UCLA Computer Science 35L Final Exam - Winter 2019
Open book, open notes, closed computer.
100 points total, 180 minutes total, 1 point = 1.8 minutes
Each top-level question is 10 points and should take 18 minutes.
Write answers on the exam.

Name:_____

Student ID:_____

1	2	3	4	5	6	7	8	9	10	total

1. You are Dr. Eggert with username ‘eggert’ and group ‘csfac’, and are trying to review some basic concepts. Your current working directory is '~/exam/'.

There are two files you are particularly interested in: foo and bar. The following are the contents and permissions on those files. (The flag -i prints the file's file inode number.)

```
$ cat part1/foo
This is the finals for CS35L
I am foo
$ cat part2/bar
Checking hard and symbolic links
I am bar
$ ls -li part1/
total 0
3165230 -rw-r--r-- 1 eggert csfac 38 Mar 15 02:37 foo
$ ls -li part2/
total 0
3165232 -r--r--r-- 1 eggert csfac 42 Mar 14 14:35 bar
```

You execute the following commands :

```
$ ln part1/foo baz
$ ln -s part2/bar qux
```

Answer the following questions based on the information above. If you think there is any error while executing the commands above or the commands mentioned later, you can briefly describe the error.

1a (1 point). What do the above commands mean?

1b (1 point). Write a shell command or commands to count the total symbolic links from the current directory up to 2 levels of directories below the current directory. (Hint : 'find -maxdepth N' descends at most N directory levels below the command line arguments.)

1c (2 points). What will be the inode (index-node) of the files baz and qux respectively?

1d (2 points). After executing the below commands, what will be the contents of the files baz and qux?

```
$ mv part1/foo part1/foobar
$ mv part2/bar part2/barfoo
```

1e (2 points). You execute the commands in step (d). What will be the content of the files baz and qux after executing the below commands?

```
$ rm part1/foobar
$ rm part2/barfoo
```

1f (2 points). For this question assume that we do not execute the commands in parts (d) and (e).

What will be the output of the below mentioned commands?

```
$ echo “Updating files” >> baz
$ echo “Updating files” >> qux
$ cat part1/foo
$ cat part2/bar
```

2. We have a directory with many genetic data files, each named like this:

    FirstnameLastname.gene_ID(chromosome_number)

FirstnameLastname contains only alphabetic characters (no spaces). Firstname and Lastname both begin with one capitalized letter, and both Firstname and Lastname must be at least 2 letters long. The gene_ID is a combination of letters and digits and chromosome_number is a number from "1" to "22". E.g.:

  JoeBruin.ENSG00000112137(6)

A FirstnameLastname pair may appear in multiple files, and a gene_ID may appear in multiple files.

2a (3 points). Assume that we decide to list the contents of the directory and store the result in a variable. Which of the following would successfully accomplish this (if any)? Explain your reasoning for each. (Assume that the directory path is stored in the variable GDIR).

  A. LIST=`ls $GDIR`
  B. LIST='`ls $GDIR`'
  C. LIST="`ls $GDIR`"

2b (2 points). Assume for this part, that one of the above commands worked, and that we now have the results of the ls command in LIST. Write a Bash script (you may skip the shebang line) that outputs to stdout every filename in LIST that does not have read permission granted. (Note: the "-r FILE" option of the test command returns true if FILE exists and read permission is granted.)

2c (1 point). Now assume that we did not do the steps in A and B. Instead, we have decided to run our ls command and pipe it into a grep command. Given our specific problem, would it be more advantageous to use Basic Regular Expression, or Extended Regular Expression? Briefly explain.

2d (4 points). Using whichever mode (BRE or ERE) you chose in part C, write a regular expression that could be used in a grep command to output only the files that meet the following criteria. Lastname begins with an "R", "C", "S", or "Q". The gene ID begins with "ENSG" followed only by 1 or more numerical digits. Lastly, the chromosome number has at least 2 digits.

3a. A Makefile has the following contents:

```
move : car
car : fuel.o wheels.o car.o
        g++ -o car fuel.o wheels.o car.o
wheels.o : wheels.cpp wheels.h
        g++ -c wheels.cpp
fuel.o : fuel.cpp fuel.h
        g++ -c fuel.cpp
car.o : car.cpp wheels.h fuel.h
        g++ -c car.cpp
clean :
        rm -f wheels.o fuel.o car.o car
```

Assume that all the header files are present in the same directory as the Makefile. Look at the contents of the Makefile thoroughly and answer the following questions:

3a1 (2 points). What happens when you run the command ‘make all’ on the terminal? Explain.

3a2 (1 point). Why do we need makefiles? List any two distinct reasons.

3b (2 points). The first two lines of a patchfile (exam_patch.patch) are as follows:

```
---abc/cs35l/Documents/exam.c
+++xyz/cs35l/Documents/exam.c
```

Given that exam_patch.patch is present in the cs35l directory, write the appropriate patch command to update exam.c if the current working directory is Documents.

3c (5 points). A Python 3 file (armstrong.py) has the following statements:

```
    def numDigits(n):
        #Write your code here

    def isArmstrong(n):
        #Write your code here

    n = int(input())
    print("Is", n, "an Armstrong number?", isArmstrong(n))
```

Complete this Python 3 file by defining the isArmstrong(n) and numDigits(n) functions such that the Python script does not throw any errors when it is run. Throw an appropriate exception for negative numbers. It is guaranteed that the input is an integer.

Note:

a) isArmstrong(n) should return "Yes" or "No".

b) An Armstrong number of x digits is a positive integer such that the sum of the nth power of its digits is equal to the number itself.

Examples:

153 is an armstrong number because 13 + 53 + 33 = 153 (the number itself).

Similarly, 1634 is an Armstrong number too since 14 + 64 + 34 + 44 = 1634.

4. Look at the code snippet below and write what gets printed at every line. Use underscores for spaces:

```
  #include <stdio.h>
  char *c[] = {"the", "quick brown fox", "jumped", "over the", "lazy dog"};
  char **cp[] = {c+3, c+2, c+1, c, c+4};
  char ***cpp = cp;

  int main(void)
  {
      printf("%s\n", **(cpp+3));   //Line 1
      printf("%s\n", **(cp+4)+4);  //Line 2
      printf("%s\n", **(++cpp));   //Line 3
      printf("%s\n", **cp);        //Line 4
      printf("%s\n", **(++cpp)+5); //Line 5
      return 0;
  }
```

4a (2 points). Line 1 output:

_____

4b (2 points). Line 2 output:

_____

4c (2 points). Line 3 output:

_____

4d (2 points). Line 4 output:

_____

4e (2 points). Line 5 output:

_____

5. Consider the following program and two text files:

question.c:

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int
main(void)
{
  int fd1,fd2,sz1,sz2;
  char *c1 = (char *) calloc(100, sizeof(char));
  char *c2 = (char *) calloc(100, sizeof(char));

  fd1 = open("content1.txt", O_RDWR);
  fd2 = open("content2.txt", O_RDONLY);

  if (fd1 < 0){
     perror("Error in opening content1.txt");
     exit(1);
  }
  if (fd2 < 0){
     perror("Error in opening content2.txt");
     exit(1);
  }
  sz1 = read(fd1, c1, 12);
  sz2 = read(fd2, c2, 8);
  printf("fd1 = %d, fd2 = %d \n", fd1, fd2);
  printf("Called read which returned %d \n", sz1);
  printf("Content read: %s \n", c2);
  write(fd2, c1, sz1);
  write(fd1, c2, sz2);
  close(fd1);
  close(fd2);
}
```

content1.txt:

```
Lord_of_the_Rings
```

content2.txt:

```
Charlie_Chaplin
```

Each text file consists of just one line; underscores (_) denote spaces, and the text lines do not have leading spaces.

5a (1 point). Will the above program execute? If yes, what is the output of the program? If no, why would it not execute?

5b (1 point). What are the contents of content1.txt and content2.txt post execution of the above code? (Rewrite the original contents of the file if you said that the program would not execute.)

Let’s say I change the open functions in this problem as given below. (The rest of the code remains unchanged.)

```
fd1 = open("content1.txt", O_RDWR | O_APPEND);
fd2 = open("content2.txt", O_RDWR);
```

After execution of the modified program on the original data:

5c (1 point). What are the contents of content1.txt?

5d (1 point). What are the contents of content2.txt?

5e (2 points). What functions would you change in the above programs to move the program from using system calls to inbuilt library functions? Mention both the functions you would change and their equivalent replacements.

5f (2 points). Comment on the performance of the code if you had written it using the library functions you listed in part (e). Which would be faster and why?

5g (2 points). Rewrite the above code to print the entire contents of content1.txt and content2.txt using system salls. Do not use functions printf(), puts(), etc.

6. Parallel Computing & Synchronization

6a (2 points). Overhead of mutex lock and unlock. In the table below, you are given 3 different snippets of using a mutex to solve critical sections. Assume they are running on the same machine. Please give the rank of their running time (e.g. “A takes more time than B, which takes more time than C”) and briefly explain why.

```
/* A */
int N = 10000000;
pthread_mutex_lock(&m);
for (int i = 0; i < N; i++)
  sum++;
pthread_mutex_unlock(&m);

/* B */
int N = 10000000;
for (int i = 0; i < N; i++) {
  pthread_mutex_lock(&m);
  sum++;
  pthread_mutex_unlock(&m);
}

/* C */
int N = 10000000;
int local = 0;
for (int i = 0; i < N; i++)
   local++;
pthread_mutex_lock(&m);
sum += local;
pthread_mutex_unlock(&m);
```

6b. Multithreaded time. Here are the outputs of the 'time' command when the multi-threaded version of the sort program above was executed on the 16 core server.

```
Threads used : 1 real 0m31.698s user 0m31.607s sys 0m0.018s
Threads used : 2 real 0m16.403s user 0m31.796s sys 0m0.017s
Threads used : 4 real 0m10.803s user 0m31.750s sys 0m0.017s
Threads used : 8 real 0m7.012s   user 0m32.989s  sys 0m0.015s
```

Please answer the following questions:

6b1 (1 point). What could be a possible reason for the 'user' time being almost the same in all the cases?

6b2 (1 point). Why is the 'user' time greater than the 'real' time in some of the cases?

6b3 (1 point). What could happen to the above times, if the server administrators set a policy that allowed your programs to use at most 2 of the cores?

6c (5 points). Multithreaded Sort. You are required to sort a set of 1 billion integer scores in descending order. This can take a very long time if your program is single threaded. Your task is to write a multi-threaded program, using pthreads, that distributes the workload across a set of 16 threads. Some starter code is given below. The standard library qsort() function, with a compare() function, should be used to do the sorting per thread. You don't have to rewrite anything that is already given below.

```
#include <stdio.h>
#include <math.h>
enum { SCORE_COUNT = 1000000000 };
enum { NTHREADS = 10 };

//Your code here [1]

int main(int argc, char** argv) {
  int *scores = (int*) malloc(sizeof(int) * SCORE _COUNT);
  if(!scores) {
    fprintf(stderr, “Cannot allocate memory!\n”);
    exit(-1);
  }

  // Assume the following function call loads values into 'scores'
  LoadScores(scores);


  // Assume the following function call prints the sorted values
  PrintScores(scores);
  free(scores);
  return 0;
}
```

7. You have maintained a C project. The below is the output of ls command when applied on your project directory:

```
$ ls
main.c    foo.c    bar.c    foo.h    bar.h    type.h
```

And you know that

foo.h includes type.h

main.c includes foo.h and bar.h

bar.c does not include any header files.

7a (6 points). Write a Makefile that:

A. When you type make, it will generate an executable called criu compiled from main.c, foo.c and bar.c.

B. When you change one of the .c or .h files, only minimal number of the .c files got recompiled and the newly created hello can correctly reflect your change.

C. When you type make clean, it will remove all the intermediate file generated by the compilation process.

D. When you type make tarball, it will generate a bzip2 compressed file named clean.tar.gz that contains all the .c file .h file listed by the ls command and the Makefile itself.

7b (4 points). A buggy Makefile may overwrite your own source code! For example, consider the following Makefile that compiles hello.c to executable hello:

```
default: hello.c
        gcc -o hello.c hello.c
```

When you type 'make', the hello.c will be overwritten with the hello.c executable.

You are afraid that there may be potential bugs in the Makefile written in part (a) which will overwrite the existing source code.

Add a new target called protection into the Makefile you have written in part (a) to prevent Makefile from overwriting your source code files. Once you have typed make protection in the terminal to execute the target, even if there is a bug in the Makefile that will overwrite the source code files, the corresponding target will fail.

You are not allowed to move any files outside the current directory or create any new files.

Write down the protection target and related target (if any) below.

8. This question has multiple-choice subproblems.
For each such problem, write the single best answer,
which will be (i), (ii), (iii), (iv), or (v).

8a (2 points).

(i). If you don't trust your server, you cannot use OpenSSH to connect to it, as it can easily corrupt your client.
(ii). If you don't trust your client, you can still use OpenSSH to connect to a trusted server.
(iii). If you don't know the name or IP address of your server, you can use OpenSSH to discover this info in a secure way.
(iv). If you don't know the name or IP address of your client, you can still use OpenSSH to connect to a server.
(v). If you don't trust your network, you can still use OpenSSH to discover whether your server is running.

8b (2 points).

(i). ssh-agent improves security by making a copy of private keys.
(ii). ssh-agent acts on your behalf by running on the server and executing commands there, under your direction.
(iii). Even if the attacker surreptitiously replaces the ssh-agent program with a modified version, your communications will still be secure.
(iv). ssh-agent eliminates all need for password authentication when communicating to the SEASnet GNU/Linux hosts.
(v). If you successfully use a typically-configured ssh-agent and then log out from the client and then log back in again, you can then connect to the same SSH server again without typing any additional passwords or passphrases.

8c (2 points).

(i). OpenSSH typically uses public-key encryption for authentication, because private-key encryption is less secure.
(ii). OpenSSH typically uses private-key encryption for data communication, because public-key encryption is less efficient.
(iii). When you run 'ssh', it chooses its authentication key randomly from a large key space, to make eavesdropping harder.
(iv). The OpenSSH client and server are essentially symmetric, so that it's easy and common to use the same program as either a client or a server.
(v). Once your private keys are 1024 bits long, there's no point making them any longer, as they're impossible to break.

8d (2 points).

(i). OpenSSH is not limited to just one client-server connection; for example, a team of five people can use OpenSSH to communicate information to each other.

(ii). For security, OpenSSH refuses to connect to programs written by other people; for example, a client running the OpenSSH code will connect only to a server running the OpenSSH code.

(iii). Although port forwarding can be used to display from an OpenSSH server to an OpenSSH client, the reverse is not possible: you cannot use port forwarding to display from an OpenSSH client to an OpenSSH server.

(iv). For security, port forwarding cannot be chained: that is, you cannot ssh from A to B, and then from B to C, and use port forwarding to let a program run on C and display on A.

(v). When using port forwarding when connecting to SEASnet, one should take care not to create a forwarding loop, as this can lead to a cycle of packets endlessly circulating on the Internet.

8e (2 points).

(i). It’s not a good idea to connect to a SEASnet GNU/Linux server and use GPG on the server to sign a file, because then an attacker on the network can easily snoop your GPG passphrase.

(ii). A detached signature file must be protected as securely as the private key it’s based upon; otherwise an attacker will be able to forge your signature more easily.

(iii). When generating a key pair it’s important to use a private entropy pool on SEASnet, not the shared pool that everybody can access, because otherwise an attacker on SEASnet might be able to guess your key more easily by inspecting the public entropy pool.

(iv). Exporting a GPG public key to ASCII format neither improves nor reduces its security.

(v). Because a detached cleartext signature isn’t encrypted, it is easily forged by an attacker with access to the file being signed.

9. Ava and Max are working on a project, using git as a versioning tool. The project is in a repository called Asymmetry, and is stored locally, since it is just a small project. Ava has checked out from the master into a branch called issue42, and Max into a branch called issue49 (both these branches are set to track the master) The state of the repo at this time is that it has three commits, arranged as follows:

  c41fa is the oldest (original) commit.
  8ac11's parent commit is c41fa.
  6ab77's parent commit is 8ac11.
  6ab77 is tagged by 'master', 'issue42', and 'issue49'.

At the commit 6ab77, the repository has the following files:

  Application.py
  dbconn.py
  frontend.py
  index.html
  README.md
  structure.css

Ava, in her branch issue42, creates two files - format.py and clean.py; modifies frontend.py, ensures her code works as expected, and then runs the following commands:

  git status
  git add format.py frontend.py
  git clean --force
  git commit -m "issue42: adding prettify patch"
  git checkout master
  git merge issue42 (this merges the code with the master branch)
  git tag -a iss42 -m "commit for issue42"

While Ava is working on her fixes, Max is also developing his end of the code, on his branch. He modifies dbconn.py and creates test_data.csv and testpage.html, tests his code, and runs the following commands:

  git status
  git add dbconn.py test_data.csv
  git commit -m "issue49: testcases for the database"

However, he does not run the merge commands:

  git checkout master
  git merge issue49

until the next day, by which time Ava is done with her development (and git commands).

[continued on next page]

9a (1 point). What command(s) should Max run before running the checkout & merge commands above to ensure that he has (and tests) the latest version of the code from the master, including Ava's? Assume he knows there will be no conflict with their files. Select any one:

A. git fetch
B. git fetch && git status
C. git fetch && git merge
D. git revert

9b (3 points). List the files in the repository (in the master branch) once Ava, Max have pushed their code. Indicate which file(s) are modified and which have been added.

9c (3 points). A week or so later, QA finds a problem with the way Ava has fixed issue 42. She has to fix it again, but unfortunately, her laptop disk has been reformatted since then and she's lost all her local changes. What's the best way Ava can get her specific fix back and work on it? Specify the command(s) for the same.

9d (3 points). There is also an issue, it is later discovered, with the way Max has written some of his testcases. Unfortunately, he hadn't Ava's foresight when he did the git push, so he must do things differently. How can he go about retrieving his commit ID, given that he remembers that he put the issue ID (issue49) first, and the word "testcases" somewhere later, in his commit message? (Use any git and/or Linux commands necessary.)

10 (10 points). Consider the following ACM TechNews summary of an article published in the New York Times on March 14. Explain the relationship between this article's topic and the ACM TechNews topic that you summarized in your solution to Assignment 10. Or, if the two topics are completely unrelated, explain why they are unrelated.

"Facebook's Daylong Malfunction Is Reminder of Internet's Fragility"

Facebook said it has corrected a technical error that caused a nearly 24-hour-long service interruption for Instagram, WhatsApp, Messenger, and other Facebook properties this week. According to a Facebook spokesperson, a "server configuration change" had a cascading effect throughout the company's network, triggering a recurrent loop of problems that kept escalating. The incident serves as a reminder that the Internet can still be hobbled by human error. For years, Facebook has recruited engineers on the idea that, within weeks, they can release computer code that reaches billions of people, especially as the company devises a strategy to consolidate the infrastructure of its "family of apps." However, the outage demonstrated that the more tightly intertwined a network becomes, the more likely a small technical problem caused by a single employee can have far-reaching consequences.