Assignment 4

Name : Fatima Fiaz

SAP ID : 45470

BS CS 2nd sem 2A

Subject : OOP

[Pick the date]

OOP Assignment 4

➢ Part 1: theory

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

Polymorphism is a feature of object-oriented programming languages that allows a specific routine to use variables of different types at different times.
Polymorphism in programming gives a program the ability to redefine methods for derived classes. It allows different objects to be used interchangeably, which makes code more flexible and reusable.

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

Static Polymorphism in Java is a type of polymorphism that collects the information for calling a method at compilation time, whereas Dynamic Polymorphism is a type of polymorphism that collects the information for calling a method at runtime.

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

Two types: Compile-time Polymorphism and Runtime Polymorphism.

- In Compile Time Polymorphism, the function to be invoked is decided at the compile time only. It is achieved using a function or operator overloading.
- In Runtime Polymorphism, the function invoked is decided at the Run-time.

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

A virtual function is a member function that you expect to be redefined in derived classes.
When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

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

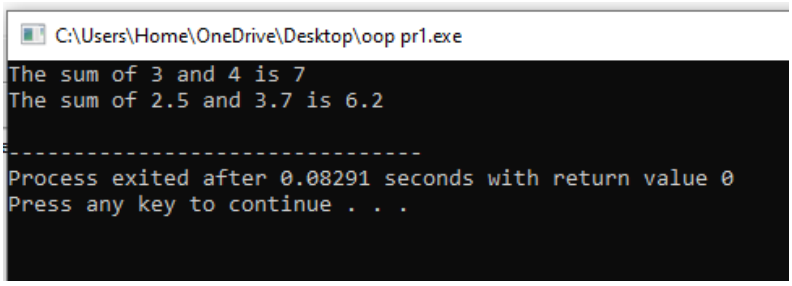
YES, a class can have both virtual and non-virtual functions, and each serves a different purpose. Non-virtual functions are resolved at compile-time, while virtual functions participate in dynamic binding and are resolved at runtime based on the actual object type.

• Part 2: Implementation

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

```
#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 x = add(3, 4);
  double y = add(2.5, 3.7);
  cout << "The sum of 3 and 4 is " << x << endl;
  cout << "The sum of 2.5 and 3.7 is " << y << endl;
  return 0;
}
```

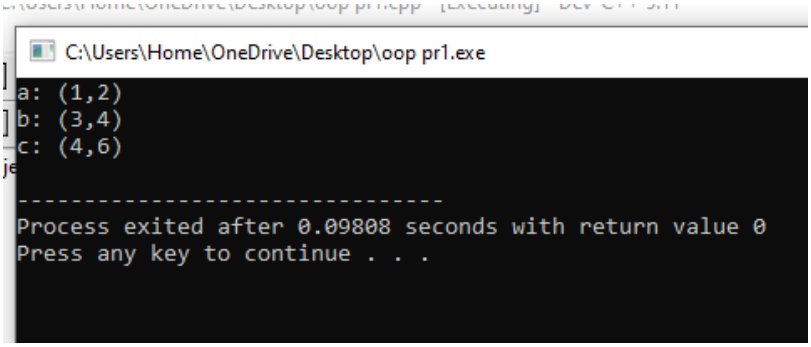
- **Output**



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

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

class Point {
 public:
   Point(int x=0, int y=0) : x(x), y(y) {}
   Point operator+(const Point& p) const {
      return Point(x+p.x, y+p.y);
   }
   friend ostream& operator<<(ostream& os, const Point& p) {
      os << "(" << p.x << "," << p.y << ")";
      return os;
   }
 private:
   int x, y;
};
int main() {
   Point a(1, 2);
   Point b(3, 4);
   Point c = a + b;
   cout << "a: " << a << endl;
   cout << "b: " << b << endl;
   cout << "c: " << c << endl;
   return 0;
}
```

- **Output**

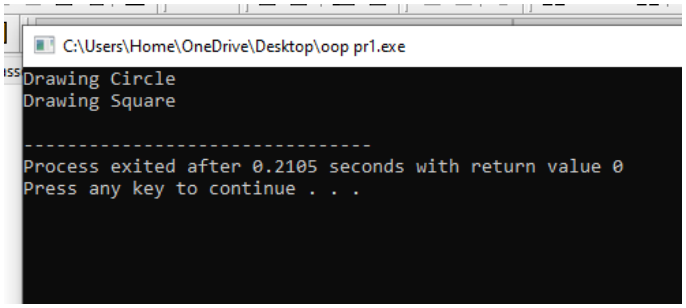


3. Write a C++ program that demonstrates the concept of runtime polymorphism using virtual functions.

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

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

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

class Square : public Shape {
  public:
    void draw() const override {
        cout << "Drawing 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();
    }
    return 0;
}
```

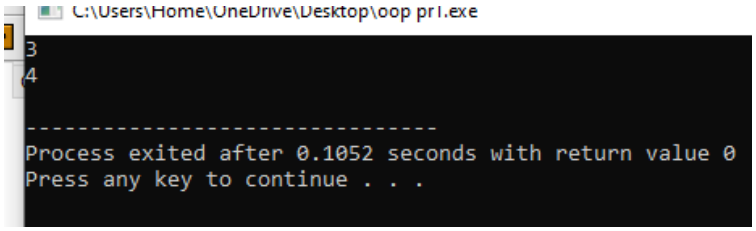
- **Output**



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

```
#include <iostream>
template <typename T>
T add(T a, T b) {
    return a + b;
}
int main() {
    int x = 1, y = 2;
    std::cout << add<int>(x, y) << std::endl;
    double a = 1.5, b = 2.5;
    std::cout << add<double>(a, b) << std::endl;
    return 0;
}
```

- **Output**



- **Part 3: Application**

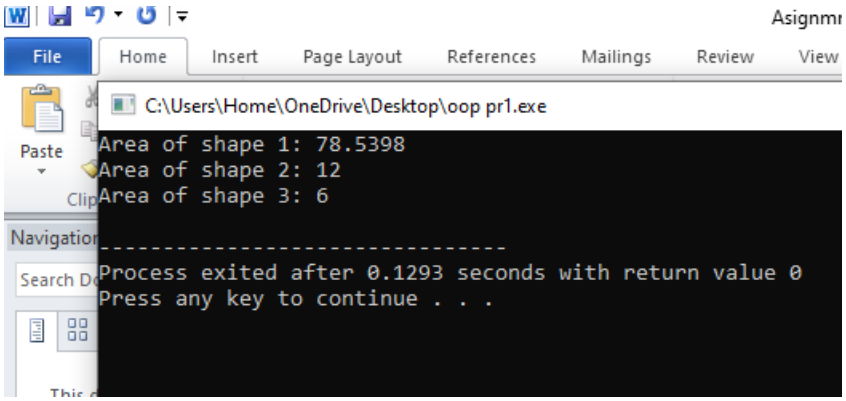
1. **Write a C++ program that uses polymorphism to create a hierarchy of shapes. The program should have a base class called `Shape` and derived classes for different types of shapes (e.g. `Circle`, `Rectangle`, `Triangle`). Each derived class should implement a function called `area()` that calculates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

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

class Shape {
public:
   virtual double area() = 0; // pure virtual function
};
class Circle : public Shape {
private:
   double radius;
public:
   Circle(double r) {
      radius = r;
   }
   double area() {
      return M_PI * pow(radius, 2);
   }
};
```

```
class Rectangle : public Shape {
private:
   double width;
 double height;
public:
   Rectangle(double w, double h) {
      width = w;
      height = h;
   }
   double area() {
      return width * height;
   }
};
class Triangle : public Shape {
private:
   double base;
   double 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(3, 4);
   shapes[2] = new Triangle(2, 6);

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

```
    cout << "Area of shape " << i + 1 << ": " << shapes[i]->area() << endl;
        delete shapes[i];
    }
    return 0;
}
```

- **Output**



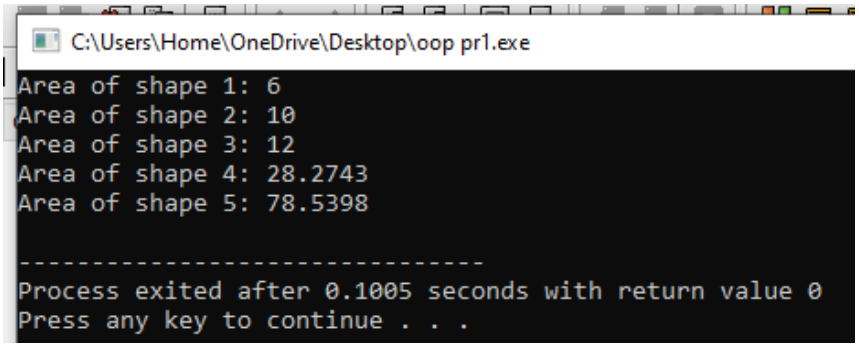
2. **Extend the previous program to include a function that sorts an array of shapes based on their area. The function should use polymorphism to determine the area of each shape and compare them. The program should allow the user to create an array of shapes of different types and sizes and sort them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

```
#include <iostream>
#include <cmath>
#include <algorithm>

using namespace std;

class Shape {
public:
    virtual double area() = 0; // pure virtual function
    virtual bool operator<(Shape& other) {
        return area() < other.area();
    }
```

```
};

class Circle : public Shape {
private:
   double radius;
public:
   Circle(double r) {
      radius = r;
   }
   double area() {
      return M_PI * pow(radius, 2);
   }
};

class Rectangle : public Shape {
private:
   double width;
   double height;
public:
   Rectangle(double w, double h) {
      width = w;
      height = h;
   }
   double area() {
      return width * height;
   }
};

class Triangle : public Shape {
private:
   double base;
   double height;
public:
   Triangle(double b, double h) {
      base = b;
      height = h;
   }
```

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

bool compareShapes(Shape* s1, Shape* s2) {
    return *s1 < *s2;
}

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

    sort(shapes, shapes + 5, compareShapes);

    for (int i = 0; i < 5; i++) {
        cout << "Area of shape " << i + 1 << ": " << shapes[i]->area() << endl;
        delete shapes[i];
    }

    return 0;
}
```

- **Output**



- **Part 4: Reflection**

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

   This assignment provides me a good introduction to the concepts of polymorphism in OOP.

   The challenging and an interesting thing was the idea of virtual functions and abstract classes and then implementing it.

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

   The concepts of polymorphism, abstract classes, inheritance etc. are the mains of OOP language and they all will use in software development related to the field of CS (Computer Science).

   In future projects and future career, I can apply these methods to create programs in flexible and modular way.

   It will be valuable for any future programming projects or career.
