ECE 244 Programming Fundamentals Fall 2021

Lab #5: Inheritance and Virtual Functions

1. Objectives

This lab is an exercise on inheritance and virtual functions, which are powerful code reuse mechanisms of C++. More specifically, you will enhance a program that implements database of shapes that stores, manipulates, and "draws" new shapes that did not exist when the program was written.

2. Problem Statement

You are provided with a framework that implements a database of shapes, in the form of an object file. Without changing the code of this framework (you really cannot since you do not have the source), you will add new functionality simply by adding three new classes (`Circle`, `Triangle` and `Rectangle`). The framework provides the "higher-level" decision making of the program, and you add new features by creating new classes that the framework calls when appropriate, via virtual functions. This is a powerful form of code re-use; the framework is making use of objects that did not exist when it was written.

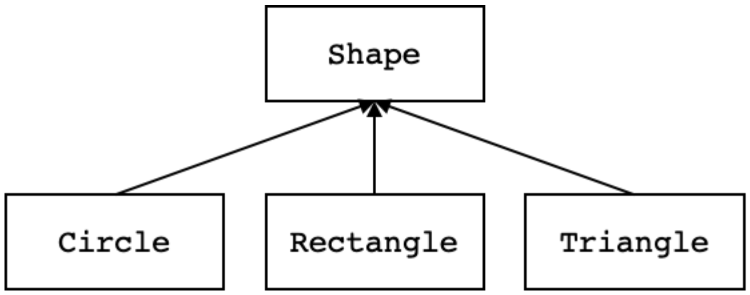
3. The `Shape` Class

The `Shape` class is an abstract class that serves as a root for inheritance. The definition of this class is given in the file `Shape.h`. The class has several `protected` variables and a number of abstract virtual functions that must be implemented by derived classes. The implementation of the class is considered part of the framework, and as such you are only provided with `Shape.o`.

Please note that you must not modify the file `Shape.h` in any shape or fashion. Doing so will result in a mark of 0 for the assignment.

4. The Derived Classes

You will extend the framework to maintain and manipulate three types of shapes: circles, rectangles and triangles. Thus, you must define and implement three classes: `Circle` (in `Circle.h` and `Circle.cpp`), `Rectangle` (in `Rectangle.h` and `Rectangle.cpp`) and `Triangle` (in `Triangle.h` and `Triangle.cpp`). The following figure shows the inheritance hierarchy.



The definition of the `Circle` class as well as its implementation are given to you as an example of how to inherit from `Shape` and extend its functionality to implement a `Circle`. Thus, you can follow this example to define and implement the remaining two classes.

5. The `ShapesDB` Framework

The `ShapesDB` framework implements a simple database for shapes. Indeed, it uses the array of shape pointers you implemented in Assignment 3. As released, it has no specific shapes defined in it. You will create new shapes (circles, rectangles and triangles) and the framework store and manipulate these new shapes, even though they did not exist when the framework was written.

The framework accepts and executes a number of commands (see Section 6 below). In addition to defining new shapes, you will extend the commands the framework accepts and processes. This is done through a simple "call back" mechanism, where you *register* your function that parses a command with the framework and the framework *calls back* this function when it receives the corresponding command. An example of doing so is given to you to illustrate this process and also to demonstrate the type of error checking required.

You are not given the source code for the framework (i.e., the file `ShapesDB.cpp` is hidden from you). However, you do get the file `ShapesDB.h` to see what functionality the framework provides and the file `ShapesDB.o` to link with.

6. Command Reference

The following are the commands that you will need to extend the framework with. Text in *italics* indicate arguments that should be replaced by the appropriate strings or numbers.

- `circle` *name xcen ycen radius*

  Create a new `Circle` object, with the specified *name*, with a center at (*xcen,* ycen) and the specified *radius. name* is a string (which cannot be a reserved word), while *xcen, ycen* and *radius* are `floats`. The code to process the `circle` command is included in the `main` function of the program (in `Parser.cpp`) as an example. You will need to extend this file to process the rest of the commands in a similar way.

- `rectangle` *name xcen ycen width height*

  Create a new `Rectangle` object, with the specified name, a center at (*xcen, ycen*) and the specified *width* and *height. name* is a string, while *xcen, ycen, width* and *height* are `float`s.

- `triangle` *name x1 y1 x2 y2 x3 y3*

  Create a new `Triangle` object, with the specified name and with 3 vertices given by (*x1, y1*), (*x2, y2*) and (*x3, y3*). *name* is a string while *x1* through *y3* are `float`s.

The following commands are already implemented in the framework, and you need not implement them.

- `area`

  Compute the total area of all the shapes in the database and outputs this sum. The computation is done in float arithmetic and is printed out as a `float` with two decimal digits (see the `Circle` class for doing this).

- `draw`

  "Draws" all the shapes by printing them to `cout`. For all shapes, the shape type is printed, followed by a "`:`", followed by the shape *name*, followed by the *x* and *y* locations of the centre of the shape. In the case of a circle the *radius* follows. In the case of a rectangle, the *width* and *height* follow. Finally, in the case of a triangle, the coordinates of the *vertices* follow. Finally, for all shapes the area of the shape is printed. Thus, the output format for the three shapes is:

  `circle:` *name xcen ycen radius area*
  `rectangle:` *name xcen ycen width height area*
  `triangle:` *name xcen ycen x1 y1 x2 y2 x3 y3 area*

  All `floats` are printed using a fixed floating-point notation with 2 decimal precision. See the sample session below for examples of printing and `Circle.cpp` for how to print using the required format.

7. Sample Session Output

The following is a sample session. It illustrates the format of "drawing" the shapes, the response generated for each command and the various error messages that may arise. Please note that an example of error checking is given in `Parser.cpp` and you can just mimic it for the other classes. Your code will not be tested for any other errors. The ">" is the prompt printed by the command parser.

```
> circle c1 1 1 4
created circle
```

```
> rectangle r1 5 5 4 8
created rectangle
> triangle t1 8 8 10 10 12 8
created triangle
> draw
circle: c1 1.00 1.00 4.00 50.27
rectangle: r1 5.00 5.00 4.00 8.00 32.00
triangle: t1 10.00 8.67 8.00 8.00 10.00 10.00 12.00 8.00 4.00
> area
Total area = 86.27
> circles c2 1 1 1
Error: invalid command
> circle triangle 1 1 1
Error: triangle is a reserved word
> triangle r1 10 10 20 20 40 30
Error: a shape with the name r1 already exists
```

Please note that rounding may cause some variations between the sum of the rounded areas of the shapes and the total area printed. Here is an example:

```
> triangle t2 1.00 1.00 2.00 2.00 0.00 3.00
created triangle
> triangle t3 1.30 4.10 0.10 9.90 3.41 3.14
created triangle
> triangle t4 0.00 0.00 0.00 0.99 0.45 0.13
created triangle
> draw
triangle: t2 1.00 2.00 1.00 1.00 2.00 2.00 0.00 3.00 1.50
triangle: t3 1.60 5.71 1.30 4.10 0.10 9.90 3.41 3.14 5.54
triangle: t4 0.15 0.37 0.00 0.00 0.00 0.99 0.45 0.13 0.22
> area
Total area = 7.27
```

The sum of the rounded areas of the three triangles is 7.26. However, the total area is 7.28. The same variations may happen across platforms (Windows, MacOS and Linux). This is acceptable in both cases.

8. Procedure

Download the release zip file from Quercus and unzip it in your `ece244` directory. This will create a subdirectory for the assignment that contains the various source/object files as well as a `Makefile`. Specifically, the release contains: `Shape.h`, `Circle.h` and `ShapesDB.h`, which you must **not** modify. It also contains the object files `Shape.o` and `ShapesDB.o`. Finally, it contains the files `Rectangle.h` and `Rectangle.cpp`, in which you should define and implement the `Rectangle` class, and `Triangle.h` and `Triangle.cpp`, in which you should define and implement the `Triangle` class. There is some skeletal code in each of these files. The `Makefile` is used by `NetBeans` to separately compile your project. Do not modify this file either.

You are to implement two classes, `Rectangle` and `Triangle` in the respective `.h` and `.cpp` files within the release. You may use the definition/implementation of Circle as a guide. You will also have to add a small amount of code to `Parser.cpp` to parse and create new Rectangle, and Triangle objects after parsing the `rectangle` and `triangle` commands, respectively.

The centre point of a triangle is computed as follows. The *x* coordinate of the centre point is the average of the *x* coordinates of its three vertices. Similarly, the *y* coordinate of the centre point is the average of the *y* coordinates of its three vertices. You may want to look up how to calculate the area of a triangle from its three vertices (e.g., https://sciencing.com/area-triangle-its-vertices-8489292.html).

If you wish, you may include `cmath` (`#include <cmath>`) to aid with the calculations.

The `~ece244i/public/exercise` command will also be helpful in testing your program. You should exercise the executable, i.e., `Parser.exe`, using the command:

```
~ece244i/public/exercise 5 Parser.exe
```

As with previous assignments, some of the exercise test cases will be used by the `autotester` during marking of your assignment. We will not provide all the `autotester` test cases in exercise, however, so you should create additional test cases yourself and ensure you fully meet the specification listed above.

Your program must not leak memory. Please use `valgrind` to test for memory leaks and eliminate them. Programs that leak memory will be penalized.

9. Deliverables

Submit your `Rectangle.h`, `Rectangle.cpp`, `Triangle.h`, `Triangle.cpp` and `Parser.cpp` using the command:

```
   ~ece244i/public/submit 5
```