Programming in Perl
Lecture 5 Examples

File modes:

```
Mode Symbol        Meaning
-----------        ---------------------
(nothing)          Open for reading
<                  Open for reading
>                  Open for writing
                     (destroys current contents)
>>                 Open for appending
                     (preserves current contents;
                      ready to write at end)
+<                 Open for both reading & writing
                     (preserves current contents)
+>                 Open for both reading & writing
                     (destroys current contents)
+>>                Open for both reading & writing
                     (preserves current contents;
                      ready to write at end)
```

Die examples:

```
$tempfile = "temp.out";
open(TEMP, ">$tempfile") or die("Cannot open $tempfile\n");

o $! is the error string from the most recent operating system
error

$errorfile = "i.do.not.exist";
open(IN, "$errorfile") or die("Cannot open $errorfile : $!\n");
# Cannot open i.do.not.exist : No such file or directory

o if the error message to be printed does not include a
newline, Perl will automatically give the script name and line
number from which die was called

$errorfile = "i.do.not.exist";
open(IN, "$errorfile") or die("Cannot open $errorfile : $!;");
# Cannot open i.do.not.exist : No such file or directory; at
./die.pl line 80.
```

Upcasing a file (output to second file):

```
print("This script stores the contents of file1 in file2\n");
print("with all letters made uppercase.\n");
print("Enter file1 (the input file): ");
chomp($file = <STDIN>);
print("Enter file2 (the output file): ");
chomp($upper_file = <STDIN>);

open(INFILE, "<$file") or die("Cannot open $file for input : $!\n");
open(UPFILE, ">$upper_file")
  or die("Cannot open $upper_file for output : $!\n");

while(<INFILE>) {
    print UPFILE ("\U$_");
}

close(INFILE) or die("Cannot close $file : $!\n");
close(UPFILE) or die("Cannot close $upper_file : $!\n");
```

Upcasing a file (output to the same file):

```
print("This script changes the contents of the file entered\n");
print("making all letters uppercase.\n");
print("Enter a file name to be upcased: ");
chomp($file = <STDIN>);

# can't use +> because contents would be erased before we did
# anything with them
open(INOUTFILE, "+<$file")
  or die("Cannot open $file for modification: $!\n");

@in_array = <INOUTFILE>;

for($i=0; $i < @in_array; $i++) {
    $in_array[$i] = "\U$in_array[$i]";
}


# seek(FILEHANDLE, OFFSET, WHENCE)
# (OFFSET in bytes; WHENCE=0 means relative to beginning of file)
# positions file pointer back at start of INOUTFILE
seek(INOUTFILE, 0, 0);

print INOUTFILE (@in_array);

close(INOUTFILE) or die("Cannot close $file : $!\n");
```

Function example (return values):

```
sub max_of_x_and_y {
  if($x > $y) {
    return $x;
  } else {
    return $y;
  }
}
```

Function example (arguments):

```
sub say {
  print("$_[0], $_[1]!\n");
}

say("hello", "world");
```

Function example (more arguments and pass by reference):

```
 sub largest {
  if (@_ == 0) {
    print("Warning no parameters passed to largest\n");
    return undef;
  }
  $max = $_[0];
  for($i = 1; $i < @_; $i++) {
    if($_[$i] > $max) {
      $max = $_[$i];
    }
  }
  return $max;
}

sub pass_by_ref {
 for($i = 0; $i < @_; $i++) {
   $_[$i] = "hi";
 }
 return $max;
}

$a = largest(10,200,300,49);
print("$a\n");
@g = (10,200,300,49);
$a = largest(@g);
print("$a\n");

@g = (10,200,3,49);
$a = pass_by_ref(@g);
print("$a\n"); # 300 (global !)
print("@g\n"); # hi hi hi hi
$a = pass_by_ref(10,200,300,49);
print("$a\n");
# Modification of a read-only value attempted at ./largest.pl line 39
# nothing printed
```

MY operator (private variable – lexically scoped):

```
sub largest2 {
 # local variable to allow call by value
 my (@values) = @_;
 # local variable; any assignment will
 # not effect a global $max
 my ($max);
 if (@values == 0) {
   print("Warning no parameters passed to largest\n");
   return undef;
 }
 $max = $values[0];
 for($i = 1; $i < @values; $i++) {
   if($values[$i] > $max) {
     $max = $values[$i];
   }
 }
 return $max;
}
```

LOCAL function (semiprivate variable – dynamically scoped):

```
$value = "original";

tellme();
spoof();
tellme();

sub spoof {
  local ($value) = "temporary";
  tellme();
}

sub tellme {
  print("Value is $value.\n");
}

# prints
```

```
Value is original.
Value is temporary.
Value is orginal.
```

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;

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

```
 Taken from T H E   B A R E   B O N E S   G U I D E   T O   H T M L
                            by Kevin Werbach
              http://werbach.com/barebones/barebone.txt

SYMBOLS USED
----------------------------------------------------------------------------
URL    URL of an external file (or just file name if in the same directory)
?      Arbitrary number (i.e. <H?> means <H1>, <H2>, <H3>, etc.)
***    Arbitrary text (i.e. ALT="***" means fill in with text)
|      Alternatives (i.e. ALIGN=LEFT|RIGHT|CENTER means pick one of these)

GENERAL  (all HTML documents should have these)
----------------------------------------------------------------------------
       Document Type    <HTML></HTML>      (beginning and end of file)
       Title            <TITLE></TITLE>    (must be in header;
Netscape displays this at the top of the Netscape window)
       Header           <HEAD></HEAD>      (descriptive info, such as title)
       Body             <BODY></BODY>      (bulk of the page)

STRUCTURAL DEFINITION  (appearance controlled by the browser's preferences)
----------------------------------------------------------------------------
       Heading          <H?></H?>          (the spec. defines 6
levels; headings are displayed in larger and/or bolder
fonts than normal body text; H1 is largest)
       Emphasis         <EM></EM>          (usually displayed as italic)
       Strong Emphasis  <STRONG></STRONG>  (usually displayed as bold)
       Author's Address <ADDRESS></ADDRESS>
(See website for more)

PRESENTATION FORMATTING  (author specifies text appearance)
----------------------------------------------------------------------------
       Bold             <B></B>
       Italic           <I></I>
       Typewriter       <TT></TT>          (displays in a typewriter font)
       Preformatted     <PRE></PRE>        (display text spacing as-is)
       Center           <CENTER></CENTER>  (for both text and images)
(See website for more)
```

```
LINKS AND GRAPHICS
-----------------------------------------------------------------------------
       Link Something   <A HREF="URL"></A>
(example: <A HREF="http://www.cs.rpi.edu/~ziantzl/perl.hmtl"> Click Me </A>)
       Link to Target   <A HREF="URL#***"></A>  (if in another document)
                          <A HREF="#***"></A>     (if in current document)
       Define Target    <A NAME="***"></A>
(example: <A NAME="loc"> Some text to jump to </A>
          <A HREF="#loc"> Goto loc in same file </A>
)
       Display Image    <IMG SRC="URL">
(See website for more)

DIVIDERS
-----------------------------------------------------------------------------
       Paragraph        <P></P>            (skips lines before and
after text; closing tag often unnecessary)
       Line Break       <BR>               (a single carriage return)
       Horizontal Rule  <HR>               (like BR but draws a line)
(See website for more)

LISTS
------------------------------------------------------------------------------
(See website)

BACKGROUNDS AND COLORS
-----------------------------------------------------------------------------
(See website plus more info at <http://werbach.com/web/wwwhelp.html#color>)

SPECIAL CHARACTERS  (these must all be in lower case)
-----------------------------------------------------------------------------
(Complete list at <http://www.uni-passau.de/%7Eramsch/iso8859-1.html>)
```

Notes on CGI scripts:

```
     -- when the form is submitted, the browser tacks onto the end of
      the URL of the CGI program a QUERY STRING which consists of
      NAME=VALUE pairs; for example

        http://cgi.cs.rpi.edu/ziantl/test.cgi?field1=3&field2=4

      -- CGI is not just limited to forms, can write the following
      HTML code

Click <A HREF="http://www.place.org/cgi-bin/fortune.cgi">here</A> to
receive your fortune.

      -- the HTML document can also pass arguments to the CGI program
      directly
<A HREF="http://www.place.org/cgi-bin/fortune_or_date.cgi?fortune">
<A HREF="http://www.place.org/cgi-bin/fortune_or_date.cgi?date">
<!-- the user can select between getting a fortune or the date -->
```

Script to say hello (name comes from separate form):

```
#!/usr/local/bin/perl
# Personalized greeting cgi script
use strict;
use CGI qw(param);

print <<End_of_Start;
Content-type: text/html

<HTML>

  <HEAD>
    <TITLE>Hello World</TITLE>
  </HEAD>

  <BODY>
    <H1>Get ready for a hearty hello...</H1>
End_of_Start

my($name) = param("name");
print("    <P>HELLO \U$name\E!</P>");
print <<All_Done;
  </BODY>

</HTML>
All_Done
```

Same script using CGI.pm:

```
#!/usr/local/bin/perl
# hello cgi script with CGI.pm functions used
# to produce HTML
use strict;
use CGI qw(:standard);
print header(), start_html("Hello World");
print h1("Get ready for a hearty hello...");

my($name) = param("name");
print("    <P>HELLO \U$name\E!</P>");

print end_html();
```

CGI.pm Import Tags:

```
Tag          Methods Imported
----------   ----------------
:cgi         CGI parameter-handling methods
:form        Form generating methods
:html2       Methods to generate HTML 2.0 elements
:html3       Methods to generate HTML 3.0 elements
:netscape    Netscape-specific extensions
:shortcuts   html2 + html3 + netscape
:standard    html2 + form + cgi
:all         everything
```

Script to generate AND process a form (add.cgi):

```
#!/usr/local/bin/perl
#ex_19-1
#Learning Perl Appendix A, Exercise 19.1
use strict;
use CGI qw(:standard);
print header(), start_html("Add Me");
print h1("Add Me");
if(param()) {
    my $n1 = param('field1');
    my $n2 = param('field2');
    my $n3 = $n2 + $n1;
    print p("$n1 + $n2 = <strong>$n3</strong>\n");
} else {
    print hr(), start_form();
    print p("First Number:", textfield("field1"));
    print p("Second Number:", textfield("field2"));
    print p(submit("add"), reset("clear"));
    print end_form(), hr();
}
print end_html();
```

HTML generated by else portion of add.cgi [modified from original]:

```
<!-- header() -->
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<!-- start_html("Add Me") -->
<HTML>
  <HEAD>
    <TITLE>Add Me</TITLE>
  </HEAD>
  <BODY>
    <!-- h1("Add Me") -->
    <H1>Add Me</H1>
    <!-- hr() -->
    <HR>
    <!-- start_form() -->
    <FORM METHOD="POST"  ENCTYPE="application/x-www-form-urlencoded">
      <!-- p("First Number:", textfield("field1")) -->
      <P>
      First Number: <INPUT TYPE="text" NAME="field1" VALUE="">
      </P>
      <!-- p("Second Number:", textfield("field2")) -->
      <P>
      Second Number: <INPUT TYPE="text" NAME="field2" VALUE="">
      </P>
      <!-- p(submit("add"), reset("clear")) -->
      <P>
      <INPUT TYPE="submit" NAME="add" VALUE="add">
      <INPUT TYPE="reset" VALUE="clear">
      </P>
    <!-- end_form() -->
    </FORM>
    <!-- hr() -->
```

```
    <HR>
  <!-- end_html() -->
  </BODY>
</HTML>
```