Writing Perl Programs using Control Structures
Worked Examples

Louise Dennis

October 27, 2004

These notes describe my attempts to do some Perl programming exercises using control structures and HTML Forms. They include notes on how I constructed the algorithms and on the mistakes I made. I hope they will be useful.

1 Processing an HTML Form

The Exercise Write a web page that asks someone to enter their name and then echos this back to them. You should use perl and CGI in your answer.

1.1 Writing the Web Page

I can write a simple web page quite easily. I don't know how the data entry bit will work yet, but I can write the stuff that goes around it. In fact I will start with the web page in the notes from the last lecture (Introduction to Perl)

```
<HTML><HEAD>
<TITLE>output of HTML from CGI script</TITLE>
</HEAD><BODY>
<H1>Sample output</H1>
What do you think of <STRONG>this?</STRONG>
</BODY></HTML>
```

In the notes this was generated by a program but I know it works fine in a browser so I'll just enter it into emacs as a new file called `form1.html`.

Now if I look at the bit in today's lecture notes on forms (Slide 18) I find that this needs to be in my `public_html` directory. I already have one of these, but you may need to create one to get the example to work. So I typed the following:

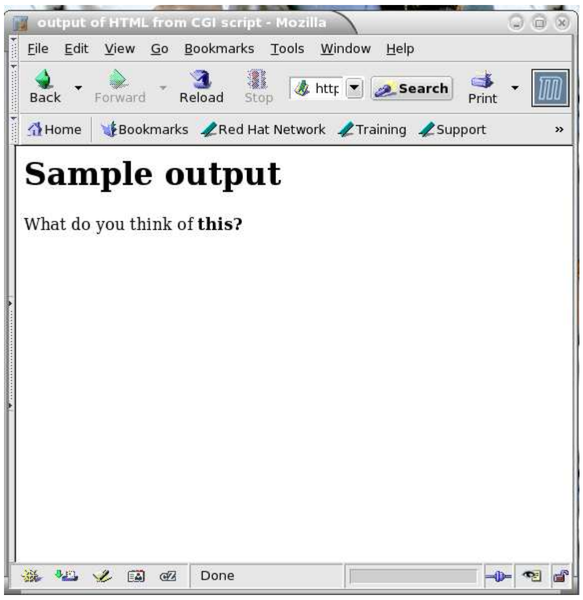
```
$ cd public_html/
$ emacs form1.html &
```

I want to check that I can see this in my browser before I do anything else so I open up Netscape and go to the URL

```
http://www.cs.nott.ac.uk/~lad/form1.html
```

If you are doing this exercise then you would need to substitute your username where I have written `lad`.

This produced the page shown in figure 1


output of HTML from CGI script - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop Search Print
Home Bookmarks Red Hat Network Training Support
Done

Sample output

What do you think of **this?**

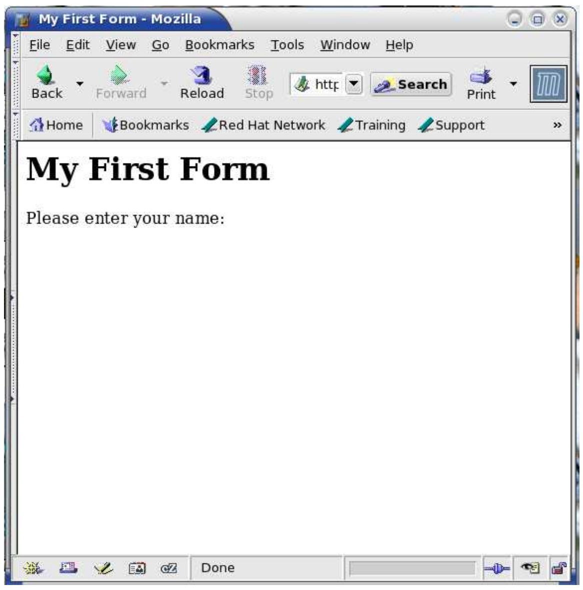
Figure 1: My First Web Page

So that was fine, but obviously the content was not what I wanted. So I edited it to

```
<html>
  <head>
    <title>My First Form</title>
  </head>

  <body>

<H1>My First Form</H1>
Please enter your name:

  </body>
</html>
```

and checked it again on the web as shown in figure 2.

Now to start the serious stuff. Looking at page 146 in the text book there is information on creating HTML forms – in particular in creating a text input field and


My First Form - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop Search Print
Home Bookmarks Red Hat Network Training Support
Done

My First Form

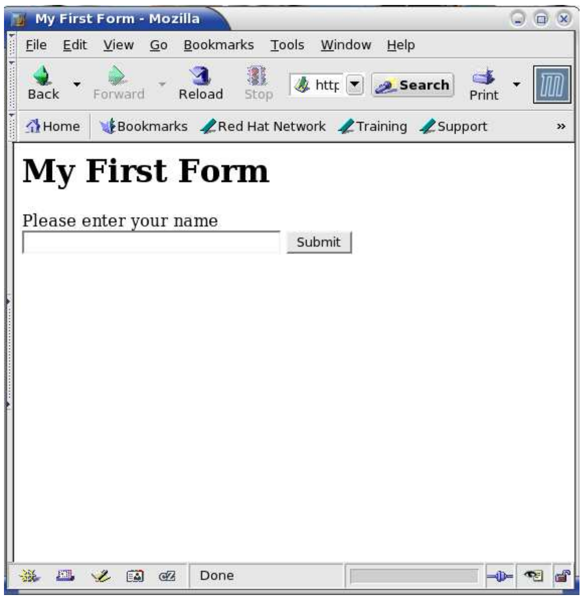
Please enter your name:

Figure 2: My First Web Page

a submit button which is what I want. I copy this stuff into my web page. I leave a question mark in the form `ACTION` – I'll worry about this in a minute.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>My First Form</title>
  </head>

  <body>

<H1>My First Form</H1>
Please enter your name

<FORM ACTION=? METHOD="POST">
<INPUT TYPE="text" NAME="username" SIZE=30 MAXLENGTH=60>
<INPUT TYPE="submit" VALUE="Submit">
</FORM>

  </body>
</html>
```

This appears as shown in figure 3 in my browser (it still doesn't do anything but at least it looks right.


My First Form - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop Search Print
Home Bookmarks Red Hat Network Training Support
Submit
Done

My First Form

Please enter your name

Figure 3: My First Form

So how do I get this to do something? Well looking again at the textbook on page 145 it says that `ACTION` should be the URL of the perl file to be executed. Now I remember from the previous lecture that this URL as a bit different for CGI scripts. I look this up – it's in the first set of perl slides on slide 11. It says the URL should be

```
http://robin.cs.nott.ac.uk/˜lad/cgi-bin/filename.pl
```

I shall call my script `form1.pl` so that means I edit my web page to

```
<html>
  <head>
    <title>My First Form</title>
  </head>

  <body>

<H1>My First Form</H1>
Please enter your name

<FORM
      ACTION="http://robin.cs.nott.ac.uk/˜lad/cgi-bin/form1.pl"
      METHOD="POST">
<INPUT TYPE="text" NAME="username" SIZE=30 MAXLENGTH=60>
<INPUT TYPE="submit" VALUE="Submit">
```

```
</FORM>

  </body>
</html>
```

1.2 Creating the CGI Script

So now for the CGI script itself. I change directory into my `cgi-bin` directory (where this script must live according to the notes) and open a new file `form1.pl`.

Page 153 tells me how to get a name and a value from the form so I copy that into my script as a start

```
#!/usr/local/bin/perl

use CGI qw(:standard :html3);

my ($name, $fruit) = (param("name"), param("fruit"));
```

This is for a form with two inputs given the `NAME`s `name` and `fruit` respectively. When I look at my HTML page I've got one input and set `NAME` to `username`. So my code becomes:

```
#!/usr/local/bin/perl

use CGI qw(:standard :html3);

my $username = param('username');
```

If I understand everything correctly `$username` should now hold the value entered by the user. What next? I want to print this back to the user. Going back to the first set of perl notes, slides 12 and 13, I see how to print stuff to the web page. It seems I just use regular print statements but I should include the line

```
print "Content-type: text/html\n\n";
```

first, so my code becomes

```
#!/usr/local/bin/perl

use CGI qw(:standard :html3);

my $username = param('username');

print "Content-type: text/html\n\n";

print $username;
```

Now to test the program I do this in figures 4 and 5.

This isn't very good. I've got an error. So I have to check the error logs

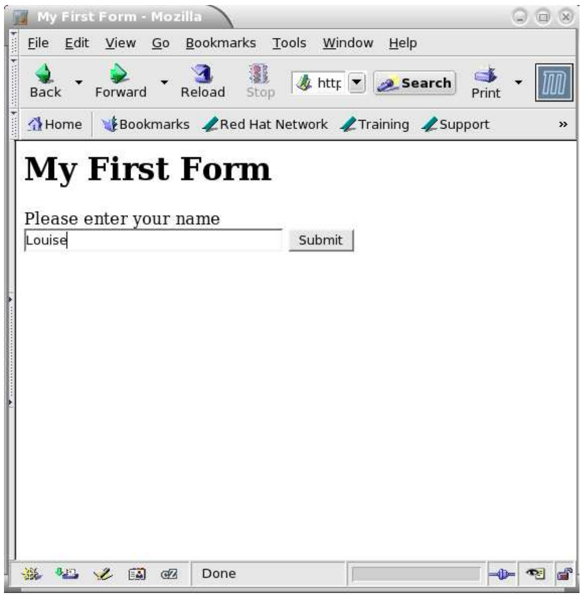

My First Form - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop Search Print
Home Bookmarks Red Hat Network Training Support
My First Form
Please enter your name
Louise
Submit
Done

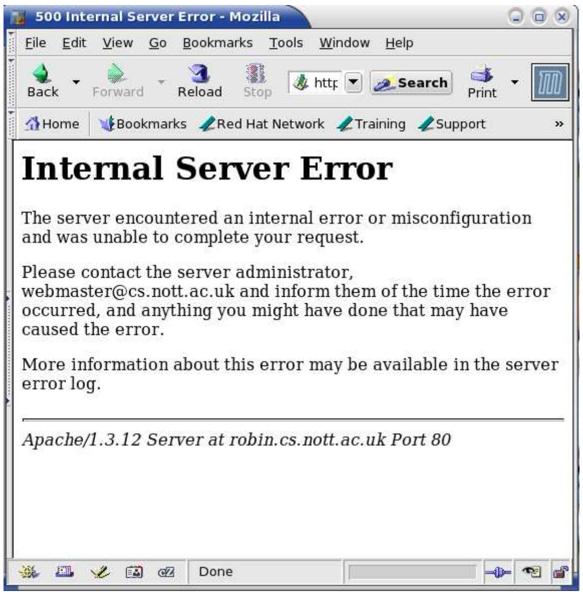
Figure 4: My First Form

```
robin$ tail /usr/local/apache/logs/error_log
Printer het not responding
couldn't find HOME environment variable to expand path
[Wed Nov 13 12:47:06 2002] [error] [client 160.80.34.237] client denied by se
[Wed Nov 13 12:49:26 2002] [error] [client 160.80.34.237] client denied by se
[Wed Nov 13 13:02:03 2002] [error] [client 213.45.56.219] Client sent malform
couldn't find HOME environment variable to expand path
Printer het not responding
couldn't find HOME environment variable to expand path
Printer het not responding
[Wed Nov 13 16:50:54 2002] [error] [client 128.243.20.243] Premature end of s
```

Right at the end there I have a message `Premature end of script headers` which isn't terribly enlightening. So I do some standard checks. First I check that both the perl script and the HTML file have been saved recently and that the URL in the HTML file is correct. Then I look in `cgi-bin` to see if the perl script is executable.

```
robin$ ls -lt
total 49
-rw-r--r--   1 lad           139 Nov 13 16:49 form1.pl
```

The lack of any `x` symbols means its not executable. This is one of the most basic errors and the fact that I made it means even experienced programmers can forget this.


500 Internal Server Error - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop Search Print
Home Bookmarks Red Hat Network Training Support
Done

Internal Server Error

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

Please contact the server administrator, webmaster@cs.nott.ac.uk and inform them of the time the error occurred, and anything you might have done that may have caused the error.

More information about this error may be available in the server error log.

Apache/1.3.12 Server at robin.cs.nott.ac.uk Port 80

Figure 5: My First Form Output

```
robin$ chmod u+x form1.pl
```

So I make it executable. Then, for good measure, because I don't want any of you looking at it

```
robin$ chmod a-r form1.pl
robin$ chmod u+r form1.pl
```

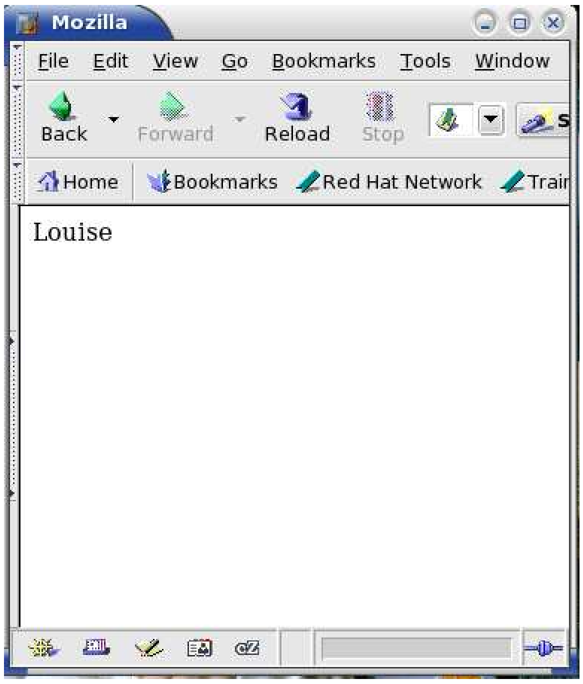
Now when I check the permissions I get

```
robin$ ls -l1
total 49
-rwx------   1 lad           139 Nov 13 16:49 form1.pl
```

I retry the form and get the output shown in figure 6

Now I know I ought to be printing HTML to the screen not just plain text. So I'll try putting in some correct HTML information copied pretty much from my web page which I know displays properly.

```
#!/usr/local/bin/perl

use CGI;
$cgi=new CGI;
$username = $cgi->param('username');
```


Mozilla
File Edit View Go Bookmarks Tools Window
Back Forward Reload Stop
Home Bookmarks Red Hat Network Trair

Louise

Figure 6: My First Form Output

```
print "Content-type: text/html\n\n";

print "<html><head><title>Returning User Name</title></head>";
print "<body>";
print $username;
print "</body></html>";
```

and I try again (shown in figure 7)

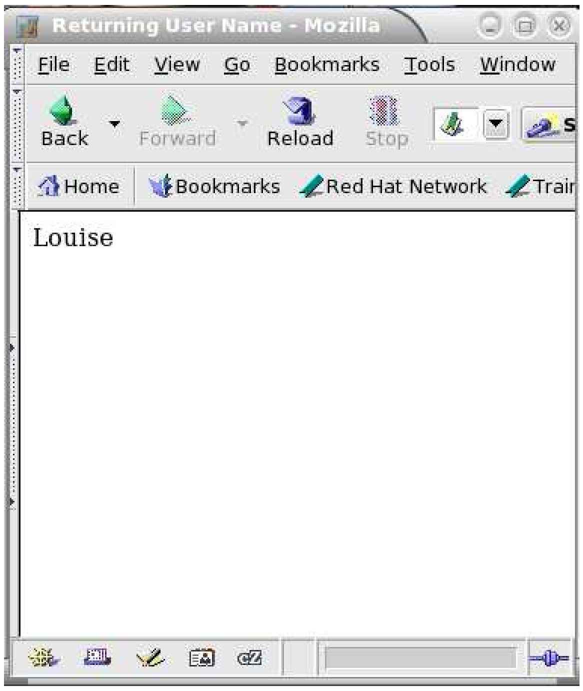
Almost the same but I've now got the title displaying at the top of the browser window.

2 Using Some Control Structures

The Exercise Write a web page that asks someone to enter their favourite number. It then prints out all the numbers from zero to that number.

2.1 The Web Page

I reckon I can do this by copying my first web page and then editing it slightly


Returning User Name - Mozilla
File Edit View Go Bookmarks Tools Window
Back Forward Reload Stop
Home Bookmarks Red Hat Network Trair

Louise

Figure 7: My First Form Output

```
$ cp form1.html form2.html
$ emacs form2.html &
```

This is very simple. I change all the ones to twos (because I'm now on the second exercise) and `name` to `number`

```
<html>
  <head>
    <title>My Second Form</title>
  </head>

  <body>

<H1>My Second Form</H1>
Please enter your number

<FORM
      ACTION="http://robin.cs.nott.ac.uk/~lad/cgi-bin/form2.pl"
      METHOD="POST">
<INPUT TYPE="text" NAME="usernumber" SIZE=30 MAXLENGTH=60>
```

```
<INPUT TYPE="submit" VALUE="Submit">
</FORM>

  </body>
</html>
```

2.2 The Perl Script

It occurs to me I can do the same with the script

```
$ cd
$ cd cgi-bin
$ cp form1.pl form2.pl
```

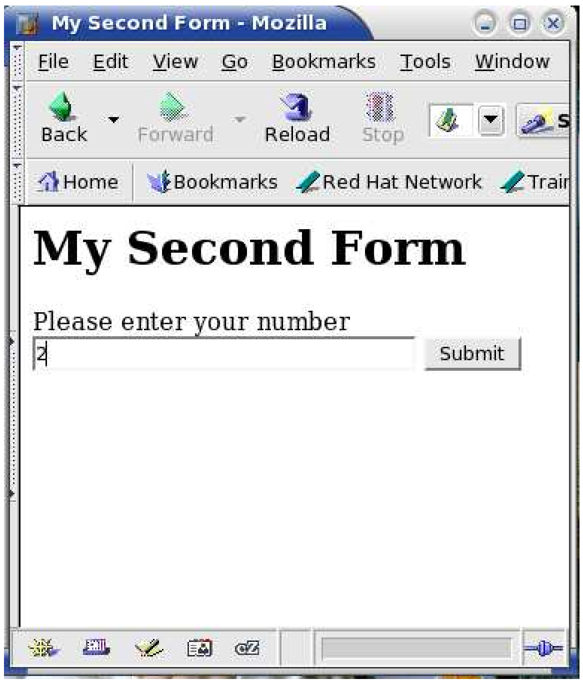
and I edit this the same way

```
#!/usr/local/bin/perl

use CGI;
$cgi=new CGI;
$usernumber = $cgi->param('usernumber');

print "Content-type: text/html\n\n";

print "<html><head><title>Returning User Number</title></head>";
print "<body>";
print $usernumber;
print "</body></html>";
```

Now this will only echo back the number at the moment, but I decide to test it anyway to make sure all the HTML and CGI bits are working (this is shown in figures 8 and 9). Excellent! and I didn't even have to mess around with permissions because I copied the file.

My Java programming experience tells me that to complete the exercise I need something like a `for` loop. So I look through chapter 3 and section 3.3.2 "The `for` Statement" and there is an example so I can see what the syntax looks like. I copy this into my code

```
#!/usr/local/bin/perl

use CGI qw(:standard :html3);

my $usernumber = param('usernumber');

print "<html><head><title>Returning User Number</title></head>";
print "<body>";
print $usernumber;
```


My Second Form - Mozilla
File Edit View Go Bookmarks Tools Window
Back Forward Reload Stop
Home Bookmarks Red Hat Network Trair

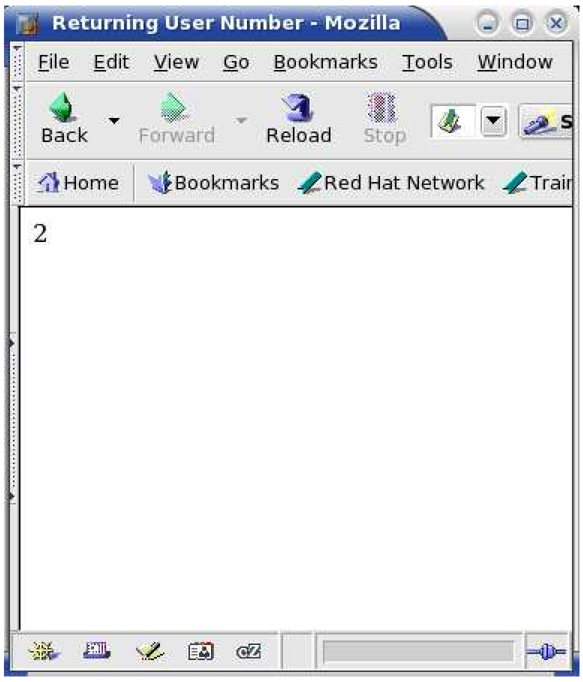
My Second Form

Please enter your number

2

Submit

Figure 8: My Second Form


Returning User Number - Mozilla
File Edit View Go Bookmarks Tools Window
Back
Forward
Reload
Stop
Home
Bookmarks
Red Hat Network
Trair

```
2
```

Figure 9: My Second Form Output

```
for ($count = 0; $count < 10; $count++) {
     $value = <STDIN>;
     $sum += $value;
}
print "</body></html>";
```

I’m going to have to do some work here. For a start `<STDIN>` is prompting the user for information and I don’t know what it’ll do in a CGI situation but its almost certain to go wrong. Instead I’ll delete it and the next line and instead I’ll print out the value of `$count` just to check its doing what I think it is (i.e. counting from zero to 9).

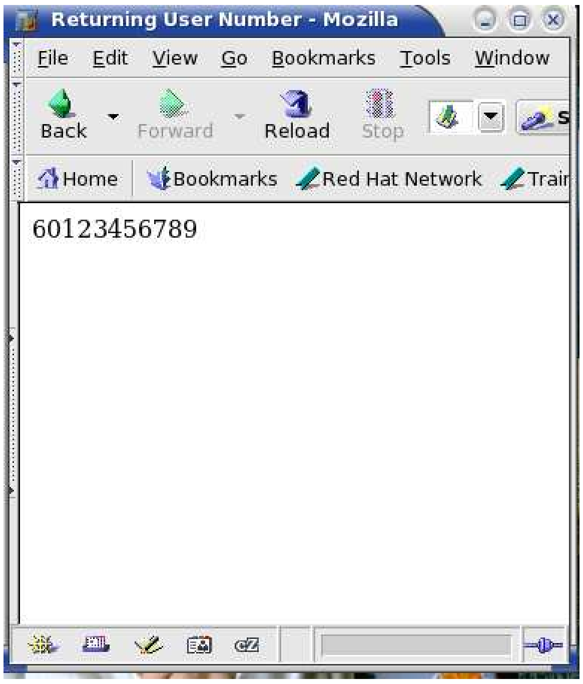
```
#!/usr/local/bin/perl

use CGI qw(:standard :html3);

my $usernumber = param('usernumber');

print "Content-type: text/html\n\n";

print "<html><head><title>Returning User Number</title></head>";
print "<body>";
print $usernumber;

for ($count = 0; $count < 10; $count++) {
     print "$count\n";
}
print "</body></html>";
```

Before I do anything else, I’ll just check it does what I expect (see figure 10).

OK, so that wasn’t quite what I expected. I was expecting the `\n` to put each row on a separate line. Then I remember that HTML doesn’t work like that. If you go an look at some HTML documentation you’ll find you need to tell it when to start new lines. However the exercise didn’t say anything about formatting so for the time being I’ll leave it as it is.

Now I don’t want it to start by printing the user number and I want it to print from zero to the user number rather than from 1 to 9. I can edit the appropriate bits to do this...

```
#!/usr/local/bin/perl

use CGI qw(:standard :html3);

my $usernumber = param('usernumber');

print "Content-type: text/html\n\n";
```


Returning User Number - Mozilla
File Edit View Go Bookmarks Tools Window
Back Forward Reload Stop
Home Bookmarks Red Hat Network Trair

```
60123456789
```

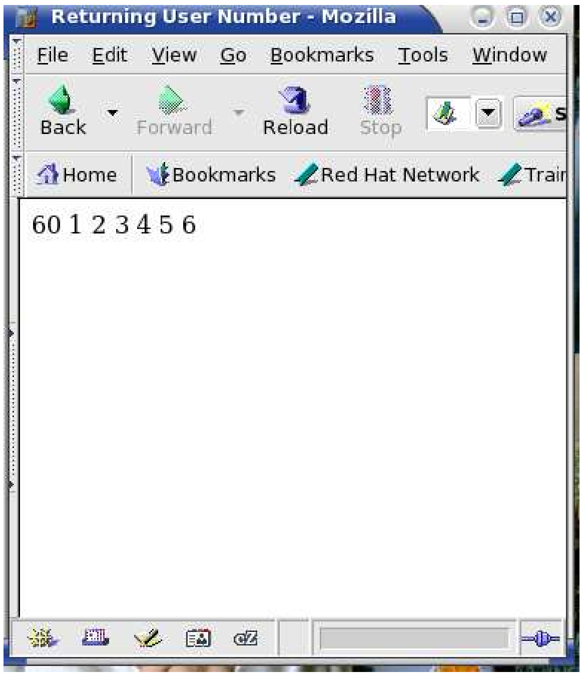
Figure 10: My Second Form Output

```
print "<html><head><title>Returning User Number</title></head>";
print "<body>";
print $usernumber;

for ($count = 0; $count <= $usernumber; $count++) {
     print "$count\n";
}
print "</body></html>";
```

and test again (figure 11). and I'm done.

Using While Loops

The Exercise Write a command line perl program that asks the user to enter a message a line at a time ending with the line `quit`. It then prints out the message except for the last time (which is the one that said `quit`).


Returning User Number - Mozilla
File Edit View Go Bookmarks Tools Window
Back Forward Reload Stop
Home Bookmarks Red Hat Network Trair

```
60 1 2 3 4 5 6
```

Figure 11: My Second Form Output

Designing the Program

I need to think a little before I start on this. Firstly my Java experience tells me I probably want to use a `while` loop of some sort since I can't tell, until I reach `quit`, how many lines the message will have. I'll draw a little control flow diagram. There are going to be four things the program does

1. Asks the user to enter a message (it does this first so it should happen before any looping.
2. Reads in a line (this will happen lots of times - probably once each time round the loop).
3. Checks if the line says `quit` and exits the loop if it does.
4. Prints out the final message

This gives me the diagram in figure 12. I'm not going to worry initially about where the message pieces will be stored, I'm going to start with working out how the basic skeleton of this will work.

I find `while` loops on page 42. So I copy that into a file called `ex3.pl` and then take out all the bits to do with `$sum`. I'm also going to need some sort of if-then-else construct for detecting a quit line and I get the syntax for this off page 39.

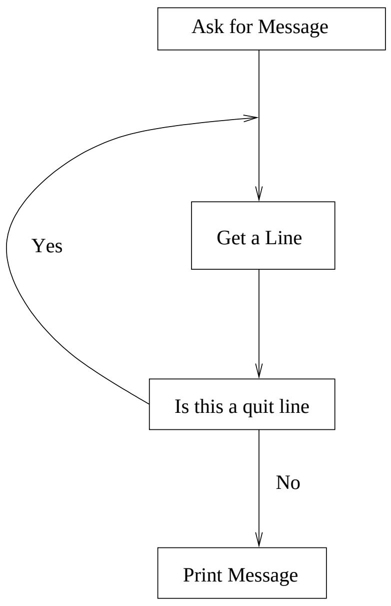


Figure 12: A Control Flow Diagram

```
#!/usr/local/bin/perl

while (????) { # not quitting
  # get line
  if ( ???? ) { # quit line
    ???
  } else {  # carry on
    }
}
```

You can see I've put in question marks and comments for all the bits I'm not sure about (i.e. most of it) . While I'm doing the easy bits I might as well put in the print statements which start and end the program

```
#!/usr/local/bin/perl

print "Please enter a message\n";

while (????) { # not quitting
  # get line
  if ( ???? ) { # quit line
    ???
  } else {  # carry on
```

```
    }
}

print $message;
```

Next I want to decide what stops the while loop going round. For the time being lets have a scalar called `$quit` which is equal to true if the program should quit and false otherwise. It reminds me on page 39 that 0 is false and flicking through the chapter I see in section 3.1.1 that everything else is true. I also find in the following sections that `!` means "not". Puting all this together I get:

```
#!/usr/local/bin/perl

print "Please enter a message\n";
$quit = 0;

while (!$quit) { # not quitting
  # get line
  if ( ???? ) { # quit line
    $quit = 1;
  } else {  # carry on
    }
}

print $message;
```

So far so good. Now, how do I get perl to read in a line from the keyboard. Well I remember this from Chapter 2 (which of course you read for last time :-). This says "The expression `<STDIN>` reads keyboard input, up to, and including, an input record separator, and this string value replaces the appearance of `<STDIN>`" which sounds like exactly what I want. So I'll add `$input = <STDIN>` to the program.

```
#!/usr/local/bin/perl

print "Please enter a message\n";
$quit = 0;

while (!$quit) { # not quitting
  $input = <STDIN>;  # get line
  if ( ???? ) { # quit line
    $quit = 1;
  } else {  # carry on
    }
}


print $message;
```

Now I only have to get rid of one set of question marks and I can test the program. I want that to be true if the input is equal to `quit`. I flick through the textbook and on page 36 I find relational operators which tell me that I need to use `eq` to compare strings so my code now is.

```
#!/usr/local/bin/perl

print "Please enter a message\n";
$quit = 0;

while (!$quit) { # not quitting
  $input = <STDIN>;  # get line
  if ( $input eq "quit" ) { # quit line
    $quit = 1;
  } else {  # carry on
    }
}

print $input;
```

I've changed the `print $message` to `print $input` at the end. I know it won't print out the whole message, just the last line but I want to check the program first before I start worrying about that.

So I run the program

```
[lad@bartok handouts]$ perl ex3.pl
Please enter a message
lakdsjf
kdkkd
quit
lasdf
quit
```

You'll notice I type `quit` twice in the above. That's because it didn't exit either time I typed it. In the end I had to press Control-C to exit the program.

At this point I remember that under Keyboard input it said "up to and including the input separator". This is a standard Perl Gotcha. When I'm typing `quit` `$input` is getting bound to `quit\n`. I edit my code accordingly.

```
#!/usr/local/bin/perl

print "Please enter a message\n";
$quit = 0;

while (!$quit) { # not quitting
  $input = <STDIN>;  # get line
  if ( $input eq "quit\n" ) { # quit line
```

```
    $quit = 1;
  } else {  # carry on
    }
}


print $input;
```

and try running the program again

```
[lad@bartok handouts]$ perl whileloop.pl
Please enter a message
sdf
sgag
quit
quit
[lad@bartok handouts]$
```

OK, so it's printing out the wrong message `quit` instead of `sdf sgag` but at least I've got it detecting the `quit` line properly - so that's one problem solved.

When scalars were first introduced (last lecture) there was a bit about built-in functions and operators for strings. So I go back and have a look to see if any of these will help and there on slide 35 is the concatenation operator which joins strings together and is a dot. So I'll introduce a new variable `$message` and gradually build it up each time I go round the loop and then print it out at the end.

```
#!/usr/local/bin/perl

print "Please enter a message\n";
$quit = 0;
$message = "";

while (!$quit) { # not quitting
  $input = <STDIN>;  # get line
  if ( $input eq "quit\n" ) { # quit line
    $quit = 1;
  } else {  # carry on
    $message = $message . $input
    }
}


print $message;
```

And then I test it

```
[lad@bartok handouts]$ perl whileloop.pl
Please enter a message
```

```
sfd
;asugfoi
skdlkj afllkd
quit
sfd
;asugfoi
skdlkj afllkd
[lad@bartok handouts]$
```

Hooray! it works.

As an exercise the following program works the same way and would probably gain more marks in an exercise. How does it work

```
#!/usr/local/bin/perl

print "Please enter a message\n";
$quit = 0;
$message = "";

while (chomp ($input = <STDIN>)) { # not quitting
  if ( $input eq "quit" ) { # quit line
    last;
  } else {  # carry on
    $message = $message . $input . "\n";
    }
}


print $message;
```