Classes:

Shape: Base class representing a generic shape.

Rectangle: Derived class representing a rectangle with length and width.

Circle: Derived class representing a circle with radius.

Concepts:

Constructors and Destructors:

Define a default constructor for Shape to initialize common properties.

Overload constructors for Rectangle and Circle to take specific dimensions as input during object creation.

Implement destructors for all classes to handle memory cleanup (if applicable).

Overriding:

Override the area() function in Rectangle and Circle to calculate their respective areas using appropriate formulas. The base class Shape can have a pure virtual area() function to enforce implementation in derived classes.

Operator Overloading:

Overload the == operator for Shape to compare shapes based on a chosen criterion (e.g., area for simplicity).

Consider overloading other operators (like +) for specific shapes if applicable (e.g., combining rectangles).

Friend Function:

Define a friend function totalArea outside the class hierarchy that takes an array of Shape pointers and calculates the total area of all shapes. This function needs access to private member variables of Shape and its derived classes.

Template (Optional):

(Optional) Create a template class Point to represent a point in 2D space with x and y coordinates. Use this template class within the Shape hierarchy if needed.

Implementation:

Design the Shape class with appropriate member variables and functions, including a pure virtual area() function.

Implement derived classes Rectangle and Circle with constructors, destructors, overridden area() functions, and potentially overloaded operators.

Define a friend function totalArea that takes an array of Shape pointers and calculates the total area.

(Optional) Implement a template class Point for representing points.

Testing:

Create objects of different shapes (rectangle, circle) and test their constructors, destructors, and overridden area() functions.

Use the overloaded == operator to compare shapes.

Call the totalArea friend function to calculate the total area of an array of shapes.

(Optional) Test the functionality of the Point template class (if implemented).

**Explanation** :

In this program i have one base class name shape and two derived class rectangle and circle. In shape class we have one virtual function name area (virtual keyword use because this function can override in both derived class ) and also have the friend function to calculate the area of all shapes. In first derived class name rectangle in this we have two private variable (length and breadth) and one constructor to get the value in variable and override function name area for calculate the area of rectangle. In second derived class name circle in this we have one private variable ( radius) and one constructor to get the value in variable and override function name area for calculate the area of rectangle. Now in main function we pass the value and calculate the area of both derived class and after we get area we call friend function with all shapes as argument and calculate the area of shapes.

**Design** :

Base class : Shape

Derived class : Circle

Derived class : Rectangle

**Program**

// Shape class - base class

#include<iostream>

#include <cmath>

using namespace std;

class Shape {

public:

Shape() {}

virtual double area() const = 0;

friend double totalArea(const Shape\* shapes[], int count);

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

// Constructors

Rectangle() : length(0), width(0) {}

Rectangle(double l, double w) : length(l), width(w) {}

// Override area calculation

double area() const override {

return length \* width;

}

friend bool operator==(const Rectangle& r1, const Rectangle& r2) {

return r1.area() == r2.area();

}

};

class Circle : public Shape {

private:

double radius;

public:

Circle() : radius(0) {}

Circle(double r) : radius(r) {}

double area() const override {

return 3.14 \* radius \* radius;

}

friend bool operator==(const Circle& c1, const Circle& c2) {

return c1.area() == c2.area();

}

};

// Friend function area of shapes

double totalArea(const Shape\* shapes[], int count) {

double total = 0.0;

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

total += shapes[i]->area();

}

return total;

}

int main() {

Rectangle rect(3.0, 4.0);

Circle circle(5.0);

std::cout<< "Area of Rectangle : " <<rect.area() << std::endl;

std::cout<< "Area of Circle : " <<circle.area() << std::endl;

Rectangle rect2(4.0, 3.0);

Circle circle2(5.0);

std::cout<< "Area of Rectangle2 : " << rect2.area() << std::endl;

std::cout<< "Area of Circle2 : " << circle2.area() << std::endl;

if (rect == rect2) {

std::cout<< "Rectangles are equal" << std::endl;

} else {

std::cout<< "Rectangles are not equal" << std::endl;

}

if (circle == circle2) {

std::cout<< "Circles are equal" << std::endl;

} else {

std::cout<< "Circles are not equal" << std::endl;

}

const Shape\* shapeArr[] = { &rect, &circle, &rect2, &circle2 };

int numShapes = sizeof(shapeArr) / sizeof(shapeArr[0]);

std::cout<< "Total area of shapes : " <<totalArea(shapeArr, numShapes) << std::endl;

return 0;

}