Week 05 Tutorial Questions

COMP(2041|9044) 19T2

1. The assignment specification doesn't fully explain the assignment - what can I do?

2. How hard are the subsets?

3. What is `gitlab.cse.unsw.edu.au` and how do I want use it for ass1?

4. What does **git init** do?
   How does this differ from **legit-init**

5. What do **git add *file*** and **legit-add *file*** do?

6. What is the index in **legit** (and **git**), and where does it get stored?

7. What is a commit in **legit** (and **git**), and where does it get stored?

8. Discuss what **./legit-status** prints below?

```
$ ./legit-init
Initialized empty legit repository in .legit
$ touch a b c d e f g h
$ ./legit-add a b c d e f
$ ./legit-commit -m 'first commit'
Commited as commit 0
$ echo hello >a
$ echo hello >b
$ echo hello >c
$ ./legit-add a b
$ echo world >a
$ rm d
$ ./legit-rm e
$ ./legit-add g
$ ./legit-status
```

9. Write a Perl function which takes an integer argument **n** and reads the next **n** lines of input and returns them as a string.

10. Write a Perl program which given the name of a C function searches the C source files (*.c) in the current directory for calls of the function, declarations & definitons of the function and prints a summary indicating the file and line number, in the format below. You can assume functions are defined with the type, name and paramaters on a single non-indented line. You can assume function bodies are always indented.

    You don't have to handle multi line comments. Try to avoid matching the function name in strings or single line comments. For example:

```
$ cat half.c
double half(double x) {
    return  x/2;
}
$ cat main.c
#include <stdio.h>
#include <stdlib.h>

double half(double x);

int main(int argc, char *argv[]) {
    return  half(atoi(argv[1]));
}
$ ./print_function_uses.pl half
a.c:1 function half defined
half.c:1 function half defined
main.c:4 function half declared
main.c:7 function half used
```

11. Write a Perl program which given a C program as input finds the definitions of single parameter functions and prints separately the function's type, name and the parameters name & type. Assume all these occur on a single non-indented line in the C source code. You can assume function bodies are always indented. Allow for white space occuring anywhere in the function header. You can assume that types in the program don't contain square or round brackets. For example:

```
$ cat a.c
double half(int *x) {
    return  *x/2.0;
}
$ ./print_function_types.pl a.c
function type='double'
function name='half'
parameter type='int *'
parameter name='x'
```

12. Write a Perl script C_include.pl which given the name of a C source file prints the file replacing any '#include' lines with the contents of the included file, if the included file itself contains a #include line these should also be processed.
Assume the source files contain only quoted ("") include directives which contain the files's actual path name. For example:

```
$ cat f.c
#include "true.h"

int main(int argc, char *argv[]) {
    return  TRUE || FALSE;
}
$ cat true.h
#define TRUE 1
#include "false.h"
$ cat false.h
#define FALSE 0
$ ./C_include.pl f.c
#define TRUE 1
#define FALSE 0

int main(int argc, char *argv[]) {
    return  TRUE || FALSE;
}
```

13. Modify C_include.pl so that it handles both "" and <> directives. It should search the directories `/usr/include/`, `/usr/local/include` and `/usr/include/x86_64-linux-gnu` for include files specified in <> directives and for files specified in "" directives which do not exist locally. For example:

```
$ cat g.c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("hello world\n");
    exit(0);
}
$ ./C_include.pl g.c|head
./C_include.pl: can not find: bits/libc-header-start.h
/* Define ISO C stdio on top of C++ iostreams.
   Copyright (C) 1991-2018 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be usefu
l,
```

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.

14. Write a Perl program `source_count.pl` which prints the number of lines of C source code in the current directory. In other words, this Perl program should behave similarly to `wc -l *.[ch]`. (Note: you are not allowed to use `wc` or other Unix programs from within the Perl script). For example:

```
$ ./source_count.pl
   383 cyclorana.c
   280 cyclorana.h
    15 enum.c
   194 frequency.c
   624 model.c
   293 parse.c
   115 random.c
    51 smooth.c
   132 util.c
    16 util.h
   410 waveform.c
  2513 total
```

15. Write a Perl program, `cut.pl` which takes three arguments, *n*, *m* and a file name. It should print characters *n*-*m* of each line of the file. For example:

```
$ ./cut.pl 1 8 file
```

should print the 8 characters of every line in `file`.

Implement a version of the program which invokes `/usr/bin/cut` and a version which performs the operations directly in Perl.

16. Implement a Perl script to solve the marks-to-grades problem that was solved as a shell script in a previous tutorial. Reminder: the script reads a sequence of (studentID,mark) pairs from its standard input and writes (studentID,grade) pairs to its standard output. The input pairs are written on a single line, separated by spaces, and the output should use a similar format. The script

should also check whether the second value on each line looks like a valid grade, and output an appropriate message if it doesn't. The script can ignore any extra data occuring after the mark on each line.
Consider the following input and corresponding output to the program:

Input

```
2212345 65
2198765 74
2199999 48
2234567 50 ok
2265432 99
2121212 hello
2222111 120
2524232 -1
```

Output

```
2212345 CR
2198765 CR
2199999 FL
2234567 PS
2265432 HD
2121212 ?? (hello)
2222111 ?? (120)
2524232 ?? (-1)
```

17. Write a program `addressbook.pl` that reads two files `people.txt` and `phones.txt` containing data in CSV (comma-separated values) format and uses this data to print an address book in the format below:

```
$ cat people.txt
andrew,Andrew Taylor,42 Railway St,Petersham
arun,Arun Sharma,94 Leafy Close,Brisbane
gernot,Gernot Heiser,64 Trendy Tce,Newtown
jas,John Shepherd,16/256 Busy Rd,Alexandria
$ cat phones.txt
jas,home,9665 6432
arun,work,9385 5518
jas,work,9385 6494
arun,home,(07) 9314 6543
andrew,work,9385 4321
arun,mobile,0803 123 432
$ ./addressbook.pl
Andrew Taylor
42 Railway St, Petersham
Phones: 9385 4321 (work)

Arun Sharma
94 Leafy Close, Brisbane
Phones: 0803 123 432 (mobile), (07) 9314 6543 (home), 9385 5518 (work)

Gernot Heiser
64 Trendy Tce, Newtown
Phones: ?

John Shepherd
```

Assume that there are only three types of phone (mobile, home and work) and we always display them in that order.

Hint: because the phone types are fixed, login name and phone type together can be the key used to look up a number. In this situation some suitable separator is used to create a composite, unambiguous key for the hash.