Programming in Perl
Lecture 5 Examples

Some pragmas:

```
use strict;

-- strict flags variables not declared with my,
as well as bareword strings at compile time

use integer;

-- forces match operations to be done in integer

10/3 now returns 3, not 3.333..

use diagnostics;

-- compiler messages now more verbose along with turning on -w
```

The default package (main):

```
        use strict;

        $sum = 0;
# Global symbol "sum" requires explicit package name at ./largest2.plx line 9.
# expects sum to come from another package, can eliminate using
#  my($sum) = 0;
# or
#  $main::sum = 0;
```

Examples of modules (see Chapter 7, Programming Perl):

```
use English;

-- allows you to use English Names for many special variables

e.g., instead of $& use $MATCH
                 $`     $PREMATCH
                 $'     $POSTMATCH
(a full list is on Prog. Perl, p. 403)

use Env;

-- gives access to environment variables like and allows them
   to be treated as simple variables (e.g., $HOSTNAME, $PATH,
   $USER etc.) rather than by using the %ENV hash
```

Example of use of a class from a module:

```
use Math::Complex;
# under UNIX, will look for module Math/Complex.pm
# in some standard paths

o a module generally defines a particular type called a CLASS
```

```
  # in this case, the class is Math::Complex

  o the module provides a way to create instantiations of the
class, called OBJECTS,

  $cnum1 = Math::Complex->new(1,1);
           # called a CLASS METHOD; called as
           #   <classname>-><method>(<params>)
           # new() is the name often given to
           # an object constructor
           # new IS NOT a built-in function in Perl
  $x = 2;
  $y = 3;
  $cnum2 = new Math::Complex($x,$y);
           # another way to call new

  o module has other functions called OBJECT METHODS which
operate on objects of the class

  # examples of object methods;
  # called as
  #   <objectname>-><method>(<params>)
  $x = $cnum1->Re(); # returns real portion
  $y = $cnum1->Im(); # returns imaginary portion

  o some modules overload operators

  # create an empty complex number object
  $cnum3 = Math::Complex->new();

  $cnum3 = $cnum1 + $cnum2;
```

Creation of Hard References

```
-- backslash operator (functions much like & in C/C++)

$ref_scalar = \$scalar;
$ref_array = \@array;
$ref_hash = \%hash;
$ref_sub = \&subname;

-- ANONYMOUS ARRAY COMPOSER

$ref_array = [$a, $b, $c];
# reference to an unnamed array consisting of
# of the values of $a, $b, and $c at the time of creation

# contrast with
@array_of_refs = \($a, $b, $c);
# a three element array containing references
# to $a, $b, and $c
# same as
@array_of_refs = (\$a, \$b, $\c);
```

```
-- ANONYMOUS HASH COMPOSER

$ref_hash = { # note the brackets!
    lunch => 'noon',
    dinner => 'six pm'
};
```

Using Hard References

```
$c = 3;
$ref_scalar = \$c;
# rightmost $ dereferences, leftmost $ indicates scalar value
$value = $$ref_scalar; # $value is 3
# could also write
$value = ${$ref_scalar}; # same as above
$ref_array = [1, 2, 3];
$$ref_array[2] = 4;
# the following produces the same result
# it says $ref_array is a reference;
# dereference it and return the second element
# of the result (it always defaults to a scalar
# context -- i.e., always returns a scalar value)
$ref_array->[2] = 4;
print("@$ref_array\n"); # prints 1 2 4<newline>
$$ref_hash{'breakfast'} = 'seven am';
@meals = keys(%$ref_hash);
$c = &$ref_sub(2);  # calls a sub
```

To illustrate the difference between `[` and `\(`:

```
$a = 1; $b = 2; $c = 3;

$ref_array = [$a, $b, $c];
# assign 100 to the first array location
$$ref_array[0] = 100;
print("@$ref_array\n");
print("$a\n");
# 100 2 3
# 1

@array_of_refs = \($a, $b, $c);
# note $$array_of_refs[0] == ${$array_of_refs}[0],
#   which would be incorrect because @array_of_refs
#   is NOT a reference; therefore, must use {}
${$array_of_refs[0]} = 100;
# assign 100 to the location referred to
# by the first element of the array
foreach $ref (@array_of_refs) {
    print ($$ref, " ");
}
print("\n");
print("$a\n");
# 100 2 3
# 100
```

Passing References as Parameters

```
($ref_first, $ref_second) = order_by_means(\@a, \@b);
print("First: @$ref_first\n");
print("Second: @$ref_second\n");

sub order_by_means {
   my($ref_a) = $_[0];
   my($ref_b) = $_[1];

   my($mean_a) = calc_mean($ref_a);
   my($mean_b) = calc_mean($ref_b);

   if ($mean_a > $mean_b) {
     return ($ref_b, $ref_a);
   }
   else {
     return ($ref_a, $ref_b);
   }
}

sub calc_mean {
    my($ref_array) = $_[0];

    my($sum) = 0;

    if ( @$ref_array == 0 ) {
      return 0;
    }
    foreach $elem (@$ref_array) {
      $sum += $elem;
    }
    return ( $sum / @$ref_array ) ;
}
```

Passing Filehandles as Parameters:

```
  o  special syntax used (typeglob)

  $rec = get_rec(\*STDIN);
  sub get_rec {
    my($handle) = $_[0];
    return (scalar <$handle>);
  }
```