Practice Implementation Exam Questions

1. In the context of computing, a *shell* is

   a. part of the Unix operating system

   b. a program that arranges the execution of other programs

   c. a component of a window manager such as `fvwm`

   d. an object-oriented wrapper for a procedural program

   a. Incorrect ... the shell runs as a normal process under Unix; it is not part of the O/S

   b. Correct.

   c. Incorrect ... the shell is completely separate from any window manager; you can run a shell without having any window manager running.

   d. Incorrect ... meaningless answer; certainly not shell-related

2. A shell script can be executed by placing it in a file called e.g. `script` and then

   a. executing the command `run script`

   b. clicking on an icon for that script in a window manager

   c. executing the command `sh script`

   d. making the file readable and then simply typing its name

   a. Incorrect ... on any Unix system I've ever seen `run` gives a `Command not found` error message. I guess it's remotely possible that some Unix system somewhere has a command called `run`. The chances of it doing what's described above is even more remote.

   b. Incorrect ... I haven't seen any window managers where shell scripts automatically get turned into icons

   c. Correct ... guaranteed to work on any Unix system I've seen.

   d. Incorrect ... you'd need to make it executable as well.

3. Which one of the following regular expressions would match a non-empty string consisting only of the letters `x`, `y` and `z`, in any order?

   a. `[xyz]+`

   b. `x+y+z+`

   c. `(xyz)*`

   d. `x*y*z*`

   a. Correct

   b. Incorrect ... this matches strings like `xxx...yyy...zzz...`

   c. Incorrect ... this matches strings like `xyzxyzxyz...`

   d. Incorrect ... this matches strings like `xxx...yyy...zzz...`

   The difference between (b) and (d) is that (b) requires there to be at least one of each `x`, `y` and `z`.

4. Which one of the following commands would extract the student id field from a file in the following format:

```
COMP3311;2122987;David Smith;95
COMP3231;2233445;John Smith;51
COMP3311;2233445;John Smith;76
```

   a. `cut -f 2`

   b. `cut -d; -f 2`

   c. `sed -e 's/.*;//'`

   d. None of the above.

a. Incorrect ... this gives the entire data file; the default field-separator is tab, and since the lines contain no tabs, they are treated as a single large field; if an invalid field number is specified, `cut` simply prints the first

b. Incorrect ... this runs two separate commands `cut -d` followed by `-f 2`, and neither of them makes sense on its own

c. Incorrect ... this removes all chars up to and including the final semicolon in the line, and this gives the 4th field on each line

d. Correct

5. Which one of the following Perl commands would acheive the same effect as in the previous question (i.e. extract the student id field)?

   a. `perl -e '{while ( ) { split /;/; print;}}'`

   b. `perl -e '{while ( ) { split /;/; print $2;}}'`

   c. `perl -e '{while ( ) { @x = split /;/; print "$x[1]\n";}}'`

   d. `perl -e '{while ( ) { @x = split /;/; print "$x[2]\n";}}'`

   a. Incorrect ... this splits the line, but doesn't save the result of the splitting, and then prints the default value, which is the whole line read

   b. Incorrect ... `$2` does not refer to the second field in Perl

   c. Correct ... the `split` saves the result in the `@x` list, and the index `[1]` selects the second value from the list

   d. Incorrect ... the `split` saves the result in the `@x` list, but the index `[2]` selects the third value from the list

6. What is the URL-encoded version of the string `Dad & Dave` (the hexadecimal ascii code for the ampersand charcter is `26`)?

   a. `Dad+&+Dave`

   b. `dad+%26+dave`

   c. `Dad+%26+Dave`

   d. `%Dad+%26+%Dave`

   a. Incorrect ... the `&` character is used to separate parameters in a URL, so this would "pass" `"Dad "` and `" Dave"`

   b. Incorrect ... URL parameter values are case-sensitive

   c. Correct ... the spaces are replaced by `+`'s and the ampersand is replaced by `%26`, as required by URL-encoding

   d. Incorrect ... the code `%D` refers to the newline character, not to the letter capital- `D`

7. Which of the following CGI.pm function calls produces a popup menu associated with the parameter `choice` and with the numbers `1`, `2` and `3` as choices?

   a. `popup_menu(-name=>'choice',-values=>{'1','2','3'})`

   b. `popup_menu(-name=>'choice',-values=>['1','2','3'])`

   c. `popup_menu(-name=>'choice',-values=>{'One','Two','Three'})`

   d. `popup_menu(-name=>'choice',-values=>['One','Two','Three'])`

   a. Incorrect ... the type of the `values` parameter value must be a reference to an array; the `{'1','2','3'}` is a block

   b. Correct

   c. Incorrect ... the type of the `values` parameter is incorrect and the values themselves are also incorrect

   d. Incorrect ... the type of the `values` parameter is correct but the values themselves are incorrect

8. For each of the following questions, give *brief* written answers preferably in point form. Around half a page should be sufficient for each question, and you should certainly write **no more than one page** for any one question. Where appropriate, give examples to illustrate your points.

   a. Consider the following Perl program that processes its standard input:

```
#!/usr/local/bin/perl
while (<STDIN>) {
    @marks = split;
    $studentID = $marks[0];
    for (i = 0; i < $#marks; i++) {
        $totalMark += $marks[$i];
    }
    printf "%s %d\n", $studentID, $totalMark;
}
```

This program has several common mistakes in it. Indicate and describe the nature of each of these mistakes, and say what the program is attempting to do.

Sample solution:

- The `for` loop uses the "variable" `i` but forgets to prefix it with the `$` symbol, so it will be treated as a constant and an error message generated
- The iteration over the marks is incorrect; the value `$#marks` gives the index of the last array element; since the loop runs to less than `$#marks` it will miss the last element
- A related point: since the first element in the array is the student ID and not a mark, it should not be included in the `$totalMark`; the loop iteration should start from `$i = 1`.
- The value of `$totalMark` is not reset for each student, so the total simply increases continually and does not reflect the sum of marks for any individual except the first student

b. Does the CGI.pm library really make it easier to write Perl code to produce HTML output? Discuss this *briefly*. Provide at least one example where the CGI.pm functions provide a definite advantage over writing the HTML yourself, and at least one example where HTML strings are simpler than CGI.pm functions.

Sample solution:

- For some constructs, `CGI.pm` requires smaller expressions.

  Example: generating a regular table ...

  ```
  table(
      tr([td(["1,1",1,2",1,3",1,4"]),
      tr([td(["2,1",2,2",2,3",2,4"]),
      tr([td(["3,1",3,2",3,3",3,4"]),
      tr([td(["4,1",4,2",4,3",4,4"]),
  )
  ```

  compared to

  ```
  <table>
      <tr> <td>1,1</td><td>1,2</td><td>1,3</td><td>1,4</td> </tr>
      <tr> <td>2,1</td><td>2,2</td><td>2,3</td><td>2,4</td> </tr>
      <tr> <td>3,1</td><td>3,2</td><td>3,3</td><td>3,4</td> </tr>
      <tr> <td>4,1</td><td>4,2</td><td>4,3</td><td>4,4</td> </tr>
  </table>
  ```

- For other constructs, `CGI.pm` requires similar or longer expressions, and gives you less control over the outcome.

  Example: generating an ordinary paragaph

  ```
  p("This is some text")
  ```

  compared to

  ```
  <P>This is some text</P>
  ```

c. You are given a program that claims to compute the average of exam marks that appear as the final item of each line of a space-separated file of student information e.g.

```
John Smith 22334455 75
Jane Alison Smith 2133567 75
```

What kind of validity checking should such a program be performing? Suggest suitable test data for testing the correctness and reliability of this program. Distinguish between reliability and correctness tests.

Points to note:

- Correctness tests aim to establish whether the system meets it requirement (i.e. whether given valid input it produces valid output). Reliability tests aim to check whether the system behaves in a "reasonable" way when it is given invalid input.
- Examples of invalid input for reliability testing

- A line with no name (just student ID and mark)
- A student with invalid student ID (no indentity)
- A student with invalid marks value (outside 0..100 range)
- A completely empty line
- Completely empty input
- etc. etc.

Examples of correct input for correctness testing

- Input with exactly zero, one, two students
- Input with all students with same/different marks
- Students with 1/2/3 components to their names
- etc. etc.

9. Write a *shell script* called `rmall.sh` that removes all of the files and directories below the directory supplied as its single command-line argument. The script should prompt the user with `Delete` *X* `?` before it starts deleting the contents of any directory *X*. If the user responds `yes` to the prompt, `rmall` should remove all of the plain files in the directory, and then check whether the contents of the subdirectories should be removed. The script should also check the validity of its command-line arguments.

   Sample solution

```
#!/bin/sh

# check whether there is a cmd line arg
case $# in
1) # ok ... requires exactly one arg
    ;;
*)
    echo "Usage: $0 dir"
    exit 1
esac

# then make sure that it is a directory
if test ! -d $1
then
    echo "$1 is not a directory"
    echo "Usage: $0 dir"
    exit 1
fi

# change into the specified directory
cd $1

# for each plain file in the directory
for f in .* *
do
    case $f in
    .|..) # ignore . and ..
        ;;
    *)
        if test -f $f
        then
            rm $f
        fi
        ;;
    esac
done

# for each subdirectory
for d in .* *
do
    case $d in
    .|..) # ignore . and ..
        ;;
    *)
        if test -d $d
        then
            echo -n "Delete $d? "
            read answer
            if test "$answer" = "yes"
            then
                rmall $d
            fi
        fi
        ;;
    esac
done
```

10. Write a *shell script* called `check` that looks for duplicated student ids in a file of marks for a particular subject. The file consists of lines in the following format:

```
2233445 David Smith 80
2155443 Peter Smith 73
2244668 Anne Smith 98
2198765 Linda Smith 65
```

The output should be a list of student ids that occur 2+ times, separated by newlines. (i.e. any student id that occurs more than once should be displayed on a line by itself on the standard output).

Sample solution

```
#!/bin/sh

cut -d' ' -f1 < Marks | sort | uniq -c | egrep -v '^  *1 ' | sed -e 's/^.*   //'
```

Explanation:

1. `cut -d' ' -f1 < Marks` ... extracts the student ID from each line
2. `sort | uniq -c` ... sorts and counts the occurrences of each ID
3. IDs that occur once will be on a line that begins with spaces followed by `1` followed by a TAB
4. `grep -v '^ *1 '` removes such lines, leaving only IDs that occur multiple times
5. `sed -e 's/^.* //'` gets rid of the counts that `uniq -c` placed at the start of each line

11. Write a *Perl script* that reverses the fields on each line of its standard input. Assume that the fields are separated by spaces, and that only one space is required between fields in the output.

Sample solution

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

while (<STDIN>)
{
    chomp;
    @fields = split;
    @fields = reverse @fields;
    $outline = join(' ',@fields);
    print "$outline\n";
}
```

or, as a one-liner

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

while (<STDIN>) { chomp; print join(' ', reverse(split)),"\n"; }
```

12. Consider the following table of student enrolment data:

StudentID	Course	Year	Session	Mark	Grade
2201440	COMP1011	1999	S1	57	PS
2201440	MATH1141	1999	S1	51	PS
2201440	MATH1081	1999	S1	60	PS
2201440	PHYS1131	1999	S1	52	PS
...	...	...	...	...	...

A file containing a large data set in this format for the years 1999 to 2001 and ordered by student ID is available in the file *data*.

Write a program that computes the average mark for a specified course for each of the sessions that it has run. The course code is specified as a command-line argument, and the data is read from standard input. All output from the program should be written to the standard output.

If no command-line argument is given, the program should write the following message and quit:

```
    Usage: ex3 Course
```

The program does *not* have to check whether the argument is valid (i.e. whether it looks like a real course code). However, if the specified course code (*CCODE*) does not appear anywhere in the data file, the program should write the following message:

```
    No marks for course CCODE
```

Otherwise, it should write one line for each session that the course was offered. The line should contain the course code, the year, the session and the average mark for the course (with one digit after the decimal point). You can assume that a course will not be offered more than 100 times. The entries should be written in chronological order.

The following shows an example input/output pair for this program:

Sample Input Data	Corresponding Output

`COMP1011`	`COMP1011 1999 S1 62.5`
	`COMP1011 2000 S1 69.1`
	`COMP1011 2001 S1 66.8`

Sample Perl solution

```
#!/usr/bin/perl

if (@ARGV < 1) {
    print "Usage: ex3 Course\n"; exit 0
}
else {
    $c = $ARGV[0];
}

while (<STDIN>) {
    chomp;
    my ($sid,$course,$year,$sess,$mark,$grade) = split;

    if ($course eq $c) {
        $sum{"$year $sess"} += $mark;
        $count{"$year $sess"}++;
        $nofferings++;
    }
}

if ($nofferings == 0) {
    print "No marks for course $c\n";
}
else {
    foreach $s (sort keys %sum) {
        printf "$c $s %0.1f\n", $sum{"$s"}/$count{"$s"};
    }
}
```

Sample Shell solution (advanced: piping to control structures and use of awk as calculator)

```
#!/bin/sh

case $# in
1) ;;
*) echo "Usage: ex3 Course"; exit 1;;
esac

cat > tmp

course="$1"
nrecs=`egrep "$1" tmp | wc -l`
if [ "$nrecs" -eq "0" ]
then
    echo "No marks for course $course"
    rm tmp
    exit 0
fi

egrep "$1" tmp | awk '{print $3" "$4}' | sort | uniq | \
while read year sess
do
#    patt="$course  $year  $sess"
    patt="$course[[:space:]]*$year[[:space:]]*$sess"
    count=`egrep "$patt" tmp | wc  -l`
    total=`egrep "$patt" tmp | awk '{sum+=$5}END{print sum}'`
    if [ "$count" -ne "0" ]
    then
        avg=`expr $total / $count`
        printf "$course $year $sess %0.1f\n" $avg
    fi
done

rm tmp
```

13. Write a program that prints a count of how often each letter ('a'..'z' and 'A'..'Z') and digit ('0'..'9') occurs in its input. Your program should follow the output format indicated in the examples below exactly.
No count should be printed for letters or digits which do not occur in the input.

    The counts should be printed in dictionary order ('0'..'9','A'..'Z','a'..'z').

    Characters other than letters and digits should be ignored.

    The following shows an example input/output pair for this program:

Sample Input Data	Corresponding Output

The Mississippi is 1800 miles long!	'0' occurred 2 times '1' occurred 1 times '8' occurred 1 times 'M' occurred 1 times 'T' occurred 1 times 'e' occurred 2 times 'g' occurred 1 times 'h' occurred 1 times 'i' occurred 6 times 'l' occurred 2 times 'm' occurred 1 times 'n' occurred 1 times 'o' occurred 1 times 'p' occurred 2 times 's' occurred 6 times

Sample Perl solution

```perl
#!/usr/bin/perl -w
while (<>) {
    for (split //) {
        $count{$_}++ if /[a-zA-Z0-9]/;
    }
}
print "'$_' occurred $count{$_} times\n" for sort keys %count;
```

Another Sample Perl solution

```perl
#!/usr/bin/perl -w
# courtesy aek@cse.unsw.EDU.AU
# letter count- count number of occurrences of each letter

# map letters to counts
my %lettercount = ();
while (<>) {
        chomp;

        # remove anything but letters and numbers
        s/[^A-Za-z0-9]//g;

        # split the line into an array of characters
        @chars = split //;
        foreach $letter (@chars) {
                # record count in hash table
                $lettercount{$letter}++;
        }
}

# output count of each letter, sorted on the keys (letters)
foreach $letter (sort keys %lettercount) {
        print "'$letter' occurred $lettercount{$letter} times\n";
        # (look up count for each letter from table)
}
```

14. Write a program that maps all lower-case vowels (a,e,i,o,u) in its standard input into their upper-case equivalents and, at the same time, maps all upper-case vowels (A, E, I, O, U) into their lower-case equivalents.
The following shows an example input/output pair for this program:

Sample Input Data	**Corresponding Output**
This is some boring text. A little foolish perhaps?	ThIs Is sOmE bOrIng tExt. a lIttlE fOOlIsh pErhAps?

Sample C solution

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int main(int argc, char *argv[])
{
    int   ch;
    char *smallVowels = "aeiou";
    char *largeVowels = "AEIOU";

    while ((ch = getchar()) != EOF) {
        if (index(smallVowels,ch) != NULL)
            putchar(toupper(ch));
        else if (index(largeVowels,ch) != NULL)
            putchar(tolower(ch));
        else
            putchar(ch);
    }
    return 0;
}
```

Sample Perl solution

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

@lines = <STDIN>;
map {tr /aeiouAEIOU/AEIOUaeiou/} @lines;
print @lines;
```

Another Sample Shell solution

```sh
#!/bin/sh

tr aeiouAEIOU AEIOUaeiou
```

15. A "hill vector" is structured as an *ascent*, followed by an *apex*, followed by a *descent*, where

    - the *ascent* is a non-empty strictly ascending sequence that ends with the apex
    - the *apex* is the maximum value, and must occur only once
    - the *descent* is a non-empty strictly descending sequence that starts with the apex

    For example, [1,2,3,4,3,2,1] is a hill vector (with apex=4) and [2,4,6,8,5] is a hill vector (with apex=8). The following vectors are not hill vectors: [1,1,2,3,3,2,1] (not strictly ascending and multiple apexes), [1,2,3,4] (no descent), and [2,6,3,7,8,4] (not ascent then descent). No vector with less than three elements is considered to be a hill.

    Write a program that determines whether a sequence of numbers (integers) read from standard input forms a "hill vector". The program should write "hill" if the input *does* form a hill vector and write "not hill" otherwise.

    Your program's input will only contain digits and white space. Any amount of whitespace may precede or follow integers.

    Multiple integers may occur on the same line.

    A line may contain no integers.

    You can assume all the integers are positive. The following shows all example input/output pairs for this program:

Sample Input Data	Corresponding Output
`1 2 4 8 5 3 2`	`hill`
`1 2`	`not hill`
`1 3 1`	`hill`

```
  3
1   1
```
	`not hill`

`2 4 6 8 10 10 9 7 5 3 1`	`not hill`

Sample Perl solution

```perl
#!/usr/bin/perl -w
@n = split /\D+/, join(' ', <>);
$i= 0;
$i++ while $i < $#n && $n[$i] < $n[$i+1];
$j = $#n;
$j-- while $j > 0 && $n[$j] < $n[$j-1];
print "not " if $i != $j || $i == 0 || $j == $#n;
print "hill\n";
```

Sample C solution

```
#include <stdio.h>

int main(void)
{
    int i, n, v[101], max, ok;

    n = 0;
    while (scanf("%d",&i) == 1 && n < 100)
        v[n++] = i;
    if (n < 3) {
        printf("not hill\\n");
        return 0;
    }
    max = 0;
    for (i = 1; i < n; i++)
        if (v[i] > v[max])
            max = i;

    ok = max > 0 && max < n-1;
    for (i = 0; ok && i < max; i++)
        if (v[i] >= v[i+1])
            ok = 0;
    for (i = max; ok && i < n-1; i++)
        if (v[i] <= v[i+1])
            ok = 0;

    if (ok)
        printf("hill\n");
    else
        printf("not hill\n");
    return 0;
}
```

(GRW, 05s1): a solution closer to the classic partition algorithm (a la Quicksort) is

```
#include <stdio.h>

int main(void)
{
    int i, j, n, v[101];

    n = 0;
    while (scanf("%d",&i) == 1 && n < 100)
        v[n++] = i;
    if (n < 3) {
        printf("not hill\\n");
        return 0;
    }
    i = 0;
    while (i < n-1 && v[i] < v[i+1]) i++;
    // v[i] >= v[i+1] && ascending(v[0..i-1])

    j = n-1;
    while (j > 0 && v[j] < v[j-1]) j--;
    // v[j] <= v[j-1] && descending(v[j+1..n-1])

    // v is a hill iff i and j identify a single, interior peak
    if (i == j && i > 0 && j < n-1)
        printf("hill\n");
    else
        printf("not hill\n");
    return 0;
}
```

This version could be easily extended to identify a "plateau vector" as well.

16. A list $a_1, a_2, \dots a_n$ is said to be *converging* if

$$a_1 > a_2 > \dots > a_{n-1} > a_n$$

and

$$\forall i, 1 < i < n, \; a_{i-1} - a_i > a_i - a_{i+1}$$

In other words, the list is strictly decreasing and the difference between consecuctive list elements always decreases as you go down the list.

Write a program that determines whether a sequence of positive integers read from standard input is converging. The program should write "converging" if the input is converging and write "not converging" otherwise. It should produce no other output.

Sample Input Data	Corresponding Output
2010 6 4 3	converging

```
20
15
9
```

```
not converging
```

```
1000
     100   10
     1
```

```
converging
```

```
  6
  5
2 2
```

```
not converging
```

```
  1 2 4 8
```

```
not converging
```

Your program's input will only contain digits and white space. Any amount of whitespace may precede or follow integers.

Multiple integers may occur on the same line.

A line may contain no integers.

You can assume your input contains at least 2 integers.all the integers are positive.

You can assume all the integers are positive.

Sample Perl solution

```
#!/usr/bin/perl -w
@n = split /\D+/, join(' ', <>);
foreach $i (1..$#n) {
    if ($n[$i] >= $n[$i+1]) {
        print "not converging\n";
        exit;
    }
}
foreach $i (2..$#n) {
    if ($n[$i-2] - $n[$i-1] <= $n[$i-1] - $n[$i]) {
        print "not converging\n";
        exit;
    }
}
print "converging\n";
```

17. The *weight* of a number in a list is its value multiplied by how many times it occurs in the list. Consider the list `1 6 4 7 3 4 6 3 3]` . The number 7 occurs once so it has weight 7. The number 3 occurs 3 times so it has weight 9. The number 4 occurs twice so it has weight 8.
Write a program which takes 1 or more positive integers as arguments and prints the heaviest.

    Your program should print one integer and no other output.

    Your program it is given only positive integers as arguments

    For example, if you program is named `a.out` here is how it should behave :

```
$ ./heaviest.pl 1 6 4 7 3 4 6 3 3
6
$  ./heaviest.pl 1 6 4 7 3 4 3 3
3
$  ./heaviest.pl 1 6 4 7 3 4 3
4
$  ./heaviest.pl 1 6 4 7 3 3
7
```

Sample Perl solution

```
#!/usr/bin/perl -w
foreach $n (@ARGV) {
    $w{$n * grep {$_ == $n} @ARGV} = $n;
}
print $w{(sort {$b <=> $a} keys %w)[0]}, "\n";
```

18. We wish to create a web site named myAddressBook where users can stores their address book.
After users login to the web site it should show them their current addressbook.

Then should be able to add and delete name/address pairs from their addressbook.

Write a CGI script `myAddressBook.cgi` to perform this task.

Here is an example implementation (http://cgi.cse.unsw.edu.au/~cs2041cgi/tut/perlcgi/myAddressBook.cgi).

Sample solution

```
#!/usr/bin/perl
# Simple CGI script written by andrewt@cse.unsw.edu.au

use CGI qw/:all/;
use CGI::Carp qw(fatalsToBrowser warningsToBrowser);

print header, start_html('My addressbook');
warningsToBrowser(1);

$data_directory = "./addresses/";
mkdir $data_directory or die "Cannot create $data_directory: $!\n" if !-d $data_directory;

$login = param('login');

if (!$login) {
    print    start_form,
            'Enter your login: ', textfield('login'),
            end_form,
            end_html;
    exit 0;
}

$login =~ s/[^\w\s]//g;          # remove all but expected characters
$login = substr $login, 0, 64;   # limit login to 64 characters
print h2("My Addressbook for $login");

$user_directory = "$data_directory/$login/";
mkdir $user_directory or die "Cannot create $user_directory: $!\n" if !-d $user_directory;

if (param('add_name') && param('add_address')) {
    my $name = param('add_name');
    $name =~ s/[^\w\s-_]//g;        # remove all but expected characters
    $name = substr $name, 0, 256;   # limit name to 256 characters
    $address = param('add_address');
    $address =~ s/[^\w\s-_\/]//g;          # remove all but expected characters
    $address = substr $address, 0, 1024;   # limit address to 1024 characters
    open F, ">$user_directory/$name" or die "Cannot create $user_directory/$name: $!\n";
    print F $address;
    close F;
}

if (param('Delete')) {
    my $name = param('delete_name');
    $name =~ s/[^\w\s-_]//g;        # remove all but expected characters
    $name = substr $name, 0, 256;   # limit name to 256 characters
    unlink "$user_directory/$name" or die "Cannot unlink $user_directory/$name: $!\n";
}

@address_files = glob "$user_directory/*";

if (!@address_files) {
    print "Your addressbook is empty, $login.";
} else {
    print "<table border=1>";
    foreach $address_file (@address_files) {
        open F, $address_file or die "Cannot access $address_file: $!\n";
        my @address = <F>;
        close F;
        my $name = $address_file;
        $name =~ s/.*\///;
        print "<tr><td>$name<td>@address\n";
    }
    print "</table>";
}

print start_form,
    hr,
    h4('Add a new address'),
    'Name: ', textfield('add_name'),
    ' Address: ', textfield('add_address'), ' ',
    hidden(login),
    submit('Add'),
    end_form;

if (@address_files) {
    my @names = @address_files;
    s/.*\/// foreach @names;
    print start_form,
        hr,
        h4('Delete the address for:'),
        'Name: ',   popup_menu('delete_name', \@names),
        hidden(login),
        submit('Delete'),
        hr,
        end_form;
}
print end_html;
```

19. Modify the CGI script to store the user's login in a cookie and log them in automatically on future visits.

Sample solution

```
#!/usr/bin/perl
# Simple CGI script written by andrewt@cse.unsw.edu.au


use CGI qw/:all/;
use CGI::Carp qw(fatalsToBrowser warningsToBrowser);
use CGI::Cookie;

%cookies = CGI::Cookie->fetch;
$login = '';
if (defined param('login')) {
    $login =  param('login');
} elsif ($cookies{'myAddressBookLogin'}) {
    $login = $cookies{'myAddressBookLogin'}->value
}

$login =~ s/[^\w\s]//g;          # remove all but expected characters
$login = substr $login, 0, 64;   # limit login to 64 characters

if (!$login) {
    print   header,
            start_html('My addressbook'),
            start_form,
            'Enter your login: ', textfield('login'),
            end_form,
            end_html;
    exit 0;
}

my $cookie = CGI::Cookie->new(-name => 'myAddressBookLogin', -value => $login, -expires => '+3M');
print   header(-cookie=>$cookie),
        start_html('My addressbook'),
        h2("My Addressbook for $login");
warningsToBrowser(1);

$data_directory = "./addresses/";
mkdir $data_directory or die "Cannot create $data_directory: $!\n" if !-d $data_directory;
$user_directory = "$data_directory/$login/";
mkdir $user_directory or die "Cannot create $user_directory: $!\n" if !-d $user_directory;

if (param('add_name') && param('add_address')) {
    my $name = param('add_name');
    $name =~ s/[^\w\s-_]//g;        # remove all but expected characters
    $name = substr $name, 0, 256;   # limit name to 256 characters
    $address = param('add_address');
    $address =~ s/[^\w\s-_\/]//g;          # remove all but expected characters
    $address = substr $address, 0, 1024;   # limit address to 1024 characters
    open F, ">$user_directory/$name" or die "Cannot create $user_directory/$name: $!\n";
    print F $address;
    close F;
}

if (param('Delete')) {
    my $name = param('delete_name');
    $name =~ s/[^\w\s-_]//g;        # remove all but expected characters
    $name = substr $name, 0, 256;   # limit name to 256 characters
    unlink "$user_directory/$name" or die "Cannot unlink $user_directory/$name: $!\n";
}

@address_files = glob "$user_directory/*";

if (!@address_files) {
    print "Your addressbook is empty, $login.";
} else {
    print "<table border=1>";
    foreach $address_file (@address_files) {
        open F, $address_file or die "Cannot access $address_file: $!\n";
        my @address = <F>;
        close F;
        my $name = $address_file;
        $name =~ s/.*\///;
        print "<tr><td>$name<td>@address\n";
    }
    print "</table>";
}

print start_form,
    hr,
    h4('Add a new address'),
    'Name: ', textfield('add_name'),
    ' Address: ', textfield('add_address'), ' ',
    hidden(login), # in case cookies are disbaled
    submit('Add'),
    end_form;

if (@address_files) {
    my @names = @address_files;
    s/.*\/// foreach @address_files;
    print start_form,
        hr,
        h4('Delete the address for:'),
        'Name: ',   popup_menu('delete_name', \@names),
        hidden(login), # in case cookies are disbaled
        submit('Delete'),
        hr,
        end_form;
}
print end_html;
```

20. If there is any remaining time revisit any uncompleted questions on CGI scripts from last week.