**Date:11/07/2024**

**Day\_14\_Test**

Q.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:

**Inheritance**: Derived classes (Rectangle and Circle) inherit from a base class (Shape).

**Polymorphism**: Using base class pointers to call derived class methods.

**Virtual Functions**: Methods in the base class that can be overridden in derived classes to achieve polymorphism.

1. **Shape Class**:
   * Shape is a base class with two pure virtual functions: area() and display(), making it an abstract class.
   * A virtual destructor ensures proper cleanup of derived class objects when deleted via a base class pointer.
2. **Rectangle Class**:
   * Rectangle inherits from Shape and overrides the area() and display() functions.
   * The constructor initializes the length and width of the rectangle.
   * area() computes the area of the rectangle.
   * display() prints the rectangle's dimensions and area.
3. **Circle Class**:
   * Circle also inherits from Shape and overrides the area() and display() functions.
   * The constructor initializes the radius of the circle.
   * area() computes the area of the circle using the formula πr².
   * display() prints the circle's radius and area.
4. **Main Function**:
   * Two pointers to Shape are created, each pointing to a new Rectangle and Circle object.
   * A loop iterates through the shapes, calling their display() methods to print their details and areas.
   * The allocated memory for the shapes is freed using delete to prevent memory leaks.

#include <iostream>

#include <cmath>

class Shape {

public:

virtual double area() const = 0;

// Virtual function to display the shape details

virtual void display() const = 0;

// Virtual destructor

virtual ~Shape() {}

};

// Derived class representing a rectangle

class Rectangle : public Shape {

private:

double length;

double width;

public:

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

// Override the area function

double area() const override {

return length \* width;

}

// Override the display function

void display() const override {

std::cout << "Rectangle: length = " << length << ", width = " << width << ", area = " << area() << std::endl;

}

};

// Derived class representing a circle

class Circle : public Shape {

private:

double radius;

public:

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

// Override the area function

double area() const override {

return M\_PI \* radius \* radius;

}

// Override the display function

void display() const override {

std::cout << "Circle: radius = " << radius << ", area = " << area() << std::endl;

}

};

int main() {

// Create instances of Rectangle and Circle

Shape\* shapes[2];

shapes[0] = new Rectangle(10.0, 5.0);

shapes[1] = new Circle(7.0);

for (int i = 0; i < 2; ++i) {

shapes[i]->display();

}

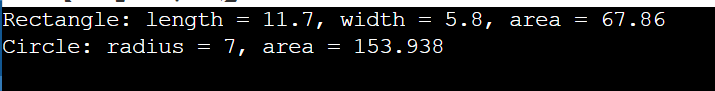
for (int i = 0; i < 2; ++i) {

delete shapes[i];

}

return 0;

}



Q.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).

The Shape base class now has a default constructor to initialize common properties, and the Rectangle and Circle classes have overloaded constructors to take specific dimensions as input during object creation.

**Shape Class**:

* The default constructor Shape() prints a message indicating that a Shape object has been created.
* The virtual destructor ~Shape() prints a message indicating that a Shape object has been destroyed.

**Rectangle Class**:

* The overloaded constructor Rectangle(double l, double w) initializes the length and width of the rectangle and prints a message indicating that a Rectangle object has been created.
* The destructor ~Rectangle() prints a message indicating that a Rectangle object has been destroyed.

**Circle Class**:

* The overloaded constructor Circle(double r) initializes the radius of the circle and prints a message indicating that a Circle object has been created.
* The destructor ~Circle() prints a message indicating that a Circle object has been destroyed.

**Main Function**:

* Two pointers to Shape are created, each pointing to a new Rectangle and Circle object.
* A loop iterates through the shapes, calling their display() methods to print their details and areas.
* The allocated memory for the shapes is freed using delete to invoke the destructors and prevent memory leaks.

Code:

#include <iostream>

#include <cmath>

// Base class representing a generic shape

class Shape {

public:

// Default constructor

Shape() {

std::cout << "Shape created." << std::endl;

}

// Virtual destructor

virtual ~Shape() {

std::cout << "Shape destroyed." << std::endl;

}

// Pure virtual function to calculate area

virtual double area() const = 0;

// Pure virtual function to display the shape details

virtual void display() const = 0;

};

// Derived class representing a rectangle

class Rectangle : public Shape {

private:

double length;

double width;

public:

// Overloaded constructor

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

std::cout << "Rectangle created." << std::endl;

}

// Destructor

~Rectangle() {

std::cout << "Rectangle destroyed." << std::endl;

}

// Override the area function

double area() const override {

return length \* width;

}

// Override the display function

void display() const override {

std::cout << "Rectangle: length = " << length << ", width = " << width << ", area = " << area() << std::endl;

}

};

// Derived class representing a circle

class Circle : public Shape {

private:

double radius;

public:

// Overloaded constructor

Circle(double r) : radius(r) {

std::cout << "Circle created." << std::endl;

}

// Destructor

~Circle() {

std::cout << "Circle destroyed." << std::endl;

}

// Override the area function

double area() const override {

return M\_PI \* radius \* radius;

}

// Override the display function

void display() const override {

std::cout << "Circle: radius = " << radius << ", area = " << area() << std::endl;

}

};

int main() {

// Create instances of Rectangle and Circle

Shape\* shapes[2];

shapes[0] = new Rectangle(10.0, 5.0);

shapes[1] = new Circle(7.0);

// Display the shapes and their areas

for (int i = 0; i < 2; ++i) {

shapes[i]->display();

}

// Clean up

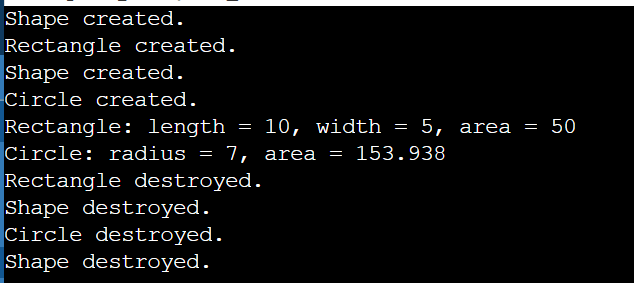
for (int i = 0; i < 2; ++i) {

delete shapes[i];

}

return 0;

}



Q.3 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.

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

 **Shape Class**:

* The Shape class has a default constructor that prints a message when a Shape object is created.
* The virtual destructor ~Shape() prints a message when a Shape object is destroyed.
* The pure virtual function area() enforces that derived classes must provide their own implementation of the area() function.
* The pure virtual function display() enforces that derived classes must provide their own implementation of the display() function.

 **Rectangle Class**:

* The Rectangle class inherits from Shape.
* The overloaded constructor Rectangle(double l, double w) initializes the length and width of the rectangle and prints a message when a Rectangle object is created.
* The destructor ~Rectangle() prints a message when a Rectangle object is destroyed.
* The overridden area() function calculates the area of the rectangle using the formula length \* width.
* The overridden display() function prints the dimensions and area of the rectangle.

 **Circle Class**:

* The Circle class inherits from Shape.
* The overloaded constructor Circle(double r) initializes the radius of the circle and prints a message when a Circle object is created.
* The destructor ~Circle() prints a message when a Circle object is destroyed.
* The overridden area() function calculates the area of the circle using the formula π \* radius^2.
* The overridden display() function prints the radius and area of the circle.

 **Main Function**:

* Two pointers to Shape are created, each pointing to a new Rectangle and Circle object.
* A loop iterates through the shapes, calling their display() methods to print their details and areas.
* The allocated memory for the shapes is freed using delete to invoke the destructors and prevent memory leaks.

#include <iostream>

#include <cmath>

class Shape {

public:

// Default constructor

Shape() {

std::cout << "Shape created." << std::endl;

}

// Virtual destructor

virtual ~Shape() {

std::cout << "Shape destroyed." << std::endl;

}

// Pure virtual function to calculate area

virtual double area() const = 0;

// Pure virtual function to display the shape details

virtual void display() const = 0;

};

// Derived class representing a rectangle

class Rectangle : public Shape {

private:

double length;

double width;

public:

// Overloaded constructor

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

std::cout << "Rectangle created." << std::endl;

}

// Destructor

~Rectangle() {

std::cout << "Rectangle destroyed." << std::endl;

}

// Override the area function to calculate the area of a rectangle

double area() const override {

return length \* width;

}

// Override the display function

void display() const override {

std::cout << "Rectangle: length = " << length << ", width = " << width << ", area = " << area() << std::endl;

}

};

// Derived class representing a circle

class Circle : public Shape {

private:

double radius;

public:

// Overloaded constructor

Circle(double r) : radius(r) {

std::cout << "Circle created." << std::endl;

}

// Destructor

~Circle() {

std::cout << "Circle destroyed." << std::endl;

}

double area() const override {

return M\_PI \* radius \* radius;

}

void display() const override {

std::cout << "Circle: radius = " << radius << ", area = " << area() << std::endl;

}

};

int main() {

// Create instances of Rectangle and Circle

Shape\* shapes[2];

shapes[0] = new Rectangle(10.0, 5.0);

shapes[1] = new Circle(7.0);

// Display the shapes and their areas

for (int i = 0; i < 2; ++i) {

shapes[i]->display();

}

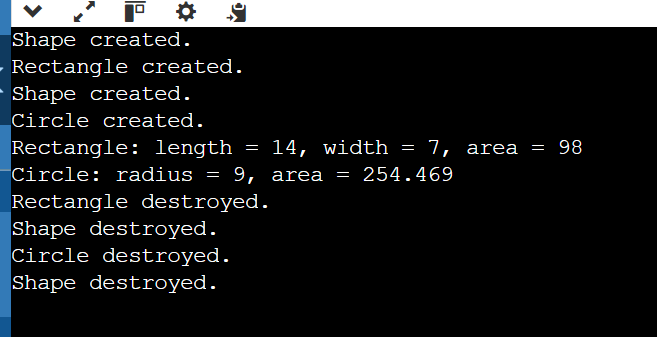
for (int i = 0; i < 2; ++i) {

delete shapes[i];

}

return 0;

}



Q.4 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).

 **Shape Class**:

* The Shape class has a default constructor and a virtual destructor.
* The pure virtual function area() enforces that derived classes must provide their own implementation of the area() function.
* The pure virtual function display() enforces that derived classes must provide their own implementation of the display() function.
* The == operator is overloaded to compare shapes based on their areas.

**Rectangle Class**:

* The Rectangle class inherits from Shape.
* The overloaded constructor initializes the length and width of the rectangle.
* The destructor prints a message when a Rectangle object is destroyed.
* The overridden area() function calculates the area of the rectangle using the formula length \* width.
* The overridden display() function prints the dimensions and area of the rectangle.
* The + operator is overloaded to combine two rectangles by summing their areas. The new rectangle is assumed to be a square with the combined area for simplicity.

**Circle Class**:

* The Circle class inherits from Shape.
* The overloaded constructor initializes the radius of the circle.
* The destructor prints a message when a Circle object is destroyed.
* The overridden area() function calculates the area of the circle using the formula π \* radius^2.
* The overridden display() function prints the radius and area of the circle.

**Main Function**:

* Two pointers to Shape are created, each pointing to a new Rectangle and Circle object.
* A loop iterates through the shapes, calling their display() methods to print their details and areas.
* The shapes are compared based on their areas using the overloaded == operator.
* Two rectangles are created and combined using the overloaded + operator, and the combined rectangle's details are displayed.
* The allocated memory for the shapes is freed using delete to invoke the destructors and prevent memory leaks.

#include <iostream>

#include <cmath>

// Base class representing a generic shape

class Shape {

public:

// Default constructor

Shape() {

std::cout << "Shape created." << std::endl;

}

// Virtual destructor

virtual ~Shape() {

std::cout << "Shape destroyed." << std::endl;

}

// Pure virtual function to calculate area

virtual double area() const = 0;

// Pure virtual function to display the shape details

virtual void display() const = 0;

// Overload the == operator to compare shapes based on area

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

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

}

};

// Derived class representing a rectangle

class Rectangle : public Shape {

private:

double length;

double width;

public:

// Overloaded constructor

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

std::cout << "Rectangle created." << std::endl;

}

// Destructor

~Rectangle() {

std::cout << "Rectangle destroyed." << std::endl;

}

// Override the area function to calculate the area of a rectangle

double area() const override {

return length \* width;

}

// Override the display function

void display() const override {

std::cout << "Rectangle: length = " << length << ", width = " << width << ", area = " << area() << std::endl;

}

// Overload the + operator to combine rectangles by summing their areas

Rectangle operator+(const Rectangle& other) const {

double newArea = this->area() + other.area();

// Assume the new rectangle is a square with the combined area for simplicity

double newLength = std::sqrt(newArea);

return Rectangle(newLength, newLength);

}

};

// Derived class representing a circle

class Circle : public Shape {

private:

double radius;

public:

// Overloaded constructor

Circle(double r) : radius(r) {

std::cout << "Circle created." << std::endl;

}

// Destructor

~Circle() {

std::cout << "Circle destroyed." << std::endl;

}

// Override the area function to calculate the area of a circle

double area() const override {

return M\_PI \* radius \* radius;

}

// Override the display function

void display() const override {

std::cout << "Circle: radius = " << radius << ", area = " << area() << std::endl;

}

};

int main() {

// Create instances of Rectangle and Circle

Shape\* shapes[2];

shapes[0] = new Rectangle(10.0, 5.0);

shapes[1] = new Circle(7.0);

// Display the shapes and their areas

for (int i = 0; i < 2; ++i) {

shapes[i]->display();

}

// Compare shapes based on area

if (\*shapes[0] == \*shapes[1]) {

std::cout << "The shapes have the same area." << std::endl;

} else {

std::cout << "The shapes have different areas." << std::endl;

}

// Create two rectangles and combine them using the + operator

Rectangle rect1(4.0, 5.0);

Rectangle rect2(6.0, 3.0);

Rectangle combinedRect = rect1 + rect2;

std::cout << "Combined rectangle: ";

combinedRect.display();

// Clean up

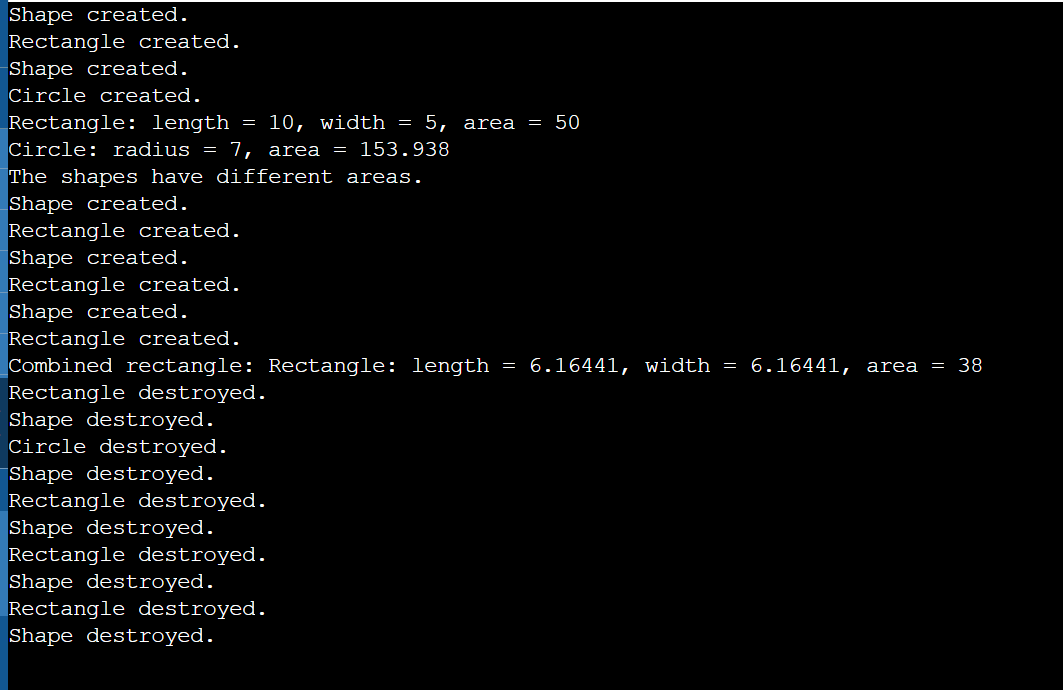
for (int i = 0; i < 2; ++i) {

delete shapes[i];

}

return 0;

}



Q.5 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.

To define a friend function totalArea that calculates the total area of all shapes, we need to give this function access to the private member variables of Shape and its derived classes. Here’s how to achieve this in C++:

1. Declare totalArea as a friend function in the Shape class.
2. Define the totalArea function outside the class hierarchy.

#include <iostream>

#include <cmath>

class Shape;

double totalArea(Shape\* shapes[], int size);

// Base class representing a generic shape

class Shape {

public:

// Default constructor

Shape() {

std::cout << "Shape created." << std::endl;

}

// Virtual destructor

virtual ~Shape() {

std::cout << "Shape destroyed." << std::endl;

}

// Pure virtual function to calculate area

virtual double area() const = 0;

// Pure virtual function to display the shape details

virtual void display() const = 0;

// Friend function declaration

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

};

// Derived class representing a rectangle

class Rectangle : public Shape {

private:

double length;

double width;

public:

// Overloaded constructor

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

std::cout << "Rectangle created." << std::endl;

}

// Destructor

~Rectangle() {

std::cout << "Rectangle destroyed." << std::endl;

}

// Override the area function to calculate the area of a rectangle

double area() const override {

return length \* width;

}

// Override the display function

void display() const override {

std::cout << "Rectangle: length = " << length << ", width = " << width << ", area = " << area() << std::endl;

}

// Friend function declaration

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

};

// Derived class representing a circle

class Circle : public Shape {

private:

double radius;

public:

// Overloaded constructor

Circle(double r) : radius(r) {

std::cout << "Circle created." << std::endl;

}

// Destructor

~Circle() {

std::cout << "Circle destroyed." << std::endl;

}

// Override the area function to calculate the area of a circle

double area() const override {

return M\_PI \* radius \* radius;

}

// Override the display function

void display() const override {

std::cout << "Circle: radius = " << radius << ", area = " << area() << std::endl;

}

// Friend function declaration

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

};

// Friend function to calculate the total area of all shapes

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

double total = 0.0;

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

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

}

return total;

}

int main() {

// Create instances of Rectangle and Circle

Shape\* shapes[3];

shapes[0] = new Rectangle(10.0, 5.0);

shapes[1] = new Circle(7.0);

shapes[2] = new Rectangle(4.0, 3.0);

// Display the shapes and their areas

for (int i = 0; i < 3; ++i) {

shapes[i]->display();

}

// Calculate the total area of all shapes

double total = totalArea(shapes, 3);

std::cout << "Total area of all shapes: " << total << std::endl;

// Clean up

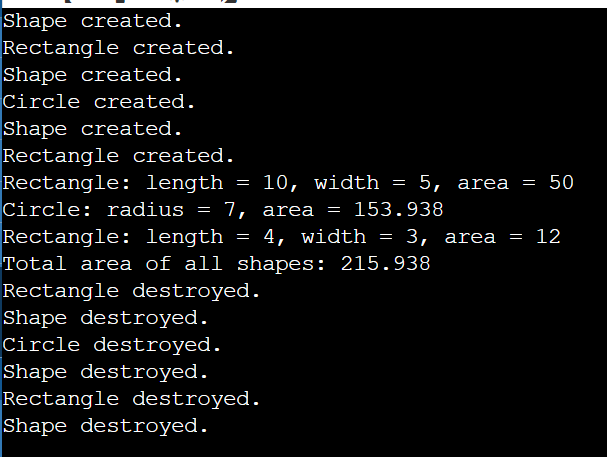
for (int i = 0; i < 3; ++i) {

delete shapes[i];

}

return 0;

}



Q.6 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:

 **Point Template Class**:

* The Point class template is defined with a type parameter T.
* It has two member variables, x and y, to represent the coordinates.
* A default constructor initializes x and y to default values of T.
* A parameterized constructor initializes x and y with given values.
* A display function prints the coordinates of the point.

 **Shape Class**:

* The Shape class has a default constructor and a virtual destructor.
* The pure virtual function area() enforces that derived classes must provide their own implementation of the area() function.
* The pure virtual function display() enforces that derived classes must provide their own implementation of the display() function.
* The totalArea function is declared as a friend function to allow access to private members.

 **Rectangle Class**:

* The Rectangle class inherits from Shape.
* It uses the Point class template to represent the top-left corner of the rectangle.
* The overloaded constructor initializes the top-left corner, length, and width.
* The destructor prints a message when a Rectangle object is destroyed.
* The overridden area() function calculates the area of the rectangle using the formula length \* width.
* The overridden display() function prints the top-left corner coordinates, dimensions, and area of the rectangle.
* The + operator is overloaded to combine two rectangles by summing their areas. The new rectangle is assumed to be a square with the combined area for simplicity.
* The totalArea function is declared as a friend function to allow access to private members.

 **Circle Class**:

* The Circle class inherits from Shape.
* It uses the Point class template to represent the center of the circle.
* The overloaded constructor initializes the center and radius.
* The destructor prints a message when a Circle object is destroyed.
* The overridden area() function calculates the area of the circle using the formula π \* radius^2.
* The overridden display() function prints the center coordinates, radius, and area of the circle.
* The totalArea function is declared as a friend function to allow access to private members.

 **Friend Function totalArea**:

* The totalArea function takes an array of Shape pointers and the size of the array as parameters.
* It iterates through the array, summing the areas of all shapes using their respective area() functions.
* It returns the total area.

**Main Function**:

* Creates instances of Rectangle and Circle.
* Displays the details and areas of the shapes.
* Calculates and displays the total area of all shapes using the totalArea function.
* Frees the allocated memory for the shapes using delete to invoke the destructors and prevent memory leaks.

#include <iostream>

#include <cmath>

// Template class representing a point in 2D space

template <typename T>

class Point {

private:

T x;

T y;

public:

// Default constructor

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

// Overloaded constructor

Point(T xCoord, T yCoord) : x(xCoord), y(yCoord) {}

// Getter for x

T getX() const {

return x;

}

// Getter for y

T getY() const {

return y;

}

// Setter for x

void setX(T xCoord) {

x = xCoord;

}

// Setter for y

void setY(T yCoord) {

y = yCoord;

}

// Display the point

void display() const {

std::cout << "Point(" << x << ", " << y << ")" << std::endl;

}

};

// Base class representing a generic shape

class Shape {

public:

// Default constructor

Shape() {

std::cout << "Shape created." << std::endl;

}

// Virtual destructor

virtual ~Shape() {

std::cout << "Shape destroyed." << std::endl;

}

// Pure virtual function to calculate area

virtual double area() const = 0;

// Pure virtual function to display the shape details

virtual void display() const = 0;

// Friend function declaration

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

};

// Derived class representing a rectangle

class Rectangle : public Shape {

private:

Point<double> bottomLeft;

Point<double> topRight;

public:

// Overloaded constructor

Rectangle(double x1, double y1, double x2, double y2)

: bottomLeft(x1, y1), topRight(x2, y2) {

std::cout << "Rectangle created." << std::endl;

}

// Destructor

~Rectangle() {

std::cout << "Rectangle destroyed." << std::endl;

}

// Override the area function to calculate the area of a rectangle

double area() const override {

double length = std::abs(topRight.getX() - bottomLeft.getX());

double width = std::abs(topRight.getY() - bottomLeft.getY());

return length \* width;

}

// Override the display function

void display() const override {

std::cout << "Rectangle: ";

bottomLeft.display();

topRight.display();

std::cout << "Area: " << area() << std::endl;

}

// Friend function declaration

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

};

// Derived class representing a circle

class Circle : public Shape {

private:

Point<double> center;

double radius;

public:

// Overloaded constructor

Circle(double x, double y, double r)

: center(x, y), radius(r) {

std::cout << "Circle created." << std::endl;

}

// Destructor

~Circle() {

std::cout << "Circle destroyed." << std::endl;

}

// Override the area function to calculate the area of a circle

double area() const override {

return M\_PI \* radius \* radius;

}

// Override the display function

void display() const override {

std::cout << "Circle: ";

center.display();

std::cout << "Radius: " << radius << ", Area: " << area() << std::endl;

}

// Friend function declaration

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

};

// Friend function to calculate the total area of all shapes

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

double total = 0.0;

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

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

}

return total;

}

int main() {

// Create instances of Rectangle and Circle

Shape\* shapes[3];

shapes[0] = new Rectangle(0.0, 0.0, 10.0, 5.0);

shapes[1] = new Circle(0.0, 0.0, 7.0);

shapes[2] = new Rectangle(2.0, 3.0, 6.0, 8.0);

// Display the shapes and their areas

for (int i = 0; i < 3; ++i) {

shapes[i]->display();

}

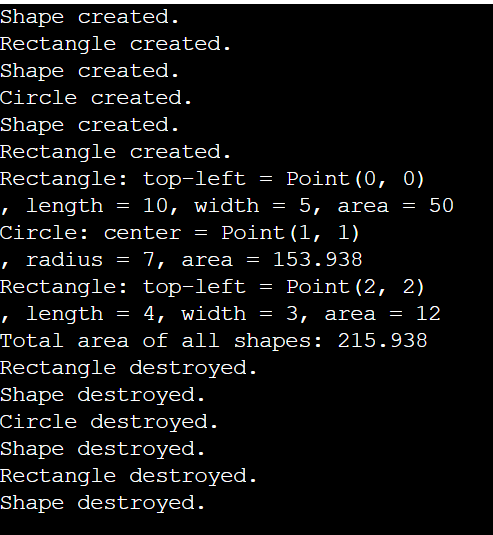
// Calculate the total area of all shapes

double total = totalArea(shapes, 3);

std::cout << "Total area of all shapes: " << total << std::endl;

return 0;

}



Q.7 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.

#include <iostream>

#include <cmath>

// Template class to represent a point in 2D space

template <typename T>

class Point {

public:

T x;

T y;

// Default constructor

Point() : x(T()), y(T()) {}

// Parameterized constructor

Point(T x, T y) : x(x), y(y) {}

// Display the coordinates of the point

void display() const {

std::cout << "Point(" << x << ", " << y << ")" << std::endl;

}

};

// Base class representing a generic shape

class Shape {

protected:

Point<double> position; // Position of the shape in 2D space

public:

// Default constructor

Shape(double x = 0.0, double y = 0.0) : position(x, y) {

std::cout << "Shape created at position ";

position.display();

}

// Virtual destructor

virtual ~Shape() {

std::cout << "Shape destroyed." << std::endl;

}

// Pure virtual function to calculate area

virtual double area() const = 0;

// Pure virtual function to display the shape details

virtual void display() const = 0;

// Friend function declaration

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

// Overload the == operator to compare shapes based on area

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

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

}

};

// Derived class representing a rectangle

class Rectangle : public Shape {

private:

double length;

double width;

public:

// Overloaded constructor

Rectangle(double x, double y, double l, double w)

: Shape(x, y), length(l), width(w) {

std::cout << "Rectangle created." << std::endl;

}

// Destructor

~Rectangle() {

std::cout << "Rectangle destroyed." << std::endl;

}

// Override the area function to calculate the area of a rectangle

double area() const override {

return length \* width;

}

// Override the display function

void display() const override {

std::cout << "Rectangle: position = ";

position.display();

std::cout << ", length = " << length << ", width = " << width << ", area = " << area() << std::endl;

}

// Overload the + operator to combine rectangles by summing their areas

Rectangle operator+(const Rectangle& other) const {

double newArea = this->area() + other.area();

// Assume the new rectangle is a square with the combined area for simplicity

double newLength = std::sqrt(newArea);

return Rectangle(position.x, position.y, newLength, newLength);

}

// Friend function declaration

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

};

// Derived class representing a circle

class Circle : public Shape {

private:

double radius;

public:

// Overloaded constructor

Circle(double x, double y, double r)

: Shape(x, y), radius(r) {

std::cout << "Circle created." << std::endl;

}

// Destructor

~Circle() {

std::cout << "Circle destroyed." << std::endl;

}

// Override the area function to calculate the area of a circle

double area() const override {

return M\_PI \* radius \* radius;

}

// Override the display function

void display() const override {

std::cout << "Circle: position = ";

position.display();

std::cout << ", radius = " << radius << ", area = " << area() << std::endl;

}

// Friend function declaration

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

};

// Friend function to calculate the total area of all shapes

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

double total = 0.0;

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

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

}

return total;

}

int main() {

// Create instances of Rectangle and Circle

Shape\* shapes[3];

shapes[0] = new Rectangle(0.0, 0.0, 10.0, 5.0);

shapes[1] = new Circle(1.0, 1.0, 7.0);

shapes[2] = new Rectangle(2.0, 2.0, 4.0, 3.0);

// Display the shapes and their areas

for (int i = 0; i < 3; ++i) {

shapes[i]->display();

}

// Calculate the total area of all shapes

double total = totalArea(shapes, 3);

std::cout << "Total area of all shapes: " << total << std::endl;

// Compare shapes based on area

if (\*shapes[0] == \*shapes[1]) {

std::cout << "The first two shapes have the same area." << std::endl;

} else {

std::cout << "The first two shapes have different areas." << std::endl;

}

// Combine two rectangles using the + operator

Rectangle\* rect1 = dynamic\_cast<Rectangle\*>(shapes[0]);

Rectangle\* rect2 = dynamic\_cast<Rectangle\*>(shapes[2]);

if (rect1 && rect2) {

Rectangle combinedRect = \*rect1 + \*rect2;

std::cout << "Combined rectangle: ";

combinedRect.display();

}

// Clean up

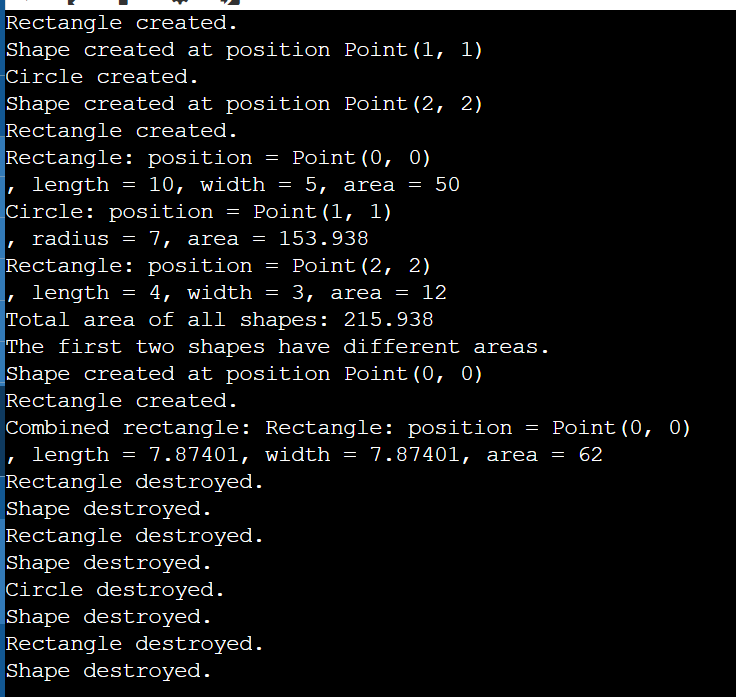
for (int i = 0; i < 3; ++i) {

delete shapes[i];

}

return 0;

}



Q.9 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).

#include <iostream>

#include <cmath>

// Template class to represent a point in 2D space

template <typename T>

class Point {

public:

T x;

T y;

// Default constructor

Point() : x(T()), y(T()) {}

// Parameterized constructor

Point(T x, T y) : x(x), y(y) {}

// Display the coordinates of the point

void display() const {

std::cout << "Point(" << x << ", " << y << ")" << std::endl;

}

};

// Base class representing a generic shape

class Shape {

protected:

Point<double> position; // Position of the shape in 2D space

public:

// Default constructor

Shape(double x = 0.0, double y = 0.0) : position(x, y) {

std::cout << "Shape created at position ";

position.display();

}

// Virtual destructor

virtual ~Shape() {

std::cout << "Shape destroyed." << std::endl;

}

// Pure virtual function to calculate area

virtual double area() const = 0;

// Pure virtual function to display the shape details

virtual void display() const = 0;

// Friend function declaration

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

// Overload the == operator to compare shapes based on area

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

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

}

};

// Derived class representing a rectangle

class Rectangle : public Shape {

private:

double length;

double width;

public:

// Overloaded constructor

Rectangle(double x, double y, double l, double w)

: Shape(x, y), length(l), width(w) {

std::cout << "Rectangle created." << std::endl;

}

// Destructor

~Rectangle() {

std::cout << "Rectangle destroyed." << std::endl;

}

// Override the area function to calculate the area of a rectangle

double area() const override {

return length \* width;

}

// Override the display function

void display() const override {

std::cout << "Rectangle: position = ";

position.display();

std::cout << ", length = " << length << ", width = " << width << ", area = " << area() << std::endl;

}

// Friend function declaration

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

};

// Derived class representing a circle

class Circle : public Shape {

private:

double radius;

public:

// Overloaded constructor

Circle(double x, double y, double r)

: Shape(x, y), radius(r) {

std::cout << "Circle created." << std::endl;

}

// Destructor

~Circle() {

std::cout << "Circle destroyed." << std::endl;

}

// Override the area function to calculate the area of a circle

double area() const override {

return M\_PI \* radius \* radius;

}

// Override the display function

void display() const override {

std::cout << "Circle: position = ";

position.display();

std::cout << ", radius = " << radius << ", area = " << area() << std::endl;

}

// Friend function declaration

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

};

// Friend function to calculate the total area of all shapes

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

double total = 0.0;

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

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

}

return total;

}

int main() {

// Create instances of Rectangle and Circle

Shape\* shapes[3];

shapes[0] = new Rectangle(0.0, 0.0, 10.0, 5.0);

shapes[1] = new Circle(1.0, 1.0, 7.0);

shapes[2] = new Rectangle(2.0, 2.0, 4.0, 3.0);

// Display the shapes and their areas

for (int i = 0; i < 3; ++i) {

shapes[i]->display();

}

// Calculate the total area of all shapes

double total = totalArea(shapes, 3);

std::cout << "Total area of all shapes: " << total << std::endl;

// Compare shapes based on area

if (\*shapes[0] == \*shapes[1]) {

std::cout << "The first two shapes have the same area." << std::endl;

} else {

std::cout << "The first two shapes have different areas." << std::endl;

}

// Clean up

for (int i = 0; i < 3; ++i) {

delete shapes[i];

}

// Test the Point template class

Point<int> p1(3, 4);

p1.display();

Point<double> p2(5.5, 6.6);

p2.display();

return 0;

}

