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).

Solution:

#include <iostream>

#include <cmath>

using namespace std;

class Shape{ // Base class

public:

virtual ~Shape() {} // Virtual destructor

virtual double area() const=0; // Pure virtual function for area

bool operator==(const Shape& other) const // Overload the == operator to compare shapes based on their area

{

return this->area() == other.area();

}

friend double totalarea(Shape\* shapes[], int count); // Friend function declaration

};

class Rectangle:public Shape { // Derived class

private:

double length;

double width;

public:

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

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

~Rectangle(){}

double area() const override{

return length\*width;

}

};

class Circle:public Shape{

private:

double radius;

public:

Circle():radius(0){}

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

~Circle() {}

double area() const override{

return M\_PI\*radius\*radius;

}

};

double totalarea(Shape\* shapes[], int count)

{

double total=0;

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

{

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

}

return total;

}

int main()

{

Rectangle r1(7,4); // Create objects of different shapes

Rectangle r2(9,5);

Circle c1(6);

Circle c2(8);

Shape\* shapes[]={ &r1, &r2, &c1, &c2 }; // Create an array of Shape pointers

cout<<"Area of Rectangle 1: "<<r1.area()<<endl;

cout<<"Area of Rectangle 2: "<<r2.area()<<endl;

cout<<"Area of Circle 1: "<<c1.area()<<endl;

cout<<"Area of Circle 2: "<<c2.area()<<endl;

cout<<"Rectangle 1 equal to Rectangle 2 is "<<(r1==r2?"True":"False")<<endl;

cout<<"Circle 1 equal to Circle 2 is "<<(c1==c2?"True":"False")<<endl;

double total=totalarea(shapes,4);

cout<<"Total area of shapes is "<<total;

return 0;

}

OUTPUT:

