EECS 398: Computing for Computer Scientists

Practice Final Exam

Honor Code

I have neither given nor received any unaggregated aid on this exam.

______________________________ (Print Name)

______________________________ (Sign Name)

______________________________ ← WRITE THIS VERY VERY VERY CLEARLY, PLEASE.
(uniqname)

You have 80 minutes to complete this exam.

Mark your answers clearly and write neatly. If we can't read it, we can't grade it.

Remember to fill out your Name and Uniqname on every page of this exam.

If you are uncertain about what a question is asking, state your assumptions and give the best answer possible.

Question 1	/2	Question 6	/2
Question 2	/2	Question 7	/2
Question 3	/2	Question 8	2/2
Question 4	/2	Question 9	/2
Question 5	/2	Question 10	/2

Final Score	/20

Name: ______________________ Uniqname: ____________________

1. Introduction and Virtual Machines

When setting up the virtual machine for this class, we had you install the Guest Additions.
Give an example of feature enable enabled by Guest Additions.

The readings presented two opposing views on “command-line bullshittery”.
Give an argument either for or against learning modern “command-line bullshittery” based on your experiences in this class.

2. Unix / Scripting

When you type “ls” into a terminal, the program that actually runs is “/bin/ls”.
What environment variable helps turn “ls” into “/bin/ls”?

We introduced several special shell variables over the course of the term.
Pick any <u>one</u> of “$?”, “$_”, or “$#” and explain what that variable does.

3. Editors

Explain how to do each of the following operations in these editors

vim	**Emacs**
Save File	Save File
Quit without saving edits	Quit without saving edits
Save and quit (two commands fine)	Save and quit (two commands fine)
Find the text “TODO”	Find the text “TODO”

Fill in the missing part of the following command

```
$ cat input.txt
One one one
$ sed s/___/___/___ input.txt # replace all ‘one’ with ‘two’
One two two
```

4. Revision Control Basics and Git

`git add` moves a file from the ______________________ to the _____________________.

`git commit` moves a file from the _____________________ to the _____________________.

The following is an excerpt from a terminal session, fill in the missing commands:

```
$ mkdir /tmp/g
$ cd /tmp/g
$ vi main.c
$ gcc main.c
$ ___________________
Initialized empty Git repository in /tmp/g/.git/
$ ___________________
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

      main.c

nothing added to commit but untracked files present (use "git add" to
track)
$ ___________________
$ ___________________
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

      new file:   main.c
```

```
$ ______________________
[master (root-commit) 6ce31c7] Initial Commit
 1 file changed, 2 insertions(+)
 create mode 100644 main.c
$ ______________________
On branch master
nothing to commit, working directory clean
```

5. Shells II, Unix Tools & Philosophy

Given this file list:

```
$ ls
Binary_tree.cpp         filter_test             recursive.cpp           test_helpers.h
Binary_tree.h           filter_test.cpp         recursive.h             tree_insert_test
Makefile                p2-tests                simple_test             tree_insert_test.cpp
Recursive_list.cpp      p2-tests.cpp            simple_test.cpp
Recursive_list.h        p2.cpp                  simple_test.out.correct
eecs280-w15-p2.tgz      p2.h                    test_helpers.cpp
```

Using shell glob(s) complete the following command:

```
$ ls ______________________
filter_test      p2-tests         simple_test      tree_insert_test
```

Using regular expressions, complete the following command:

```
$ ls | grep ____________________________________________________________________
filter_test      p2-tests         simple_test      tree_insert_test
```

You may find some parts of the grep man page helpful for this. You may add additional pipes if it is helpful.

```
-C[num, --context=num]
        Print num lines of leading and trailing context surrounding each
        match.  The default is 2 and is equivalent to -A 2 -B 2.  Note:
        no whitespace may be given between the option and its argument.
-c, --count
        Only a count of selected lines is written to standard output.
-v, --invert-match
        Selected lines are those not matching any of the specified pat-
        Terns.
-w, --word-regexp
        The expression is searched for as a word (as if surrounded by
        `[[:<:]]' and `[[:>:]]'; see re_format(7)).
```

6. Build Systems

The following is a snippet from the Makefile in EECS 280 W15 project that we used the term:

```
simple_test: simple_test.cpp p2.cpp Recursive_list.cpp Binary_tree.cpp \
             recursive.cpp test_helpers.cpp
       g++ -Wall -Werror -pedantic -O2 \
             simple_test.cpp p2.cpp Recursive_list.cpp Binary_tree.cpp \
             recursive.cpp test_helpers.cpp -o simple_test
```

After building, a student edits “simple_test.cpp” and re-runs make.

Does make rebuild “simple_test”, why or why not?

After building, a student edits “p2.h”, which is #include’d by p2.cpp, and re-runs make.

Does make rebuild “simple_test”, why or why not?

7. Debuggers

During a debugging session, gdb prints…

```
(gdb) run
Breakpoint 2, main () at test.c:11
11 temp = add ( values[ j ], values[ k ] );
(gdb)
```

If you type “next”, gdb will…

A. Enter, but not execute, the add() function
B. Execute the whole add() function
C. Print an error

Name: ______________________ Uniqname: ____________________

You are debugging the following program with gdb:
(no, this is not a very useful program)

```
$ cat main.c | nl -ba
     1	#include <stdio.h>
     2	
     3	#ifdef NDEBUG
     4	#define DBG(...)
     5	#else
     6	#define DBG(...) printf(__VA_ARGS__)
     7	#endif
     8	
     9	int recurse(int add_me) {
    10		DBG("add_me: %d\n", add_me);
    11		if (add_me == 1) {
    12			return add_me;
    13		}
    14		return recurse(add_me + add_me);
    15	}
    16	
    17	int main() {
    18		printf("%d\n", recurse(2));
    19	}
    20	
```

When you run the program, this is the output:

```
$ gcc main.c && ./a.out
add_me: 2
add_me: 4
add_me: 8
add_me: 16
[ … many lines skipped … ]
add_me: 0
add_me: 0
add_me: 0
Segmentation fault: 11
```

At some point, add_me became 0 and never stopped. If you then run your program under gdb, **what command can you give gdb to stop your program once add_me first becomes 0?**

```
$ gdb -q ./a.out
Reading symbols from ./a.out...(no debugging symbols found)...done.

(gdb) _______________________
```

Name: ____________________ Uniqname: ____________________

8. Spring Break

Just kidding. Who asks questions during spring break? That's just cruel.

9. Using Git Effectively

The "git branch" command…
Circle all that apply

A. Creates a new commit
B. Changes the contents of the working directory
C. Changes the contents of the staging area
D. Changes the contents of the .git folder
E. Creates a new "pointer" to a commit

The "git checkout" command…
Circle all that apply

A. Creates a new commit
B. Changes the contents of the working directory
C. Changes the contents of the staging area
D. Changes the contents of the .git folder
E. Creates a new "pointer" to a commit

10. Profiling

Your code doesn't produce the output you were expecting, the best tool to check and track it down is:

A. gprof
B. valgrind
C. gdb
D. gcov

You're setting up your code for profiling using the GNU profiler, aka "gprof", so you add "-pg" to the compiler flags when you build your program.
Profile data is stored in a file named "gmon.out", what generates the "gmon.out" file?

A. The gprof utility, i.e. running, `gprof myprogram`
B. Your program itself, i.e. running, `./myprogram`
C. The gcov utility, a companion utility that helps profiling, i.e. running, `gcov myprogram`