Cppexam:

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).

#include <iostream>

#include <string>

using namespace std;

template <typename T>

class Point {

public:

T x;

T y;

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

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

void display() const {

cout << "(" << x << ", " << y << ")";

}

};

class Shape;

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

class Shape {

protected:

string name;

Point<double> position;

public:

Shape() : name("Shape"), position(0, 0) {

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

}

Shape(double x, double y) : name("Shape"), position(x, y) {

cout << "Shape created at position ";

position.display();

cout << "." << endl;

}

virtual double area() const = 0;

virtual void display() const = 0;

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

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

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

}

virtual ~Shape() {

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

}

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

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

name = "Rectangle";

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

}

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

name = "Rectangle";

cout << "Rectangle created with length " << l << " and width " << w << "." << endl;

}

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

name = "Rectangle";

cout << "Rectangle created at position ";

position.display();

cout << " with length " << l << " and width " << w << "." << endl;

}

double area() const override {

return length \* width;

}

void display() const override {

cout << name << ": " << endl;

cout << "Position: ";

position.display();

cout << endl;

cout << "Length: " << length << endl;

cout << "Width: " << width << endl;

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

}

Rectangle operator+(const Rectangle& other) const {

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

double side = combinedArea / length; // Approximation for simplicity

return Rectangle(side, side);

}

~Rectangle() {

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

}

};

class Circle : public Shape {

private:

double radius;

public:

Circle() : Shape(), radius(0) {

name = "Circle";

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

}

Circle(double r) : Shape(), radius(r) {

name = "Circle";

cout << "Circle created with radius " << r << "." << endl;

}

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

name = "Circle";

cout << "Circle created at position ";

position.display();

cout << " with radius " << r << "." << endl;

}

double area() const override {

const double PI = 3.141592653589793;

return PI \* radius \* radius;

}

void display() const override {

cout << name << ": " << endl;

cout << "Position: ";

position.display();

cout << endl;

cout << "Radius: " << radius << endl;

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

}

~Circle() {

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

}

};

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

double total = 0;

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

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

}

return total;

}

int main() {

Shape\* shapes[4];

shapes[0] = new Rectangle(5.0, 3.0);

shapes[1] = new Rectangle(2.0, 2.0);

shapes[2] = new Circle(4.0);

shapes[3] = new Circle(2.0);

for (int i = 0; i < 4; ++i) {

shapes[i]->display();

cout << endl;

}

double total = totalArea(shapes, 4);

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

for (int i = 0; i < 4; ++i) {

delete shapes[i];

}

return 0;

}

2. Problem 1: Read from a File

Task:

Write a C++ program that reads a text file named input.txt and prints its content to the console.

Questions:

How do you open a file for reading in C++?

What is the purpose of the ifstream class in C++?

How can you check if a file was successfully opened?

What function do you use to read a line from a file?

How do you properly close a file after reading?

Problem 2: Write to a File

Task:

Write a C++ program that writes the following lines to a file named output.txt:

bash

Copy code

Hello, world!

This is a test file.

Questions:

How do you open a file for writing in C++?

What is the purpose of the ofstream class in C++?

How can you handle errors if the file fails to open for writing?

How do you write a string to a file in C++?

What is the importance of closing a file after writing to it?

Problem 3: Append to a File

Task:

Write a C++ program that appends the following line to a file named log.txt:

bash

Copy code

New log entry.

Questions:

How do you open a file for appending in C++?

What is the difference between opening a file in write mode and append mode?

How do you use the ofstream class to append data to a file?

What happens if the file does not exist when you try to open it in append mode?

How can you ensure data integrity when appending to a file?

Problem 4: Copy a File

Task:

Write a C++ program that copies the content of a file named source.txt to another file named destination.txt.

Questions:

How do you read from one file and write to another file in C++?

How can you efficiently copy the contents of a file in C++?

What are the potential errors you should handle when copying a file?

How do you check the end-of-file (EOF) condition when reading a file?

How do you ensure both files are properly closed after the copy operation?

Problem 5: Count Words in a File

Task:

Write a C++ program that reads a file named data.txt and counts the number of words in the file.

Questions:

How do you define a word in the context of reading from a file?

What functions can you use to read words from a file in C++?

How do you handle different word delimiters (spaces, newlines, etc.)?

How can you keep track of the word count while reading the file?

How do you handle large files to avoid memory issues while counting words?

These problems and questions should help you understand the basics of file operations in C++.

PROBLEM 1:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

ifstream inputFile("input.txt");

// Check if file was successfully opened

if (!inputFile.is\_open()) {

cerr << "Failed to open the file." << endl;

return 1;

}

string line;

// Read the file line by line

while (getline(inputFile, line)) {

cout << line << endl;

}

// Close the file

inputFile.close();

return 0;

}

PROBLEM 2:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream outputFile("output.txt");

// Check if file was successfully opened

if (!outputFile.is\_open()) {

cerr << "Failed to open the file for writing." << endl;

return 1;

}

// Write to the file

outputFile << "Hello, world!" << endl;

outputFile << "This is a test file." << endl;

// Close the file

outputFile.close();

return 0;

}

PROBLEM 3:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream logFile("log.txt", ios::app);

// Check if file was successfully opened

if (!logFile.is\_open()) {

cerr << "Failed to open the file for appending." << endl;

return 1;

}

// Append to the file

logFile << "New log entry." << endl;

// Close the file

logFile.close();

return 0;

}