ENSF 607 – Software Design and Architecture I
Winter 2020

Project: Inheritance in C++

Due Dates	
Post-Lab	Submit electronically on D2L **before 11:59 PM on Monday April 20**

This is a Group assignment. Students can work in groups of two. You can pair up regardless of your TA groups.

The objective of this lab is to:

- Understand the concepts of inheritance, multiple inheritance, and polymorphism in C++.

The following rules apply to this lab and all other lab assignments in future:

1. Before submitting your lab reports, take a moment to make sure that you are handing in all the material that is required. If you forget to hand something in, that is your fault; you can't use `I forgot' as an excuse to hand in parts of the assignment late.

2. **20% marks** will be deducted from the assignments handed in up to **24 hours** after each due date. It means if your mark is X out of Y, you will only gain 0.8 times X. There will be no credit for assignments turned in later than 24 hours after the due dates; they will be returned unmarked.

Lab (55 Marks)

Exercise – 1: Following inheritance with an AR diagram (20 marks)

Important Note: The material in this lab (Inheritance, polymorphism, and multiple inheritance), is one of the very important topics in object oriented-programming and particularly in C++. If you are working in a group, you are strongly advised to ensure both partner participate in developing the code, and equally understand the details of these concepts.

What to do: Download file `Lab5_Exe_1.cpp` from D2L and draw memory diagrams for point **one** (14 marks) and point **two** (6 marks).

What to submit: Submit an electronic image of the diagram as part of your in-lab report (in a PDF file) on D2L.

Exercise 2 – Part I – Single Inheritance in C++ (25 Marks)

Introduction: The concepts of aggregation, composition, and inheritance in C++ and Java in terms of concepts are similar, but they are completely different in terms of implementation and syntax. Also another major difference between the two languages is that C++ unlike Java supports multiple-inheritance.

What to Do: In this part, first you will write the definitions of following classes: `Point`, `Shape`, `Rectangle`, `Square`, `GraphicsWorld`, as explained below.

`Class Point:`

This class is supposed to represent a point in a Cartesian plane. It should have three data members: x, and y coordinates and an automatically created id number. The first object's id number should be 1001, second one should 1002, and so on. `Point` should have at least the following member functions:

- `display` - that displays x, and y coordinates of the point in the following format:
  `X-coordinate:`
  `Y-coordinate:`
- A constructor that initializes its data members.

  Note: You are not supposed to define a default constructor in this class. Automatic calls to the default constructor will hide some of the important aspects of this assignment (marks will be deducted if you define a default constructor this class).

- Access functions `get` and `set`, for all data member as needed.

- Function `counter()` that returns the number of objects of class `Point`.

- Two distance functions that return the distance between two points. One of the two must be a static function.

`Class Shape:`

This class is the base class or the ancestor of several classes, including class `Rectangle`, `Square`, `Circle`, etc. It should support basic operations and structures that are common among the children of this class. This class is similar to what you did in one of the earlier lab assignments and should have an object of class `Point` called `origin`, and a `char` pointer called `shapeName` that points to a dynamically allocated memory space, allocated by the class constructor. This class should also have several functions and a constructor as follows:

- A constructor that initializes its data members.
- No default constructor
- A destructor that de-allocates the memory space allocated dynamically for `shapeName`
- `getOrigin` – that returns a reference to point origin. The reference should not allow the x and y values of point to be modified through this reference.
- `getName` – that returns the name of the shape.
- `display` – that prints on the screen the shape's name, x and y coordinates of point origin, in the following format:

```
Shape Name:
X-coordinate:
Y-coordinate:
```

- two `distance` functions
```
double distance (Shape& other);
static double distance (Shape& the_shape, Shape& other);
```
- `move`: that changes the position of the shape, the current x and y coordinate, to `x+dx`, and `y+dx`. The function's interface (prototype) should be:
```
void move (double dx, double dy);
```
- More functions, as needed. Don't forget about the law of **Bi**g 3.

Class Square:

This class is supposed to be derived from class `Shape`, and should have one data member, called `side_a`, and in addition to its constructor should have a constructor and several member functions as follows:

- One constructor that initializes its data members with its arguments supplied by the user.
- No default constructor. **Marks will be deducted if a default constructor is defined for this class**
- `area` – that returns the area of a square
- `perimeter`: that returns the perimeter of a square
- `get` and `set` - as needed.
- `display` – that displays the name, x and y coordinates of the origin, in the following format:

```
Square Name:
X-coordinate:
Y-coordinate:
```

- More Functions, if needed.

Class Rectangle:

This class that is derived from class Square needs to have one more side called `side_b`. this class addition to its constructor (no default constructor), it should support the following functions:

- `area` – that calculates and returns the area of a rectangle
- `perimeter` – that calculates and returns the perimeter of a rectangle
- `get` and `set` – that retrieves or changes the values of its private data members.
- `display` – that displays the name, and x and y coordinate origin of the shape, in the following format:

```
Rectangle Name:
X-coordinate:
Y-coordinate:
```

- More Functions, if needed.

`Class GraphicsWorld`:

This class should have only a function called *run*, which declares instances of the above-mentioned classes as its local variable. This function should first display a short message about the author(s) of the program and then start testing all of the functions of the classes in this system. A sample code segment of function *run* is given below (you can add more code to this function if you need to show additional features of your program):

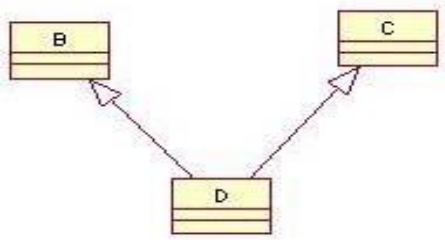
```
void GraphicsWorld::run(){
 // ================================== PART 1 ======================================
          cout << "\nThis program has been written by: student name(s)." ;
          cout << "\nSubmitted at: 11:00 pm, February 3 , 2008\n";
          cout << "\nTesting Functions in class Point:" <<endl;
          Point m (6, 8);
          Point n (6,8);

          n.setx(9);
          m.display();
          n.display();

         cout << "\nThe distance between two points m and n is: " << m.distance(n);
```

```
        // Testing the second version of the function distance.
        // Put the necessary code in this place

         cout << "\nTesting Functions in class Square:" <<endl;
         Square s(5, 7, 12, "SQUARE - S");
         s.display();
        cout << "the area of " << s.getName() <<" is: "<< s.area() << "\n";
       cout << "the perimeter of " << s.getName() <<" is: " << s.perimeter() << "\n";

        cout << "\nTesting Functions in class Rectangle:" <<endl;
        Rectangle a(5, 7, 12, 15, "RECTANGLE A");
        a.display();
        Rectangle b(16 , 7, 8, 9, "RECTANGLE B");
        b.display();
        cout << "the area of " << a.getName() <<" is: "<< a.area() << "\n";
        cout << "the perimeter of " << a.getName() <<" is: "<< a.perimeter() << "\n";
        double d = a.distance(b);
        cout << "\nThe distance between two rectangles is: " <<d;
        cout << "\nTesting copy constructor in class Rectangle:" <<endl;
        Rectangle rec1 = a;
        rec1.display();


       cout << "\nTesting assignment operator in class Rectangle:" <<endl;
      Rectangle rec2 (3, 4, 11, 7, "RECTANGLE rec2");
       rec2 = a;
       rec2.display();
       cout << "\nTesting Functions in class Circle:" <<endl;
       Circle c (3, 5, 9, "CIRCLE C");
       c.display();
       cout << "the area of " << c.getName() <<" is: "<< c.area() << endl;
       cout << "the perimeter of " << c.getName() << " is: "<< c.perimeter() << endl;
       d = a.distance(c);
       cout << "\nThe distance between rectangle a and circle c is: " <<d;
       // YOU MAY ADD ADDITIONAL CODE HERE, IF NEEDED
}
```

Exercise 2 – Part II – Multiple-Inheritance (10 Marks)

This part is the continuation of Part I. You must complete Part I and then start this part.

C++ allows a class to have more than one parent:



This is called multiple-inheritance. For example, if we have two classes called `B` and `C`, and class `D` is derived from both classes (`B` and `C`), we say class `D` has two parents (multiple inheritance).

This is a powerful feature of the C++ language that other languages such as Java do not support. For more details please refer to your class notes.

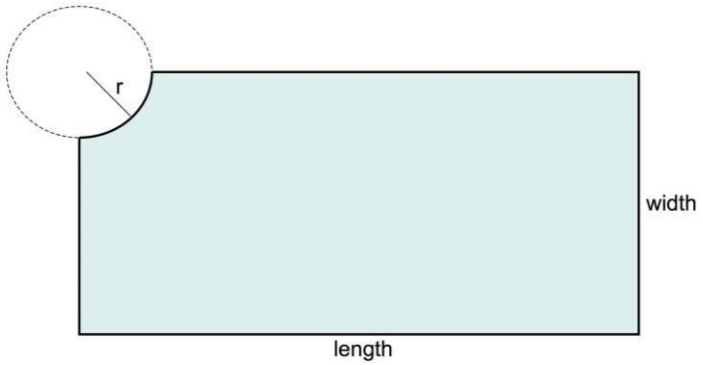
What to Do: In this part of the exercise you should add two new classes to your program:

`Class Circle:`

This class is supposed to be derived from class `Shape`, and should have a data member `radius`. In addition to its constructor it should support similar functions as class `Rectangle`, such as `area, perimeter, get, set`, and `display`.

`Class CornerCut:`

This class represents a partial rectangle with its top left corner cut away (see the following picture).



This class must be derived from class `Rectangle` and class `Circle`. Where the origins (x and y coordinates) of both shapes are the same. This class in addition to a constructor should support the following functions:

- `area` – that calculates the highlighted area of the above figure.
- `perimeter` – that calculates and returns the perimeter of highlighted areas.
- `display` – that displays the name, x, y origin of the shape, in the following format:

```
CornerCut Name:
X-coordinate:
Y-coordinate:
Width:
Length:
Radius of the cut:
```

Note: The radius of the circle must always be less than or equal to the width of the rectangle. Otherwise the program should display an error message and terminate the program.

You should also add some code to function `run` in class `GraphicsWorld` to:

- test the member functions of class `CornerCut`.
- use an array of pointers to `Shape` objects, where each pointer points to a **different object** in the shapes hierarchy. Then test the functions of each class again.

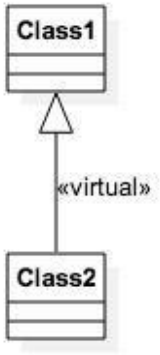
A sample code segment that you can use to test your program is given in the following box (you can add more code to this function if you want to show additional features of your program).

```
void GraphicsWorld::run(){
        // ASSUME CODE SEGMENT FOR PART I IS HERE
        // ================================== PART 2 ==========================================
        CornerCut rc (6, 5, 10, 12, 9, "CornerCut rc");
        rc.display();
        cout << "the area of " << rc.getName() <<" is: "<< rc.area();
        cout << "the perimeter of " << rc.getName() << " is: "<< rc.perimeter();
        d = rc.distance(c);
        cout << "\nThe distance between rc and c is: " <<d;
        // Using array of Shape pointers:
        Shape* sh[4];
        sh[0] = &s;
        sh[1] = &a;
        sh [2] = &c;
        sh [3] = &rc;
        sh [0]->display();
        cout << "\nthe area of "<< sh[0]->getName() << "is: "<< sh[0] ->area();
        cout << "\nthe perimeter of " << sh[0]->getName () << " is: "<< sh[0]->perimeter();
        sh [1]->display();
        cout << "\nthe area of "<< sh[1]->getName() << "is: "<< sh[1] ->area();
        cout << "\nthe perimeter of " << sh[0]->getName () << " is: "<< sh[1]->perimeter();
        sh [2]->display();
        cout << "\nthe area of "<< sh[2]->getName() << "is: "<< sh[2] ->area();
        cout << "\nthe circumference of " << sh[2]->getName ()<< " is: "<< sh[2]->perimeter();
        sh [3]->display();
        cout << "\nthe area of "<< sh[3]->getName() << "is: "<< sh[3] ->area();
        cout << "\nthe perimeter of " << sh[3]->getName () << " is: "<< sh[3]->perimeter();
        cout << "\nTesting copy constructor in class CornerCut:" <<endl;
        CornerCut cc = rc;
        cc.display();
        cout << "\nTesting assignment operator in class CornerCut:" <<endl;
        CornerCut cc2(2, 5, 100,  12, 9,  "CornerCut cc2");
        cc2.display();
        cc2 = cc;
        cc2.display();
        // ADD ADDITIONAL CODE TO TEST MORE FEATRUES, IF NEEDED
}       // END OF FUNCTION run
```

What to Submit for Part I, and Part II:

1. As part of your post-lab report (pdf format) you should submit:
   - A class diagram, that shows the relationships between the classes in this program (you should use UML notations). Make sure your diagram uses appropriate notation for abstract classes. Also for clarification purposes, if there are any virtual derivations among the classes, please mark it with the letter V or the stereotype <<virtual>> on the line between the two classes.

     For example:



   - A copy of your source code (.cpp and .h files)
   - The program output showing your code in part I and II works

2. A zipped file with the source code: all .cpp and .h files

How to submit: Include all your files for the post-lab section in one folder, zip your folder and upload it in D2L before the deadline.