ECE 4122/6122 Lab #1

(100 pts)

Section	Due Date
ECE 4122 - A	Sept 22nd, 2021 by 11:59 PM
ECE 6122 - A	Sept 22nd, 2021 by 11:59 PM
ECE 6122 – Q, QSZ, PDO	Sept 22nd, 2021 by 11:59 PM

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. Your programs need to check for invalid inputs and if an invalid entry occurs then your program should output "Invalid Input" in place of the normal output.

Submitting the Assignment:

The solution to each problem should be contained in a single file and you must use C\C++ as the programming language. Each file needs to be able to be compiled and run by itself. Each file should be labeled Lab1_Problem**#**.cpp, where **#** is problem number. **Do not zip up the files, instead upload the individual files to canvas.**

Useful links:

http://www.cplusplus.com/reference/fstream/ofstream/ofstream/ example of outputting to file

http://www.cplusplus.com/articles/DEN36Up4/ example of how to parse command line arguments.

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
Files named incorrectly	10%	Per problem.
Does Not Compile	30%	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)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score - 0.5*H	H = number of hours (ceiling function) passed deadline.

Problem 1: Prime Factors (25 pts)

(www.projecteuler.net) The prime factors of 13195 are 5, 7, 13 and 29.

Write a program, in a file named **Lab1_Problem1.cpp**, that takes one command-line argument variable of type ***unsigned long***.

Your program needs to contain two functions:

1. int *main*(int argc, char* argv[]) – parses the input parameter and calls the function

   GetPrimeFactors(const unsigned long ulInputNumber, string &strOutput) to calculate the prime factors of *ulInputNumber*.

2. bool *GetPrimeFactors* (*const unsigned long ulInputNumber, string &strOutput*) – determines the prime factors of *ulInputNumber.* If there are no prime factors then the function returns false, otherwise the function returns true. Any prime factors are concatenated into the function parameter *strOutput* using a comma to separate the factors.
   For example, if *ulInputNumber* = 13195, then *strOutput = "5,7,13,29"*

Your program must then output just *strOutput* to a text file called **output1.txt**. If the output1.txt previously existed, then your code needs to overwrite any existing information in the file. If there are no prime factors then your code needs to output "No prime factors"

This link describes how to parse command line arguments: http://www.cplusplus.com/articles/DEN36Up4/

This link describes how to output to a text file: http://www.cplusplus.com/doc/tutorial/files/

You can use this link as a guide but you can not just copy the code: https://www.geeksforgeeks.org/prime-factor/

Problem 2: Langton's Ant (25 pts)

(www.projecteuler.net) An ant moves on a regular grid of squares that are colored either black or white. The ant is always oriented in one of the cardinal directions (left, right, up or down) and moves from square to adjacent square according to the following rules:

- if it is on a black square, it flips the color of the square to white, rotates 90 degrees counterclockwise and moves forward one square.
- if it is on a white square, it flips the color of the square to black, rotates 90 degrees clockwise and moves forward one square.

Starting with a grid that is entirely white, you will need to determine how many squares are black after some number of moves of the ant?

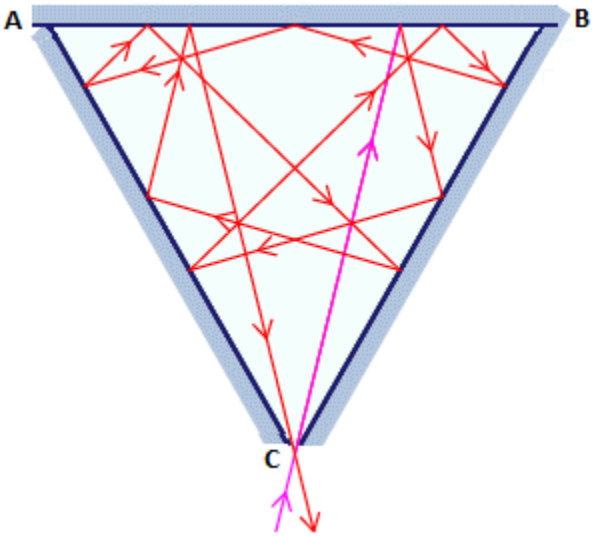
Write a program, in a file named **Lab1_Problem2.cpp**, that takes one command-line argument variable of type ***unsigned long*** that indicates the number of moves the ant makes. You might consider using a std::unordered_map (https://www.cplusplus.com/reference/unordered_map/unordered_map/) to record which squares are black.

Your program must then output just the number of black squares to a text file called **output2.txt**.

Problem3: Investigating multiple reflections of a laser beam (50 pts)

(www.projecteuler.net) In laser physics, a "white cell" is a mirror system that acts as a delay line for the laser beam. The beam enters the cell, bounces around on the mirrors, and eventually works its way back out. The specific white cell we will be considering is an equilateral triangle with a side length of 20 cm suspended symmetrically on the origin (the lower vertex C is at the origin) with the +x axis pointing to the right and the +y axis pointing up.

The sections corresponding to y ≤ 0.01 cm are missing, allowing the laser beam to enter and exit through the gap. The light beam always passes through the origin when entering the “cell”.



Each time the laser beam hits an interior surface, it follows the usual law of reflection "angle of incidence equals angle of reflection." That is, both the incident and reflected beams make the same angle with the normal line at the point of incidence.

The figure above illustrates a possible scenario.

Write a C++ program, in a file named **Lab1_Problem3.cpp** that takes a single command line argument that represents the initial reflection’s x location along the AB segment and calculates the number of times the beam is reflected off an internal surface of the white cell before exiting.

Your program must then output just this number of reflections to a text file called **output3.txt**.

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. *pace-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**@pace-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:
   pace-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**@ pace-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.