University of Pittsburgh
Department of Electrical and Computer Engineering
ECE 0301 – Spring 2021

In-Class Assignment #7

Programming concepts:

- Classes and objects
- Data hiding: private member variables and public member functions
- Constructors

ECE concepts:

- Complex number
- NOT gate
- Flashlight

Develop software according to the following specifications and submit whatever portion you have completed to the class repository before midnight tonight.

1. Obtain the `C++` program `Pr13-1.cpp` from course website (under Modules `->` Example Programs `->` chapter 13.zip). This file contains a complete definition for a Rectangle class, and a simple program that allows the user to choose the width and length, then uses member functions to set these variables with the user data, and uses other member functions to access the width, length and area.
   Read through the file and run the program. Experiment with it and make sure you understand how the class is defined and how the member functions are used.

2. Save this program under the name `ece0301_ICA07_cmplx_nmbr.cpp`, and modify it so that it defines a class for a complex number, and a short program to allow the user to define a complex number, and then display information about the complex number object.
   Your class must have private member variables for the real and imaginary parts of the complex number. It must also have public member functions as follows:
   - Mutator functions that set the real and imaginary parts
   - Accessor functions that return the real and imaginary parts
   - Accessor functions that compute and return the magnitude and phase angle. Use the `atan2` function from the `cmath` library to compute the phase.

Your `main` function must:

- Prompt the user with this message:
```
This program will calculate the magnitude and phase
angle of a complex number.

What is the real part?
```
- Read user input from the keyboard.
- Prompt the user with this message:
```
What is the imaginary part?
```
- Read user input from the keyboard.
- Use the mutator functions to set the real and imaginary parts accordingly.
- Use the accessor functions to get the real and imaginary parts.
- Use the accessor functions to get the magnitude and phase angle.
- Print messages to standard output to report information about the complex number. The proper output format is shown below, for the complex number $z = 5 + j12$.
```
Here are the data on complex number z:
Real part: 5
Imaginary part: 12
Magnitude: 13
Phase angle (radians): 1.17601
```

Test your program and demonstrate correct output.

3. Write a program named `ece0301_ICA07_NOT_gate.cpp` that includes the definition of a class for a NOT gate, and a `main` function that that uses the class. The private member variables are a `string` variable to store a text label used to identify the input node, and a `bool` variable used to store the logic value at the input node. The public member functions are:
   - A mutator function that takes a `string` parameter and sets the input label.
   - A mutator function that takes a `bool` parameter and sets the input logic value
   - Accessor functions that return the input label and input logic value.
   - An accessor function that returns a `string` variable containing the label for the output node, which is equal to the input label with the string literal "`_not`" appended.
   - An accessor function that returns a `bool` variable containing the logic value at the output node, which is equal to the complement of the logic value at the input node.

   Your `main` function must:
   - Prompt the user with this message:
```
This program will simulate a not gate.
What is the label for the gate input?
```
   - Read user input from the keyboard.
   - Prompt the user with this message:
```
What is the logic value at the input (0/1)?
```
   - Read user input from the keyboard.

- Print messages to standard output to report information about the NOT gate, following the format shown below. Each `cout` statement should use the accessor functions to obtain the information needed.

```
Here are the data on the not gate:
Input label: x
Output label: x_not
Logic value at input: 0
Logic value at output: 1
```

4. Write a program named `ece0301_ICA07_flashlight.cpp` that includes the definition of a class to simulate a flashlight, and a `main` function that uses the class. The name of the class must be `FlashLight`.

   The private member variables are two `double` variables that store the battery voltage and bulb resistance, and a `bool` variable that stores the switch state. When the switch state is `true`, the switch is closed, the flashlight is on, and current flows from the battery through the switch and bulb, and back to the battery. When the switch state is `false`, the switch is open, the flashlight is off, and no current flows.

   The public member functions are:
   - A Constructor that initializes the battery voltage to 3.0 V, the bulb resistance to 100 Ohms, and the switch state to `false`.
   - Two mutator functions that take a `double` parameter and set the battery voltage and bulb resistance, respectively.
   - A mutator function that toggles the switch state, i.e. changes the switch state to the complement of its current value.
   - Three accessor functions that return each of the member variables.
   - Two accessor functions that return the current flowing in the bulb and the power dissipated in the bulb.
   - A function that prints to standard output all available information about the current state of the flashlight, by calling the various accessor functions and reporting the values that are returned. The output format is shown below, for the case when a flashlight object has been created with the default values for the member variables used in the constructor:

```
Here are the data on the FlashLight:
The battery voltage is 3 Volts.
The bulb resistance is 100 Ohms.
The switch is open.
The FlashLight is off.
The bulb current is 0 Amperes.
The bulb power is 0 Watts.
```

Write a `main` function that is a menu-driven interactive simulation of a flashlight.

- Declare an instance of your flashlight class, so that the constructor is called.
- Call the member function to display all information about the flashlight.
- Implement a loop structure that will terminate when the user chooses to end the program. During each iteration of the loop, the user must be presented with a menu of four options:

```
(1) Change the battery voltage
(2) Change the bulb resistance
(3) Toggle the switch state
(4) Exit the program
```

- After the user makes their selection, update the simulated flashlight as instructed by the user. If the user chooses to change the battery voltage, prompt the user with:

```
Enter the new battery voltage:
```

  Then read the user response from the keyboard, and call the mutator function to update the battery voltage.
- If the user chooses to change the bulb resistance, prompt the user with:

```
Enter the new bulb resistance:
```

  Then read the user response from the keyboard, and call the mutator function to update the bulb resistance.
- If the user chooses to toggle the switch state, call the mutator function that does this. There is no need to prompt the user here.
- If the user chose any of the options 1-3, call the member function (at the end of each option) to display all information about the flashlight, and return to the top of the loop, so that the menu is displayed again.
- If the user chooses to end the program, do nothing. Exit the loop, so that the program ends.
- If the user enters anything other than 1-4, return to the top of the loop, so that the menu is displayed again.

Test your program and confirm that it operates as expected, and provides the correct values for all circuit parameters, for several different values of the member variables.

5. Break your program up into three files:
   - A *class specification* file: a header file named `FlashLight.h` that contains the class declaration. It declares the member variables and member functions, indicates whether they are public or private, and indicates their data types (where appropriate). It does **not** include the definitions of the member functions.
   - A *class implementation* file named `FlashLight.cpp` that contains the definitions of the member functions.
   - A program file named `ece0301_ICA07_flashlight.cpp` that contains **only** the `main` function. Make sure you use the `#include` preprocessor directive with the header file for your class. If you are using the Geany IDE to develop your programs, you will have to provide `#include` preprocessor directives for both the class specification file and the class implementation file.

Test your program and confirm that it operates as expected.

6. Submit all of the following files to the class repository:
   `ece0301_ICA07_cmplx_nmbr.cpp`
   `ece0301_ICA07_NOT_gate.cpp`
   `ece0301_ICA07_flashlight.cpp`
   `FlashLight.h`
   `FlashLight.cpp`