ECE 4122/6122 Lab 0: Getting PACE-ICE Access and Using the g++ Compiler

(100 pts)

Category: Getting Started
Due: Monday September 12th, 2022 by 11:59 PM

Problem 1:

This problem is very simple. Write a C++ program using the insertion stream operator and escape sequences that outputs the following text to your terminal screen when executed:

```
My name is: (your first and last name separated by a space)
This (") is a double quote.
This (') is a single quote.
This (\) is a backslash.
This (/) is a forward slash.
```

This program is very simple with no user input, command arguments, or file output. You can place all the code in your *main*() function in a file called **Lab0_Problem1.cpp**.

Your PACE-ICE accounts have already been created.

If you are trying to logon off campus, you will need to setup a VPN. Instructions are on the OIT website at Georgia Tech.

Problem 2: 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 continuously 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.

Entering a 0 ends the program.

Make sure code check for valid input values.

Place your code in the file **Lab0_Problem2.cpp**.

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 Program Input\Output:

```
Please enter the starting number n: 125

The sequence had 8 instances of the number 1 being added.

Please enter the starting number n: -1

Invalid input!! Please try again.

Please enter the starting number n:
```

Turn-In Instructions

Place your two cpp files in a zip file called **Lab0.zip** and upload this zip file on the assignment section of Canvas.

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 look through your code for other elements needed to meet 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 – 0.5 * H	H = number of hours (ceiling function) passed deadline

<u>Appendix A: Coding Standards</u>

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 (section)
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.