ECE 4122/6122 Lab #1

(100 pts)

Section	Due Date
89313 - ECE 4122 - A	Sept 14st, 2020 by 11:59 PM
89314 - ECE 6122 - A	Sept 14st, 2020 by 11:59 PM
89340 - ECE 6122 – Q, QSZ, Q3	Sept 17th, 2020 by 11:59 PM

ECE 4122 students need to do Problems 1, 2, 3.

ECE 6122 students need to do Problems 1, 2, 4.

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:

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 - (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: Summation of Primes (25 pts)

(www.projecteuler.net) The sum of the primes below 10 is 2 + 3 + 5 + 7 = 17.

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 uses a **for** loop to call the function *IsPrime()* to calculate the sum of numbers that are prime.
2. bool *IsPrime*(unsigned long lNum) – determines if the input number lNum is a prime number

Your program must calculate the sum of the prime numbers less than or equal to this input value. For input values of 0 or 1 the sum is 0.

Your program must then output just this sum value to a text file called **output1.txt**.

The link below describes how to parse command line arguments:

http://www.cplusplus.com/articles/DEN36Up4/

Problem 2: Collatz Sequence (25 pts)

(www.projecteuler.net) The following iterative sequence is defined for the set of positive integers:

n → n/2 (n is even)

n → 3n + 1 (n is odd)

As an example, using the rule above and starting with 13, we generate the following Collatz sequence:

13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

Write a program, in a file named **Lab1_Problem2.cpp**, that takes one command-line argument variable of type ***unsigned long***.

Your program must calculate the sum of the resulting Collatz Sequence.

Your program must then output just this sum value to a text file called **output2.txt**.

Problem 3 (ECE 4122 students only): An ant on the move (50 pts)

(www.projecteuler.net)

On the Euclidean plane, an ant travels from point A(0, 1) to point B(**d**, 1) for an integer **d**.

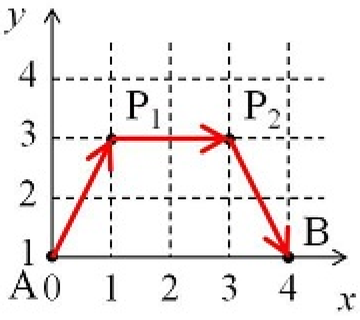
In each step, the ant at point (x0, y0) chooses one of the lattice points (x1, y1) which satisfy x1 ≥ 0 and y1 ≥ 1 and goes straight to (x1, y1) at a constant velocity v. The value of v depends on y0 and y1 as follows:

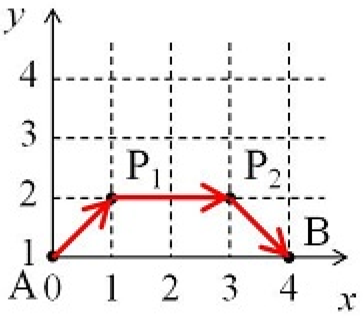
If y0 = y1, the value of v equals y0.
If y0 ≠ y1, the value of v equals (y1 - y0) / (ln(y1) - ln(y0)).

The left image below is one of the possible paths for d = 4. First the ant goes from A(0, 1) to P1(1, 3) at velocity (3 - 1) / (ln(3) - ln(1)) ≈ 1.8205. Then the required time is sqrt(5) / 1.8205 ≈ 1.2283.

From P1(1, 3) to P2(3, 3) the ant travels at velocity 3 so the required time is 2 / 3 ≈ 0.6667. From P2(3, 3) to B(4, 1) the ant travels at velocity (1 - 3) / (ln(1) - ln(3)) ≈ 1.8205 so the required time is sqrt(5) / 1.8205 ≈ 1.2283.
Thus the total required time for this path is 1.2283 + 0.6667 + 1.2283 = 3.1233.

The right image below is another path. The total required time is calculated as 0.98026 + 1 + 0.98026 = 2.96052. It can be shown that this is the quickest path for d = 4.





Let F(d) be the total required time if the ant chooses the quickest path. For example, F(4) ≈ 2.960516287.
We can verify that F(10) ≈ 4.668187834 and F(100) ≈ 9.217221972.

Write a program, in a file named **Lab1_Problem3.cpp**, that take one command-line argument variable of type ***unsigned long*** that represents the value of **d**.

Your program must calculate the time for the quickest path.

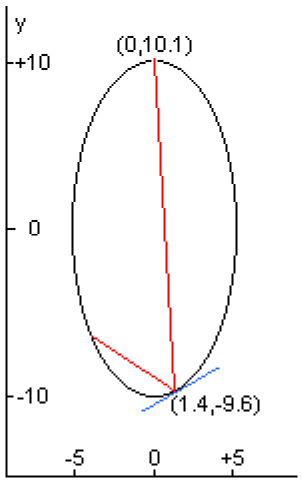
Your program must then output just this time value (to 8 decimal places) to a text file called **output3.txt**.

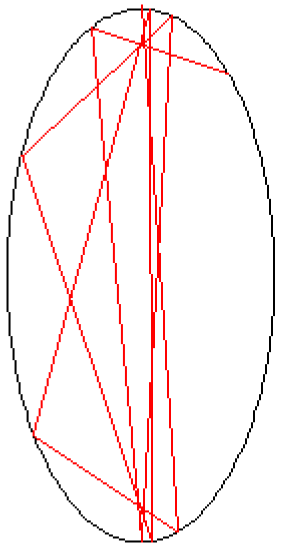
Problem 4 (ECE 6122 students only): Investigating multiple reflections of a laser beam (50 pts)

(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 ellipse with the equation **4x2 + y2 = 100**

The section corresponding to −0.01 ≤ x ≤ +0.01 at the top is missing, allowing the light to enter and exit through the hole. The light beam in this problem starts at the point (0.0,10.1) just outside the white cell, and the beam first impacts the mirror at (1.4,-9.6).

Each time the laser beam hits the surface of the ellipse, 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.





In the figure on the left, the red line shows the first two points of contact between the laser beam and the wall of the white cell; the blue line shows the line tangent to the ellipse at the point of incidence of the first bounce.

The slope m of the tangent line at any point (x,y) of the given ellipse is: m = −4x/y

The normal line is perpendicular to this tangent line at the point of incidence. The figure on the right shows the first several reflections of the beam.

Write a program, in a file named **Lab1_Problem4.cpp** that 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 **output4.txt**.

https://www.emathzone.com/tutorials/geometry/intersection-of-line-and-ellipse.html

https://www.wikihow.com/Find-the-Equation-of-a-Perpendicular-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. *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.