**Programming in Perl**
**Lecture 6 Examples**

```
  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)

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)
```

```
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)

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:

```
      -- in a form, each input field has a name and an associated
      value (which the user has entered)

      -- the form itself is associated with a CGI program that
      processes the form input

      -- when the form is submitted, the browser creates a QUERY
      STRING consisting of NAME=VALUE pairs; it is either tacked on to
      the end of the URL of the CGI program or made available to the
      CGI script via STDIN, e.g.,

        http://cgi.test.org/test.cgi?field1=3&field2=4&resp=hello+there

      -- the CGI program takes these arguments, performs some
      processing, and usually returns HTML code to the server

      -- the format in which CGI program expects to receive its
      arguments from the browser via the sever is governed by the
      Common Gateway Interface specification

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

Click <A HREF="http://www.place.org/cgi-bin/fort.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/f_or_d.cgi?opt=fortune">
<A HREF="http://www.place.org/cgi-bin/f_or_d.cgi?opt=date">
<!-- the user can select between getting a fortune or the date -->
```

HTML form to enter a name:

```
<HTML>

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

  <BODY>
    <H1>What is your name?</H1>
    <HR>
    <FORM ACTION="hello2.cgi" METHOD="POST"  ENCTYPE="application/x-www-form-urlencoded">
      <P>
      Name: <INPUT TYPE="text" NAME="name" VALUE="">
      </P>
      <P>
      <INPUT TYPE="submit" NAME="submit" VALUE="submit">
      <INPUT TYPE="reset" VALUE="clear">
      </P>
    </FORM>
    <HR>
  </BODY>

</HTML>
```

Script to say hello (name comes from form given above):

```
#!/usr/local/bin/perl -w
# 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 -w
# 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!");

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
```

Basic HTML element generation:

```
Methods that generate begin/end sequences for HTML documents

Method                 Returns
--------------         ----------------
header()               "Content type: text/html\n\n"
start_html($string)    HTML source to start a document
                       (<HTML> <HEADER> <TITLE> $string </TITLE> </HEADER> <BODY>)
end_html()             HTML source to end a document
                       (</BODY> </HTML>)


Commonly used HTML element generating methods (may be placed between
start_html and end_html)

Method                 Returns
--------------         ----------------
p($string)             $string list in HTML paragraph (<P>$string</P>)
h1($string)            $string as HTML header 1 [largest] (<H1>$string</H1>)
br()                   an HTML line break (<BR>)
hr()                   an HTML horizontal rule (<HR>)
b($string)             $string set in HTML bold face (<B>$string</B>)
i($string)             $string set in HTML italic face (<I>$string</I>)

Note: $string can be replaced with a comma-separated list in most
cases.
```

HTML form generation:

```
Methods that generate begin/end sequences for HTML forms

Method                 Returns
--------------         ----------------
start_form()           HTML source to start a form (default
                       script to be executed: current cgi script,
                       default method: POST)
end_form()             HTML source to end a form
```

```
Reference list of methods that generate HTML form elements

Text Entry: textfield, textarea, password_field
Checkboxes: checkbox, checkbox_group, radio_group
Lists/Menus: popup_menu, scrolling_list
Buttons: submit, reset

(More details to come.)
```

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 + $n2 = ", b($n3), "\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>
```

The Class CGI

```
# Adapted from Lincoln D. Stein., How To Set Up and Maintain a Web Site
use CGI qw(:standard);

$query = CGI->new();
print $query->header();
print $query->start_html('Skeleton Script');

unless ($query->param()) {
   &display_prompt($query);
} else {
   &process_query($query);
}

print $query->end_html();
```

```
# --------------------- subroutines --------------------
# print the prompt
sub display_prompt {
    my ($query) = $_[0];
    print $query->h1("Skeleton Script");
    print $query->hr();
    print $query->start_form();
    print $query->p("Type something: ",$query->textfield('something'));
    print $query->submit();
    print $query->end_form();
}

# do the processing
sub process_query {
    my($query)=$_[0];

    my($value) = $query->param('something');
    print "The value returned is $value\n";
    print p(b("Your code here!")), "\n";
}
```

More Verbose Calling Sequences

```
      -- instead of

    print $query->p("Type something: ",$query->textfield('something'));

           we could have written

    print $query->p("Type something: ",
                    $query->textfield( -NAME => 'something' ));

      -- the -NAME is called a named parameter; what follows the => is
called a parameter value

      -- these types of actual argument lists allow arguments to be
passed in any order while being more descriptive

      -- of course you have to know the proper literal names to use
named parameters
```

Example of Named Calling Sequence

```
print $query->scrolling_list(
        -NAME => "flavors",
        -VALUES => [ qw(mint chocolate cherry vanilla peach) ],
        -LABELS => {
           mint => "Mighty Mint",
           chocolate => "Cherished Chocolate",
           cherry => "Cheery Chery",
           vanilla => "Very Vanilla",
           peach => "Perfectly Peachy",
         },
        -SIZE => 3,
        -MULTIPLE => 1,
      );
```

```
     -NAME : string which can be used to retrieve user data from the
form with param()

     -VALUES : a reference to an array which contains a list of the
possible values passed back to the CGI script (accessible using
param() with the appropriate name string)

     -LABELS : a reference to a hash which relates the values defined
above to the labels actually displayed to the user in the scrolling
list (optional)

     -SIZE : an integer giving the number of list items displayed to
the user at one time (optional)

     -MULTIPLE : set to 1 (true) to allow the user to choose more than
one item (false means the user can select only one item)

     -DEFAULT : [not used here] selects values that are highlighted as
already selected when the user first looks at the list (optional)
```

Instead on anonymous hash...

```
%flavors = (
           mint => "Mighty Mint",
           chocolate => "Cherished Chocolate",
           cherry => "Cheery Chery",
           vanilla => "Very Vanilla",
           peach => "Perfectly Peachy",
);

print $query->scrolling_list(
        -NAME => "flavors",
        -LABELS => \%flavors,
        -VALUES => [ keys %flavors ],
        -SIZE => 3,
        -MULTIPLE => 1,
      );
```

Example List of Widgets and Parameters

```
  -- TEXTFIELD

    o prompts for a single line of text input

print $query->p("Name:",$query->textfield(
                           -NAME => 'something',
                           -DEFAULT => 'unknown',
                           -SIZE => 20,
                           -MAXLENGTH => 80
                        )
               );

-NAME : as for scrolling list
-DEFAULT : starting value displayed in field (optional)
-SIZE : size of the field in characters (optional)
-MAXLENGTH : maximum number of characters accepted (optional)

  -- PASSWORD_FIELD

    o as textfield (characters entered appear as stars)

print $query->p("Password:",$query->password_field(
                            -NAME => 'secret',
                            -VALUE => 'swordfish',
                            -SIZE => 20,
                            -MAXLENGTH => 80
                        )
               );

-NAME : as for scrolling list
-VALUE : starting value displayed in field, as textfield -DEFAULT (optional)
-SIZE : size of the field in characters (optional)
-MAXLENGTH : maximum number of characters accepted (optional)

  -- TEXTAREA

    o prompts for text input, allowing multiple lines

    o displays a rectangular box to enter text into

print $query->p("Anything to say for yourself?", br(),
                $query->textarea( -NAME => 'comments',
                                  -DEFAULT => "A while back...",
                                  -ROWS => 10,
                                  -COLUMNS => 50
                                )
                );

-NAME : as for scrolling list
-DEFAULT : as for textfield (optional)
-ROWS : height in number of lines the text area  will occupy
-COLUMNS : width in number of characters  the text area  will occupy
```

```
      -- STANDALONE CHECKBOXES

        o creates an isolated checkbox

        o user can check the box or leave it unchecked

    print $query->p($query->checkbox( -NAME => 'sample',
      -CHECKED => 1,
      -VALUE => 'YES',
      -LABEL => "Free sample"
    )
    ), "\n";

    -NAME : as for scrolling list
    -CHECKED : 1 if the box should start as checked (optional)
    -VALUE : value of the checkbox when checked; default is 'on'
(optional)
    -LABEL : label to be attached to the box for the user to see ;
default is the name of the box (optional)


      -- CHECKBOX GROUP

        o creates a list of logically related checkboxes

        o any number can be checked or unchecked at once

    print $query->p($query->checkbox_group(
                        -NAME => 'operating_systems',
                        -VALUES => [ 'IRIX', 'AIX', 'Solaris', 'Sun OS' ],
                        -DEFAULT => [ 'AIX' ],
      -LINEBREAK => 1,
 )
   ), "\n";

        o note: param('operating_systems') returns a list of values

    -NAME : as for scrolling list
    -VALUES : as for scrolling list
    -DEFAULT : reference to a list containing values to be checked at
start or a single value to be checked; default is that nothing is
selected initially (optional)
    -LINEBREAK : set to 1 if boxes are to be displayed on separate
lines; default is single line (optional)
    -LABELS : [not used above] as for scrolling list (optional)
```

```
  -- RADIO GROUP

    o creates a list of logically related radio buttons

    o only one radio box can be selected at a time

print $query->p($query->radio_group(
                 -NAME => 'monty_hall',
                 -VALUES => [ "a new car", "a fabulous trip", "a donkey" ],
                 -DEFAULT => '-',
                 -LINEBREAK => 1,
 -LABELS => {
                    "a new car" => "door #1",
                    "a fabulous trip" => "door #2",
                    "a donkey" => "door #3",
},
                                  )
               ), "\n";


-NAME : as for scrolling list
-VALUES : as for scrolling list
-DEFAULT : name of default button to turn on; specify non-existent
button to have none turned on
-LINEBREAK : as for checkbox group (optional)
-LABELS : as for scrolling list (optional)


  -- POPUP MENU

    o creates a popup menu the user can select from

    o only one menu item can be selected at a time

print $query->p($query->popup_menu(
                 -NAME => 'pick',
                 -VALUES => [ "eenie", "meenie", "minie", "moe" ],
                 -DEFAULT => 'minie',
 )
), "\n";

-NAME : as for scrolling list
-VALUES : as for scrolling list
-DEFAULT : name of default button to turn on; default is first
item
-LABELS : [not used above] as for scrolling list (optional)
```

```
   -- SUBMIT

     o creates button used to submit the form

     o every form should have one

  -- RESET

     o creates a button to reset the form

     o every form should have one

    print p($query->submit( -NAME => "submit",
                            -VALUE => "go" ),
            $query->reset( -NAME => "clear" )
           ), "\n";

    -NAME : as for scrolling list (also used to label button)
    -VALUE : value returned to CGI script; overrides label given by
name on button [submit only] (optional)
```

To display current settings of named fields from a form:

```
sub display_values {
    my($query)=$_[0];

    my(@values, $key);

    print h2("Here are the current settings:"), "\n";
    print hr();

    foreach $key ($query->param()) {

        print b("$key ->");
        @values = $query->param($key);

        print (join("; ", @values), br(), "\n");
    }

}
```

To display a simple error page :

```
# Note that this routine assumes that header() and start_html() have already been called
sub bail {
    my($error) = "@_";

    print h1("Unexpected Error"), "\n";
    print p($error), "\n";
    print end_html(), "\n";;
    die($error);
}
```

Locking Files

```
use Fcntl qw (:flock);
# variables imported from Fcntl
# LOCK_SH -- value for concurrent read lock
# used when file is open for reading only
# LOCK_EX -- value for exclusive write lock
# used when file is open for writing

# we will talk about bail soon
open(INFILE, "<$produ_filename")
  or &bail("Cannot open file $product_filename for input : $!");
# concurrent read lock after opening for read
# anyone can still read; no one can write
flock(INFILE, LOCK_SH) # there is no $ in front of LOCK_SH
  or &bail("Cannot lock file $prod_filename for reading : $!");

# do stuff

# closing unlocks the file
close(INFILE)
  or &bail("Error in closing file $prod_filename for input : $!");


open(OUTFILE, ">$orders_filename")
  or &bail("Cannot open file $orders_filename for output : $!");
# exclusive write lock after opening for write
# no one can write or read
flock(OUTFILE, LOCK_EX) # there is no $ in front of LOCK_EX
  or &bail("Cannot lock file $orders_filename for writing : $!");
# do stuff

# closing unlocks the file
close(OUTFILE)
  or &bail("Error in closing file $orders_filename for input : $!");
```

Guestbook example CGI script:

```
use strict;

use CGI qw(:standard);
use Fcntl qw (:flock);
use CGI::Carp qw(fatalsToBrowser);

my (
    $CHATNAME, # name of guestbook file
    $MAXSAVE,  # how may entries to keep
    $TITLE,    # page title and header
    $cur,     # new entry in guestbook
    @entries,  # all current entries
    $entry     # a particular entry
    );
```

```
$TITLE = "Simple Guestbook";
$CHATNAME = "chatfile";
$MAXSAVE = 2;

print header(), start_html($TITLE), h1($TITLE);

$cur = CGI->new();
if($cur->param("message")) {
    $cur->param("date", scalar(localtime())); # set to current time
    @entries = ($cur); # initializes an array of pointers
}

open(CHANDLE, "+<$CHATNAME") or bail("cannot open $CHATNAME: $!");

flock(CHANDLE, LOCK_EX) or bail("cannot lock $CHATNAME: $!");

# grab up to $MAXSAVE old entries, newest first
while ( !eof(CHANDLE) && (@entries < $MAXSAVE) ) {
    $entry = CGI->new(\*CHANDLE); # create a new entry read from the file
    push (@entries, $entry);
}

# eliminate old entries from the file
seek(CHANDLE, 0, 0) or bail("cannot read $CHATNAME: $!");
foreach $entry (@entries) {
    $entry->save(\*CHANDLE); # save the most recent entries to the file,
                             # with the newest one at the start
}
truncate(CHANDLE, tell(CHANDLE)) or bail("cannot truncate $CHATNAME: $!");
close(CHANDLE) or bail("cannot close $CHATNAME: $!");

print hr(), start_form();
print p("Name:", $cur->textfield ( -NAME => "name" ) );
print p("Message:", $cur->textfield( -NAME => "message",
     -OVERRIDE => 1,
     -SIZE => 50 ) );
print p(submit("send"), reset("clear"));
print end_form(), hr();

print h2("Prior Messages");
foreach $entry (@entries) {
    printf("%s \[%s]: %s", $entry->param("date"),
                  $entry->param("name"),
                  $entry->param("message") );
    print br();
}

print end_html();
```