COMP(2041|9044) 19T2

Week 06 Tutorial Questions

COMP(2041|9044) 19T2

1. How is the assignment going?
   Does anyone have hints or advice for other students?

   Has anyone discovered interesting cases that have to be handled?

2. Apart from the **legit-*** and **diary.txt**scripts what else do you need to submit (and give an example)?

3. You work on the assignment for an hour tonight. What do you need to do when you are finished?

4. What is a merge conflict - and how do they get handled in git and legit?

5. Write a Perl program `word_frequency.pl` which prints a count of all the words found in its input. Your program should ignore case. It should treat any sequence of alphabetic characters ([a-z]) as a word. It should treat any non-alphabetic character as a space. It should print words and their counts sorted in increasing order of frequency in the format shown in this example:

```
$ ./word_frequency.pl
Peter Piper picked a peck of pickled peppers;
A peck of pickled peppers Peter Piper picked;
If Peter Piper picked a peck of pickled peppers,
Where's the peck of pickled peppers Peter Piper picked?
Ctrl-D
1 s
1 the
1 where
1 if
3 a
4 picked
4 peppers
4 peter
4 piper
4 pickled
4 peck
4 of
```

6. Write a Perl program `missing_words.pl` which given two files as arguments prints, in sorted order, all the words found in file1 but not file2.
   You can assume words occur one per line in each file.

7. A citizen science project monitoring whale populations has files containing large numbers of whale observations. Each line in these files contains:

   - the date the observation was made
   - the number of whales in the pod ("pod" is the collective number for a group of whales)
   - the species of whale

   For example:

```
$ cat whale_observations.txt
01/09/18 21 Southern right whale
01/09/18  5 Southern right whale
02/09/18  7 Southern right whale
05/09/18  4 Common dolphin
05/09/18  9 Pygmy right whale
05/09/18  4 Striped dolphin
05/09/18 35 Striped dolphin
05/09/18  4 Blue whale
```

Write a Perl program `./merge_whales.pl` which reads a file of whale observations and prints them to its standard output, merging adjacent counts from the same day of the same species. For example:

```
$ ./merge_whales.pl whale_observations.txt
01/09/18 26 Southern right whale
02/09/18 7 Southern right whale
05/09/18 4 Common dolphin
05/09/18 9 Pygmy right whale
05/09/18 39 Striped dolphin
05/09/18 4 Blue whale
```

8. Write a Perl program `./whale_last_seen.pl` which reads a file of whale observations in the same format as the last question and prints the species in alphabetical order, with the date they were last seen.

```
$ ./whale_last_seen.pl whale_observations.txt
Blue whale 05/09/18
Common dolphin 05/09/18
Pygmy right whale 05/09/18
Southern right whale 02/09/18
Striped dolphin 05/09/18
```

You can assume the file is in chronological order, so the last line in the file for a species will be the last date for the species.

9. Write a Perl program, `phone_numbers.pl` which given the URL of a web page fetches it by running *wget* and prints any strings that might be phone numbers in the web page.
Assume the digits of phone numbers may be separated by zero or more spaces or hyphens ('-') and can contain between 8 and 15 digits inclusive.

You should print the phone numbers one per line with spaces & hyphens removed.

For example

```
$ ./phone_numbers.pl http://www.unsw.edu.au
20151028
11187777
841430912571345
413200225
61293851000
57195873179
```

Revision questions

The remaining tutorial questions are primarily intended for revision - either this week or later in session.
Your tutor may still choose to cover some of the questions time permitting.

10. What does each of the following Perl code fragments print (no, don't just clip the lines and pass them to Perl, think about what they're doing):

a.
```
$x = 'x';
for ($i = 1; $i <= 3; $i++) {
    $x = "($x)";
}
print "$x\n";
```

b.
```
@hi = split //,"hello";
for ($i = 0; $i < 4; $i++) {
    print $hi[$i];
}
print "\n";
```

c.
```
@vec = (10 .. 20);
print "@vec[1..3]\n";
```

d.
```
foreach $n (1..10) {
    last if ($n > 5);
    print "$n ";
    next if ($n % 2 == 0);
    print "$n ";
}
print "\n";
```

11. Write a Perl program that given the road distances between a number of towns (on standard input) calculates the shortest journey between two towns specified as arguments. Here is an example of how your program should behave.

```
$ ./shortest_path.pl Parkes Gilgandra
Bourke Broken-Hill   217
Bourke Dubbo          23
Bourke Gilgandra      62
Bourke Parkes         71
Canowindra Dubbo      35
Canowindra Gilgandra  13
Canowindra Parkes    112
Dubbo Gilgandra       91
Dubbo Parkes          57
Ctrl-D
Shortest route is length = 105: Parkes Dubbo Canowindra Gilgandra.
```

12. Write a Perl function `print_hash()` that displays the contents of a Perl associative array (hash) in the format below (its the format used by the PHP function `print_r()` e.g. the hash table ...

```
%colours = ("John" => "blue", "Anne" => "red", "Andrew" => "green");
```

and the function call ...

```
print_hash(%colours);
```

should produce the output ...

```
[Andrew] => green
[Anne] => red
[John] => blue
```

Since the function achieves its effect via `print`, it doesn't really need to return any value, but since Perl functions typically return *something*, `print_hash` should return a count of the number of items displayed (i.e. the number of keys in the hash table). Note that the hash should be displayed in ascending alphabetical order on key values.

13. A bigram is two words occuring consecutively in a piece of text. Some pairs of words tend to occur more commonly than others as bigrams, e.g `cold beer` or `programming language`.
Your task is to write a Perl program `bigrams.pl` which reads a piece of text, and prints the words which occur in the text in sorted order, one per line. Each word should be accompanied by the word which most frequently follows it in the text - if several words occur equally often after the word, any of them can be printed. The number of times the word occurs in the text should be indicated as should the number of times the second word follows it. Case should be ignored. For example given this text:

```
$ ./bigrams.pl
Peter Piper picked a peck of pickled peppers;
A peck of pickled peppers Peter Piper picked;
If Peter Piper picked a peck of pickled peppers,
Where's the peck of pickled peppers Peter Piper picked?
Ctrl-D
a(3) peck(3)
if(1) peter(1)
of(4) pickled(4)
peck(4) of(4)
peppers(4) peter(2)
peter(4) piper(4)
picked(3) a(2)
pickled(4) peppers(4)
piper(4) picked(4)
s(1) the(1)
the(1) peck(1)
where(1) s(1)
```

14. Write a Perl program, `times.pl` which prints a table of multiplications.
    Your program will be given the dimension of the table and the width of the columns to be printed. For example:

```
$ ./times.pl 4 5 3
  1   1   2   3   4   5
  2   2   4   6   8  10
  3   3   6   9  12  15
  4   4   8  12  16  20
```

15. Write a Perl program which deletes blank lines from each of the files specified as arguments. For example, if run like this:

```
$ deblank.pl file1 file2 file3
```

    your program should delete any blank lines in `file1`, `file2` and `file3`. Note that this program *changes* the files, it doesn't just write the "de-blanked" versions to standard output.

16. Write a Perl function `listToHTML()` that given a list of values returns a string of HTML code as an unordered list. For example

```
$out = listToHTML('The', 'Quick', 'Brown', 'Fox');
```

    would result in `$out` having the value ...

```
<ul>
<li>The
<li>Quick
<li>Brown
<li>Fox
</ul>
```

    As part of an HTML page, this would display as:

    - The
    - Quick
    - Brown
    - Fox

    P.S. A Perl syntactic short cut can be used to construct the list above:

```
$out = listToHTML(qw/The Quick Brown Fox/);
```

17. Write a Perl function `hashToHTML()` that returns a string of HTML code that could be used to display a Perl associative array (hash) as an HTML table, e.g.

```
# the hash table ...
%colours = ("John"=>"blue", "Anne"=>"red", "Andrew"=>"green");
# and the function call ...
$out = hashToHTML(%colours);
```

    would result in `$out` having the value ...

```
<table border="1" cellpadding="5">
<tr><th> Key </th><th> Value </th></tr>
<tr><td> Andrew </td><td> green </td></tr>
<tr><td> Anne </td><td> red </td></tr>
<tr><td> John </td><td> blue </td></tr>
</table>
```

    As part of an HTML page, this would display as:

Key	**Value**
Andrew	green
Anne	red
John	blue

    Note that the hash should be displayed in ascending alphabetical order on key values.

18. Write a Perl program that will read in a HTML document and output a new HTML document that contains a table with two cells (in one row). In the left cell should be a copy of the complete original HTML document inside <pre> tags so we can see the raw HTML. You will need to replace all "<" characters with the sequence "<" and all ">" characters with the sequence ">", otherwise the browser will think they are HTML tags (and we want to see the tags in the left cell). In the right cell just include the HTML body of the document, so we can see what it will look like when rendered by a browser.

19. Write a Perl program that reads in data about student performance in a Prac Exam consisting of 3 exercises and computes the overall result for each student. The program takes a *single command line argument*, which is the name of a file containing space-separated text records of the form:

```
studentID  exerciseID  testsPassed  numWarnings
```

There will be one line in the file for each exercise submitted by a student, so a given student may have one, two or three lines of data.
The output should be ordered by student ID and should contain a single line for each student, in the format:

```
studentID  totalMark  passOrFail
```

The *totalMark* value is computed as follows:

- if an exercise passes all 5 tests, it is awarded a mark of 10 and is *correct*
- if an exercise passes less than 5 tests, it is awarded a mark of *testsPassed/2* and is *incorrect*
- if there are *any* warnings on an exercise, the mark is reduced by 2
- the minimum mark for a given exercise is zero
- the *totalMark* is the sum of the marks for the individual exercises

The *totalMark* value should be display using the `printf` format `"%4.1f"`. A student is awarded a `PASS` if they have 2 or 3 *correct* exercises and is awarded a `FAIL` otherwise. Note that warnings do not cause an exercise to be treated as incorrect.

Sample Marks File	Corresponding Output
Command line argument: `marks1`	

`2121211 ex1 5 0`
`2121211 ex2 5 0`
`2121211 ex3 5 0`
`2233455 ex1 5 0`
`2233455 ex2 5 1`
`2233455 ex3 0 1`
`2277688 ex1 4 0`
`2277688 ex2 3 0`
`2277688 ex3 2 1`
`2277689 ex1 5 0`
`2277689 ex2 5 0`
`2277689 ex3 1 1`
	`2121211 30.0 PASS`
	`2233455 18.0 PASS`
	`2277688  3.5 FAIL`
	`2277689 20.0 PASS`

20. What does this Perl print and why?

```
@a = (1..5);
@b = grep { $_ = $_ - 3; $_ > 0} @a;
print "@a\n";
print "@b\n";
```

21. What does this Perl print?

```
@vec = map { $_ ** 2 } (1,2,3,4,5);
print "@vec\n";
```

22. Write a Perl program, `tags.pl` which given the URL of a web page fetches it by running *wget* and prints the HTML tags it uses. The tag should be converted to lower case and printed in sorted order with a count of how often each is used.

    Don't count closing tags.

    Make sure you don't print tags within HTML comments.

    For example:

```
$ ./tags.pl http://www.cse.unsw.edu.au
a 141
body 1
br 14
div 161
em 3
footer 1
form 1
h2 2
h4 3
h5 3
head 1
header 1
hr 3
html 1
img 12
input 5
li 99
link 3
meta 4
noscript 1
p 18
script 14
small 3
span 3
strong 4
title 1
ul 25
```

Note the counts in the above example will not be current - the CSE pages change almost daily.

23. Add an -f option to tags.pl which indicates the tags are to printed in order of frequency.

```
$ tags.pl -f http://www.cse.unsw.edu.au
head 1
noscript 1
html 1
form 1
title 1
footer 1
header 1
body 1
h2 2
hr 3
h4 3
span 3
link 3
small 3
h5 3
em 3
meta 4
strong 4
input 5
img 12
br 14
script 14
p 18
ul 25
li 99
a 141
div 161
```