Name:

|  |  |  |
| --- | --- | --- |
| Problem | Points | Score |
| Base Score | 50 |  |
| P1: Level 1 | +5 |  |
| P1: Level 2 | +10 |  |
| P1: Level 3 | +15 |  |
| P1: Level 4 | +20 |  |
| P1: Level 5 | +25 |  |
| P2: Level 1 | +5 |  |
| P2: Level 2 | +10 |  |
| P2: Level 3 | +15 |  |
| P2: Level 4 | +20 |  |
| P2: Level 5 | +25 |  |
| 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. As explained in class, this exam is structured so that you only get credit for a problem if you successfully pass the level you claim you have completed. If you apply for credit for a particular level and fail to meet the requirements of that level, you will receive no credit for that problem.
3. You cannot skip levels. For example, you must complete level 2 before you can apply for level 3.
4. Note that even though specific test cases are provided, your code must work for general cases. To receive full credit at a given level, you will need to pass previously unseen test cases.

**Deliverables:**

In this directory:

/data/courses/ece\_1111/2019\_spring/exams/exam\_04/<lastname\_firstname>

Create the following directories:

p01/l01 p02/l01

p01/l02 p02/l02

p01/l03 p02/l03

p01/l04 p02/l04

p01/l05 p02/l05

Deposit a working version of your code that must compile with make (except for the l05 solution). Once you complete a level, leave a working version of your code at that level, and copy your code to the next directory. This way you always have a working version of your code for the previous level.

**Problem 1:** This problem involves implementation of a framing and windowing approach for processing data from a binary file.

**P01.Level 1:** Create a main program, p01\_l01.exe, that supports the following interface:

**p01\_l01.exe -help -f 3 -w 5 <any filename>**

The “-help” option can appear anywhere in the command. If present, a help message is printed and the program exits. The help message that is displayed must be this:

**name: p01\_l01**

**synopsis: p01\_l01 [options] file(s)**

**descr: processes a binary file**

and must be stored in a file named p01\_l01.help.

**P01.Level 2:** Create a program, p01\_l02.exe, that supports the following interface:

**p01\_l02.exe -start 3 -N 5 <any filename>**

This program creates a binary file containing short integers. The values start at “start” and continue for N values. For example, for the command above, the file would contain the values [3, 4, 5, 6, 7]. Use “od -s” to verify that your program is working.

Run this program and generate a file called p01\_l02.dat, containing the following data: [-2, -1, 0, 1, 2, 3, 4, 5, 6, 7]. The file should be 20 bytes in size.

**P01.Level 3:** Create a program, p01\_l03.exe, that reads the data using a “right-aligned” window and prints the content of each window. Your interface should be as shown below and produce the results shown:

**Test Case 1: p01\_l03.exe -f 3 -w 3 p01\_l02.dat**

**-2 -1 0**

 **1 2 3**

 **4 5 6**

 **7 0 0**

**Test Case 2: p01\_l03.exe -f 3 -w 6 p01\_l02.dat**

 **0 0 0 -2 -1 0**

**-2 -1 0 1 2 3**

 **1 2 3 4 5 6**

 **4 5 6 7 0 0**

**Test Case 3: p01\_l03.exe -f 4 -w 2 p01\_l02.dat**

 **0 1**

 **4 5**

 **0 0**

**P01.Level 4:** Create a program, p01\_l04.exe, that computes a sum of squares for the data in the window. Your output should be as shown below:

**Test Case 1: p01\_l04.exe -f 3 -w 3 p01\_l02.dat**

**Frame: 0 SumSq = 5.0000**

**Frame: 1 SumSq = 14.0000**

**Frame: 2 SumSq = 77.0000**

**Frame: 3 Sumsq = 49.0000**

**P01.Level 5:** Compile your program debug. Using gdb, set a conditional breakpoint so that the program stops on frame 2 for the example above in Level 4. Print the value of the squared sum from inside the debugger. Copy the entire session from your terminal window as text into a file p01\_l05.txt as proof you were able to do this. Your file should contain all the steps, including how you set the breakpoint, how you ran the program and how you printed the value.

**Problem 2:** This problem involves implementing a matrix class in C++.

**P02.Level 1:** Create a main program, p02\_l01.exe, that supports the following interface:

**p02\_l01.exe -help -operation multiply <any filename> <any filename>**

The “-help” option can appear anywhere in the command. If present, a help message is printed and the program exits. The help message that is displayed must be this:

**name: p02\_l01**

**synopsis: p01 [options] file(s)**

**descr: processes a binary file**

and must be stored in a file named p02.help. The help message should also be printed if less than two filenames are specified from the command line.

You must support values of “multiply”, “add” and “subtract” for the option “-operation”.

You must use your Matrix class implementation in C++ for this problem.

**P02.Level 2:** Create two text files containing the values of two matrices:

**f01.txt: f02.txt**

**1 1 1 2 2**

**1 1 1 2 2**

 **2 2**

Demonstrate that you can correctly read this data from a file and initialize your matrices. Your program should print the values of the matrices in an easily readable format before it exits:

**nedc\_999\_[1]: p02\_l02.exe -operation multiply f01.txt f02.txt**

**f01.txt:**

 **1 1 1**

 **1 1 1**

**f02.txt:**

 **2 2**

 **2 2**

You can deviate from the above format as long as your output is easy to read and nicely formatted.

Obviously, your program should work for any data. Each row does not need to have the same number of values, and the values for each row can be different.

**P02.Level 3:** For the example shown in P02.Level 2, demonstrate that you can correctly multiply the matrices. Your output should be:

**nedc\_999\_[1]: p02\_l02.exe -operation multiply f01.txt f02.txt**

**6 6**

**6 6**

**P02.Level 4:** Create two text files containing the values of two matrices:

**f03.txt: f03.txt**

**1 2 1 1**

**3 4 5 1 1 1**

**6 7 1 1**

Demonstrate that you can add and subtract these matrices. Since these matrices cannot be multiplied, you should display an appropriate error message. Note that a seg fault is not an appropriate error message ☺

**P02.Level 5:** Compile your program for the previous step debug. Using gdb or the Cloud9 IDE, demonstrate that you can stop the program inside your multiply method and display the values of the matrices before multiplication. You can submit a screenshot or a text file depending on the debugger that you use.