# OOP Assignment 4

USMAN KHAN (46729)

## ### Part 1: Theory

1. Explain what polymorphism is and how it relates to object-oriented programming?

ANS: Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows objects of different classes to be treated as if they were objects of a common superclass or interface. Polymorphism is enabled in OOP by the use of inheritance and interfaces. Inheritance allows a subclass to inherit the properties and methods of its superclass, while interfaces provide a common contract that specifies a set of methods and properties that a class must implement.

1. What is the difference between static and dynamic polymorphism?

Ans: The main difference between static and dynamic polymorphism is the point at which the binding of the method or operator occurs. **Static polymorphism**, also known as compile-time polymorphism, occurs when the binding of the method or operator is determined at compile time. **Dynamic polymorphism**, also known as run-time polymorphism, occurs when the binding of the method or operator is determined at run time.

1. Describe the two types of polymorphism in C++?

Ans: In C++, there are two types of polymorphism: compile-time polymorphism, which is also known as static polymorphism, and run-time polymorphism, which is also known as dynamic polymorphism. **Static polymorphism**, also known as compile-time polymorphism, occurs when the binding of the method or operator is determined at compile time. **Dynamic polymorphism**, also known as run-time polymorphism, occurs when the binding of the method or operator is determined at run time.

1. What is a virtual function? Explain why it is used?

Ans: In object-oriented programming, a virtual function is a function that is declared in a base class and can be overridden in a derived class. When a virtual function is called on a pointer or reference to an object, the actual type of the object determines which implementation of the function to call. This allows for dynamic dispatch of the function call at run time.

1. Can a class have both virtual and non-virtual functions? Explain your answer?

Ans: Yes, a class in C++ can have both virtual and non-virtual functions.Virtual functions are used to enable run-time polymorphism, which allows a function to be called on an object of a base class, but the appropriate implementation to be selected based on the actual type of the object. Non-virtual functions, on the other hand, are resolved at compile time and are not intended to be overridden in derived classes.It is common for a class to have a mix of virtual and non-virtual functions. Non-virtual functions are often used for utility functions or to enforce constraints on the behavior of the class, while virtual functions are used to enable polymorphism

## ### Part 2: Implementation

1. Write a C++ program that demonstrates the concept of function overloading.

Ans: #include <iostream>

using namespace std;

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

int main() {

int sum1 = add(5, 10);

cout << "Sum of integers: " << sum1 << endl;

double sum2 = add(3.14, 2.71);

cout << "Sum of doubles: " << sum2 << endl;

return 0;

}

1. Write a C++ program that demonstrates the concept of operator overloading?

Ans: #include <iostream>

using namespace std;

class Box {

public:

double length;

double width;

double height;

Box() {

length = 0;

width = 0;

height = 0;

}

Box(double l, double w, double h) {

length = l;

width = w;

height = h;

}

Box operator +(const Box& b) {

Box box;

box.length = length + b.length;

box.width = width + b.width;

box.height = height + b.height;

return box;

}

};

int main() {

Box box1(2, 3, 4);

Box box2(3, 4, 5);

Box box3 = box1 + box2;

cout << "Length: " << box3.length << endl;

cout << "Width: " << box3.width << endl;

cout << "Height: " << box3.height << endl;

return 0;

}

1. Write a C++ program that demonstrates the concept of runtime polymorphism using virtual functions?

Ans: #include <iostream>

using namespace std;

class Shape {

public:

virtual double area() {

cout << "Parent class area function" << endl;

return 0;

}

};

class Rectangle : public Shape {

public:

double length;

double width;

Rectangle(double l, double w) {

length = l;

width = w;

}

double area() {

cout << "Area of Rectangle" << endl;

return length \* width;

}

};

class Circle : public Shape {

public:

double radius;

Circle(double r) {

radius = r;

}

double area() {

cout << "Area of Circle" << endl;

return 3.14 \* radius \* radius;

}

};

int main() {

Shape \*shapes[2];

Rectangle rectangle(5, 10);

shapes[0] = &rectangle;

Circle circle(5);

shapes[1] = &circle;

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

cout << "Area: " << shapes[i]->area() << endl;

}

return 0;

}

1. Write a C++ program that demonstrates the concept of compile-time polymorphism using templates?

Ans: #include <iostream>

using namespace std;

template<typename T>

T add(T a, T b) {

return a + b;

}

int main() {

int num1 = 5, num2 = 10;

double decimal1 = 3.5, decimal2 = 2.5;

string str1 = "Hello", str2 = "World";

cout << add(num1, num2) << endl;

cout << add(decimal1, decimal2) << endl;

cout << add(str1, str2) << endl;

return 0;

}

## ### Part 3: Application

1. Write a C++ program that uses polymorphism to create a hierarchy of shapes. The program should have a base class called `Shape` and derived classes for different types of shapes (e.g. `Circle`, `Rectangle`, `Triangle`). Each derived class should implement a function called `area()` that calculates the area of the shape. The program should allow the user to create objects of different shapes and calculate their areas using polymorphism.

Ans: #include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

virtual double area() = 0;

};

class Circle : public Shape {

public:

double radius;

Circle(double r) {

radius = r;

}

double area() {

return M\_PI \* radius \* radius;

}

};

class Rectangle : public Shape {

public:

double length;

double width;

Rectangle(double l, double w) {

length = l;

width = w;

}

double area() {

return length \* width;

}

};

class Triangle : public Shape {

public:

double base;

double height;

Triangle(double b, double h) {

base = b;

height = h;

}

double area() {

return 0.5 \* base \* height;

}

};

int main() {

Shape \*shapes[3];

Circle circle(5);

Rectangle rectangle(5, 10);

Triangle triangle(6, 8);

shapes[0] = &circle;

shapes[1] = &rectangle;

shapes[2] = &triangle;

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

cout << "Area: " << shapes[i]->area() << endl;

}

return 0;

}

1. Extend the previous program to include a function that sorts an array of shapes based on their area. The function should use polymorphism to determine the area of each shape and compare them. The program should allow the user to create an array of shapes of different types and sizes and sort them by area?

Ans: #include <iostream>

#include <cmath>

#include <algorithm>

using namespace std;

class Shape {

public:

virtual double area() = 0;

};

class Circle : public Shape {

public:

double radius;

Circle(double r) {

radius = r;

}

double area() {

return M\_PI \* radius \* radius;

}

};

class Rectangle : public Shape {

public:

double length;

double width;

Rectangle(double l, double w) {

length = l;

width = w;

}

double area() {

return length \* width;

}

};

class Triangle : public Shape {

public:

double base;

double height;

Triangle(double b, double h) {

base = b;

height = h;

}

double area() {

return 0.5 \* base \* height;

}

};

bool compareShapes(Shape\* a, Shape\* b) {

return a->area() < b->area();

}

int main() {

Shape\* shapes[3];

Circle circle(5);

Rectangle rectangle(5, 10);

Triangle triangle(6, 8);

shapes[0] = &circle;

shapes[1] = &rectangle;

shapes[2] = &triangle;

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

cout << "Area: " << shapes[i]->area() << endl;

}

cout << "Sorting shapes by area..." << endl;

sort(shapes, shapes + 3, compareShapes);

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

cout << "Area: " << shapes[i]->area() << endl;

}

return 0;

}

## ### Part 4: Reflection

1. Reflect on what you learned in this assignment. What was challenging, and what did you find interesting?

Ans: This assignment covered the topic of polymorphism in object-oriented programming, including function overloading, operator overloading, and runtime and compile-time polymorphism. Overall, this assignment helped me deepen my understanding of polymorphism and how it can be used to create more flexible and extensible code.

1. How can you apply what you learned in this assignment to future projects or your future career?

Ans: By applying the concepts of polymorphism, function overloading, operator overloading, and runtime and compile-time polymorphism learned in this assignment, I can write more flexible, reusable, and maintainable object-oriented code in my future projects or career as a programmer. This can improve the scalability, efficiency, and readability of the codebase and help me avoid duplicating code.