CPSC 2150 – Algorithms and Data Structures II
Lab1: Circles
Total - 40 Marks

Learning Outcomes

- Design and implement classes in C++
- Design and implement the methods of a given class
- Identify and define variables and methods with the least scopes
- Implement information hiding, encapsulation and access control concepts to make a robust object-oriented program

Resources

- Since this lab is a review, the resources on CPSC 1160 can be applied.
- Chapter 1 of the textbook

Background - Writing a C++ Class

.h and .cpp Files

C++ classes are made up of a header file and an implementation file. Both files should have the same name except that the header file has a .h extension while the implementation file has a .cpp extension. The header file contains the class definition, i.e., the class name, the name (and type) of the data members (variables) and the header (prototype) for each of the class methods. The .cpp file consists of the implementation of these class methods.

Public or Private?

Class data members and class methods should be categorised as being either **private** or **public**. Private data members and class methods can only be accessed from within the class, whereas public data members and methods can be accessed from outside the class.

There are a couple of good general design principles to follow when deciding whether to make a data member or method public:

- Only make something public if it needs to be public.
- Designed classes as ADT classes: make all data member private (behind the wall), if necessary write public getter and setter methods for each data members, if appropriate. This allows you to write the setter methods to ensure that any class invariants are maintained (e.g. such as ensuring that a circle's radius is a positive number).

Constructors and Destructors

Every class requires a constructor to create new objects of that class. A class will often have multiple constructors that build new objects in slightly different ways: a default constructor and some parameterised constructors. A constructor is a method that has the same name as the class and has no return type, it is responsible for setting the initial values of the data members of an object.

C++ classes require destructors. A destructor is responsible for de-allocating any dynamic memory that an object uses. If you don't write a destructor for a class a default one is created for you. It is OK to rely on the default destructor if your class does not use any dynamic memory. If your class does, then you must write a destructor.

Syntax

It is easier to show the syntax by presenting a simple example. Here is a class that models a rectangle. It has the following data members and methods:

- height
- width
- Rectangle - constructor to create a new rectangle with the given height and width
- getHeight - returns the height of the rectangle
- getWidth - returns the width of the rectangle
- setHeight - sets the height of the rectangle
- setWidth - sets the width of the rectangle
- computeArea - computes and returns the area of the rectangle
- displayRectangle - prints the height and width of the rectangle.

Here is its header file, C++ keywords are in bold.

```
// Rectangle.h file
// Description: This class models a rectangle ...
// Author:
// Creation date:

class Rectangle {
private: // Everything that follows is private and cannot be "seen"
            //and directly accessed from outside the class
           // To access these private data members, client code needs to use the
           // getter and setter methods
           // Some classes have private methods. This one does not!
  int width;
  int height;

public: //everything that follows here is public
 //Default constructor
  Rectangle();    //the default constructor has no parameters.
  // Constructor to create a new rectangle with the given values
  Rectangle(int w, int h);

  // Getters that return information about the rectangle,
  // note the const at the end of the method, this guarantees that the method
  // cannot alter the member variables
  int getWidth() const;
  int getHeight() const;

  // Setters that change the values of the attributes
  void setWidth(int w);
  void setHeight(int h);

  // Compute and return the area of "this" rectangle
  int computeArea() const;

  // Display method that prints the rectangle's height and width
  void displayRectangle() const;

};  //Rectangle - note the ";" - do not forget it!
   // End of the header file
```

And here is the implementation file:

```
// Rectangle.cpp file
// Description: This class models a rectangle ...
// Author:
// Creation date:

#include <iostream>     // As we need to print data
#include "Rectangle.h" // The header file for the class - we need this!

using namespace std;

// Now follows each of the method implementations.
// The <class>:: that precedes each method indicates that the method belongs to the   // class.
If it is omitted, the compiler will attempt to create a separate function,  // which is not what
we want.

//Default constructor
Rectangle::Rectangle() : width(1), height(1) {}//Rectangle()

// Constructor to create a new rectangle with the given values
Rectangle::Rectangle(int w, int h) {    if
(w > 0)
     width = w; // don't need {}s if there is only one line in the body    else
     width = 1;

  if (h > 0){
     height = h; // but you can use them if you want
  }else{
     height = 1;
  }
}  //Rectangle(int, int)
// Getters that return information about the rectangle int
Rectangle::getWidth() const {
  return width;
}//getWidth

int Rectangle::getHeight() const {
  return height;
}//getHeight

// Setters that change the values of the attributes void
Rectangle::setWidth(int w) {    if (w > 0)       width = w;
else      width = 1;
}//setWidth
```

```
void Rectangle::setHeight(int h){    if
(h > 0)      height = h;    else
height = 1;
}//setHeight

// Compute and return the area of "this" rectangle int
Rectangle::computeArea() const {
  return width * height;
}//computeArea

// Display method that prints the rectangle's height and width void
Rectangle::displayRectangle() const {    cout << endl << "width = " <<
getWidth();    cout << ", height = " << getHeight() << endl;
}//displayRectangle

// End of the implementation file
```

Note There is (a lot) more about C++ classes than what is described above.

Description

A. [25 marks] Write a complete C++ class to represent a circle in two-dimensional space. Your class should consist of a header (C**ircle.h**) and an implementation (**Circle.cpp**) file as described below. The circle class should have the following (private) attributes (or data members) and (public) methods:

Attributes

- x coordinate (an int), can be negative
- y coordinate (an int), can be negative
- radius (a double), must be greater than 0.0

Methods

- A default constructor that creates a circle of radius 10 at position (0,0)
- A constructor with parameters for x, y, and radius
- int getX() - returns the circle's x coordinate
- int getY() - returns the circle's y coordinate
- double getRadius() - returns the circle's radius

- void translateXY(int X, int Y) – Moves the center of circle by X and Y. For example, if circle c1 is centered in (-5,2), calling the translate(4, 3) on c1 will move the center of c1 to (-1, 5), since -5+4=-1 and 2+3=5.
- void setRadius(double r) - changes the circle's radius to r, if r is invalid it throws an exception to the caller function.
- double computeArea() - computes and returns the circle's area.
- void displayCircle() - displays the circle's attributes like this: "[x = 12, y = 17, redius=10.0]"
- **[2 bonus marks]** bool intersect(Circle c2) – takes a circle object, c2, and returns true if c2 intersects with the current circle (calling circle). If two circles touch each other or if they are centered on the same point, the method returns true.

  This is an optional method, so finish everything else first!

B. **[10 marks]** Modify class Circle developed in part A to count all the objects that exist in the program.

Implement body of the following function:

int getNumberOfCurrentCircles(); // returns number of current circles

Call the above function in the test program to display the number of circles created. (**testCircle.cpp**)

C. **[5 marks]** Change your class to count the circles which are in left side of y-axis. Ignore circles that have intersection with y-axis.

Your programming style (http://geosoft.no/development/cppstyle.html) is important.

Submit to D2L

Submit a zip file named **StudentNumber-lab1.zip** including all the files you have made to answer this lab by the end of the lab time. For example, if your student number is 10023449, the submitted file must be named as **10023449-lab1.zip**.