Lab 3: Introduction to classes and objects

CPSC 1021

School of COMPUTING Clemson University

Lab Objective

- Learn how to write a UML class diagram
- Practice writing classes in C++
- Practice using objects in a C++ program
- Practice separating class specification, implementation, and client code
- Practice using member functions that return a Boolean value for input validation
- Practice writing a Makefile for a multi-file project

Introduction

In this week's lab, we will revisit our first programming assignment, the Areas Calculation program from Lab 1. This time, you will create a solution using classes and objects. To design and plan your program, we will introduce the *Unified Modeling Language* that you will use to design a class diagram for each class in your program.

Collaboration Policy

Work with your assigned team members from Lab 0.

You and your partners can work on all assignments together, but every student has to submit their own files to Canvas and Gradescope for this lab.

Tutorial: UML Class Diagrams

The Unified Modeling Language (UML) is a modeling language used in software engineering do illustrate and visualize a system design. UML is *not* a programming language; UML is language agnostic. It is a standardized way to plan and communicate the design of a system.

Here we will go through the steps of writing a class diagram for our Square class from lecture.

1) Start with a box divided into three sections. Enter the following information:

- top section: name of the class
- middle section: list of the class attributes/member variables
- bottom section: list of the class member functions

Square
side
setSide() getSide()

2) For each class member, we indicate whether the member is private ("-") or public ("+"). For now, we set all member variables to private and all member functions to public.

Square
- side
+ setSide() + getSide()

3) UML provides a way to indicate a variable's data type. This is done by placing a colon after the variable followed by the data type. (Note that this notation is different from C++ syntax where a variable's data type is listed before its name.) Here we declare `side` to be an integer variable. We use the same notation to indicate the data types of function parameters and a function's return type.

Square
- side: int
+ setSide(s: int): void + getSide(): int

That's it! You will practice writing UML Class Diagrams for the following programming assignment.

Areas Calculation

Create a menu-driven program that calculates and displays the areas of four different shapes. The menu should have the following five choices (note that these choices are slightly different from lab 1):

1 -- circle

2 -- square

3 -- rectangle

4 -- trapezoid

5 -- quit

Make a separate class for each of the four shapes. For each class, create a .h file that contains the class declaration and a .cpp file that contains the function definitions. Use the class name as the file name for both the .h and .cpp file.

> *Before you start working on this assignment, carefully read chapter 7.11 "Separating Class Specification, Implementation, and Client Code" in our textbook. This chapter provides an example of the Rectangle class that you can re-use for this assignment. Use the Rectangle class as an example for how you need to design and implement the other three classes.*

Teamwork Instructions

There is a lot of code to write for this assignment. Use a divide and conquer strategy. Work together on the design document (the class diagrams and pseudocode), then have each team member work on one class. Once all classes are written, collaborate on the coding of the main program.

Class Specification

- In each class, set the member variables to private and the member functions to public.
- Declare each member variable as double.
- Each class should have the following member functions:
  - A separate setter-function for each member variable. These functions should return a Boolean value. If the argument passed to the function is valid (i.e., a value >= 0), the value passed to the function should be stored in the member variable and `true` returned; if an invalid argument is received, the member variable should be left unchanged and `false` returned.
  - A separate getter-function for each member variable that returns the value of the variable.
  - A single function that returns the area for the shape defined by the class.
- Do not use *cin* or *cout* in any of your member functions. I/O operations should be defined in the main program. Read subsection "Performing Input/Output in a Class Object" in chapter 7.11 for an explanation why I/O operations should, in most cases, be avoided inside a class.

Client Program Specification

Your main program should have the following functionality:

- A menu from which the user can select a shape (menu choices 1-4, in the order displayed above) or quit the program (menu choice 5). If the user selects an invalid choice, your program should prompt the user for a valid menu selection. Implement this menu validation inside main().
- When selecting a valid choice 1-4, your program should:
  - Instantiate an object from the correct class and declare all variables needed to collect user input.
  - Use `cin` to collect user input.
  - Call the correct member function(s) to set the values for the member variable(s). Remember that the setter functions should return a Boolean value. Your program must test the returned Boolean value and prompt the user for new input if the user has entered an invalid value.
  - After all necessary user input was collected, your program should use the correct member functions to display the results, following the example output below.

Example output

Follow these examples when formatting your output.

- For menu choice 1, a circle with radius 2:
```
Radius: 2.0
Area: 12.6
```
- For menu choice 2, a square with side=2:
```
Side: 2.0
Area: 4.0
```
- For menu choice 3, a rectangle with length=2.5 and width=3:
```
Length: 2.5
Width: 3.0
Area: 7.5
```
- For menu choice 4, a trapezoid with parallel base1 = 2.5, parallel base2=3.5, and height = 4.0:
```
Base1: 2.5
Base2: 3.5
Height: 4.0
Area: 12.0
```

Makefile

Create a Makefile for your program that includes the following three targets: all, run, clean. Follow the format for lab 2 when writing your Makefile. **Your Makefile should create an executable named main.out**.

Submission Instructions

1. Complete the Word template "Lab3_Design_Document.docx" and submit it via Canvas. Follow the naming convention explained in this document when designing and implementing your classes.
2. Submit your complete and correct program to Gradescope. Your submission must include the following 10 files:
    1. areaCalc.cpp (this file includes `main()`)
    2. Circle.h
    3. Circle.cpp
    4. Rectangle.h
    5. Rectangle.cpp
    6. Square.h
    7. Square.cpp
    8. Trapezoid.h
    9. Trapezoid.cpp
    10. Makefile
3. Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.
4. Code style (proper indentation, consistency, readability, use of comments, etc.) will be considered when grading your submission.

The Gradescope autograder will check that your submission is complete and that your code compiles with the Makefile you have submitted. There are also tests to check that your program produces the expected output (please follow the above examples when formatting your output). Additional tests that check your classes and objects for correctness may be added after the submission deadline.