Polymorphism in c++

What is Polymorphism?

- The word polymorphism can be described as **an object having many forms**
- To understand polymorphism, you can consider a real-life example. You can relate it to the relationship of a person with different people. A man can be a father to someone, a husband, a boss, an employee, a son, a brother, or can have many other relationships with various people. Here, this man represents the object, and his relationships display the ability of this object to be represented in many forms with totally different characteristics.

Why do we need polymorphism

- Polymorphism has several benefits associated with it.
- One of them is that it helps you write the code consistently. For example, suppose you want to find the area of a circle and a rectangle. For that, your first approach will be to create a base class that will deal with the type of polygon. This base class contains two derived classes, one for the circle and another one for the rectangle. Now, instead of declaring two separate functions with separate names in both the derived classes to calculate the area of each polygon, you can just declare one function in the base class and override it in the child classes to calculate the area. In this way, you increase the code consistency using polymorphism.

Types of Polymorphism

- Based on the functionality, you can categorize polymorphism into two types
- Compile-Time Polymorphism (static)
- Runtime Polymorphism (dynamic)

Compile-Time Polymorphism

- When the relationship between the definition of different functions and their function calls, is determined during the compile-time, it is known as compile-time polymorphism.
- This type of polymorphism is also known as **static** or **early binding** polymorphism.
- All the methods of compile-time polymorphism get called or invoked during the compile time.
- Examples are **Function overloading** and **Operator Overloading**

Runtime Polymorphism

- In runtime polymorphism, the compiler resolves the object at run time and then it decides which function call should be associated with that object.
- It is also known as dynamic or late binding polymorphism.

- The implementation of run time polymorphism can be achieved in two ways:
- Function overriding
- Virtual functions

Method overriding

- Method overriding is an application of run time polymorphism where two or more functions with **the same name, arguments, and return type accompany different classes of the same structure.**
- This method has a comparatively **slower** execution rate than compile-time polymorphism since all the methods that need to be executed are called during run time.
- Runtime polymorphism is known to be **better for dealing with complex problems** since all the methods and details turn up during the runtime itself
- Functions that are overridden acquire different scopes.
- **For function overriding, inheritance is a must**

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

class Base {
public:
    void display() {
        cout << "Display method in Base class" << endl;
    }
};

class Derived : public Base {
public:
    void display() {
        cout << "Display method in Derived class" << endl;
    }
};

int main() {
    Base baseObj;
    Derived derivedObj;

    baseObj.display();
    derivedObj.display();

    return 0;
}
```