Name : Zeeshan Shoukat

OOP Assignment # 04 Final Term

Sap ID: 47170

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

In object-oriented programming, polymorphism, which is defined as "having multiple forms" in Greek, refers to the ability to give something a different meaning or usage in different contexts. More specifically, it refers to the ability to allow an entity, such as a variable, a function, or an object, to have more than one form. It is possible for a variable to exist in more than one form, and the programmer can choose which form to use when the variable is used. A named function's behavior can also change based on the input arguments.

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

The primary distinction between static and dynamic polymorphism is that static polymorphism resolves at compile time whereas dynamic polymorphism resolves at run time.

Q. Describe the two types of polymorphism in C++.

Compile-time and run-time polymorphism in C++ can be further subdivided into function overloading and virtual functions, respectively. Compile-time polymorphism can also be further subdivided into operator overloading and function overloading.

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

Regardless of the expression used to invoke the function, virtual functions guarantee sure the right function is called for an object. Let us say a derived class defines a function that is contained in a base class and is marked as virtual. When defining overriding functions in a derived class, the virtual keyword can be used, but it is not necessary because overrides of virtual functions are always virtual.

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

While virtual functions are dynamically resolved at run-time, non-virtual functions are resolved statically at compile time. There is a small expense in the case of virtual functions to accomplish this flexibility of being able to choose which function to invoke at run-time.

QUESTION NO 2: IMPLEMENTATION

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

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  class Math {
5.    public:
6.      int add(int x, int y) {
7.        return x + y;
8.      }
9.      double add(double x, double y) {
10.       return x + y;
```

```
    }
};

int main() {
    Math obj;
    int a = 10, b = 20;
    double c = 1.5, d = 2.5;

    cout << "Addition of two integers: " << obj.add(a, b) << endl;
    cout << "Addition of two doubles: " << obj.add(c, d) << endl;

    return 0;
}
```

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

```
#include <iostream>
using namespace std;

class Complex {
  private:
    double real;
    double imag;

  public:
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

    Complex operator+(Complex const &obj) {
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }

    void display() { cout << real << " + " << imag << "i" << endl; }
};

int main() {
    Complex c1(2.5, 3.0);
    Complex c2(1.5, 2.5);

    Complex c3 = c1 + c2;

    c1.display();
    c2.display();
    c3.display();

    return 0;
}
```

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

```
#include <iostream>
using namespace std;

class Shape {
   public:
    virtual void draw() = 0;
};

class Circle : public Shape {
   public:
    void draw() {
       cout << "Drawing a circle" << endl;
    }
};

class Square : public Shape {
   public:
    void draw() {
       cout << "Drawing a square" << endl;
    }
};

int main() {
    Shape* shapes[2];
    shapes[0] = new Circle();
    shapes[1] = new Square();

    for (int i = 0; i < 2; i++) {
       shapes[i]->draw();
       delete shapes[i];
    }

    return 0;
}
```

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

```
#include <iostream>
using namespace std;

template<typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}

int main() {
```

```
    int x = 5, y = 10;
    double a = 1.23, b = 4.56;

    cout << "Max of " << x << " and " << y << " is " << max(x, y) << endl;
    cout << "Max of " << a << " and " << b << " is " << max(a, b) << endl;

    return 0;
}
```

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.

```
#include <iostream>
using namespace std;

class Shape {
  public:
   virtual double area() = 0;
};

class Circle : public Shape {
  private:
   double radius;

  public:
   Circle(double r) {
      radius = r;
   }

   double area() {
      return 3.14159 * radius * radius;
   }
};

class Rectangle : public Shape {
  private:
   double width, height;

  public:
   Rectangle(double w, double h) {
      width = w;
      height = h;
   }

   double area() {
      return width * height;
```

```
    }
};

class Triangle : public Shape {
  private:
    double base, height;

  public:
    Triangle(double b, double h) {
        base = b;
        height = h;
    }

    double area() {
        return 0.5 * base * height;
    }
};

int main() {
    Shape* shapes[3];
    shapes[0] = new Circle(5);
    shapes[1] = new Rectangle(4, 6);
    shapes[2] = new Triangle(3, 8);

    for (int i = 0; i < 3; i++) {
        cout << "Area of shape " << i + 1 << " is " << shapes[i]->area() << endl;
        delete shapes[i];
    }

    return 0;
}
```

REFLECTION

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

While working on this assignment, I picked up a lot of new programming skills. Additionally, I learned about variables, functions, inheritance, and its various sorts. Static and dynamic inheitance was also learned. I also had a lot of difficulties with this task, but learning about inheritance and its various forms is simple and fascinating.

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

In C++, I learned a lot about inheritance and the fundamentals of OOP. Both virtual and non-virtual functionalities as well as many more subjects for upcoming developments.