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

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

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*`

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.

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";}}'`

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`

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'])`

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.

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.

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.

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.

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).

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.

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`

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`

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?`

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

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

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

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.

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
```

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).

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

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