**ECE 1111: Engineering Computation I**

**Laboratory No. 14: Application Programming**

|  |  |  |
| --- | --- | --- |
| Problem | Cumulative Points | Score |
| 0 | 50 | Attend the lab and make a reasonable attempt at solving the problems. |
| 1 | 25 |  |
| 2 | 25 |  |
| Total | 100 |  |

Notes:

1. You must code your exam solution in Python.
2. 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. You can use software tools like ChatGPT to solve particular problems, but your code must be your own original work **and should use concepts discussed in this course**.
3. Your code and results should be placed in subdirectories **p01** and **p02**.
4. You must work the problems in order – you cannot skip a problem. For example, you cannot complete **p02** until you have finished **p01**.
5. If **p01** doesn’t work, then you will not receive credit for **p02**.
6. Most importantly, only turn in code that you are 100% sure works and is fully debugged. If you turn in a program that doesn’t work, you get a 50. If you start **p02** and determine your code doesn’t work, rename the directory “**\_p02**” so I know you are not submitting this as part of your solution.

**P01:** Write a function, named **convert\_mask()**, that converts a text string of the form **“1, 5-7, 3”** to an ordered list **“1, 3, 5, 6, 7”**. Some test cases are below:

**“1, 3, 2” => “1, 2, 3”**

**“4-6, 0, 3-5” => “0, 3, 4, 5, 6”**

**“1, 2, 3” => “1, 2, 3”**

Provide a driver script, **p01.py**, that operates like this:

**ece-000\_[1]: p01.py “4-6, 0, 3-5”**

**0, 3, 4, 5, 6**

You can assume the input numbers are positive integers.

**P02:** The following file:

**/data/courses/ece\_1111/resources/data/binary/f03.dat**

contains binary data. The format of the file is a bit unusual. The file consists of records. Each record contains a variable number of characters terminated with a null character, followed by a 16-bit signed integer value, and then a 32-bit float value. The file can contain a variable number of records.

For example, the above file contains these records:

 **“a”, 1, 1.0**

**“abc”, 2, 2.0**

**“abcde”, 3, 3.0**

We can verify this with **od**:

**ece-000\_[1]: d f03.dat**

**-rw-rw-r--. 1 picone ece\_1111 30 Dec 5 19:18 f03.dat**

**ece-000\_[1]: od -c -j 0 -N 2 f03.dat; od -s -j 2 -N 2 f03.dat; od -f -j 4 -N 4 f03.dat;**

**0000000 a \0**

**0000002**

**0000002 1**

**0000004**

**0000004 1**

**0000010**

**ece-000\_[1]: od -c -j 8 -N 4 f03.dat; od -s -j 12 -N 2 f03.dat; od -f -j 14 -N 4 f03.dat;**

**0000010 a b c \0**

**0000014**

**0000014 2**

**0000016**

**0000016 2**

**0000022**

**ece-000\_[1]: od -c -j 18 -N 6 f03.dat; od -s -j 24 -N 2 f03.dat; od -f -j 26 -N 4 f03.dat;**

**0000022 a b c d e \0**

**0000030**

**0000030 3**

**0000032**

**0000032 3**

**0000036**

Write a program, named **p02.py**, that reads the records identified by the mask and prints them to *stdout*:

**p02.py <filename> <mask>**

For example, this command:

**p02.py /data/courses/ece\_1111/resources/data/binary/f03.dat “0-2”**

prints:

**“a”, 1, 1.0**

**“abc”, 2, 2.0**

**“abcde”, 3, 3.0**

The program should call your solution to **p01** as a function and work for any filename of this format.