**C++ Part II (INFO1-CE9265) Spring 2015 – Midterm**

Clement Chan

**Question 1:**

Shape.h

#ifndef SHAPE\_H

#define SHAPE\_H

#include <iostream>

#include <string>

#include <stdio.h>

#include <cmath>

class Shape{

protected:

int X;

int Y;

int color;

enum color{Red, Orange, Yellow, Green, Blue, Indigo, Violet};

float Area;

std::string shapetype;

public:

Shape(); //Default Constructor

~Shape(); // Default Destructor

static int shapecount; //Shape counters

//Setters, Getters and

void setX(int);

void setY(int);

int getX() const;

int getY() const;

void displaytotalshapecount();

void setshapetype(std::string shape);

void setColor(int);

//Virtual Functions

virtual float CalcArea() = 0;

virtual void DrawObject();

//Other Functions for Color Settings etc

void ResetLocation(int, int);

void ShowColor() const;

float getArea() const;

//Overloading operator to calculate distance

float get\_distance(const Shape \*rhs, const Shape \*lhs);

};

#endif

Shape.cpp

#include "Shape.h"

Shape :: Shape(){

std::cout << "This is Shape Default Constructor ... " << std::endl;

X = 0;

Y = 0;

Area = 0;

shapetype = "Shape" ;

}

Shape :: ~Shape(){

std::cout << "This is Shape Default Destructor ... " << std::endl;

}

void Shape :: setX(int deltaX){

X = deltaX;

}

void Shape :: setY(int deltaY){

Y = deltaY;

}

void Shape :: setshapetype(std::string shape){

shapetype = shape;

}

int Shape :: getX() const{

return X;

}

int Shape :: getY() const{

return Y;

}

//Shape and reset location

void Shape :: ResetLocation(int deltaX, int deltaY){

X = deltaX;

Y = deltaY;

}

float Shape :: getArea()const {

return Area;

}

void Shape :: DrawObject(){

std::cout << "Drawing : " << shapetype << std::endl;

}

//Setting the static shape count to 0;

int Shape :: shapecount = 0;

void Shape :: displaytotalshapecount(){

std::cout <<"The number of shapes created is: " << shapecount << std::endl;

std::cout << " " << std::endl;

}

//Setting color code

void Shape :: setColor (int i){

color = i;

}

void Shape :: ShowColor()const{

if(color == 0){

std::cout << "The Color of Shape is Red." << std::endl;

}

else if(color == 1){

std::cout << "The Color of Shape is Orange." << std::endl;

}

else if(color == 2){

std::cout << "The Color of Shape is Yellow." << std::endl;

}

else if(color == 3){

std::cout << "The Color of Shape is Green." << std::endl;

}

else if(color == 4){

std::cout << "The Color of Shape is Blue." << std::endl;

}

else if(color == 5){

std::cout << "The Color of Shape is Indigo." << std::endl;

}

else if(color == 6){

std::cout << "The Color of Shape is Violet." << std::endl;

}

else{

std::cout << "Color Not Set." << std::endl;

}

}

float Shape :: get\_distance(const Shape \*S1, const Shape \*S2){

return sqrt((S1->getX() - S2->getX())\*(S1->getX() - S2->getX()) - (S1->getY() - S2->getY()));

}

Circle.h

#ifndef Circle\_H

#define Circle\_H

#include "Shape.h"

#include <iostream>

#include <string.h>

#include <stdio.h>

class Circle : public Shape{

protected:

double radius; //Only Circle has Radius

public:

Circle(); //Default Constructors

Circle(double, int, int); //Parameterized Constructor that sets radius without color

Circle(double, int, int, int); //Parameterized Constructor that sets radius with color

~Circle(); //Destructor

Circle(const Circle &rhs);

Circle &operator = (const Circle &rhs);

//Setters and Getters

float CalcArea();

void DrawObject();

double getRadius() const;

void setradius(double);

//Operator Overloading with '\*' to Scale the size

Circle &operator \*(double scalar);

friend const Circle operator \*(const Circle &circ, const double scalar);

friend const Circle operator \*(const double scalar,const Circle &circ);

};

#endif

Circle.cpp

#include "Circle.h"

Circle :: Circle(){

std::cout << "This is Circle Constructor ... " << std::endl;

std::cout << " " << std::endl;

radius = 0;

setX(0);

setY(0);

setshapetype("Circle");

shapecount += 1;

}

Circle :: Circle(double r, int x, int y){

std::cout << "This is Circle Constructor ... " << std::endl;

std::cout << " " << std::endl;

radius = r;

setX(x);

setY(y);

setshapetype("Circle");

shapecount += 1;

}

Circle :: Circle(double r, int x, int y, int color\_code){

std::cout << "This is Circle Constructor ... " << std::endl;

std::cout << " " << std::endl;

radius = r;

setColor(color\_code);

setX(x);

setY(y);

setshapetype("Circle");

shapecount += 1;

}

//Copy Constructor

Circle :: Circle(const Circle &rhs){

radius = rhs.getRadius();

Area = rhs.getArea();

X = rhs.getX();

Y = rhs.getY();

}

//Assignment Operator

Circle &Circle :: operator = (const Circle &rhs){

radius = rhs.getRadius();

Area = rhs.getArea();

X = rhs.getX();

Y = rhs.getY();

return \*this;

}

//Destructor

Circle :: ~Circle(){

std::cout << "This is Circle Destructor ... " << std::endl;

}

//Setting Method for virtual class

double Circle :: getRadius() const{

return radius;

}

float Circle :: CalcArea(){

float pi = 3.14159265359;

Area = pi \* radius \* radius;

return pi \* radius \* radius;

}

void Circle :: DrawObject(){

std::cout << "Drawing : " << shapetype << std::endl;

}

void Circle :: setradius(double rad){

radius = rad;

}

//Scaled Shape Operator

Circle &Circle :: operator \*(double scalar){

float pi = 3.14159265359;

radius \*= scalar;

Area = pi \* radius \* radius; //The Scalar only changes radius. When multiplied, the area should show changes

return \*this;

}

const Circle operator \*(const Circle &circ, const double scalar){

Circle circ1 = circ;

float pi = 3.14159265359;

circ1.radius = scalar \* circ.radius;

circ1.Area = pi \* circ1.radius \* circ1.radius; //The Scalar only changes radius. When multiplied, the area should show changes

return circ1;

}

const Circle operator \*(const double scalar, const Circle &circ){

Circle circ1 = circ;

float pi = 3.14159265359;

circ1.radius = scalar \* circ.radius;

circ1.Area = pi \* circ1.radius \* circ1.radius; //The Scalar only changes radius. When multiplied, the area should show changes

return circ1;

}

Doughnut.h

#ifndef Doughnut\_H

#define Doughnut\_H

#include "Circle.h"

#include <iostream>

#include <string.h>

#include <stdio.h>

class Doughnut : public Circle{

protected:

double radius1; //Entering the Larger Radius that covers the smaller radius

int color\_2;

public:

Doughnut(double the\_radius, double the\_radius\_1, int the\_x, int the\_y); //Default Constructors

Doughnut(double the\_radius, double the\_radius\_1, int the\_x, int the\_y, int the\_color\_code, int the\_color\_code\_2); //Default Constructors

~Doughnut(); //Destructor

Doughnut(const Doughnut &rhs);

Doughnut &operator = (const Doughnut &rhs);

//Setters and Getters

float CalcArea();

void DrawObject();

double getRadius1() const;

double getRadius2() const;

void set\_radius(double, double);

Doughnut &operator \*(double scalar);

friend const Doughnut operator \*(const Doughnut &circ, const double scalar);

friend const Doughnut operator \*(const double scalar,const Doughnut &circ);

};

#endif

Doughnut.cpp

#include "Doughnut.h"

Doughnut :: Doughnut(double the\_radius, double the\_radius\_1, int the\_x, int the\_y) : Circle(the\_radius, the\_x , the\_y), radius1(the\_radius\_1){

std::cout << "This is Doughnut Constructor ... " << std::endl;

std::cout << " " << std::endl;

setshapetype("Doughnut");

//Left out static shape counter because circle already initialized the shape

}

Doughnut :: Doughnut(double the\_radius, double the\_radius\_1, int the\_x, int the\_y, int the\_color\_code, int the\_color\_code\_2) : Circle(the\_radius, the\_x , the\_y, the\_color\_code), radius1(the\_radius\_1), color\_2(the\_color\_code\_2){

std::cout << "This is Doughnut Constructor ... " << std::endl;

std::cout << " " << std::endl;

setshapetype("Doughnut");

//Left out static shape counter because circle already initialized the shape

}

//Copy Constructor

Doughnut :: Doughnut(const Doughnut &rhs){

radius = rhs.getRadius1();

radius1 = rhs.getRadius2();

Area = rhs.getArea();

X = rhs.getX();

Y = rhs.getY();

}

//Assignment Operator

Doughnut &Doughnut :: operator = (const Doughnut &rhs){

radius = rhs.getRadius1();

radius1 = rhs.getRadius2();

Area = rhs.getArea();

X = rhs.getX();

Y = rhs.getY();

return \*this;

}

//Destructor

Doughnut :: ~Doughnut(){

std::cout << "This is Doughnut Destructor ... " << std::endl;

}

void Doughnut :: set\_radius(double rad, double rad1){

radius = rad;

radius1 = rad1;

}

//Setting Method for virtual class

double Doughnut :: getRadius1() const{

return radius;

}

double Doughnut :: getRadius2() const{

return radius1;

}

float Doughnut :: CalcArea(){

float pi = 3.14159265359;

Area = (pi \* radius1 \* radius1) - (pi \* radius \* radius);

return Area;

}

void Doughnut :: DrawObject(){

std::cout << "Drawing : " << shapetype << std::endl;

}

//Scaled Shape Operator

Doughnut &Doughnut :: operator \*(double scalar){

float pi = 3.14159265359;

radius \*= scalar;

radius1 \*= scalar;

Area = (pi \* radius1 \* radius1) - (pi \* radius \* radius); //The Scalar only changes radius. When multiplied, the area should show changes

return \*this;

}

const Doughnut operator \*(const Doughnut &DN, const double scalar){

Doughnut DN1 = DN;

float pi = 3.14159265359;

DN1.radius = scalar \* DN.radius;

DN1.radius1 = scalar \* DN.radius1;

DN1.Area = (pi \* DN1.radius1 \* DN1.radius1) - (pi \* DN1.radius \* DN1.radius); //The Scalar only changes radius. When multiplied, the area should show changes

return DN1;

}

const Doughnut operator \*(const double scalar, const Doughnut &DN){

Doughnut DN1 = DN;

float pi = 3.14159265359;

DN1.radius = scalar \* DN.radius;

DN1.radius1 = scalar \* DN.radius1;

DN1.Area = (pi \* DN1.radius1 \* DN1.radius1) - (pi \* DN1.radius \* DN1.radius); //The Scalar only changes radius. When multiplied, the area should show changes

return DN1;

}

Main.cpp

#include "Shape.h"

#include "Circle.h"

#include "Right\_Triangle.h"

#include "Doughnut.h"

void SortShape(Shape \*arr[], int NumShapes){

std::cout<<"Sorting Objects ... "<< std::endl;

std::cout<<" "<< std::endl;

//Recalculate the area after the settings

for(int i = 0; i < NumShapes; i++){

arr[i] -> CalcArea();

}

for(int i = 1; i < NumShapes; i++){

Shape \*temp = arr[i];

int j = i;

while(j > 0 && arr[j-1]->getArea() > temp -> getArea()){

arr[j] = arr[j-1];

j--;

arr[j] = temp;

}

}

}

void printsortshape(Shape \*arr[], int NumShapes){

std::cout << "Sorted Shape order is: " << std::endl;

for(int i=0; i<NumShapes; i++){

arr[i] -> DrawObject();

arr[i] -> ShowColor();

std::cout << "The X location is: " << arr[i] -> getX() << std::endl;

std::cout << "The Y location is: " << arr[i] -> getY() << std::endl;

std::cout << "The Sorted Object's Area is: " << arr[i] -> getArea() << std::endl;

std::cout << " " << std::endl;

}

std::cout << " " << std::endl;

};

//Global Function for GetDistance

float getDistance(Shape \*S1, Shape \*S2){

float dist;

dist = sqrt((S1->getX() - S2->getX())\*(S1->getX() - S2->getX()) - (S1->getY() - S2->getY()));

return dist;

}

int main(){

//Creating 6 derived classes

int N = 6;

Circle circ(5,30,30,3), circ1(10,50,50,1);

RTriangle RTri(5,10, 150, 150,5), RTri1(10,20, 100, 100,1), RTri2(10, 5, 250, 250,6);

Doughnut DN(5,10,40,70,3,1);

Shape \*shapeptr[N];

shapeptr[0] = &circ;

shapeptr[1] = &circ1;

shapeptr[2] = &RTri;

shapeptr[3] = &RTri1;

shapeptr[4] = &RTri2;

shapeptr[5] = &DN;

//Show the total shapes being developed using static variable

shapeptr[1]-> displaytotalshapecount();

//Display all the shapes' information

for (int i=0; i < 6; i++){

if (i < 2 || i > 4){

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*This is Shape Number: " << i + 1 << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

Shape \*temp = shapeptr[i];

temp -> CalcArea();

temp -> DrawObject();

std::cout << "The X location is: " << temp -> getX() << std::endl;

std::cout << "The Y location is: " << temp -> getY() << std::endl;

//Use Dynamic Casting to Find Radius, which is information in the derived class

if(i == 0){

Shape \*temp1;

temp1 = new Circle(5,30,30,3);

Circle \*circptr = dynamic\_cast<Circle\*>(temp1);

std::cout << "The Radius of the Circle is: " << circptr -> getRadius() << std::endl;

}

else if(i == 1){

Shape \*temp1;

temp1 = new Circle(10,50,50,1);

Circle \*circptr = dynamic\_cast<Circle\*>(temp1);

std::cout << "The Radius of the Circle is: " << circptr -> getRadius() << std::endl;

}

else if(i == 5){

Shape \*temp1;

temp1 = new Doughnut(5,10,40,70);

Doughnut \*DNptr = dynamic\_cast<Doughnut\*>(temp1);

std::cout << "The Radius of the smaller circle in Doughnut is: " << DNptr -> getRadius1() << std::endl;

std::cout << "The Radius of the larger circle in Doughnut is: " << DNptr -> getRadius2() << std::endl;

}

temp -> ShowColor();

std::cout << "The Area of the Pointed Shape is: " << temp ->getArea()<<std::endl;

std::cout << " " << std::endl;;

}

else{

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*This is Shape Number: " << i + 1 << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

Shape \*temp = shapeptr[i];

temp -> CalcArea();

temp ->DrawObject();

std::cout << "The X location is: " << temp -> getX() << std::endl;

std::cout << "The Y location is: " << temp -> getY() << std::endl;

if(i == 2){

Shape \*temp1;

temp1 = new RTriangle(5,10, 150, 150);

RTriangle \*rtriptr = dynamic\_cast<RTriangle\*>(temp1);

std::cout << "The Base of the Right Triangle is: " << rtriptr -> getBase() << std::endl;

std::cout << "The Height of the Right Triangle is: " << rtriptr ->getHeight() << std::endl;

}

else if(i == 3){

Shape \*temp1;

temp1 = new RTriangle(10,20, 100, 100);

RTriangle \*rtriptr = dynamic\_cast<RTriangle\*>(temp1);

std::cout << "The Base of the Right Triangle is: " << rtriptr -> getBase() << std::endl;

std::cout << "The Height of the Right Triangle is: " << rtriptr -> getHeight() << std::endl;

}

else if(i == 4){

Shape \*temp1;

temp1 = new RTriangle(10, 5, 250, 250);

RTriangle \*rtriptr = dynamic\_cast<RTriangle\*>(temp1);

std::cout << "The Base of the Right Triangle is: " << rtriptr -> getBase() << std::endl;

std::cout << "The Height of the Right Triangle is: " << rtriptr -> getHeight() << std::endl;

}

temp -> ShowColor();

std::cout << "The Area of the Pointed Shape is: " << temp ->getArea()<<std::endl;

std::cout << " " << std::endl;

}

}

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

//Testing the multiplication functions

Circle circ3 = circ \* 2;

Shape \*circtemp = &circ3;

std::cout << "The scaled circle is: " << circtemp -> getArea() << std::endl;

std::cout << " " << std::endl;

circ.setradius(5);

Circle circ4 = 2 \* circ;

circtemp = &circ4;

std::cout << "The scaled circle is: " << circtemp -> getArea() << std::endl;

std::cout << " " << std::endl;

circ.setradius(5);

RTriangle RTri3 = RTri \* 2;

Shape \*tritemp = &RTri3;

std::cout << "The scaled Triangle is: " << tritemp-> getArea() << std::endl;

std::cout << " " << std::endl;

RTri.setbase\_height(5,10);

RTriangle RTri4 = 2 \* RTri;

tritemp = &RTri4;

std::cout << "The scaled Triangle is: " << tritemp-> getArea() << std::endl;

std::cout << " " << std::endl;

RTri.setbase\_height(5,10);

Doughnut DN2 = DN \* 2;

Shape \*DNtemp = &DN2;

std::cout << "The scaled Dougnut is: " << DNtemp-> getArea() << std::endl;

std::cout << " " << std::endl;

DN.set\_radius(5,10);

Doughnut DN3 = 2 \* DN;

DNtemp = &DN3;

std::cout << "The scaled Doughnut is: " << DNtemp-> getArea() << std::endl;

std::cout << " " << std::endl;

DN.set\_radius(5,10);

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Question 17-18 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Question 17-18

//Sorting the last base on area;

SortShape(shapeptr, N);

printsortshape(shapeptr, N);

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Question 19-20 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

std::cout << "The distance between Circle 1 and Circle 2 is: " << getDistance(shapeptr[2], shapeptr[5]) << std::endl;

std::cout << " " << std::endl;

std::cout << "The distance between Circle 1 and Right Triangle is: " << getDistance(shapeptr[2], shapeptr[0]) << std::endl;

std::cout << " " << std::endl;

float dist = shapeptr[0]->get\_distance(shapeptr[2], shapeptr[5]);

std::cout << "The distance between Circle 1 and Circle 2 using operator function is: " << dist << std::endl;

std::cout << " " << std::endl;

dist = shapeptr[0]->get\_distance(shapeptr[2], shapeptr[0]);

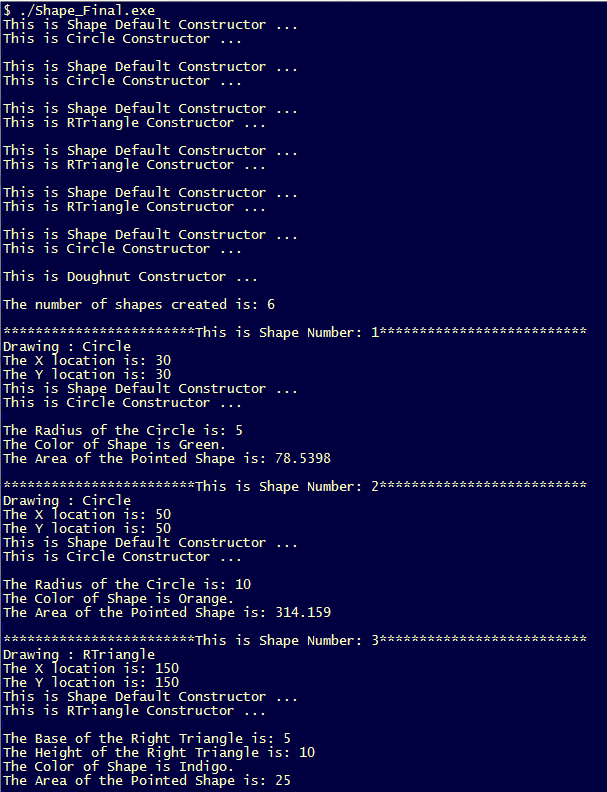
std::cout << "The distance between Circle 1 and Right Triangle using operator function is: " << dist << std::endl;

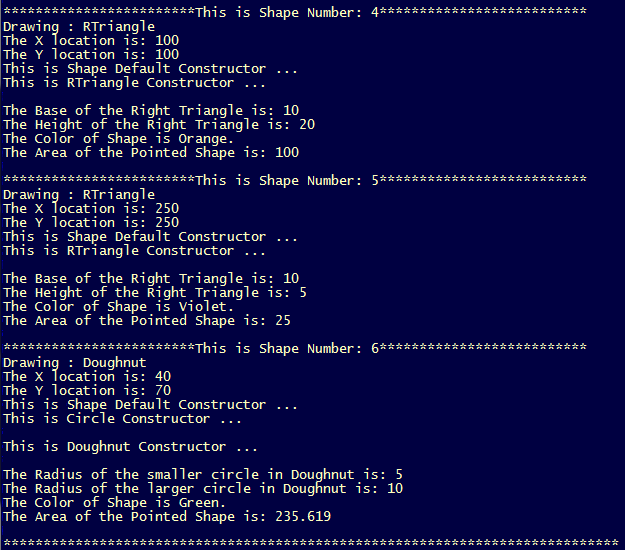
std::cout << " " << std::endl;

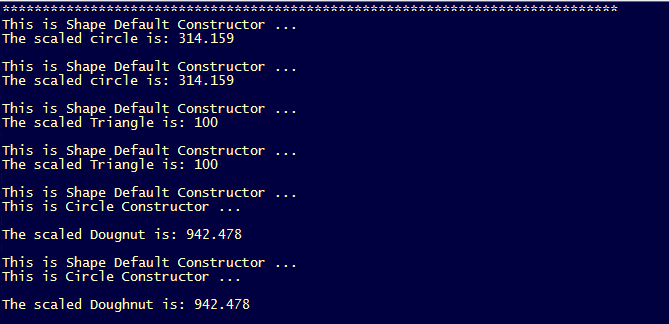
return 0;

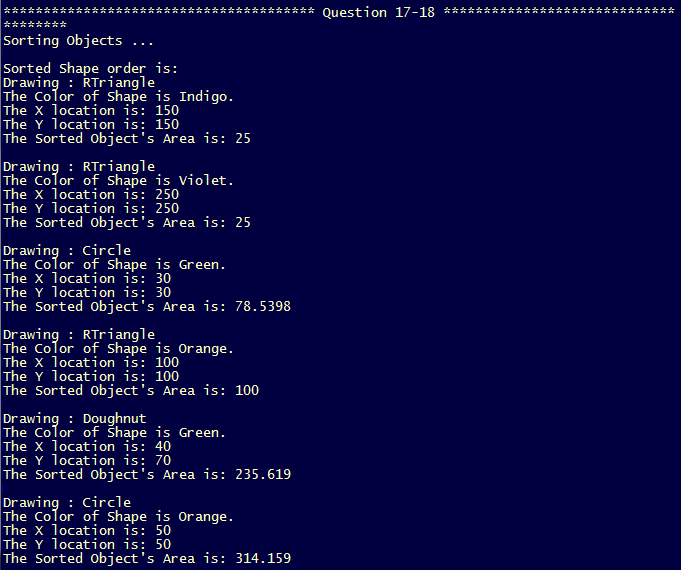
}

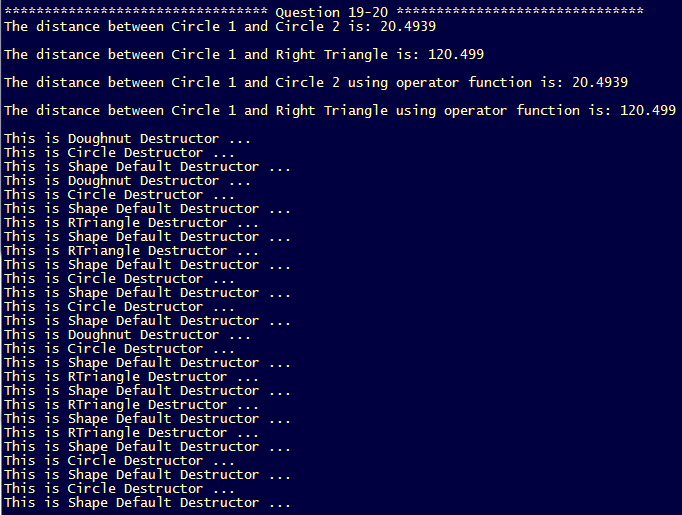
Output











**Question 2:**

Point.h

#ifndef Point\_H

#define Point\_H

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

class Point{

protected:

int x;

int y;

double distance;

public:

Point(); //Default Constructor

Point(int, int); //Parameterized Constructor

Point(const Point &rhs); //Copy Constructor

~Point(); //Destructor

Point &operator =(const Point &rhs); //Assignment Operator

Point &operator == (const Point &rhs);

//Setters and Getters

void reset\_location(int,int);

int getX() const;

int getY() const;

double getdistance() const;

Point &operator\*(int scalar); //To Perform Point \* scalar

friend const Point operator \*(const int scalar, const Point &P);

//To Perform Scalar \* Point

friend double operator-(const Point &rhs, const Point &lhs);

//Distance between Points

friend ostream &operator << (ostream &Oustream, const Point &P);

};

#endif

Point.cpp

//Method Class for Point class

#include "Point.h"

Point :: Point() : x(0), y(0){

std::cout << "This is Default Constructor ... " <<std::endl;

}

Point :: Point(int X, int Y) : x(X), y(Y){

std::cout << "This is Parameterized Constructor ... " <<std::endl;

}

Point :: Point(const Point &rhs){

x = rhs.getX();

y = rhs.getY();

}

Point &Point::operator = (const Point &rhs){

x = rhs.getX();

y = rhs.getY();

return \*this;

}

Point :: ~Point(){

std::cout << "This is Destructor..." << std::endl;

}

//Setters and Getters

void Point :: reset\_location(int X, int Y){

x = X;

y = Y;

}

int Point :: getX() const{

return x;

}

int Point :: getY() const{

return y;

}

double Point :: getdistance() const{

return distance;

}

Point &Point :: operator \* (int scalar){

x \*= scalar;

y \*= scalar;

return \*this;

}

const Point operator \* (const int scalar, const Point &P){

Point P1 = P;

P1.x \*= scalar;

P1.y \*= scalar;

return P1;

}

double operator-(const Point &rhs, const Point &lhs){

double dist;

dist = sqrt((rhs.getX() - lhs.getX())\*(rhs.getX() - lhs.getX()) + (rhs.getY() - lhs.getY())\*(rhs.getY() - lhs.getY()));

return dist;

}

ostream& operator << (ostream &Outstream, const Point &P){

Outstream << "The X co-ordinate is: " << P.getX() <<endl;

Outstream << "The Y co-ordinate is: " << P.getY() <<endl;

return Outstream;

}

Person.h

#ifndef Person\_H

#define Person\_H

#include "Point.h"

class Person : public Point{

protected:

char\* name;

int age;

Point gps\_loc;

public:

Person(); //Default Constructor

Person(char\*, int);

Person(char\*, int, int, int); //Parameterized Constructor

~Person(); //Destructor

//Setters and Getters

char\* getname() const;

int getage() const;

void gps\_setloc(int, int);

int get\_gps\_X() const;

int get\_gps\_Y() const;

double get\_distance(const Person &rhs, const Person &lhs);

};

#endif

Person.cpp

//Method Class for Point class

#include "Person.h"

Person :: Person() : Point(), name("No Name Yet"), age(0){

std::cout << "This is Default Constructor ... " <<std::endl;

}

//Parameterized Constructor with just name and age

Person:: Person(char\* the\_name, int the\_age) : name(the\_name), age(the\_age){

std::cout << "This is Parameterized Constructor (name and age only) ... " <<std::endl;

}

//Setting information using inheritance

Person:: Person(char\* the\_name, int the\_age, int the\_X, int the\_Y) : Point(the\_X, the\_Y), name(the\_name), age(the\_age){

std::cout << "This is Parameterized Constructor (all) ... " <<std::endl;

}

Person:: ~Person(){

std::cout << "This is Destructor..." << std::endl;

}

//Setters and Getters Method

char\* Person::getname() const{

return name;

}

int Person::getage() const{

return age;

}

void Person :: gps\_setloc(int X, int Y){

gps\_loc.reset\_location(X,Y);

}

int Person :: get\_gps\_X()const{

return gps\_loc.getX();

}

int Person :: get\_gps\_Y()const{

return gps\_loc.getY();

}

double Person :: get\_distance(const Person &rhs, const Person &lhs){

double dist;

dist = sqrt((rhs.get\_gps\_X() - lhs.get\_gps\_X())\*(rhs.get\_gps\_X() - lhs.get\_gps\_X()) + (rhs.get\_gps\_Y() - lhs.get\_gps\_Y())\*(rhs.get\_gps\_Y() - lhs.get\_gps\_Y()));

return dist;

}

Main.cpp

#include "Point.h"

#include "Person.h"

void foo(const Point &X, const Point Y){

cout << "X coordinate in Point X is: " << X.getX() << endl;

cout << "Y coordinate in Point X is: " << X.getY() << endl;

cout << "X coordinate in Point Y is: " << Y.getX() << endl;

cout << "Y coordinate in Point Y is: " << Y.getY() << endl;

}

int main(){

Point X(3,4), Y(10,40), Z;

double dist = Y - X;

cout<<"The distance between X and Y is: " << dist << endl;

//scalar multiplier test

Z = X \* 10;

cout<<Z;

X.reset\_location(3,4);

Z = 10 \* X;

cout<<Z;

X.reset\_location(3,4);

//Perform get functions

int x = X.getX();

int y = X.getY();

cout<<"The X coordinate in X is: " << x << endl;

cout<<"The Y coordinate in X is: " << y << endl;

cout<< X << endl;

//Passing function foo

foo(X,Y);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Method Through Composition \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Person X1("Joe Doe", 33);

Person Y1("Mary Joe", 22);

X1.gps\_setloc(10,15);

Y1.gps\_setloc(20,30);

dist = X1.get\_distance(X1,Y1);

cout<< " " <<endl;

cout<<"The name of the first person is : " << X1.getname() << " and the age is: " << X1.getage()<<endl;

cout<<"The name of the second person is : " << Y1.getname() << " and the age is: " << Y1.getage()<<endl;

cout<<"The distance between X1 and Y1 (Composition) is: " << dist << endl;

cout<< " " <<endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Method through Inheritance \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Initializing Person

Person X\_1("Joe Doe", 33, 10,15);

Person Y\_1("Mary Joe", 22, 20, 30);

//Finding the distance between Joe Doe and Mary Joe

dist = Y\_1 - X\_1;

cout<< " " <<endl;

cout<<"The name of the first person is : " << X\_1.getname() << " and the age is: " << X\_1.getage()<<endl;

cout<<"The name of the second person is : " << Y\_1.getname() << " and the age is: " << Y\_1.getage()<<endl;

cout<<"The distance between X\_1 and Y\_1 (Inheritance) is: " << dist << endl;

cout<< " " <<endl;

return 0;

}

Output

