UCLA Computer Science 97 midterm - Fall 2020

Name:_____

Student ID:_____

100 minutes total, 1 minute = 1 point. Open book, open notes, open computer. Answer all questions yourself, without assistance from others.

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.

You can print the exam, read the first page (if you haven't read the instructions already, which I sent you via email yesterday), then write your starting time on the first page. Then take at most 100 minutes to answer the questions and write your answers on the exam. (CAE students with x% extra time should add to the 100 minutes accordingly, getting 100+x minutes.) 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.

You can use other technology to take the exam, e.g., by using a tablet to write over the PDF. All we need is a PDF, preferably using the same layout.

You can pick the starting time for the exam. We will give you an extra 20 minutes for the overhead of downloading, printing, and/or scanning. Don't abuse this extra time: limit your own exam-taking time to 100 minutes. You must finish the exam by 24 hours after the exam is made available.

If you lack a printer, read the exam on your laptop's screen, write your answers on blank sheets of paper (preferably 8½″×11″) 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. You can type your answers if you like; all we need is a PDF that you can upload on Gradescope.

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 Piazza with the instructor or TAs. Do not communicate this exam or your answers to anybody other than the professor or the TAs, even after the exam is over.

IMPORTANT Before submitting the exam, certify that you have read and followed the above rules by signing and dating it.

_____ Date and time (Los Angeles time) you started the exam

_____ Date and time that you ended the exam

_____Signature

1. This problem asks you to write some shell scripts. Your scripts should be simple and clear. They should minimize the use of temporary files; if they do need to create temporary files they should remove the temporaries before exiting. If a script runs into any trouble, it should exit with status 0; otherwise, it should exit with nonzero status. Your scripts can use any shell control structures such as functions and loops, but should limit themselves to a small set of utilities, such as:

```
[ . : break cat cd chmod comm continue cp diff echo env eval exit
expr false find fmt fold grep head kill ln ls mkdir mv printf ps pwd
rm rmdir seq shuf sort tail test tr true uniq
```

That is, you should not solve the problem in C++ or Python or some other programming language; just use the shell. If you need any utility not in the above list, briefly justify why it's needed.

You can write several scripts if you like, and have your scripts invoke each other.

Along with each question answer, write a brief after-action report explaining how you came up with and debugged the scripts, including any blind alleys you explored.

The basic goal of these shell scripts will be to test the 'myspell' program that you wrote in Assignment 1. That is, you want to give 'myspell' some input data and compare the resulting output to what it should be, and you want to have lots of test cases to test 'myspell' thoroughly.

1a (12 minutes). Write a shell script 'genspelldata' that generates test cases for 'myspell'. Your script can assume the existence of the file '/usr/share/dict/linux.words' as in Assignment 1. It should take three operands G B F, where G is the number of good words in the output, B is the number of bad words in the output, and F is the name of a text file containing bad words, one per line. (All numbers are unsigned decimal integers.) Words should be chosen randomly from the respective sources, with repetition.

For example, if the file 'bad.words' contains the four lines

```
notaword
anothernonword
ain't
secretery
```

the command 'genspelldata 5 3 bad.words' might output:

```
tailorless ain't Avicennism notaword mariengroschen catadrome
underbailiff ain't
```

because this output has 5 good words and 3 bad words; two of the bad words are identical, but that sort of thing is expected.

(Put your answer on the next page; include the after-action report.)

(Put your answer to 1a here.)

1b (10 minutes). Assuming the shell scripts ‘genspelldata’ and ‘myspell’ exist in the current directory, write a shell script ‘testmyspell’ that takes three unsigned decimal number operands N G B F and tests ‘myspell’ with N different randomly chosen test cases, each containing G good words and B bad words (the latter taken from the file F). For example, ‘testmyspell 1000 5 3 bad.words’ should test ‘myspell’ 1000 times, each with a randomly-chosen test case generated by ‘genspelldata 5 3 bad.words’.

1c (10 minutes). Suppose you are maintaining ‘genspelldata’ and are worried that the changes that you make might break it. You want to have some regression tests for ‘genspelldata’, so that you can test it after changing it, and make sure that it outputs exactly the same thing after the change as before the change. Describe and implement extensions to the behavior of ‘genspelldata’ to make it regression-testable in this sense, while still keeping the functionality that the script already has. Your extensions should add an option or options to ‘genspelldata’ to make it regression-testable. (Hint: see the full documentation for ‘shuf’.)

2 (14 minutes). Reimplement the ‘genspelldata’ command in Python, using only the argparse, random, string, and sys modules. (See (1a) for the specification of ‘genspelldata’.) Or, if it’s not practical to implement ‘genspelldata’ that way, explain why not, implement it as best you can in Python some other way, and explain any shortcomings in your approach. Either way, make your solution as simple and clear as you can.

3. Consider the following Elisp code:

```
  (defun delete-horizontal-space (&optional backward-only)
    "Delete all spaces and tabs around point.
  If BACKWARD-ONLY is non-nil, delete them only before point."
    (interactive "*P")
    (let ((orig-pos (point)))
      (delete-region
       (if backward-only
           orig-pos
         (progn
           (skip-chars-forward " \t")
           (constrain-to-field nil orig-pos t)))
       (progn
         (skip-chars-backward " \t")
         (constrain-to-field nil orig-pos)))))
```

3a (6 minutes). Give two ways to call this function from an Emacs session in which you want to delete all spaces and tabs around the cursor. Prefer convenience.

3b (6 minutes). Extend the function so that the caller can optionally delete spaces and tabs only *after* the current position.

3c (6 minutes). Modify the original (non-extended) function so that it does not worry about fields, i.e., it does not call ‘constrain-to-field’ and always behaves as if the entire buffer were one field.

4 (9 minutes). Explain how the ‘wget’ shell command fits into the client-server computing model as exemplified by React. For example, what are the pros and cons of using ‘wget’ to retrieve data from a system built by React?

5. Consider the following JSX function taken from the React tutorial used as the basis of Homework 3:

```
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}
```

5a (8 minutes). Explain what this function is for and how it works. Give the types of all the identifiers and expressions used in this JSX code.

5b (4 minutes). Write a JSX class that behaves the same way as this function, when used as a React component.

6 (15 minutes). Briefly compare and contrast the role of dependencies and parallelism during development, during builds, and during installation. What do the dependencies have in common, and how do they differ? Give an example of a development dependency in your ongoing class project, and give an example of a build and of an installation dependency; take the latter two examples from the assignments.