DAY – 13 Test

// Quest : Problem Statements - **Test (Time = 2:00 – 3:00 p.m.)**

/* 1. 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:

2. 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.

3. 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). */

// Ans : of Problem Statement –

/* Steps required for implementing the above problem statement are -

Step 1: Define the Base Class ‘ Shape ‘

Step 2: Define the Derived Class ‘ Rectangle ‘

Step 3: Define the Derived Class ‘ Circle ‘

Step 4: Implement the Friend Function ‘ totalArea ‘.

Step 5: Implement the Optional Template Class ‘ Point ‘

Step 6: Implement the main Function . */

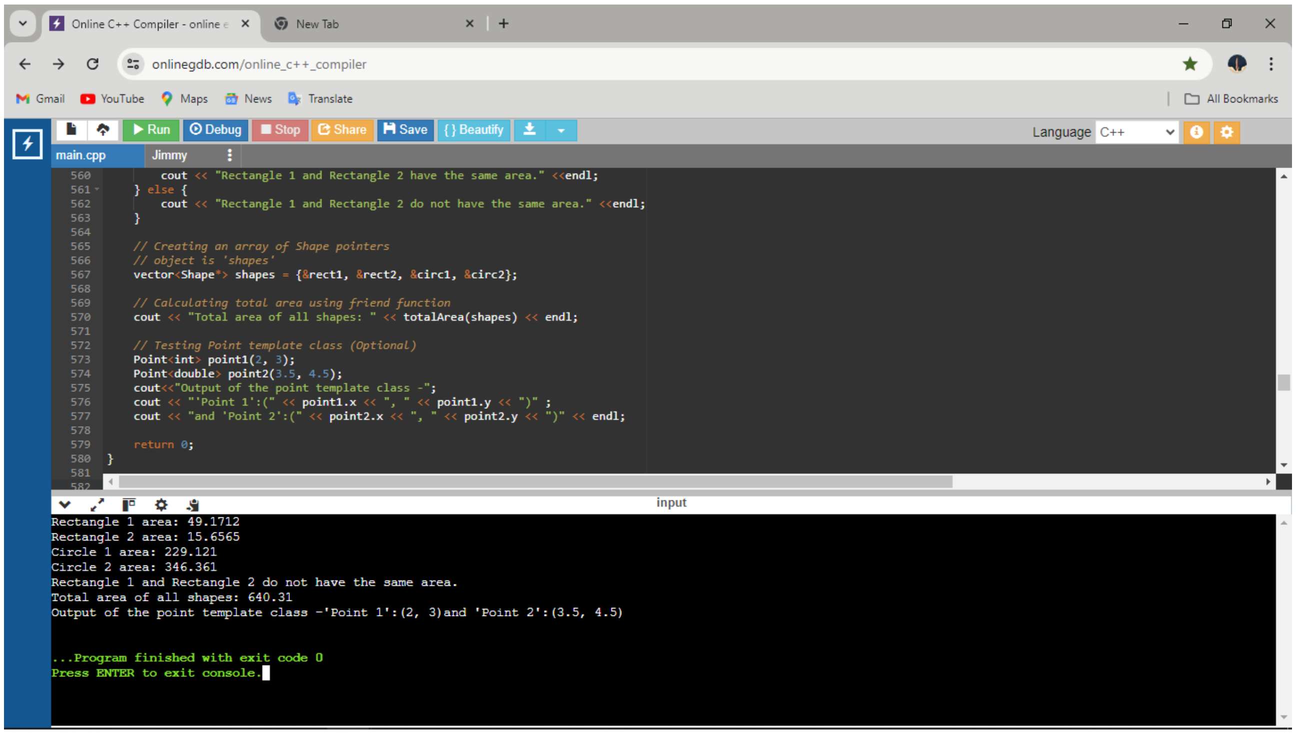
// **Code Execution :**

```
#include <iostream>

#include <cmath> // for 'M_PI' or we could ignore it and use ‘ 3.14 ‘ is place of ‘M_PI’ in below.

#include <vector>

using namespace std;


// Base class Shape becomes implementation

class Shape {

public:

    virtual ~Shape() {}

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


    // Overloading the == operator for comparing areas of shapes

    bool operator==(const Shape& other) const {

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

    }

    // Friend function to calculate total area of shapes

    friend double totalArea(const vector<Shape*>& shapes);

};


// Derived class Rectangle

class Rectangle : public Shape {
```

```
private:
    double length;
    double width;
public:
    // Constructor
    Rectangle(double l, double w) : length(l), width(w) {}

    // Destructor
    ~Rectangle() {}

    // Override area function
    double area() const override {
        return length * width;
    }
    // Overload '+' operator to combine areas of rectangles
    Rectangle operator+(const Rectangle& other) const {
        return Rectangle(length + other.length, width + other.width);
    }
};
// Derived class Circle
class Circle : public Shape {
private:
    double radius;
public:
    // Constructor
    Circle(double r) : radius(r) {}


    // Destructor
    ~Circle() {}


    // Overridden area function
    double area() const override {
        return M_PI * radius * radius;
    }
};
// Friend function to calculate total area of shapes
double totalArea(const vector<Shape*>& shapes) {
    double total = 0;
    for (const auto& shape : shapes) {
        total += shape->area();
    }
    return total;
}
// Template class Point (Optional)
template <typename T>
class Point {
public:
    T x, y;
    Point(T x = 0, T y = 0) : x(x), y(y) {}
};
```

```
int main() {
    // Creating objects of Rectangle and Circle
    Rectangle rect1(6.24, 7.88);
    Rectangle rect2(1.73, 9.05);
    Circle circ1(8.54);
    Circle circ2(10.5);

    // Testing area functions
    cout << "Rectangle 1 area: " << rect1.area() <<endl;
    cout << "Rectangle 2 area: " << rect2.area() <<endl;
    cout << "Circle 1 area: " << circ1.area() << endl;
    cout << "Circle 2 area: " << circ2.area() << endl;

    // Overloaded == operator
    if (rect1 == rect2) {
        cout << "Rectangle 1 and Rectangle 2 have the same area." <<endl;
    } else {
        cout << "Rectangle 1 and Rectangle 2 do not have the same area." <<endl;
    }

    // Creating an array of Shape pointers
    // object is 'shapes'
    vector<Shape*> shapes = {&rect1, &rect2, &circ1, &circ2};

    // Calculating total area using friend function
    cout << "Total area of all shapes: " << totalArea(shapes) << endl;

    // Testing Point template class (Optional)

    Point<int> point1(2, 3);

    Point<double> point2(3.5, 4.5);

    cout<<"Output of the point template class -";

    cout << "'Point 1':(" << point1.x << ", " << point1.y << ")" ;

    cout << "and 'Point 2':(" << point2.x << ", " << point2.y << ")" << endl;


    return 0;

}
```

// **Output of Above Code :**



```
            cout << "Rectangle 1 and Rectangle 2 have the same area." <<endl;
        } else {
            cout << "Rectangle 1 and Rectangle 2 do not have the same area." <<endl;
        }

        // Creating an array of Shape pointers
        // object is 'shapes'
        vector<Shape*> shapes = {&rect1, &rect2, &circ1, &circ2};

        // Calculating total area using friend function
        cout << "Total area of all shapes: " << totalArea(shapes) << endl;

        // Testing Point template class (Optional)
        Point<int> point1(2, 3);
        Point<double> point2(3.5, 4.5);
        cout<<"Output of the point template class -";
        cout << "'Point 1':(" << point1.x << ", " << point1.y << ")" ;
        cout << "and 'Point 2':(" << point2.x << ", " << point2.y << ")" << endl;

        return 0;
}
```

```
Rectangle 1 area: 49.1712
Rectangle 2 area: 15.6565
Circle 1 area: 229.121
Circle 2 area: 346.361
Rectangle 1 and Rectangle 2 do not have the same area.
Total area of all shapes: 640.31
Output of the point template class -'Point 1':(2, 3)and 'Point 2':(3.5, 4.5)


...Program finished with exit code 0
Press ENTER to exit console.
```