Name:

|  |  |  |
| --- | --- | --- |
| Problem | Points | Score |
| 1 | 50 |  |
| 2 | 60 |  |
| 3 | 70 |  |
| 4 | 80 |  |
| 5 | 90 |  |
| 6 | 100 |  |
| Total | 100 |  |

Notes:

1. For this exam you are allowed to open a terminal window on your computer, you are allowed to web surf with Google, but you cannot use online chat or other interactive services.
2. Your code and results should be placed in directories p01, p02, …, p06.

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

**/data/courses/ece\_1111/current/exams/exam\_04/lastname\_firstname**

**Set the permissions using “chmod -R u+rwx,g-rwx,o-rwx <lastname\_firstname>” so only you have read and write permission to this directory. Create six subdirectories within this directory: p01, p02, …, p06. Put ALL your code in these directories. Do not touch your files after the exam is over.**

**You must use Python 3 for this exam (the version of Python on the AWS server).**

This exam is structured in a tiered manner. Start with problem 1. When done, copy your code to the next problem and continue editing it. Only turn in code that is completely working. I will grade the highest level you submit. If it doesn’t work, you will get a 50 for this exam (we call this the “you didn’t debug your code and wasted my time” penalty). Therefore, what you submit must work and meet the stated requirements.

**Problem No. 1**: Create a simple class called CompareFile that consists of a default constructor with no arguments, a default destructor and a method called myprint that takes a file pointer as an argument and prints “comparing files: <name>” where “<name>” is the actual name of the file (argv[1]). Put all your code in a file p01.py. Your main program should construct an object and call its print method. It should accept an argument from the command line using argv[1].

**Problem No. 2:** Add a method called myread that accepts a filename as an argument (“myread(fname)”). Have it open a text file, read it line by line and print the contents of each line along with its length and an integer counter indicating the line number:

nedc\_999[0]: p02.py foo.txt

0 [5]: abcde

1[3]: fgh

…

Use your myprint function to do the printing (myread() calls myprint()).

**Problem No. 3:** Change your class to read the data into memory, sort it using a unique sort, and print out the file in an ascending sort order. Print out the number of times a line appears in the file. Duplicate lines should only appear once in your output:

nedc\_999[0]: p03.py foo.txt

0 [c = 7][l = 5]: abcde

1 [c = 4][l = 3]: fgh

…

Use your myprint function to do the printing (myread() calls myprint()). The first number in brackets is the number of times the line appears. The second item is the length.

**Problem No. 4:** Assume each line contains data of the form:

mary = 1, 3, 5, 7, 9

joe = 2, 4, 6

john = 5, 7, 9, 27, 999

Parse the data and store it in a dictionary where each entry is a list of integers (not strings). Print the dictionary using the format shown in problem no. 3. Again, only print the unique lines, but now you are printing them in this format:

nedc\_999[0]: p04.py foo.txt

0 [c = 7][l = 5]: mary = [1, 3, 5, 7, 9]

1[c = 4][l = 3]: joe = [2, 4, 6]

…

Instead of printing the length of the line in characters, print the number of items you found on each line.

**Problem No. 5:** Change your main program to take two arguments: file1 and file2. Read them and load them into separate dictionaries following the approach for problem no. 4, and print them as in problem no. 4:

nedc\_999[0]: p05.py f1.txt f2.txt

f1.txt:

 0 [c = 7][l = 5]: mary = [1, 3, 5, 7, 9]

 1[c = 4][l = 3]: joe = [2, 4, 6]

 …

f2.txt:

 0 [c = 7][l = 5]: mary = [1, 3, 5, 7, 9]

 1[c = 4][l = 3]: joe = [2, 4, 6]

 …

**Problem No. 6:** Compare the two dictionaries and produce a score in the range [0,1] that indicates how similar they are. A score of 1 means they are identical. A score of 0 means they are completely different. Test your code using the following use cases:

1. Create two files that are very large (100 items) but differ only by one value. This should produce a score close to 1.
2. Create two large files for which 50% of the elements are different. This should score close to 0.5.
3. Create two files for which all the elements are different except one. This should score close to 0.

Your program should run using this interface:

nedc\_999[0]: p06.py f1.txt f2.txt

f1.txt:

 0 [c = 7][l = 5]: mary = [1, 3, 5, 7, 9]

 1[c = 4][l = 3]: joe = [2, 4, 6]

 …

f2.txt:

 0 [c = 7][l = 5]: mary = [1, 3, 5, 7, 9]

 1[c = 4][l = 3]: joe = [2, 4, 6]

 …

 similarity score = 0.75

Of course, you have to implement a reasonable algorithm to produce a similarity score.