

Polymorphism in C++

Polymorphism is a feature of [OOPs](#) that allows the object to behave differently in different conditions. In C++ we have two types of polymorphism:

- 1) Compile time Polymorphism – This is also known as static (or early) binding.
- 2) Runtime Polymorphism – This is also known as dynamic (or late) binding.

1) Compile time Polymorphism

Function overloading and Operator overloading are perfect example of Compile time polymorphism.

Compile time Polymorphism Example

In this example, we have two functions with same name but different number of arguments. Based on how many parameters we pass during function call determines which function is to be called, this is why it is considered as an example of polymorphism because in different conditions the output is different. Since, the call is determined during compile time that's why it is called compile time polymorphism.

```
#include <iostream>
using namespace std;
class Add {
public:
    int sum(int num1, int num2){
        return num1+num2;
    }
    int sum(int num1, int num2, int num3){
        return num1+num2+num3;
    }
};
int main() {
    Add obj;
    //This will call the first function
    cout<<"Output: "<<obj.sum(10, 20)<<endl;
    //This will call the second function
    cout<<"Output: "<<obj.sum(11, 22, 33);
    return 0;
}
```

Output:
Output: 30
Output: 66

2) Runtime Polymorphism

Function overriding is an example of Runtime polymorphism.

Function Overriding: When child class declares a method, which is already present in the parent class then this is called function overriding, here child class overrides the parent class.

In case of function overriding we have two definitions of the same function, one is parent class and one in child class. The call to the function is determined at **runtime** to decide which definition of the function is to be called, that's the reason it is called runtime polymorphism.

Example of Runtime Polymorphism

```
#include <iostream>
using namespace std;
class A {
public:
    void disp(){
        cout<<"Super Class Function"<<endl;
    }
};
class B: public A{
public:
    void disp(){
        cout<<"Sub Class Function";
    }
};
int main() {
    //Parent class object
    A obj;
    obj.disp();
    //Child class object
    B obj2;
    obj2.disp();
    return 0;
}
```

Output:

```
Super Class Function
Sub Class Function
```