UCLA Computer Science 35L final exam - Spring 2020

Name:_____

Student ID:_____

180 minutes total, 100 points total, so 1 point = 1.8 minutes.
Open book, open notes, open computer. Answer all questions yourself, without assistance from other students or outsiders.

The exam is not easy, and you are not expected to answer all the questions completely. In your answers, overall approach and intuition will count more than trivial detail. Budget your time while taking the exam. It may help to skip questions that are harder than their point count would suggest.

Print the exam, read the first page, then write your starting time on the first page. Then take at most 180 minutes to answer the questions and write your answers on the exam. (CAE students with x% extra time should add to the 180 minutes accordingly.) When you're done, write your finishing time on the first page, sign the first page, scan the completed exam, and upload your scans to CCLE Gradescope as quickly as you can. If you lack a scanner, carefully photograph the sheets of paper with your cell phone and upload the photographs. Save your filled-out exam until the class is over, and do not give or show it to anybody other than an instructor or TA.

Alternatively, you can use a notepad computer to write your answers into the PDF and upload the modified PDF. Or you can read the exam on your laptop's screen, write your answers on blank sheets of paper (preferably $8\frac{1}{2}''\times11''$) with one page per question, and upload the scanned sheets of paper; at the end of the exam, you should have scanned and uploaded as many photographs as there are questions (if you do not answer a question, scan a blank sheet of paper as the answer).

The exam is open book, open notes, and open computer. You can use your laptop to use a search engine for answers, and to run programs designed to help you answer questions. However, do not use your computer or any other method to communicate with other students or outsiders, or anything like that. Communicate only via CCLE and Gradescope to obtain your exam and upload your scanned results, or via Zoom or email with the instructor or TAs.

_____ Time (Los Angeles time) you started the exam

_____ Time that you ended the exam

IMPORTANT Before submitting the exam, certify that you have read and abided by the above rules by signing and dating here:

Signature:_____ Date:_____

1. Suppose you've written your own version of the cat command that, when called, outputs this to the terminal:

```
   \    /\
    )  ( ')
   (  /  )
    \(__)|
```

Assume that the full path name for the the directory where the executable for your personal cat command is stored is:

```
   /u/cs/ugrad/bruin/better_commands/animals
```

1a (3 points). Suppose you're working on lnxsrv06 and decide this version of the command is good enough to be the default version of 'cat' for yourself. Write a one line command that will ensure that when you invoke the 'cat' command, you get back the personal version above.

1b (3 points). Your friends aren't nearly as impressed by your cat command as you thought they'd be, so you now decide to retire the command. You do not delete the executable; that is, the file stored in the better_commands/animals folder still exists. Write a command that will ensure that when you now invoke the 'cat' command, *after* the events of (a), you get back the actual Linux concatenate command.

2 (6 points). You and your friend decide that you've had enough of seeing USC on the news, and you'd like to create a spam filter to filter out the unneeded acronym. Write an extended regular expression that matches any string that consists entirely of the uppercase letters U and S and C, but with the following restrictions:

* The string must have a length of six.

* The string must have exactly two copies of each letter (two U, two S, two C).

* The string must start with the letter U and end with the letter C.

* The string must be the concatenation of two substrings, and in each substring every U must precede every S, and every S must precede every C. (The two substrings can differ in length and one substring can be empty.)

Here are some examples of valid matches (one on each line):

```
USCUSC
USUSCC
UUSSCC
```

Briefly explain why your regular expression is correct.

Hint: Try writing out the combinations as a tree that will help in deriving the regular expression.

3a (2 points). Write a Bash command that implements a configuration of the Caesar cipher (a very simple substitution cipher). The command should comply with the following specifications:

 * Input is from stdin.
 * Output is to stdout.
 * Only substitute lowercase and uppercase ASCII letters.
 * Use a right rotation of three places i.e. substitute ‘a’ with ‘d’, ‘b’ with ‘e’, ‘w’ with ‘z’, ‘x’ with ‘a’, and so on. Similarly, do the same with uppercase letters: substitute ‘A’ with ‘D’, ‘B’ with ‘E’, ‘W’ with ‘Z’, ‘X’ with ‘A’, and so on.

For example:

```
   Input: afC2xA
  Output: diF2aD
```

3b (2 points). Write another Bash command such that the previous specifications apply, except:

 * Substitute lowercase letters ‘x’ through ‘z’ with the ‘@’ character instead of their previous substitutions.

 * Substitute uppercase letters ‘A’ through ‘W’ with the ‘@’ character instead of their previous substitutions.

For example:

```
   Input: afC2xA
  Output: di@2@@
```

4. The following is a patch for a project you're working on.

```
  --- a/src/coolors.c  2020-05-31 09:14:41.000000000 -0700
  +++ b/src/coolors.c  2020-05-31 09:14:37.000000000 -0700
  @@ -1,12 +1,17 @@
  +/* Checks that an RGB parameter is valid */
  +int is_valid_rgb(int val) {
  +    return val >= 0 && val <= 255;
  +}
  +
   /* Convert RGB color to HEX format */
   int rgb2hex(int r, int g, int b) {
  -    if (r >= 0 & r <= 255) {
  +    if (is_valid_rgb(r)) {
           fprintf(stderr, "Invalid red value");
           return -1;
  -    } else if (g >= 0 & g <= 255) {
  +    } else if (is_valid_rgb(g)) {
           fprintf(stderr, "Invalid green value");
           return -1;
  -    } else if (b >= 0 & b <= 255) {
  +    } else if (is_valid_rgb(b)) {
           fprintf(stderr, "Invalid blue value");
           return -1;
       } else {
```

4a (1 point). Explain what this patch does at a high level.

4b (1 point). How many lines are added and/or removed by the patch?

4c (2 points). Write a shell command that applies the patch. Assume coolors.c is in src/coolors.c and the patch file, named cool.patch, is in the root of the project directory.

5. Zoom is working on a Python tool that extracts info from password-protected meeting links that follow this format:

 * Links start with the base URL "https://zoom.us/j/" or "https://SUBDOMAIN.zoom.us/j/", where SUBDOMAIN is a non-empty string of lowercase ASCII letters.

 * The base URL is followed by a 9-digit conference ID (e.g., 356209714).

 * The conference ID is followed by "?pwd=", which is followed by a 32-character alphanumeric ASCII password (e.g., JZ9u3x6FVBumItd2PxOrbAApyCv860i3).

Here are some examples that match the format above:

https://zoom.us/j/356209714?pwd=JZ9u3x6FVBumItd2PxOrbAApyCv860i3
https://ucla.zoom.us/j/701495623?pwd=5915EqMgkRtaeCyrSTXouXeRnaQlSeCm

5a (2 points). Write a regular expression that matches with a valid base URL.

5b (2 points). Write a regular expression that matches with a valid 9-digit conference ID.

5c (2 points). Write a regular expression that matches with a valid 32-character alphanumeric password.

[continued on next page]

[continued from previous page]
5d (4 points). Write a Python function parse_zoom(link) that takes a valid meeting link as a string argument, extracts the conference ID and password using regex, and returns the values as a tuple. The outputs for the above examples would be:

```
('356209714', 'JZ9u3x6FVBumItd2PxOrbAApyCv860i3')
('701495623', '5915EqMgkRtaeCyrSTXouXeRnaQlSeCm')
```

Python's standard library provides the 're' module for regex operations. To search for a regular expression match in a string, use re.search(pattern, string). You can retrieve matched substrings in a regular expression with the help of capture groups and the group() function. For example,

```
import re
s = "hello world"
match = re.search("hello (.*)", s) # search for pattern in s
match.group(1) # first capture group result, i.e., "world"
```

6a (8 points). In the space available below, explain the differences between the two different 2D arrays declared below, arr1 and arr2. Focus on the underlying memory structure, any performance implications, and use-cases. Assume that this code appears at the start of a function body.

```
  const int ROWS = 5;
  const int COLS = 4;

  //arr1
  int **arr1 = malloc(ROWS * sizeof(int *));
  for (int i = 0; i < ROWS; i ++)
      arr1[i] = malloc(COLS * sizeof(int));

  //arr2
  int arr2[ROWS][COLS];
```

6b (2 points). Do the lines of code below make sense? Why or why not? Assume arr2 is the same as the one declared in part (a) above.

```
  int **arr3 = arr2;
  printf("%p", arr3[2][3]);
```

7 (4 points). On one running instance, the code below returns 10. What are two possible reasons why? Is either of those a concern and why?

```
  return read(STDIN_FILENO, buffer, 50);
```

8 (6 points). One day you decide that it is annoying and confusing that EOF is not a real character. So you decide to make a new language encoding, ASCII-with-EOF, where EOF is a real character. You do this by replacing ASCII character 28 (the file separator) with EOF since the file separator control character is very rarely used in any programming/files. You decide you won't miss it.

Now getchar() actually can return a char and read() will also fill in EOF as a character in the buffer.

What are some cons of this idea? Would you want to permanently use ASCII-with-EOF for all of your future work?

9. You are a developer at a startup working on a project written in C. Apart from the C standard library, the project also uses a custom library you wrote called ‘coolors’. Here’s a code snippet from the project:

```
#include "coolors.h"
#include <stdlib.h>
...
/* Return a random hex color */
int random_hex_color() {
    int red = random_value(0, 256);
    int green = random_value(0, 256);
    int blue = random_value(0, 256);

    // Call function in coolors.
    int hex = rgb2hex(red, green, blue);
    return hex;
}
...
```

9a (2 points). You need to send the compiled executable to quality assurance (QA) for testing. Which linking method(s) would you use here to build the program and why? Assume that QA testers use the same architecture and Linux distribution that you do.

9b (2 points). Write a gcc command to compile the coolors source code, located in coolors.c, into a shared library, libcoolors.so.

[continued on next page]

[continued from previous page]
9c (6 points). Rewrite the code snippet to dynamically load the coolors shared library solely to be used within this function. If there is a dynamic loading error and the function doesn’t have a random hex color to return, the function should return -1.

10. You are working on a software project that uses Git as the dedicated version control system. You clone the remote repository to a local directory on your computer. Your local repository has the following commit history:

C1 ← C2 (master, origin/master)

where C1 and C2 are commits and C2 is pointed to by the ‘master’ and ‘origin/master’ branch references.

The remote repository also has the following commit history:

C1 ← C2 (master)

The following questions depend on each other and should be answered sequentially.

10a (2 points). You commit a new snapshot to your local ‘master’ branch (assume the new commit is C3). What does your local commit history look like now? Make sure to draw the commit history (you can use the format above). Also, explicitly show the updated branch references and generated commits, if any.

10b (2 points). Unfortunately, someone on your team has pushed new changes to the remote ‘master’ branch before you could push your local changes. The remote repository now looks like this:

C1 ← C2 ← C4 (master)

You pull from the remote repository. What does your local commit history look like now? Make sure to draw the commit history (you can use the format above). And explicitly show the updated branch references and generated commits, if any.

[continued on next page]

[continued from previous page]
10c (3 points). Once again, before you could push your local changes, someone on your team pushes their work to the remote repository. (Your team really needs to work on communication.) The remote repository now looks like this:

```
C1 ← C2 ← C4 ← C6 (master)
```

After pulling from the remote repository, what does your local commit history look like now? Make sure to draw the commit history (you can use the format above). And explicitly show the updated branch references and generated commits, if any.

11. Assume your local commit history looks like this:

```
   C2 (feature)
  /
C1 ← C3 (master)
```

where C1, C2, C3 are commits and ‘master’, ‘feature’ are branch references.

11a (2 points). You are currently on the ‘feature’ branch. You run ‘git rebase master’. What does your local commit history look like now? Make sure to draw the commit history (you can use the format above). And explicitly show the updated branch references and generated commits, if any.

11b (1 point). You now checkout the master branch and invoke ‘git merge feature’. What type of merge occurs?

12 (6 points). After working for too long on Assignment 9 for CS 35L, one of your friends decides that the layout of git is just awful, and they simply don’t like it. In particular, they state the following:

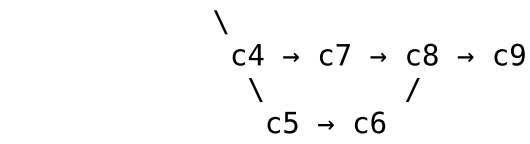
 * The .git/object file should have the entire SHA-1 hash as its file name, instead of the first two characters being used as folder names and the remaining characters used as a file name inside of the folder.

 * The files inside of .git/object should be uncompressed, so they don’t have to take the useless extra step of decompressing files.

 * Keeping all SHA-1 hashes for all git objects together inside of the .git/objects folder is too confusing; they should be separated into three different folders - .git/objects/commits, .git/objects/trees, .git/objects/blobs.

For each of the three points above, argue why your friend may be correct, or why your friend may be incorrect. Write only 1 - 2 sentences per bullet point.

[page 15]
13. For the following questions, use the commit graph below. Assume that c1 is the oldest commit, and that each commit points to its child commits.



13a (2 points). Give a valid topological ordering of this graph. You do not need to list branch names, or use the sticky start/end notation from Assignment 9.

13b (2 points). Suppose I pull a copy of commit c7 to my own local machine, and I use git log to read through commit messages for this project. Of the 10 commits, which commit messages will I not be able to read from my copy?

14. Suppose you are worried that someone has broken into your SEASnet GNU/Linux server and installed both a packet sniffer and a file sniffer. Both sniffers are malware in your server's operating system. The packet sniffer snoops on all data sent to or from the network, and the file sniffer snoops on all data being read from or written to files. While the sniffers are installed, you login into the server via SSH and do Assignment 1.

14a (1 minute). Can the attacker now get a copy of your solution to Assignment 1? Briefly explain.

14b (2 minutes). Can the attacker now pretend to be you to log into the same server via SSH? Briefly explain.

14c (2 minutes). Can the attacker now set up a fake SSH server of his own, one that you can mistakenly log into? Briefly explain.

14d (5 minutes). Suppose instead you are worried that someone has broken into your SEASnet GNU/Linux server and installed a pipe sniffer, that is, a piece of malware in your server's operating system that snoops on data going through pipes. You are worried that the pipe sniffer will get a copy of the data flowing through the pipe in the following shell script:

```
#!/bin/sh
sed 's/a/b/; /lambda/d; s/x/yy/g' | uniq -c
```

Your friend says, "Don't worry; you can easily modify your script to use ssh so that the pipe sniffer will see only encrypted data." Does your friend's remark make sense? If so, show how to modify the script along the lines that your friend suggested, so that it is as efficient as possible (for example, it works well even if a lot of data passes through the pipe); if not, explain why not.

[page 18]
19 (10 points). Consider the following discussion taken from a recent review of computer application technology:

> The doubling of the number of transistors on a chip every 2 years, a seemly inevitable trend that has been called Moore's law, has contributed immensely to improvements in computer performance. However, silicon-based transistors cannot get much smaller than they are today, and other approaches should be explored to keep performance growing.... Software performance engineering, development of algorithms, and hardware streamlining at the Top can continue to make computer applications faster in the post-Moore era. Unlike the historical gains at the Bottom, however, gains at the Top will be opportunistic, uneven, and sporadic. Moreover, they will be subject to diminishing returns as specific computations become better explored.
>
> Source:
> Leiserson CE, Thompson NC, Emer JS, et al.
> There's plenty of room at the Top:
> What will drive computer performance after Moore's law?
> Science. 2020-06-05;368(6495):eaam9744.
> https://dx.doi.org/10.1126/science.aam9744

Discuss the relationship of Leiserson et al.'s thesis about computer application development to the technology that you covered in your Assignment 10 presentation. If your Assignment 10 technology is largely unrelated to the thesis, briefly state why your technology will be unaffected even if the thesis turns out to be true, and discuss how how the thesis is related machine-learning applications that attempt to predict COVID-19 outbreaks.