1 Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Lab03, put each programming exercise into its own subdirectory of this directory, zip the entire Lab03 directory up into the file Lab03.zip, and then submit this zip file to Lab #3.

Make sure that you:

- Follow the coding and documentation standards for the course as published in the MyClasses page for the class.
- Check the contents of the zip file before uploading it. Make sure all the files are included.
- Make sure that the file was submitted correctly to MyCLasses.

2 Programming Exercise

The main idea for this assignment is to get some basic familiarity with class structures and object oriented programming in C++. All class structures are to have their own guarded specification file (.h) and implementation file (.cpp) that has the same name as the class. In addition you must create a makefile that compiles and links the project on a Linux computer with a Debian or Debian branch flavor.

For this exercise, create a class structure named `Sphere` which has the following specification.

```
class Sphere {
private:
    double x;
    double y;
    double z;
    double rad;

    double distance(double, double, double);

public:
    Sphere(double ix = 0, double iy = 0, double iz = 0, double ir = 1);

    void setCenter(double ix = 0, double iy = 0, double iz = 0);
    void setRadius(double ir = 1);
    double getRadius();
    double getX();
    double getY();
    double getZ();
    double surfaceArea();
    double volume();
    bool collide(Sphere s);
    void printSphereInfo();
};
```

- The data for the class must be in the private section as well as the function distance that is used internally. All other functions are to be in the public section. The data consists of three doubles to hold the three dimensional position of a sphere in space. The fourth data item is a double that holds the value of the radius of the sphere. Also for the calculations you will need an approximation to π, so create a constant named PI and set its value to 3.141592653589793.
- The constructor will take in 4 parameters, the x, y, and z coordinates of the sphere and the radius of the sphere. As you can see from the specification, there are default values for the constructor and hence a default constructor that creates the unit sphere at the origin. Have the radius error check the parameter value coming in, if the value of ir is negative the radius should be set to 0.
- A destructor is not needed for this class.
- setCenter: Will set the center of the sphere to the parameter values.
- setRadius: Will set the radius of the sphere to the parameter value. Have the radius error check the parameter value coming in, if the value of ir is negative the radius should be set to 0.
- getRadius: Will simply return the radius of the sphere.
- getX: Will simply return the x coordinate of the center of the sphere.
- getY: Will simply return the y coordinate of the center of the sphere.
- getZ: Will simply return the z coordinate of the center of the sphere.
- surfaceArea: Returns the surface area of the sphere. The surface area of a sphere with radius r is $S = 4\pi r^2$
- volume: Returns the volume of the sphere. The volume of a sphere with radius r is $V = \frac{4}{3}\pi r^3$
- collide: This function takes a single parameter, another sphere object, and determines if the calling sphere and the parameter sphere collide. So if S1 and S2 are two spheres then the call `S1.collide(S2)` will return true if the two spheres collide and false if not. You know that two spheres collide if the distance between their centers is less than the sum of their two radius values.
- printSphereInfo: This function will print sphere information in the following style,

  ```
  Center: (-2.5548, 4.47454, -7.68339)    Radius: 0.420059
  ```
- distance: Returns the distance between the center of the sphere and the parameter double values that represent an (x, y, z) coordinate. Recall that the distance between two points in space, (x_1, y_1, z_1) and (x_2, y_2, z_2), is calculated as

  $$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

  This function will make implementing the collide function much easier.

With the above class the following main will produce the output below.

```
#include <iostream>

#include "Sphere.h"

using namespace std;

int main() {
    Sphere s1(2, 3, 4, 1);
    Sphere s2(2.1, 3.5, 3.9, 3);
    Sphere s3(-1, -2, -3, 2);

    s1.printSphereInfo();
    s2.printSphereInfo();
    s3.printSphereInfo();

    cout << s1.getX() << " " << s1.getY() << " " << s1.getZ() << " --- ";
    cout << s1.getRadius() << endl;

    s1.setCenter(2, 5, 9);
    s1.setRadius(3.2);

    cout << s1.getX() << " " << s1.getY() << " " << s1.getZ() << " --- ";
    cout << s1.getRadius() << endl;

    cout << "Sphere 1 Volume: " << s1.volume() << endl;
    cout << "Sphere 1 Surface Area: " << s1.surfaceArea() << endl;

    cout << s1.collide(s2) << endl;
    cout << s2.collide(s3) << endl;
    cout << s1.collide(s3) << endl;

    return 0;
}
```

Output:

```
Center: (2, 3, 4)    Radius: 1
Center: (2.1, 3.5, 3.9)    Radius: 3
Center: (-1, -2, -3)    Radius: 2
2 3 4 --- 1
2 5 9 --- 3.2
Sphere 1 Volume: 137.258
Sphere 1 Surface Area: 128.68
1
0
0
```