languages, providing room for over one million different characters. Unicode 3.1 specifies 94,140 characters: The Basic Latin characters are assigned to the numbers 0 - 127. The Latin-1 Supplement with characters that are used in several European languages is in the next range, up to 255. After some more Latin extensions we find the character sets from languages using non-Roman alphabets, interspersed with a variety of symbol sets such as currency symbols, Zapf Dingbats or Braille. (You might want to visit *www.unicode.org* for a look at some of them - my personal favourites are Telugu and Kannada.)

The Unicode character sets associates characters with integers. Encoding these numbers in an equal number of bytes would more than double the requirements for storing texts written in Latin alphabets. The UTF-8 encoding avoids this by storing the most common (from a western point of view) characters in a single byte while encoding the rarer ones in three or more bytes.

So what has this got to do with `pack`? Well, if you want to convert between a Unicode number and its UTF-8 representation you can do so by using template code `U`. As an example, let's produce the UTF-8 representation of the Euro currency symbol (code number 0x20AC):