```
    package Foo;
    our $bar;           # declares $Foo::bar for rest of lexical scope
    $bar = 20;

    package Bar;
    print $bar;         # prints 20
```

Multiple `our` declarations in the same lexical scope are allowed if they are in different packages. If they happened to be in the same package, Perl will emit warnings if you have asked for them.

```
    use warnings;
    package Foo;
    our $bar;           # declares $Foo::bar for rest of lexical scope
    $bar = 20;

    package Bar;
    our $bar = 30;      # declares $Bar::bar for rest of lexical scope
    print $bar;         # prints 30

    our $bar;           # emits warning
```

An `our` declaration may also have a list of attributes associated with it.

The exact semantics and interface of TYPE and ATTRS are still evolving. TYPE is currently bound to the use of `fields` pragma, and attributes are handled using the `attributes` pragma, or starting from Perl 5.8.0 also via the `Attribute::Handlers` module. See Private Variables via my() in *perlsub* for details, and *fields*, *attributes*, and *Attribute::Handlers*.

The only currently recognized `our()` attribute is `unique` which indicates that a single copy of the global is to be used by all interpreters should the program happen to be running in a multi-interpreter environment. (The default behaviour would be for each interpreter to have its own copy of the global.) Examples:

```
    our @EXPORT : unique = qw(foo);
    our %EXPORT_TAGS : unique = (bar => [qw(aa bb cc)]);
    our $VERSION : unique = "1.00";
```

Note that this attribute also has the effect of making the global readonly when the first new interpreter is cloned (for example, when the first new thread is created).

Multi-interpreter environments can come to being either through the fork() emulation on Windows platforms, or by embedding perl in a multi-threaded application. The `unique` attribute does nothing in all other environments.

Warning: the current implementation of this attribute operates on the typeglob associated with the variable; this means that `our $x : unique` also has the effect of `our @x : unique; our %x : unique`. This may be subject to change.

pack TEMPLATE,LIST

Takes a LIST of values and converts it into a string using the rules given by the TEMPLATE. The resulting string is the concatenation of the converted values. Typically, each converted value looks like its machine-level representation. For example, on 32-bit machines a converted integer may be represented by a sequence of 4 bytes.

The TEMPLATE is a sequence of characters that give the order and type of values, as follows:

```
    a   A string with arbitrary binary data, will be null padded.
    A   A text (ASCII) string, will be space padded.
    Z   A null terminated (ASCIZ) string, will be null padded.
```