ECE 4122/6122 Lab #2

(100 pts)

Section	Due Date
89313 - ECE 4122 - A	Sept 29th, 2020 by 11:59 PM
89314 - ECE 6122 - A	Sept 29th, 2020 by 11:59 PM
89340 - ECE 6122 – Q, QSZ, Q3	Oct 1st, 2020 by 11:59 PM

ECE 4122 students need to do Problem 1.

ECE 6122 students need to do Problem 1.

Note:

You can write, debug and test your code locally on your personal computer. However, the code you submit must compile and run correctly on the PACE-ICE server.

Grading Rubric

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her homework; however, TA's will **randomly** look through this set of "perfect-output" programs to look for other elements of meeting the lab requirements. The table below shows typical deductions that could occur.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
13 member functions	6%	Per function. (Total 78 %)
Does Not Compile	Up to 30%	Code does not compile on PACE-ICE!
Clear Self-Documenting Coding Styles	Up to 22%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score - (20/24)*H	H = number of hours (ceiling function) passed deadline note : Sat/Sun count as one day; therefore $H = 0.5*H_{weekend}$

<u>Problem 1:</u> MyGrid Class

Complete a class called ***MyGrid*** that has a member variable *gridElements* that is a dynamic two-dimensional array of elements (type **long**) of m rows by n columns. The class also contains two **long** variables called *myX* and *myY* which indicates the upper left-hand location of element (0,0) of the grid. Your class needs to be contained in two files. <u>MyGrid.h</u> is provided and has the class declaration. <u>MyGrid.cpp</u> is where all your member functions need to be implemented. The main.cpp file is provided for you to test your class's functionality.

The program needs to be able to read in text files containing initialization information for a MyGrid object. The names of the input files are passed into your program as command line arguments. All values in the input file are space separated and the file format will be as follows:

- First line has the myX and myY values
- Second line has the m number of rows and n number of columns
- The next m lines will contain n column elements.

There are sample input files (input1.txt and input2.txt) included that can be used for reference.

Your program needs to be able to read in the input files and correctly create the individual *MyGrid* objects. Your program needs to support all the functionality used in the provided main.cpp file.

Your class needs to support the following operators:

MyGrid operator+(MyGrid const &) const;
MyGrid operator-(MyGrid const &) const;
these operators use the extents of the grid and the upper left-hand corner to create a new grid that has the maximum extents of the two grids (see the image below). The values of the elements for the new grid are obtained by combining the values from the two grids on an element by element basis. Any new elements are initialized to zero.

MyGrid operator+(long const &) const;
Adds a long to all the elements in the MyGrid object.

friend MyGrid operator+(long const& lhs, MyGrid const &rhs);
Adds a MyGrid object to a long. The long value is added to all the elements in the MyGrid object.

MyGrid& operator++() and MyGrid operator++(int) all the element values are increased by one
MyGrid& operator—() and MyGrid operator--(int) all the element values are decreased by one

MyGrid operator-() const;
has the same effect as multiplying all the elements by -1

bool operator==(MyGrid const &) const;
the size and all the elements must be the same, and the upper left-hand offset must be the same.

friend fstream & operator<<(fstream& os, const MyGrid & foo);

this operator outputs the class using the same format as is used in reading in a *MyGrid* object.

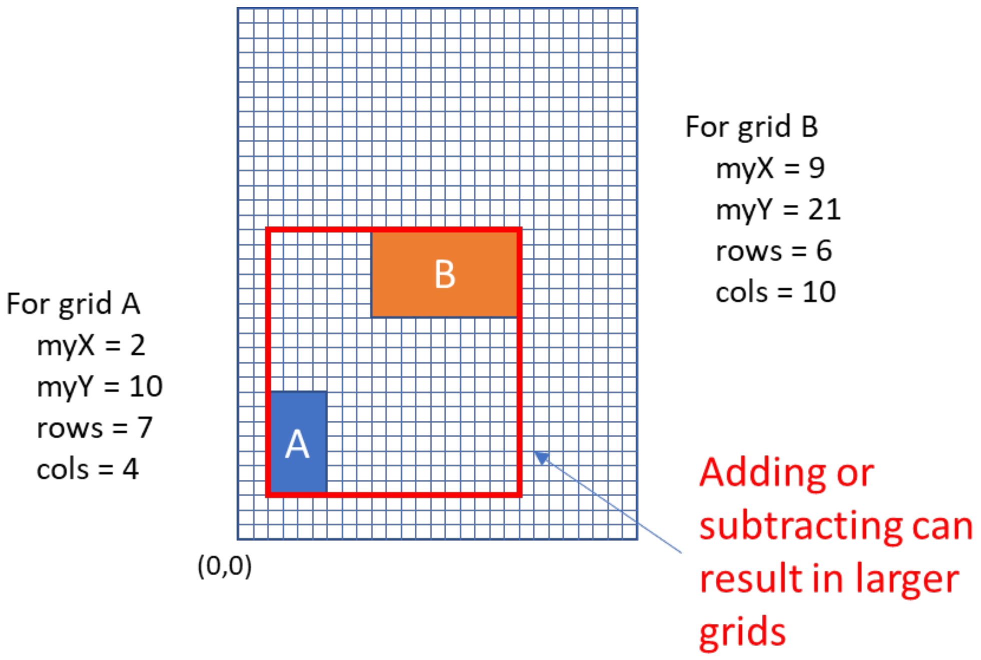
Your class also needs to have the following public member functions:

bool loadGridFromFile(const std::string &filename);

void zeroOutGrid(); sets all elements to 0

long getNumRows() const;

long getNumCols() const;



Canvas Submission:

Upload your versions of MyGrid.h and MyGrid.cpp to canvas.

Appendix A: Coding Standards

Indentation:

When using *if/for/while* statements, make sure you indent 4 spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of
“imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: <your name>
Class: ECE4122 or ECE6122
Last Date Modified: <date>

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.