28.1.3 The Arrow Operator

"->" is an infix dereference operator, just as it is in C and C++. If the right side is either a `[...]`, `{...}`, or a `(...)` subscript, then the left side must be either a hard or symbolic reference to an array, a hash, or a subroutine respectively. (Or technically speaking, a location capable of holding a hard reference, if it's an array or hash reference being used for assignment.) See *perlreftut* and *perlref*.

Otherwise, the right side is a method name or a simple scalar variable containing either the method name or a subroutine reference, and the left side must be either an object (a blessed reference) or a class name (that is, a package name). See *perlobj*.

28.1.4 Auto-increment and Auto-decrement

"++" and "–" work as in C. That is, if placed before a variable, they increment or decrement the variable by one before returning the value, and if placed after, increment or decrement after returning the value.

```
$i = 0;  $j = 0;
print $i++;  # prints 0
print ++$j;  # prints 1
```

Note that just as in C, Perl doesn't define **when** the variable is incremented or decremented. You just know it will be done sometime before or after the value is returned. This also means that modifying a variable twice in the same statement will lead to undefined behaviour. Avoid statements like:

```
$i = $i ++;
print ++ $i + $i ++;
```

Perl will not guarantee what the result of the above statements is.

The auto-increment operator has a little extra builtin magic to it. If you increment a variable that is numeric, or that has ever been used in a numeric context, you get a normal increment. If, however, the variable has been used in only string contexts since it was set, and has a value that is not the empty string and matches the pattern `/^[a-zA-Z]*[0-9]*\z/`, the increment is done as a string, preserving each character within its range, with carry:

```
print ++($foo = '99');      # prints '100'
print ++($foo = 'a0');      # prints 'a1'
print ++($foo = 'Az');      # prints 'Ba'
print ++($foo = 'zz');      # prints 'aaa'
```

`undef` is always treated as numeric, and in particular is changed to `0` before incrementing (so that a post-increment of an undef value will return `0` rather than `undef`).

The auto-decrement operator is not magical.

28.1.5 Exponentiation

Binary "**" is the exponentiation operator. It binds even more tightly than unary minus, so -2**4 is -(2**4), not (-2)**4. (This is implemented using C's pow(3) function, which actually works on doubles internally.)