Programming in Perl, Summer 98
Project 2
Due : Friday, June 12 at midnight
Weight : 20% of course grade
Instructor : Moorthy

This assignment will allow you to experiment with creating and running a CGI script using Perl. It should be submitted according to the *Submission Guidelines*. Before handing it in, make sure it works using `netscape` running on an IBM workstation. Do not forget to comment your program, providing a general description of the Perl script and any subroutines used, along with inline comments for important or complex portions of code. Please get started on the assignment as soon as possible.

1 Motivation

You have been hired by a small software company. They would like to make some of the CGI scripts they have developed available from their web page. Your first project is to develop a simple Perl script that meets the specifications detailed in this document. Your initial script will eventually be refined into a full-fledged shopping cart application.

2 Specification

They would like a single script that both generates an ordering form and processes a user's order.

2.1 Form Generation

Products are stored in a file called catalog.dat. A line of the file has the following format:

```
{product-code}:{product-name}:{commerical-price}:{noncommercial-price}:{product-description}
```

For example,

```
C05:Net Chat:50:20:New version; can handle any number of visitors!
```

Product codes consist of three characters and are unique. You will first need to read and store this data in some reasonable data structures (perhaps separate hashes for product names, commercial prices, noncommercial prices, and descriptions, each using product code as keys, or separate arrays for each data field).

The `<TITLE>` of the order page should be `Order Form for New Customers`. The page itself should consist of the following:

- A `<H1>` heading that reads `Order Form for New Customers`.
- A horizontal rule (`<HR>`).
- Text that indicates how the user should select scripts, that the form must be completely filled out to process an order, and that a username and password must also be provided.
- A horizontal rule (`<HR>`).
- A `<H2>` heading that reads `Please select the scripts you wish to purchase and the number of licenses:`. This is followed by a series of product listings, each separated by a `<P>`. The listings do not have to be displayed in any particular order.
  - Each listing should start with a checkbox. Use the product code as the `-NAME` of the checkbox. The box should not be checked at the beginning.

- The label `Item:` should appear in bold on the same line as the checkbox. The label should be followed by the product name.
- The label `Commercial price:` should appear next in bold on the same line. It should be followed by the commercial price of the product (the price should include a dollar sign).
- The label `Noncommercial price:` should appear next in bold on the same line, followed by the noncommercial price of the product (the price should include a dollar sign).
- The label `Quantity:` should appear in bold on the same line as the checkbox, followed by a textfield of width 3 with an default value of 1. I would suggest that each textfield be given a name that can be constructed from the corresponding product code. Since product codes are of fixed length, you can simply prepend a `'Q'` to the product code, for example.
- On the next line (emit a `<BR>`), the product description should appear in italics.

- A horizontal rule (`<HR>`).
- A `<H2>` heading that reads `Contact Information:`.
- An appropriate label and a textfield that allows the user to enter a name (set the width of the field to 40).
- An appropriate label and a textfield that allows the user to enter a street address (set the width of the field to 40).
- An appropriate label and a textfield that allows the user to enter a city (set the width of the field to 25). On the same line, a label and a popup menu with postal abbreviations for all 50 states followed by another label and a textfield that allows the user to enter a zip code (set the width of the field to 9 and make this the maximum number of characters that can be entered).
- A bold label `Intended use for script:` followed by a radio group list consisting of two options — Commercial and Noncommercial. Both options should be on the same line. Neither box should be checked at the beginning.
- A horizontal rule (`<HR>`).
- Text indicating that the user should now choose a username and password that will be used to access the scripts once payment is received. Point out that they should enter the password twice.
- An appropriate label and a textfield that allows the user to enter a username.
- An appropriate label and a password field that allows the user to enter a password.
- An appropriate label and a password field that allows the user to re-enter the password.
- A horizontal rule (`<HR>`).
- A submit button titled `Submit Order` and a reset button titled `Reset Order`.

2.2 Form Processing

2.2.1 Validating User Input *(Bonus — 15 points)*

Before any processing occurs, your code should verify that the contact information portion of the form was completely filled out. If the name, address, city, zip code, username or password fields have no data or if none of the "intended use" boxes were clicked by the user, an error page should be generated. The error page should indicate which field was not completed by the user (if more than one is blank, you simply need to issue a message about the first blank field encountered in your tests). In addition, if the original password entered doesn't match the password re-entered by the user, a different error page should be displayed indicating that the two passwords were not the same.

After verifying this data, the product file must be read as in the form generation stage.

The script should then determine whether anything was ordered by the user. If none of the boxes were checked or quantities less than 1 were provided for those boxes that were checked, an error page should be generated indicating that no items were ordered.

Notes on error pages:

- any error page generated should inform the user that he or she may use the Back option on a browser to go back to the form and rectify any mistakes.
- if an error page is generated, no data should be written to orders.dat (see Section 2.2.2).

2.2.2 Processing an Order

(Note that even if you are not doing the bonus part, you will need to read the product file as in the form generation stage.) A valid order should be appended to a file called orders.dat. An order consists of several lines written out to the file with the following format:

```
{name}
{street-address}
{city}:{state}:{zip-code}
{intended-use}
{username}
{encrypted-password}
{order-line-1}
...
{order-line-n}
%%
```

Notice that city, state, and zip code are separated by colons. Intended use is either Commercial or Noncommercial. You can encrypt the password by calling the Perl builtin function `crypt`. Use the text to be encrypted as the first argument, and pass the string `"bx"` as the second. A person's order ends with two percent signs. An order line is of the following form:

```
{product-code}:{quantity}
```

The script should then generate a page containing a purchase summary for the user. The `<TITLE>` of the page should be `Order Summary for {name}` where `{name}` is the name input by the user. It should have the following format:

- A `<H1>` heading `Order Summary for {name}` where `{name}` is the name input by the user.
- A horizontal rule (`<HR>`).
- A `<H2>` heading that reads `Contact Information:`.
- An appropriate label in bold and the user's street address.
- The user's city, state, and zip code with appropriate bold labels on the same line.
- A horizontal rule (`<HR>`).
- A `<H2>` heading that reads `Other Information:`.
- An appropriate label in bold and the user's intended use for the software.
- An appropriate label in bold and the user's username.
- The text `Password successfully logged.` in bold.

- A horizontal rule (<HR>).
- A <H2> heading that reads `Purchase Information:`.
- One or more lines indicating the items ordered. Each line should consist of the product name, the quantity, and the item's price (commercial or noncommercial) with appropriate bold labels.
- An appropriate label and the total number of items ordered.
- An appropriate label and the total due. Note that total due is computed based on the number of items ordered and the price of each item (either commercial or noncommercial based on the intended use of the software).

3 Details

- Name the script cart.cgi.
- Use the module CGI.pm (`use CGI qw(:standard);`).
- Include the pragma strict (`use strict;`).
- Use the module CGI::Carp.pm (`use CGI::Carp qw(fatalsToBrowser);`)
- Do not forget to call the appropriate `start_*` and `end_*` when starting/ending an HTML page or form.
- You may assume the initial data files are error free. I will e-mail the class when sample data files are made available on the course web page (`http://www.cs.rpi.edu/~ziantzl/perl.html`).
- Assume any username entered by the user does not already exist.
- You will need to lock catalog.dat for concurrent read (no write) when you read it for input.
- You will need to lock orders.dat for exclusive write (no read) when you write to it.
- If you do the bonus part, append the string `"-- Bonus"` in the <TITLE> and <H1> heading of the generated form.
- Make sure you display an HTML error message whenever there is an error that you can trap (for example, if a file can't be opened or locked). You can use the `&bail()` subroutine I gave in class or make one of your own.
- Make use of subroutines. The generation and processing of the form should be separate subroutines. The form processing subroutine should call several subroutines. Other activities, such as reading the catalog file and generating the different HTML error pages are all candidates for separate subroutines.
- Keep the use of global variables to a minimum. Global "constants" such as filenames are fine. It is acceptable to make the product data structures global (but you should at least experiment with passing a hash or an array by reference, since it may be on the exam).

4 Offline Execution

Most of the development of your CGI script should be done in this mode of execution. To get offline execution, you simply run the CGI script as a normal Perl script on, for example, an IBM workstation. Suppose you execute the script add.cgi. You will get the following prompt:

```
(offline mode: enter name=value pairs on standard input)
```

If you hit Ctrl-D at this point, the script should produce the HTML corresponding to the form that the user will use to enter input. As long as you use the routines `header()`, `start_html()`, `start_form()`, `end_form()`, and `end_html()` with HTML emitting functions placed between `start_html()` and `end_html()` and HTML form widgets only placed between `start_form()` and `end_form()`, the HTML produced should be valid (if not aesthetically pleasing when printed). However, this is a good way to check for run-time warnings and other problems. Continuing the above example, a Ctrl-D would give us

```
Content-type: text/html

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

To make your script execute as if it received user input, enter the `-NAME` that you gave each widget followed by an equals sign and a test value you would like it to have. Note that giving a widget a name containing spaces will work in online mode but will be a problem in offline mode. If the value has spaces in it, like *hello there*, it should be enclosed in quotes. When you are done with the list, type Ctrl-D. For add.cgi, we could enter

```
field1=3 RETURN
field2=4 RETURN
Ctrl-D
```

The script would emit:

```
Content-type: text/html

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>Add Me</TITLE>
</HEAD><BODY><H1>Add Me</H1><P>3 + 4 = <B>7</B>
</P></BODY></HTML>
```

Again, it could also generate Perl warnings or programmer-defined error messages.

What if you have a widget that allows the user to select several values to return to the CGI script (for example, a `checkbox_group`)? Then you simply repeat the widget-name-equals-value combination for each value:

```
operating_systems=AIX RETURN
operating_systems='Sun OS' RETURN
Ctrl-D
```

Instead of typing in the data each time, you can create several files of the proper format and then run the script by redirecting standard input to come from a file. For example, if we have a file called input.batch, with the following contents:

```
N03=yes
G11=yes
QN03=1
QG11=3
name="Krishna Moorthy"
address="CS Dept, RPI"
city=Troy
state=NY
zip=12180
comm_stat=Noncommercial
username=krishm
password=youkilledkenny
password2=youkilledkenny
```

we can run cart.cgi as `cart.cgi < input.batch`.

5 Enabling Online CGI Execution

To run a CGI script on the server, you will need to take the following steps:

1. starting at your home directory, create a directory called **public_html**;
2. `cd` to **public_html** and create a directory called **cgi-bin**;
3. `cd` to **cgi-bin** and type

   ```
   fs sa . system:anyuser rlikw
   ```

   The `rl` (read and lookup) privileges are needed to run any CGI script. The `ikw` (write and lock) privileges will be needed because you will be creating, locking, and writing to a file from a CGI script for Project 2. You should only copy your script to cgi-bin when you want to do online testing. **When testing is done, you should remove it from cgi-bin and set the directory's privileges back to `rl` or less.**
4. make sure the Perl script is contained in a file that ends with the extension `.cgi`;
5. make the CGI script executable (for example, `chmod a+x script.cgi`). Note the `a+x`. It must be executable by everyone, not just you.

Once this is complete, the script may be run as described in the following section.

6 Online Execution

Bring up `netscape`. Open the URL

`http://cgi.cs.rpi.edu/rcs/{your-rcs-id}/{your-script-prefix}.cgi`

For example, if I wanted to run a script called add.cgi, I would open

`http://cgi.cs.rpi.edu/rcs/krishm/add.cgi`

Your script should now execute, most likely generating a form for the user to enter input. Once the user submits the form, your script will produce another HTML page in response which will be displayed automatically. If you develop your script on a different CGI server, make sure that it works correctly using the above URL.

7 Script Errors

If you are using the module CGI::Carp.pm and running in *online mode*, fatal errors generated by Perl will be displayed by your browser, for example:

```
Software error:

Undefined subroutine &main::foo called at /usr/local/www/rcs/krishm/Test/cart.cgi line 14.

Please send mail to this site's webmaster for help.
```

At this point, you should go back to *offline mode* to fix the problem before running in *online mode* again. Note that Perl warning messages will not be displayed in *online mode*.

Suppose you are running in *online mode* and receive the following message or something like it:

```
Internal Server Error

The server encountered an internal error or misconfiguration and was
unable to complete your request.

Please contact the server administrator, www@cs.rpi.edu and inform
them of the time the error occurred, and anything you might have
done that may have caused the error.
```

In this case, you 1) are probably not using the module CGI::Carp.pm and 2) should go back to *offline mode* and test your script again thoroughly. You will probably find that you have an error. If everything seems to run fine offline, but you still get a server error message online, please contact me.

In either instance, **please do not contact** `www@cs.rpi.edu`**. Take note that this is not an RCS server, so sending mail to anyone at ACS will not be productive.**