

Name: Nevin Liang

Student ID: 705-575-353

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.

4:35pm Date and time (Los Angeles time) you started the exam

6:12pm Date and time that you ended the exam

Nevin Liang 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
```

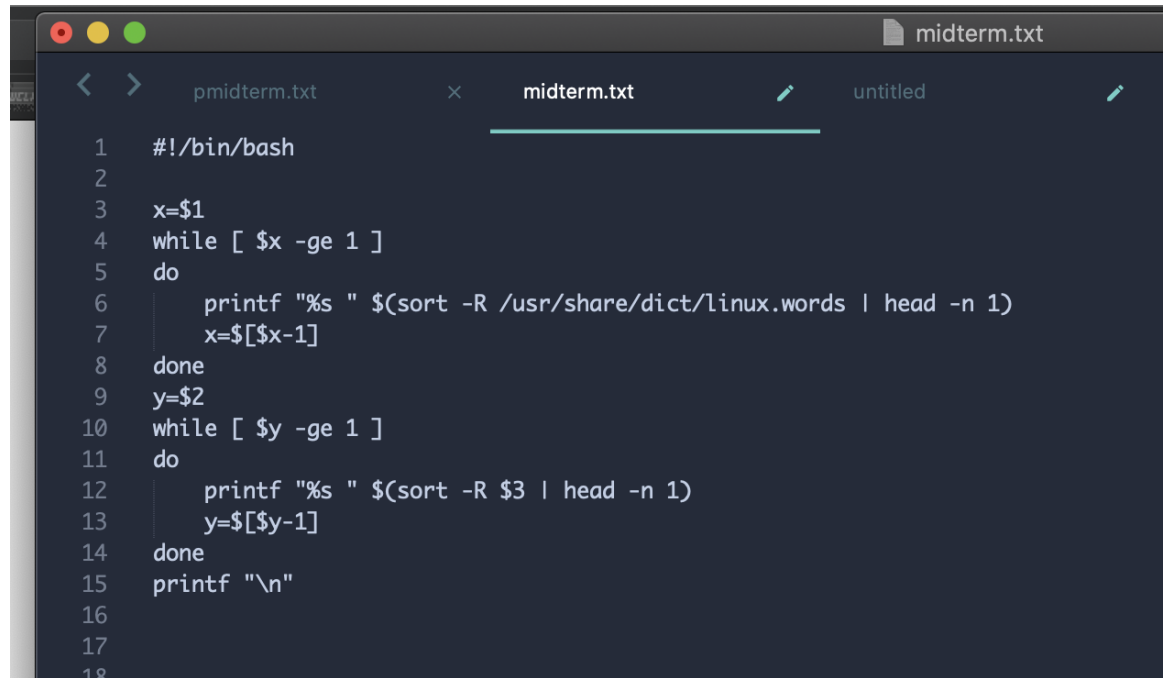
the command 'genspelldata 5 3 bad.words' might output:

```
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.)



```
1  #!/bin/bash
2
3  x=$1
4  while [ $x -ge 1 ]
5  do
6      printf "%s " $(sort -R /usr/share/dict/linux.words | head -n 1)
7      x=$((x-1))
8  done
9  y=$2
10 while [ $y -ge 1 ]
11 do
12     printf "%s " $(sort -R $3 | head -n 1)
13     y=$((y-1))
14 done
15 printf "\n"
16
17
18
```

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'.

```
20
21
22  #!/bin/bash
23
24  x=$1
25  while [ $x -ge 1 ]
26  do
27      ./genspelldata $2 $3 $4 | myspell
28      x=$((x-1))
29  done
30
31
```

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'.)

```

33
34
35  #!/bin/bash
36
37  if test $1 = '-n'; then
38      # new code that won't be affected if we don't choose the -n flag
39      x=$1
40      while [ $x -ge 1 ]
41      do
42          ./genspelldata $3 $2 $4 | myspell
43          x=$((x-1))
44      done
45  else
46      # this executes the old code.
47      x=$1
48      while [ $x -ge 1 ]
49      do
50          ./genspelldata $2 $3 $4 | myspell
51          x=$((x-1))
52      done
53  fi
54
55

```

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.

```

1  #!/usr/bin/python
2
3  import random, sys
4  from argparse import ArgumentParser
5
6  class genspelldata:
7      def __init__(self, args):
8          try:
9              f = open(args.file[0], 'r')
10             self.bad = f.readlines()
11             f.close()
12         except FileNotFoundError:
13             sys.stdout.write("FILE NOT FOUND. PROGRAM TERMINATED.\n")
14             return
15         try:
16             f = open("/usr/share/dict/linux.words", 'r')
17             self.good = f.readlines()
18             f.close()
19         except FileNotFoundError:
20             sys.stdout.write("FILE NOT FOUND. PROGRAM TERMINATED.\n")
21             return
22
23         for i in range(0, args.reps):
24             ans = ""
25             for j in range(0, args.bad):
26                 ans += random.choice(self.bad)
27             for j in range(0, args.good):
28                 ans += random.choice(self.good)
29             sys.stdout.write(ans)
30             sys.stdout.write("\n")
31         return
32
33  def main():
34      parser = ArgumentParser(description="generates spell data!")
35      parser.add_argument("reps")
36      parser.add_argument("good")
37      parser.add_argument("bad")
38      parser.add_argument("file", nargs="*", default="")
39      args = parser.parse_args()
40
41      try:
42          generator = genspelldata(args)
43      except IOError as err:
44          parser.error('I/O error({0}): {1}'.format(err.errno, err.strerror))
45
46  if __name__ == "__main__":
47      main()
48

```

3. Consider the following Emacs 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.

here are a number of ways to do it:

- 1) C-x C-e evaluates the Lisp form right at your cursor and prints the value in the echo area
- 2) If you're in emacs-lisp-mode you can do C-M-x to evaluate it before or around the point
- 3) You can also literally type it in the scratch buffer and do C-j but your cursor would have to be at the end so there's kind of no point haha
- 4) You can also put the elisp code in a file and when you're in a different file, you can type M-x load-file and Emacs will evaluate everything in the elisp file.

#2 and #4 are most convenient

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

```
add this after the "if backward-only orig-pos" statement
1.
(if forward-only
  orig-pos
  (progn
    (skip-chars-backward " \t")
    (constrain-to-field nil orig-pos t)))
2.
have there be an (&optional forward-only)
in the first line of the code.
```


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.

there are actually many ways to do this:
one, is to bind the variable: 'inhibit-field-text-motion' to a non-nil value. another way is to keep the code the exact same and still use constrain-to-field, but make sure the optional argument 'inhibit-capture-property' is non-nil and 'old-pos' has a non-nil property of that name. this way, any field boundaries are also ignored.

(just a way to keep the original code the same :))

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?

```
wget is a command line program to help retrieve web pages. I've used this command multiple times to download web pages from the linux terminal, but have only done this for static webpages because I assumed it was safe to do so since nothing was running and actively updating the website as it was being downloaded. therefore, you might only get a snapshot of the site, not the actual contents you might want. after combing the docs of wget :) I realized that you cannot use wget on a dynamically generated js. according to google, you need something like Selenium.
```

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.

this function is essentially just a button that when clicked returns the value of the property it has the props are like a object that you can pass to the react component

props.onClick is a function
props.value is a string

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

```
class Square extends React.Component {  
  render() {  
    return (  
      <button  
        className="square"  
        onClick={() => this.props.onClick()}  
      >  
        {this.props.value}  
      </button>  
    );  
  }  
}
```

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.

Dependencies basically determine whether or not things in an application can run in parallel or not. While the problem says to briefly compare and contrast, I think these two are actually very connected and that changing one of them fundamentally alters the qualities and effectiveness of the other. If A depends on B and B depends on A, they must run in parallel. If A depends on B's output, they cannot run in parallel (to a certain extent). This reminds me of pipelining and parallelization in computer architecture as well, even though the topic we are talking about right here is software construction.

Dependencies in software result from a number of things, like changes in system, external influences, the size of the project (multiple products or not), as well as how the different parts of the projects are integrated together. From a logical point of view, during development, dependencies are widespread. And they are a serious problem when it comes to development. Person A working on project A has to depend on Person B working on project B without the project B actually even being finished. They have to coordinate the end result before they even start working, or else they cannot work in parallel (to speed things up).

During building, it should be fairly straightforward to deal with parallelism (given the adequate preparation put into development). Dependencies are going to be slightly troublesome in this phase, however, as most of the time when building big projects, there are multiple "pit stops" you have to make in order to test your product before you have finished it with. (building the project all at once is a bad idea haha). coordinating this is hard. Parallelism might be disrupted. One person might wait around because he can't go on without the other person's input (in case the other person needs to make a drastic change in plans due to an unforeseeable bug). Dependencies can mess up parallelism.

(talk about the actual code dependencies)

Code dependencies are difficult to deal with as well in the building stage. not everything is integrated perfectly in the form of .txt files that can be read in and out between programs. sometimes there is more muddy, gritty output formats that have to be read as input into a program that is coded in a language that can't even read it!

(on second thought i think build-dependencies might be talking about compile-time dependences)

READ-HERE:

in the case of build-time vs run-time dependencies, build-time consists of which packages to build first and which ones depend on which other ones. run-time, which should go in the next section, are things that a package requires in order to run the code in it.

During installation, program dependencies on each other are HUGE. installation means running an executable file and when there are hundreds and hundreds of files that have to be run, dependencies in code have to be kept crystal clear in order for everything to run smoothly

development dependency in our ongoing class project could be either the algorithm for our tetris game, and the react script used to display everything on the webpage, or even the "forum-thing" we have next to our tetris game and the game itself, running at the same time on the same page. the score of our tetris game has to go into the forum often, and we will have to incorporate input and output in both codes

build dependency might be the sys library in our python code. We could not have built our code if the compiler could not have found the sys library and built it. an example of an installation dependency would be when we typed apt-get install libpython into our linux terminals.