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

(100 pts)

Category: Getting Started
Due: Wednesday September 6th, 2023 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.

<u>Problem 2:</u> Sum of the Multiples of 3 or 5

(www.projecteuler.net)

The write a console program that **continuously** takes in a **natural** number from the console and outputs to the console all the numbers below the entered number that are multiples of **3 or 5** and then **outputs the sum** of all the multiples.

Entering a 0 ends the program.

Make sure code checks for valid input values. Entries must be positive and be made up of numeric characters (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). A sample sequence is shown below and your output should **match the formatting** shown in the sample.

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

Sample Sequence:

```
> Please enter a natural number (0 to quit): 10

> The multiples of 3 below 10 are: 3, 6, 9.

> The multiples of 5 below 10 are: 5.

> The sum of all multiples is: 23.

> Please enter a natural number (0 to quit): 0

> Program terminated.

> Have a nice day!
```

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

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 represents. 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.