ECE 4122/6122 Hmk #2

(100 pts)

Section	Due Date
89313 - ECE 4122 - A	Sept 18st, 2019 by 11:59 PM
89314 - ECE 6122 - A	Sept 18st, 2019 by 11:59 PM
89340 - ECE 6122 - Q	Sept 22th, 2019 by 11:59 PM
89706 - ECE 6122 - QSZ	Sept 22th, 2019 by 11:59 PM

ECE 4122 students need to do Problems 1 & 2. Each problem is worth 50 points.

ECE 6122 students need to do Problems 1, 2, & 3. Problems 1&2 are worth 33 points, and problem 3 is worth 34 points.

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.

Submitting the Assignment:

Create a zip file for each problem with the files you created to solve the problem. Each zip file should be named <FirstName_LastName>_Hmk2Prob#.zip, where # is 1, 2, or 3. You should upload the 3 zip files to the canvas website under the homework assignment.

Grading Rubric

Speed matters in this homework assignment. For each homework problem, the students with the fastest run times and correct answer will get full credit.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Does Not Compile	40%	Code does not compile on PACE-ICE!
Does Not Match Output	10%-90%	The code compiles but does not produce correct outputs.
Clear Self-Documenting Coding Styles	10%-25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)
Execution Speed	0% < 2x fastest 5% >= 2x but < 10x fastest 10% >= 10x fastest	

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*$H_{weekend}$

Problem 1: Matrix Class

Create a class called **ECE_Matrix** which will be used to hold a dynamic two dimensional array with M rows and N columns of element type **double**. Your class must support all of the operations in the attached file Hmk2_Test1.cpp.

You must create a header file called **ECE_Matrix.h** and an implementation file called **ECE_Matrix.cpp** which contains your solution to the assignment.

Use the file **data_Problem1.txt** included in the assignment as a test input file for the constructor that takes a data file string as an input. The numbers are space delimited. You can assume that the input text file does not contain errors such as missing numbers or nonnumeric numbers.

For the + and – operators for two matrices, the resulting matrix should be sized large enough to handle all the elements from both matrices. For example, a 2x4 matrix added to a 4x2 matrix should yield a 4x4 matrix.

When outputting the matrix with the stream insertion operator, output just the elements and use something similar to the following statement to output the matrix elements with a linefeed at the end of each row.

```
cout << scientific << setw(12) << setprecision(3) << values[ii][jj];
```

Output example for a 2x4 matrix:

```
 3.333e-01   3.536e+01   6.537e+02   4.358e+03
 1.002e+02   3.537e+01   6.538e+02  1.798e+308
```

Problem 2: Largest Product in a Grid

(www.projecteuler.net) Write a console program that takes as a **command line argument** the name of a data file. The data file will contain a **M x N** grid of numbers. Your program must read in the data file and determine the **largest product** of four adjacent numbers in the same direction (up, down, left, right, or diagonally). Below is an example of 4 adjacent diagonal numbers in **Red**:

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

The product of these numbers is 26 × 63 × 78 × 14 = 1788696.

File format:

All values in the text data file are **space** delimited. You can assume that the input data file does not contain errors, such as missing numbers or nonnumeric numbers. You are free to create as many files as you like and use whatever filenames in order to create a solution. All the files *.cpp and *.h files you create should reside in a single folder that can be compiled to create the executable.

The first line in the file will have the number of M rows and N columns. The next M lines in the data file will contain the rows of N numbers.

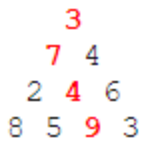
Use the file **data_Problem2.txt** included in the assignment as a test input file.

Output:

Your program should create a text file called **output2.txt,** in the same directory as the executable, with just the numerical answer on a single line.

Problem 3 (ECE 6122 students only): Maximum Path Sum

(www.projecteuler.net) By starting at the top of the triangle below and moving to adjacent numbers on the row below, the maximum total from top to bottom is 23.



That is, 3 + 7 + 4 + 9 = 23.

Similar to Problem #1, write a console program that takes as a command line argument the name of a data file. Your program must read in a triangle of arbitrary size from a data file. The first line will contain the number of levels in the triangle. A sample data file is attached (**data_triangle.txt**) to the homework assignment. Your program must find the maximum sum path through the input triangle starting at the top of the triangle.

Output:

Your program should create a text file called **output3.txt,** in the same directory as the executable, with just the numerical answer on a single line.

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.

Appendix B: Accessing PACE-ICE Instructions

ACCESSING LINUX PACE-ICE CLUSTER (SERVER)

To access the PACE-ICE cluster you need certain software on your laptop or desktop system, as described below.

Windows Users:

Option 0 (Using SecureCRT)- THIS IS THE EASIEST OPTION!

The Georgia Tech Office of Information Technology (*OIT)* maintains a web page of software that can be downloaded and installed by students and faculty. That web page is:

http://software.oit.gatech.edu

From that page you will need to install SecureCRT.

To access this software, you will first have to log in with your Georgia Tech user name and password, then answer a series of questions regarding export controls.

Connecting using SecureCRT should be easy.

- Open SecureCRT, you'll be presented with the "Quick Connect" screen.
- Choose protocol "ssh2".
- Enter the name of the PACE machine you wish to connect to in the "HostName" box (i.e. *coc-ice.pace.gatech.edu*)
- Type your username in the "Username" box.
- Click "Connect".
- A new window will open, and you'll be prompted for your password.

Option 1 (Using Ubuntu for Windows 10):

Option 1 uses the Ubuntu on Windows program. This can only be downloaded if you are running Windows 10 or above. If using Windows 8 or below, use Options 2 or 3. It also requires the use of simple bash commands.

1. Install Ubuntu for Windows 10 by following the guide from the following link:

   https://msdn.microsoft.com/en-us/commandline/wsl/install-win10

2. Once Ubuntu for Windows 10 is installed, open it and type the following into the command line:

   *ssh **YourGTUsername**@coc-ice.pace.gatech.edu* where **YourGTUsername** is

   replaced with your alphanumeric GT login. Ex: bkim334

3. When it asks if you're sure you want to connect, type in:
   yes

and type in your password when prompted (Note: When typing in your password, it will not show any characters typing)

4. You're now connected to PACE-ICE. You can edit files using vim by typing in:

   vi filename.cc OR *nano vilename.cpp*

   For a list of vim commands, use the following link:

   https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started

5. You're able to edit, compile, run, and submit your code from this command line.

Option 2 (Using PuTTY):

Option 2 uses a program called PuTTY to ssh into the PACE-ICE cluster. It is easier to set up, but doesn't allow you to access any local files from the command line. It also requires the use of simple bash commands.

1. Download and install PuTTY from the following link: www.putty.org

2. Once installed, open PuTTY and for the Host Name, type in:
   coc-ice.pace.gatech.edu and

   for the port, leave it as 22.

3. Click Open and a window will pop up asking if you trust the host. Click Yes and it will then ask you for your username and password. (Note: When typing in your password, it will not show any characters typing)

4. You're now connected to PACE-ICE. You can edit files using vim by typing in: *vim filename.cc*
   OR *nano vilename.cpp*

   For a list of vim commands, use the following link:

   https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started

5. You're able to edit, compile, run, and submit your code from this command line.

MacOS Users:.

Option 0 (Using the Terminal to SSH into PACE-ICE):

This option uses the built-in terminal in MacOS to ssh into PACE-ICE and use a command line text editor to edit your code.

1. Open Terminal from the Launchpad or Spotlight Search.

2. Once you're in the terminal, ssh into PACE-ICE by typing:

   *ssh **YourGTUsername**@ coc-ice.pace.gatech.edu* where **YourGTUsername** is replaced with your alphanumeric GT login. Ex: bkim334

3. When it asks if you're sure you want to connect, type in:
   yes

   and type in your password when prompted (Note: When typing in your password, it will not show any characters typing)

4. You're now connected to PACE-ICE. You can edit files using vim by typing in:

   vi filename.cc OR *nano filename.cpp*

   For a list of vim commands, use the following link: https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started

5. You're able to edit, compile, run, and submit your code from this command line.

Linux Users:

If you're using Linux, follow Option 0 for MacOS users.