ECE 4122/6122 Hmk #1

(100 pts)

Section	Due Date
89313 - ECE 4122 - A	Sept 1st, 2019 by 11:59 PM
89314 - ECE 6122 - A	Sept 1st, 2019 by 11:59 PM
89340 - ECE 6122 - Q	Sept 8th, 2019 by 11:59 PM
89706 - ECE 6122 - QSZ	Sept 8th, 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:

The solution to each problem should be contained in a single file. Each file needs to be able to be compiled and run by itself. Each file should be labeled Hmk1_Problem**#**.cpp, where **#** is 1, 2, or 3. All files should be zipped up into a single file named: <Your_Name>_Hmk1.zip and uploaded onto canvas under the assignment.

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
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)

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: One more One

(www.projecteuler.net) Consider the following process that can be applied recursively to any positive integer **n**:

- if **n**=1, do nothing and the process stops,
- if **n** is divisible by 7, divide it by 7,
- otherwise add 1.

The write a console program that takes in a number **n** from the console and outputs to the console **the number of 1's that must be added** to the positive integer **n** before the process above ends.

Sample Sequence:

$$125 \xrightarrow{+1} 126 \xrightarrow{\div 7} 18 \xrightarrow{+1} 19 \xrightarrow{+1} 20 \xrightarrow{+1} 21 \xrightarrow{\div 7} 3 \xrightarrow{+1} 4 \xrightarrow{+1} 5 \xrightarrow{+1} 6 \xrightarrow{+1} 7 \xrightarrow{\div 7} 1$$

Example Input:

Please enter the starting number n: 125

The sequence had 8 instances of the number 1 being added

Problem 2: Numerical Integration

Write a console program to numerically calculate the value of the following integral, using the "midpoint" rule.

$$\int_0^{\beta} \frac{4}{(1+x^2)} dx$$

The program should ask the user to enter β (type double) and the number of subdivisions, N (type unsigned long) on separate lines. The value (type double) of the integral should be displayed on the following line.

Example Input:

Please enter a value for the upper limit (beta): 1.0

Please enter the number of subdivisions to use: 10000000

The integral evaluates to: <answer>

Check your code:

When β =1.0 the integral is equal to π. Your answer should approach the exact value of π as N increases.

Midpoint Rule:

The integral is equal to the area underneath the function from 0 to β. We can approximate the integral as the sum of the area of rectangles. The height of the rectangle is F(x) at the midpoint of the rectangle and the width is Δx as shown below.

$$\int F(x)dx \cong \sum_{1}^{N} F(x_i) * \Delta x$$

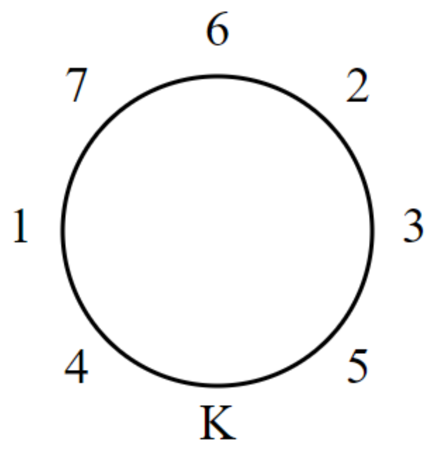
Problem 3 (ECE 6122 students only):

(www.projecteuler.net) The Knights of the Order of Fibonacci are preparing a grand feast for their king. There are **n** knights, and each knight is assigned a distinct number from **1 to n**.

When the knights sit down at the roundtable for their feast, they follow a peculiar seating rule: two knights can only sit next to each other if their respective numbers sum to a Fibonacci number.

When the **n** knights all try to sit down around a circular table with **n** chairs, they are unable to find a suitable seating arrangement for any n > 2 despite their best efforts. Just when they are about to give up, they remember that the king will sit on his throne at the table as well.

Suppose there are **n = 7** knights and 7 chairs at the roundtable, in addition to the king's throne. After some trial and error, they come up with the following seating arrangement (K represents the king):



Following the rules above, write a console program that prompts the user for the number of knights, **n** (type unsigned long). The program then determines if the rules above can be satisfied. If a solution is found, then output to the console the solution in a clockwise fashion starting with the king. If no solution is possible then output "No solution found!"

Example Input/Check your code:

Please enter the number of knights: 7

The knights should sit clockwise as follows: K, 4, 1, 7, 6, 2, 3, 5

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.

<u>File Headers</u>:

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?

*/
```

<u>Code Comments:</u>

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 CoC 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.