**C++ Programming: Class Creation Program Assignment Instructions**

**Overview**

The purpose of this assignment is to give you some practice with creating your own classes. This program serves as the basis for all of the other programming assignments in this class and your future Computer Science classes.

**Instructions**

Construct a class named **Rectangle**.

Constructor:

* Zero-argument constructor that initializes the data members to 0.

Data Members:

* **width** – double or float
* **length** – double or float

Functions:

* **calcPerimeter(),** which calculates and returns the perimeter of the rectangle
* **calcArea(),** which calculates and returns the area of the rectangle
* **calcDiagonal(),** which calculates and returns the length of the diagonal line from opposite corners of the rectangle
* **showData(),** which displays the rectangle’s width, length, diagonal length, perimeter, and area
* **setWidth(),** which sets the width data member
* **getWidth(),** which returns the width data member
* **setLength(),** which sets the length data member
* **getLength(),** which returns the length data member

Formulas:

* Area = width \* length
* Perimeter = (2 \* width) + (2 \* length)
* Diagonal length = square root of ((width squared) + (length squared))

The class should use appropriate protection levels for the member data and functions. Data members need to be in the private: section. It should also follow “principles of minimalization”: that is, no member data should be part of a class unless it is needed by most member functions of the object. A general rule of thumb is that “if you can easily calculate it, don’t store it.”

Use your class in a program that creates an instance of a **Rectangle**(utilizing the zero-argument constructor), prompts a user for **width** and **length**, calls the **setWidth()** and **setLength()** function to set the rectangle’s sides, and then calls **showData()** to display the rectangle’s width, length, diagonal length, perimeter, and area.

Your program should allow the user to enter new rectangle dimensions until the user enters **-1** for the width. Be sure to include appropriate error checking. Does it make sense to enter “abc” as the side of a square? No. Therefore, you should ensure that the user enters numeric data for the side. Negative numbers (other than the -1 to exit) should also be prevented.

Here is an example of how the input and output should look.

Graphical user interface, text

Description automatically generated

Be sure to test your program with several different sets of widths and lengths.

**Style**:

* Your lab should be constructed such that separate files are used: Rectangle.h (your class declaration file), Rectangle.cpp (your class implementation file), and RectangleDriver.cpp (the file that contains main() and any other functions that are not part of the class).

The purpose of having separate files is for code reusability. If other developers want to use your class in their programs, they don't need main() as well. They should only have to "#include" your class header file. As such, two separate files for the class are needed to be able to hide the implementation details from other developers. You want other developers to know how to use your class (i.e., what functions are available and how they are called -- this is called the "interface" of the class), but not how they are implemented. This cannot be accomplished if both the interface and the implementation code are in the same file. When you distribute your class to other developers, the implementation (.cpp) file gets compiled, but the interface (.h) doesn't. That way, the developer can use your class, but they can't see or change your code in your class functions.

* Never use “**using namespace std;**” in a header file. Here is a link that describes why: <https://stackoverflow.com/questions/14575799/using-namespace-std-in-a-header-file>
* Note that when you follow the rule above and don’t use “**using namespace std;**” in a header file, you must remember to prefix certain things with std:: (i.e., cout, cin, endl, string, istream, ostream, vector etc.). If you fail to prefix these words with std::, you will get a compilation error that is often quite cryptic. Be on the lookout for this situation. It may save you hours of debugging.
* All functions that are to return a value, “get”, “calc”, etc., should be declared as constant. Example: **string myProcess() const;**. **Any** function that does not change the data in the data members should be declared as constant. This is a security measure that prevents anyone from accidentally writing code in a function that changes underlying data when the purpose of the function is only to retrieve the data. In other words, "const" at the end of a function protects your data. The rationale is that certain functions (e.g., those that merely return a data item to the caller or even those that just print the data) should be prevented from ever changing the underlying data. If a function doesn't need "write" access to a data item, it shouldn't be granted access. This adheres to the “Principle of Least Privilege,” a security principle that helps protect the integrity of your data.

Another important thing to know for future reference. Whenever you make a function constant, it cannot call another function unless that function is also constant. It makes sense that if you lock down an object in a function, you don't want to open it up for modifications by calling another function that allows it to be inadvertently altered. If you ever try to call a function that is not constant from a constant function, you will get a compilation error.

* The function **showData()** should not refer to the data members directly. Instead, it should call **getWidth()** and **getLength()** to retrieve the values. Think of your getters and setters as middlemen. You should avoid accessing your data members directly due to maintenance issues. Pretend that you decided to change the name of your width and length variable to "wid" and “lng”. You really should only have to make that change in four functions: getWidth(), setWidth(), getLength(), and setLength(). If you refer to the private data members directly throughout your code, you will have a nightmare trying to make all necessary changes when the data member name changes. If you always use the getters to retrieve your data, any changes to the variable name will only necessitate changes to your get function rather than your whole codebase.
* The functions **calcPerimeter()**, **calcArea()**, **calcDiagonal()**, and **showData()** should **not** have the width and length passed in as arguments because **width**and**length** are data members of the class. Class functions have **direct** access to all data members in their own class through the getters. When you pass in an argument for the value of the width and length, you're not using the value that has already been stored in the corresponding data member. You're using the value that you've passed into the functions, which circumvents the whole purpose of storing data in the class.
* Use good prompting (correct spelling and clear instructions), output labeling, and modularization in your program and class.
* Be sure to avoid using global variables in your program unless they are constants.
* Finally, be sure not to include any unnecessary libraries.

**Comments and Error Handling**:

* Remember to incorporate the comments and error handling listed in the Lab 1 instructions.

**Deliverables**:

* Complete the programming assignment described above and submit your completed assignment in accordance with the assignment submission policies.

Below is a checklist you will want to go through before submitting the lab.

