UCLA CS 35L midterm, spring 2022
100 minutes total, open book, open notes, closed computer.
1 minute = 1 point. Write answers on exam.

Name:__________________________ Student ID:__________

1a (6 minutes). Explain what the following command does; give an example.

```
grep -E '^( *\([^)]+\))+ *$'
```

1b (4 minutes). Simplify the following shell command, and briefly explain why your simplification works. It's OK if the simplification is slight.

```
sed 's/^ab/cdef/g'
```

2 (6 minutes). Explain why Node can allow circular module dependencies (i.e., cycles in the module dependency graph), but a build tool like 'make' cannot allow circular dependencies.

3a (10 minutes). Write a simple shell command ‘findrange’ that accepts two arguments C and D, reads standard input, and outputs a copy of each line containing a character that is greater than or equal to C, and less than or equal to D. Your command can assume that C and D are both single printable ASCII characters. It should use case-sensitive comparison, using standard ASCII ordering; you need not worry about non-ASCII characters. Your implementation should use ‘grep’ to do the actual work of reading and writing lines.

For example, if standard input is ...

```
The University of
California requires
that UCLA students,
faculty and staff
who are living,
working or learning
on campus or at
other UCLA
properties be
vaccinated against
COVID-19 - with
limited exceptions.
```

... then ‘findrange w z’ should output ...

```
The University of
faculty and staff
who are living,
working or learning
COVID-19 - with
limited exceptions.
```

... because the above are all the input lines that contain ‘w’, ‘x’, ‘y’, or ‘z’.

3b (10 minutes). Explain what your ‘findrange’ does with arguments that specify special characters. For example, what does it do if its arguments are '"', '$', "'", '(', ')', '*', '+', '-', '.', '?', '[', '\', ']', '^', or '|'? If this exposes any bugs in your implementation, explain the bugs.

4 (10 minutes). Suppose our version of Bash is buggy, because it consistently misspells ‘export’ with an ‘s’ between the ‘x’ and the ‘p’. That is, if you try to run the command ‘export PATH’ your attempt will fail with the diagnostic ‘bash: export: command not found’. You can run the command ‘exsport PATH’ instead, and it successfully exports PATH.

You attempt to work around the problem by creating an executable shell script named ‘export’ in your PATH with the following two lines as its contents:

```
  #!/bin/sh
  exsport "$@"
```

Now when you run the command ‘export PATH’ in your buggy shell, you don’t get a diagnostic and the ‘export PATH’ command exits with status zero, indicating success. However, the command still doesn’t work like a standard ‘export’ command should. Explain what goes wrong and why.

5 (10 minutes). Consider the following HTML taken from the class web page. Diagram the elements of the document tree that it represents. Draw arrows from each element to it parent element, if any. Label each element with its tag, along with the letter A if the element has attributes and the letter T if the element contains text that is not part of any sub-element.

```
<html lang='en'>
<head>
<title>UCLA Computer Science 35L, spring 2022</title>
</head>

<body>
<section>
<h1>UCLA Computer Science 35L, spring 2022.<br>Software
Construction</h1>

<ul>
<li><a href='news.html'>News</a></li>
<li><a href='syllabus.html'>Syllabus</a></li>
</ul>

<p>Lecture, 4 hours; laboratory, 2 hours; outside study, 6 hours.</p>
</section>
</body>
</html>
```

6 (8 minutes). HTTP running atop TCP/IP is good for some client-server applications, and not so good for others. Give an example of both the former and the latter and briefly justify your examples.

7 (12 minutes). Fix three bugs in the following Emacs Lisp function, which is to be executed in View mode when you type '%'. The bugs cause the code to not agree with its documentation.

```
(defun View-goto-percent (&optional percent)
  "Move to end (or prefix PERCENT) of buffer.
Center display at point.
Also set the mark at the position where point was."
  (interactive "P")
  (push-mark)
  (view-recenter)
  (goto-char
   (if percent
       (+ (point-min)
          (floor
           (* (- (point-max) (point-min))
              (max 0 (min 100 (prefix-numeric-value
                                percent))))))
     (point-min))))
```

8. Normally in Python if you import the same module more than once, only the first import statement has an effect; the later import statements do nothing and take essentially zero time. If you want to actually import a module several times, you can use the reload function of the importlib module. For example:

```
import importlib
import mymodule
import mymodule # does no work
import mymodule # does no work
importlib.reload(mymodule)
importlib.reload(mymodule)
importlib.reload(mymodule)
```

8a (6 minutes). The above example is contrived; you'll never see those lines in practical code. Explain why you might want to use 'import mymodule' multiple times in a realistic Python application.

8b (6 minutes). Explain why you might want to use 'importlib.reload(mymodule)' instead of 'import mymodule' in a more realistic case.

9 (6 minutes). Explain why renaming a symbolic link can change the file that it points to, whereas renaming a hard link cannot do that.

10 (6 minutes). Suppose React were written in C++ rather than in JavaScript. Explain how this would affect how you'd do Assignment 3 in React. In particular, how would you debug your program, compared to debugging your program with React as it is?