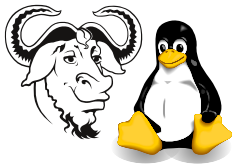

Perl Data Structures

1 Background

1.1 Scalars

You can specify scalar numerical values in the same ways as in C:

The following are all scalar constant values: `123`, `12.4`, `5E-10`, `0xff` (a hexadecimal value), `0377` (octal).

Single quotes preserve all characters unchanged, except for '\\' and '\'':

```
my $see = "very far";
my $x = "big X";
print 'What you \'$see\' is (almost) what \n you get';
print "\n";
print 'Don\'t Walk';
print "\n";
print "How are you?" . '  ' . "Substitute values of $x and \n in \" quotes.";
print "\n";
```

The output is:

```
What you '$see' is (almost) what \n you get
Don't Walk
How are you?  Substitute values of big X and
 in " quotes.
```

Back ticks work rather like they do in the shell:

```
my $total_file_size = `du -s $directory_name`;
my $date = `date`;
```

Here are three scalar values; the second is an array element, the third is a hash element:

```
$x    $list_of_things[5]     $lookup{key}
```

Single-quotes `'...'` allow no *interpolation* (substitution) except for \\and `\'`. Double-quotes `"..."` allow interpolation of variables like `$x` and control codes like `\n` (newline). Back-quotes `` `...` `` also allow interpolation, then try to execute the result as a system command, returning as the final value whatever the system command sends to standard output.

1.2 Arrays (or *Lists*)

Here are some examples of constant lists:

```
( 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
    'Saturday' )
( 13, 14, 15, 16, 17, 18, 19 )     is equivalent to  (13..19)
( 13, 14, 15, 16, 17, 18, 19 )[2..4]  equivalent to  (15, 16, 17)
```

1.3 Hashes

Here are a few examples of hashes:

```
$DaysInMonth{January} = 31;
$enrolled{'Chan Wai-yee'} = 1;
$StudentName{012345678} = 'Chan Wai-yee';
%whole_hash
```

1.4 Some More Examples

```
# A list with 12 elements:
@days = ( 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 );
%days = (
          Mon => 'SNM lecture and workshop',
          Tue => 'SNM workshops',
          Wed => 'OSSI lecture and workshops',
          Thu => 'Projects, OSSI workshops',
          Fri => 'OSSI workshops',
          Sat => 'riding bicycle with my son',
          Sun => 'Write more workshops and notes',
        );

$#days              # Last index of @days; 11 for above list @days
$#days = 7;         # shortens or lengthens list @days to 8 elements
@days               # ( $days[0], $days[1],... )
@days[3,4,5]        # = ( 30, 31, 30 )
@days{'Mon', 'Wed'} # same as ($days{Mon}, $days{Wed})
%days               # (key1, value1, key2, value2, ...)
```

1.5 Comparisons

This example program illustrates comparisons, and the "here document" (borrowed from the shell):

```
#! /usr/bin/perl

# The following "<<" variation of quoting can help simplify things sometimes
my $x = 'operator';
print <<THATSALL;
A common mistake:  Confusing the assignment $x =
and the numeric comparison $x ==, and the character
comparison $x eq.
THATSALL
$x = 7;
if ($x == 7) { print "x is $x\n"; }
if ($x = 5)  {
    print "x is now $x, ",
    "the assignment is successful.\n";
}
$x = 'stuff';
```

```
if ($x eq 'stuff') {
    print "Use eq, ne, lt, gt, le, ge for strings.\n";
}
```

The output looks like this:

```
A common mistake:  Confusing the assignment operator =
and the numeric comparison operator ==, and the character
comparison operator eq.
x is 7
x is now 5, the assignment is successful.
Use eq, ne, lt, gt, le, ge for strings.
```

Note that if the first line has a minus w '`-w`' like this:

```
#! /usr/bin/perl -w
```

then the compiler also generates a warning:

```
Found = in conditional, should be == at ./comparisons line 12.
```

1.6 `split`

It is very useful to be able to split a string into elements of an array. The Perl builtin function for this is called `split`. Here is a simple fragment to show how it works:

```
my $string = "   What a    lovely piece of    string ";
my @array = split ' ', $string;
```

Now the elements of `@array` have been filled with the words of the string:

```
$array[0] contains "What"
$array[1] contains "a"
$array[2] contains "lovely"
$array[3] contains "piece"
$array[4] contains "of"
$array[5] contains "string"
```

2 What to do

1. Type in this program and run it:

   ```
   #! /usr/bin/perl -w
   use strict;
   print "Enter numeric:  day  month  year\n";
   my ( $day, $month, $year ) = split ' ', <STDIN>;
   print "day=$day, month=$month, year=$year\n";
   ```

   Complete this program. Print an error message if the month is not valid. Print an error message if the day is not valid for the given month (31 is ok for January but not for February). See if you can reduce the use of conditional statements (`if`, `unless`, `?`, `and`, `or`, `&&`, `||`, ...) and use data structures as far as is practical without making your program hard to read.

   Approach this incrementally.

(a) On the first draft, assume that the user enters 3 numbers separated by spaces and that February has 28 days.

(b) Modify your program so that you can enter the month with *either* an integer 1...12, or a three letter value from `Jan`, `Feb`, `Mar`, `Apr`, `May`, `Jun`, `Jul`, `Aug`, `Sep`, `Oct`, `Nov` and `Dec`. You should be able to accept these in any case, so entering `jan`, `jAN`, `JAN` should all count as the month January.

(c) Subsequent refinements should account for bad input and leap year. You should also be able to provide a month as an integer in the range 1–12 or as a three-character string as described above.

> A year is a leap year if it is divisible by 400, or if it is divisible by 4 but is not also divisible by 100.

(d) Finally, use the standard Perl module `Time::Local` to perform the verification of your input. See `perldoc Time::Local`. Note that there are a number of Perl built-in functions that handle time. Of course, this should also be able to handle months in either format, as described previously.