**ECE 1111: Engineering Computation I**

**Laboratory No. 12: C++ and Python Classes**

**Deposit your work in:**

**/data/courses/ece\_1111/current/labs/lab\_12/<lastname\_firstname>/p01**

**/data/courses/ece\_1111/current/labs/lab\_12/<lastname\_firstname>/p02**

**Goals:** Representation of data types common in engineering mathematics can be nicely done using many container classes available to you in C++ or Python. In this assignment, you will update your program that reads a matrix from a file and manipulates the matrices. Tokenizing data is one of those messy little things we must do frequently when coding. In this lab, you will demonstrate the ability to tokenize an input string. This is fairly easy in Python but also not so bad in C++ once you learn a little about string tokenization.

**Deliverables:**

1. *Source Code:* a collection of files in one directory that meets the assignment’s requirements (e.g., Makefile, header file, main driver program).
2. *Check Off:* Demonstrate that your program runs successfully. Answer questions about data types and scope.

**Description:**

1. (/p01) Create a C++ class that can read a matrix from a file and operate on it. Your class must have these public methods:

**Class MyMatrix {**

**MyMatrix();**

**bool MyMatrix::read(char \*);**

**bool MyMatrix::display(FILE\* fp);**

**}**

The matrix must be stored as protected data in the class using a data structure that supports **non-rectangular matrices** (e.g., a vector of vectors). The class should have functions to do the operations described below *and should overload the “+”, “-“ and “\*” operators*.

Your main program should have this interface:

**p01.exe -operation addition <filename1> <filename2>**

similar to the previous lab. You should support matrix addition, subtraction, and multiplication. If the matrices cannot be operated on due to incompatible dimensions, you should print an informative error message.

The input matrices can be non-rectangular (e.g., the length of each row is variable). The input file only has floating-point values. It does not have the dimensions of the matrix stated explicitly:

**nedc\_000\_[1]: cat example.txt**

**2.1 3.1**

**99.27 -23.45**

**-33.3**

**-99.99 -100.01 999.999 0.35**

Your output should look like this:

**NNN: 0000.0000 0000.0000 0000.0000 ...**

For example (use %10.4f):

 **1: 2.1000 3.1000**

 **2: 99.2700 -23.4500**

 **3: -33.3000**

 **4: -99.9900 -100.0100 999.9990 0.3500**

Read each matrix into a vector of vectors stored as internal data in the class. Your matrix class should be called MyMatrix.

Your code should be organized into these files: Makefile, mymatrix.h, mat\_00.cc and mymath.cc. The header file should have your class definition. The source file, mat\_00.cc, should have implementations of all your supporting member functions.

You must also be able to step through your code using Visual Studio.

1. (/p02) Repeat the previous task in Python. Use the same structure as your C++ code (e.g., same class name, same methods). Your output should match your C++ output exactly. For example:

**p01.exe -operation addition f1.dat f2.dat > p01.dat**

**p02.py -operation addition f1.dat f2.dat > p02.dat**

**diff p01.dat p02.dat**

should produce no differences.

**Summary:**

Engineering math can be implemented very efficiently in either C/C++ or Python. High-level languages like Python are very popular for engineering mathematics today, but computationally intensive tasks can run much faster in C/C++. The ability to write flexible, data-driven programs that implement this kind of math is a major outcome from this course.