**Object-Oriented Programming Language**

**06/01/2023**

**Due:06/05/2023 11:59p.m.**

**Lab Assignment 13**

1. Create a class Vehicle with a function start() and create 3 classes derived from the base class Vehicle called Car, Bicycle and Motorcycle . This main code you should not change.

int main() {

Vehicle\* vehicle1 = new Car();

Vehicle\* vehicle2 = new Motorcycle();

Vehicle\* vehicle3 = new Bicycle();

vehicle1->start();

vehicle2->start();

vehicle3->start();

delete vehicle1;

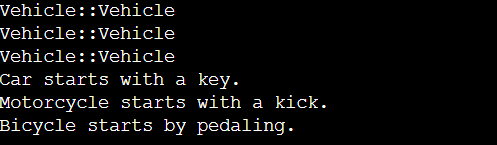
delete vehicle2;

delete vehicle3;

return 0;

}

Output should look like this



1. Following previous problem, set start() to pure virtual function. This main code you should not change.

int main() {

Vehicle\* vehicle1 = new Car();

Vehicle\* vehicle2 = new Motorcycle();

Vehicle\* vehicle3 = new Bicycle();

vehicle1->start();

vehicle2->start();

vehicle3->start();

delete vehicle1;

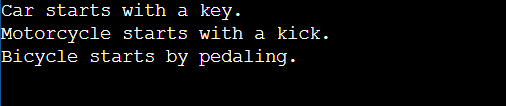
delete vehicle2;

delete vehicle3;

return 0;

}

Output should look like this



1. Define a base class called Shape has two pure virtual function double getArea() and string getClassName(). Implement three derived classes: Rectangle, Circle, and Triangle, which inherit from the Shape base class.

Rectangle class should have (len) A double representing the length of the rectangle, and (wid) A double representing the width of the rectangle. Circle class should have (rad) A double representing the radius of the circle. Triangle class have(a,b,c) Doubles representing the lengths of the three sides of the triangle.All of them override getArea() and getClassName().This main code you should not change.

int main() {

vector<Shape\*> vs;

Rectangle r(10,20);

Circle c(10);

Triangle t(18,30,24);

vs.push\_back(&r);

vs.push\_back(&c);

vs.push\_back(&t);

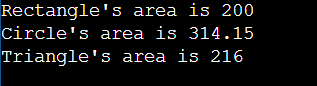
for(auto s : vs)

cout << s;

return 0;

}

Output should look like this



1. Write a C++ Program to demonstrate Run time polymorphism. In place of static binding, one would like a binding method that is capable of determining which function should be invoked at run-time, on the basis of object type making call. In place of static binding, one would like a binding method that is capable of determining which function should be invoked at run-time, on the basis of object type making call. In place of static binding, one would like a binding method that is capable of determining which function should be invoked at run-time, on the basis of object type making call. This main code you should not change.

int main()

{

B\* base\_ptr ;

D1 der1\_obj ;

base\_ptr = &der1\_obj ;

base\_ptr->Display( );

D2 der2\_obj ;

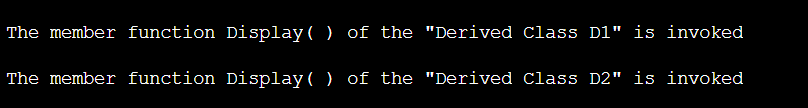
base\_ptr = &der2\_obj ;

base\_ptr->Display( );

return 0;

}

Output should look like this



1. Follow problem 3, class Shape is also an ABC, but change getArea() into void function print the area directly. Now design a new class ShapeGroupID inherits from class Shape representing a group of shapes. It should have a vector to store shape objects and another vector to store corresponding IDs. Implement the necessary functions to insert shapes into the group and retrieve their areas and class names. The structure of ShapeGroupID is listed below. This main code you should not change.

class ShapeGroupID : public Shape {

protected:

vector<Shape\*> elems;

vector<int> IDs;

public:

ShapeGroupID() = default;

void getArea() {

// input your code here

}

string getClassName() {

// input your code here

}

void insert(Shape\* s) {

elems.push\_back(s);

IDs.push\_back(rand() % 100);

}

void insert(ShapeGroupID& sgid) {

// input your code here

}

// copy constructor

ShapeGroupID(const ShapeGroupID& rhs) {

// input your code here

}

// destructor

~ShapeGroupID() {

// input your code here

}

};

int main() {

srand(420);

ShapeGroupID sgID;

Rectangle\* r1 = new Rectangle(10, 20);

Circle\* c1 = new Circle(10);

Square\* s1 = new Square(10);

sgID.insert(r1);

sgID.insert(c1);

sgID.insert(s1);

ShapeGroupID sgID2;

Circle\* c2 = new Circle(5);

Square\* s2 = new Square(5);

sgID2.insert(c2);

sgID2.insert(s2);

ShapeGroupID sgID3(sgID);

Circle\* c3 = new Circle(20);

Square\* s3 = new Square(20);

sgID3.insert(c3);

sgID3.insert(s3);

sgID3.insert(sgID2);

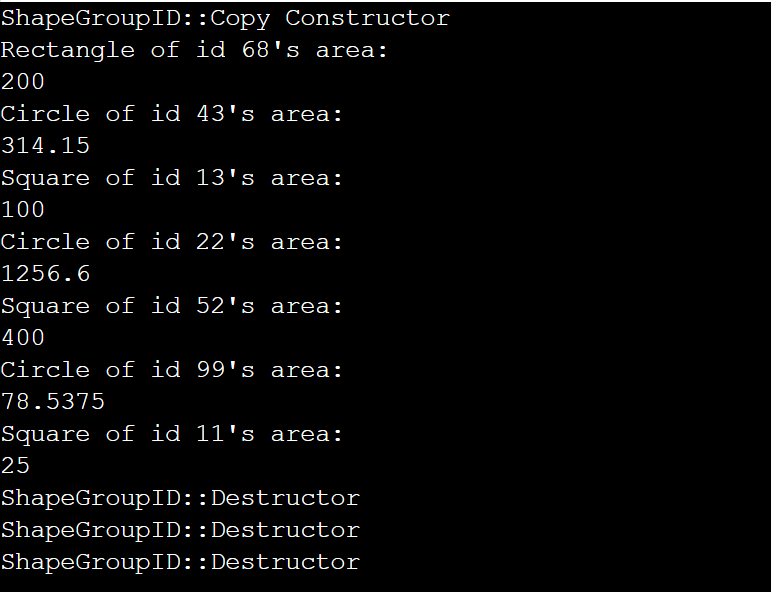
sgID3.getArea();

delete r1, c1, s1, c2, s2, c3, s3;

return 0;

}

Output should look like this



1. design a program that defines several classes, namely first\_class, second\_class, third\_class, four\_class, and five\_class. These classes inherit from each other in a hierarchical structure. first\_class has a virtual void printit(), other class might have it or not. Following is the code you shouldn’t change.

int main() {

first\_class one;

one.printit();

second\_class second;

second.printit();

third\_class third;

third.printit();

four\_class four;

four.printit();

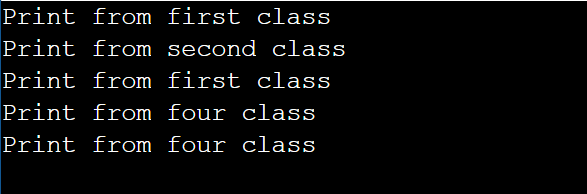
five\_class five;

five.printit();

return 0;

}

Output should look like this



1. You have a base class Shape and need to implement two derived classes Circle and Square. However, the implementation of the Draw function is not trivial. Luckily someone provides you a useful class Widget which has the functions drawCircle and drawSquare you need. Please design two classes Circle and Square inheritance from Shape and Widget.

Following is the code you shouldn’t change.

class Shape{

protected:

int \_x;

int \_y;

int \_width;

int \_height;

public:

Shape(int x=0, int y=0, int width=0, int height=0):

\_x(x), \_y(y), \_width(width), \_height(height){}

virtual void Draw () = 0 ;

} ;

class Widget {

private:

string name;

public:

Widget() : name("Widget") {}

void drawCircle(int x, int y, int r) {

cout << name << " is drawing a Circle whose radius is " << r << " at (" << x << ", " << y << ")." << endl;

}

void drawSquare(int x, int y, int l) {

cout << name << " is drawing a Square whose length is " << l << " at (" << x << ", " << y << ")." << endl;

}

};

int main() {

Circle A(25,0,0);

A.Draw();

Square B(50,100,100);

B.Draw();

return 0;

}

Output should look like this

