**Part 1: Theory**

1. Polymorphism is a concept in object-oriented programming where the same code can be used to operate on multiple types of data. It allows an object to take on many forms, allowing methods to be overridden so that they can perform different functions depending on the type of data they are given.

2. Static polymorphism is a type of polymorphism that is resolved at compile time, meaning that the code is determined when the program is compiled. Dynamic polymorphism is a type of polymorphism that is resolved at runtime, meaning that the code is determined when the program is running.

3. C++ supports two types of polymorphism: compile-time polymorphism and runtime polymorphism. Compile-time polymorphism is achieved through function overloading, operator overloading, and templates. Runtime polymorphism is achieved through virtual functions.

4. A virtual function is a function that is declared in a base class and is overridden in derived classes. It allows for dynamic dispatch, meaning that the correct version of the function is called based on the type of object being used, rather than being determined at compile time. It is used to implement polymorphism.

5. Yes, a class can have both virtual and non-virtual functions. Virtual functions are used to implement polymorphism, while non-virtual functions are used for other purposes.

**Part 2: Implementation**

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

#include <iostream>

using namespace std;

// Function with one int parameter

int add(int a) { return a + a; }

// Function with two int parameters

int add(int a, int b) { return a + b; }

// Function with three int parameters

int add(int a, int b, int c) { return a + b + c; }

int main()

{

cout << add(2) << endl;

cout << add(2, 3) << endl;

cout << add(2, 3, 5) << endl;

return 0;

}

2. Write a C++ program that demonstrates the concept of operator overloading. #include <iostream>

using namespace std;

class Complex {

private:

int real, imag;

public:

Complex(int r = 0, int i =0) { real = r; imag = i; }

// This is automatically called when '+' is used with // between two Complex objects

Complex operator + (Complex const &obj) {

Complex res;

res.real = real + obj.real;

res.imag = imag + obj.imag;

return res;

}

void print() {

cout << real << " + i" << imag << endl;

}

};

int main() {

Complex c1(10, 5), c2(2, 4);

Complex c3 = c1 + c2; // An example call to "operator+"

c3.print();

}

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

#include <iostream>

using namespace std;

class Shape {

public: // pure virtual function

virtual void draw() = 0;

};

class Circle : public Shape {

public:

void draw() {

cout<<"Drawing Circle"<<endl;

}

};

class Rectangle : public Shape {

public:

void draw() {

cout<<"Drawing Rectangle"<<endl;

}

};

class Triangle : public Shape { public: void draw() {

cout<<"Drawing Triangle"<<endl;

}

};

int main() {

Shape \*s;

Circle c;

s = &c;

s->draw();

Rectangle r;

s = &r; s->draw();

Triangle t;

s = &t;

s->draw();

return 0;

}

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

#include <iostream>

using namespace std;

// Function template to add two values template

<class T>

T add(T a, T b) {

return a + b;

}

int main()

{

// Compile-time polymorphism

cout << add(2, 3) << endl;

cout << add(2.5, 3.5) << endl;

cout << add('G', 'e') << endl;

return 0;

}

**Part 3: Application**

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;

//Base class

class Shape {

public:

//Pure virtual function

virtual double area() = 0;

};

//Derived classes

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 length;

double width;

public:

Rectangle(double l, double w) {

length = l; width = w;

}

double area() {

return length \* width;

}

};

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()

{

//Create objects of different shapes

Circle c(3.0);

Rectangle r(4.0, 5.0);

Triangle t(6.0, 7.0);

//Calculate areas of different shapes using polymorphism

cout << "Area of Circle: " << c.area() << endl;

cout << "Area of Rectangle: " << r.area() << endl;

cout << "Area of Triangle: " << t.area() << endl;

return 0;

}

Extend the 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 <vector>

#include <algorithm>

using namespace std;

//Base class

class Shape {

public: virtual double area() = 0;

};

//Derived classes

class Circle : public Shape {

private:

double radius;

public: Circle(double radius) {

this->radius = radius;

}

double area() {

return 3.14 \* radius \* radius;

}

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

Rectangle(double length, double width) {

this->length = length;

this->width = width;

}

double area() {

return length \* width;

}

};

class Triangle : public Shape {

private:

double base;

double height;

public:

Triangle(double base, double height) {

this->base = base;

this->height = height;

}

double area() {

return 0.5 \* base \* height;

}

};

//Sorting function

bool compareArea(Shape \*s1, Shape \*s2) {

return (s1->area() < s2->area());

}

int main() {

//Create vector of shapes

Vector<Shape \*> shapes;

shapes.push\_back (new Circle (3.0));

shapes.push\_back (new Rectangle (2.0, 3.0));

shapes.push\_back (new Triangle (3.0, 4.0));

//Calculate areas

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

cout << "Area of shape " << i + 1 << " is: " << shapes[i]->area() << endl;

}

//Sort vector

sort(shapes.begin(), shapes.end(), compareArea);

//Print sorted vector

cout << "Shapes sorted by area:" << endl;

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

cout << "Area of shape " << i + 1 << " is:" << 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?

We learned about the concepts of opp ,involving polymorphism ,inheritance ,vectors ,virtual functions.

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

As we are currently working on our projects we can implement the following concepts to make our program work efficiently .