Name:

|  |  |  |
| --- | --- | --- |
| Problem | Points | Score |
| 1 (in-class) | 25 |  |
| 2 (in-class) | 25 |  |
| 1 (interview) | 25 |  |
| 2 (interview) | 25 |  |
| Total | 100 |  |

Notes:

1. The first step in this exam is to create a workspace in the following directory:

/data/courses/ece\_1111/current/exams/ex\_01

Your directory should be your last name all lowercase, followed by an underscore, following by your first name (e.g, “picone\_joseph”). Set the permissions using “chmod u+rwx,g-rwx,o-rwx <lastname>” so only you have read and write permission to this directory. Create subdirectories within this directory: p01, p02, … You will use these for problems 1 and 2, … respectively. Put ALL your code in these directories. Do not touch your files after the exam is over.

1. For this exam you are allowed to open a terminal window on your computer, you are allowed to web surf with Google and use online chatbot services. However, if you choose to use these Internet resources, at the top of your file you need to list the URL for the source and the query you used for this resource. If you used multiple queries, list each one individually. Failure to cite your sources will be considered plagiarism.
2. After the exam is over, you will be given until 8 AM on the following Wednesday to meet with your TA/peer mentor. Your peer mentor will ask you questions about your solutions to problems 1 and 2. Any code that cannot be explained will result in 0 points for that problem. If you successfully explain your code, you will be awarded between 0 and 25 points depending on the thoroughness of your answers.

Hence, do not submit code you do not understand. Do not submit code that you think you will study after the exam is over – that rarely works well in practice. Use these tools as a learning tool, but not a substitute for learning how to write code.

**(25 pts) Problem No. 1 (P01)**: Bash Scripting

This file:

/data/courses/ece\_1111/resources/data/text/data\_v00.zip

contains a large text document. Write a bash shell script with an interface that accepts a command line argument containing a filename which is in a zip format:

p01/p01.sh <filename>.zip

Your script should look for all lines in the file that contain at least one occurrence of the letters [a-z], and produce a histogram of all word pairs that occur on these lines, sorted in descending order and alphabetically. For example, suppose we have the following three lines:

“12345 12345 12345”

One oh two oh one one oh two.

“6789 9876”

Your script would produce a list

oh two 2

one oh 2

oh one 1

one one 1

two oh 1

Note “oh” comes before “one” alphabetically.

You should ignore/strip punctuation, ignore case, but leave hyphenated words as a single word. You do not need to worry about the first or last word in the line as being the start or end of a line. Some tools would output “<start> one” and “two <end>” to account for the beginning and end of line – you do not need to do this.

Your script should not generate intermediate files and should work on any zip file. Your script should not use bash functions. You can, however, string together Unix commands using pipes (hint!) and embed them in your script. Just don’t try to overly complicate this by using bash functions.

Finally, note that some approaches might use vast amounts of memory. If you find your script crashing due to memory issues, rethink your approach.

**(25 pts) Problem No. 2:** Python Programming

Copy the following code into your p02 directory:

sd /data/courses/ece\_1111/2025\_01\_fall/exams/ex\_01/<your\_name>/p02

cp -r ../../picone\_joseph/p02/\* .

The Python file p02.py contains a broken program. Your job is to fix it so that this command works:

p02.py songs

This program randomly shuffles and prints filenames. If debugged, the output you get should be:

nedc\_130\_[1]: p02.py songs

1 (0, 1) playing f02.mp3

2 (1, 1) playing f01.mp3

3 (2, 1) playing f05.mp3

4 (0, 2) playing f02.mp3

5 (1, 2) playing f01.mp3

6 (2, 2) playing f05.mp3

7 (2, 3) playing f05.mp3

8 (0, 3) playing f02.mp3

^Z

[1]+ Stopped p02.py songs

ece-000\_[2]:

Note that this is what you will get the first time you run it, and that the order of the first three files might change depending on the behavior of the random number generator.

The program randomly displays filenames and keeps track of how many times the filename has been displayed. You can only kill this program by typing “ctrl Z” (^Z) and then “kill -9 %1”. The number displayed in brackets after you hit ^Z is the job number. If “[2]” is displayed, type “kill -9 %2”. Make sure you kill the job each time you run it so we don’t overload the machine.

This program writes a history file which by default is named “HISTORY.txt” in your p02 directory. When the program is started, it reads the history file to know how many times a filename has been previously displayed. The program constantly updates this file to know where it is at.

The first argument is a directory path. The program is supposed to scan this directory path for any file that ends in “.mp3” and adds it to the list of files to be processed. Note that it ignores some files in the directory named “songs”. However, the programmer made an error, and it does not find “.mp3” files right now.

Once the list is built, it selects the filename that has been displayed the least and displays it. It updates the counts and displays the next “least displayed” filename. If all goes well, every file is displayed the same number of times before it attempts to start over and display all the filenames once again. In the above example, f02.mp3, f01.mp3 and f05.mp3 are displayed in a random order, and then it goes back and displays all three names in a different order.

The code you have copied, p02.py, has several problems associated with it. Find these bugs and fix them. The order the files are displayed might change from the above, but if the program is working properly, it will display all three files exactly once and then start over.

Ignore the code from main() to the comment “start here”. That code is fine. The bugs are located somewhere after that.

Finally, you must not rewrite the entire codebase using a chatbot or your own code. You must preserve the structure of this code but find and fix the bugs. Wherever you make a change, put a comment “# bug fix” preceding the code you are changing.

Ask questions if something is not clear.