11/7/24 Day14 Assessment File Handing

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).

**Shape.h**

**// Shape.h**

**#ifndef SHAPE\_H**

**#define SHAPE\_H**

**class Shape {**

**public:**

**// Constructor**

**Shape();**

**// Destructor (virtual to ensure proper cleanup in derived classes)**

**virtual ~Shape();**

**// Pure virtual function to calculate area**

**virtual double area() const = 0;**

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

**bool operator==(const Shape& other) const;**

**// Friend function to calculate total area of an array of shapes**

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

**};**

**// Constructor definition**

**inline Shape::Shape() {**

**// Common initialization if any**

**}**

**// Destructor definition**

**inline Shape::~Shape() {**

**// Virtual destructor ensures proper cleanup**

**}**

**// Overload == operator definition**

**inline bool Shape::operator==(const Shape& other) const {**

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

**}**

**#endif // SHAPE\_H**

**Rectangle.h**

**// Rectangle.h**

**#ifndef RECTANGLE\_H**

**#define RECTANGLE\_H**

**#include "Shape.h"**

**class Rectangle : public Shape {**

**private:**

**double length;**

**double width;**

**public:**

**// Constructors**

**Rectangle(double l, double w);**

**// Destructor**

**~Rectangle() override;**

**// Override area function to calculate area of rectangle**

**double area() const override;**

**};**

**#endif // RECTANGLE\_H**

**Circle.h**

**// Circle.h**

**#ifndef CIRCLE\_H**

**#define CIRCLE\_H**

**#include "Shape.h"**

**class Circle : public Shape {**

**private:**

**double radius;**

**public:**

**// Constructors**

**Circle(double r);**

**// Destructor**

**~Circle() override;**

**// Override area function to calculate area of circle**

**double area() const override;**

**};**

**#endif // CIRCLE\_H**

**totalArea.h**

**// totalArea.h**

**#ifndef TOTALAREA\_H**

**#define TOTALAREA\_H**

**#include "Shape.h"**

**// Friend function to calculate total area of an array of shapes**

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

**#endif // TOTALAREA\_H**

**Main.cpp**

**#include <iostream>**

**#include "Rectangle.h"**

**#include "Circle.h"**

**#include "totalArea.h"**

**int main() {**

**// Create shapes**

**Rectangle rect(3.0, 4.0);**

**Circle circle(2.5);**

**// Test overridden area functions**

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

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

**// Test comparison using ==**

**if (rect == circle) {**

**std::cout << "The rectangle and circle have the same area." << std::endl;**

**} else {**

**std::cout << "The rectangle and circle do not have the same area." << std::endl;**

**}**

**// Create an array of Shape pointers**

**const Shape\* shapes[] = { &rect, &circle };**

**// Calculate total area using friend function**

**double total = totalArea(shapes, 2);**

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

**return 0;**

**}**

**Output**

**g++ -c rectancle.cpp -o rectancle.o**

**g++ -c totalArea.cpp -o totalArea.o**

**g++ -c shape.cpp -o shape.o**

**g++ -c circle.cpp -o circle.o**

**g++ main.cpp shape.o rectancle.o circle.o totalArea.o -o shapes\_program**

**PS D:\CPPAdvance\Day14> ./shapes\_program**

**Output**

**Area of Rectangle: 12**

**Area of Circle: 19.6349**

**The rectangle and circle do not have the same area.**

**Total area of shapes: 31.6349**