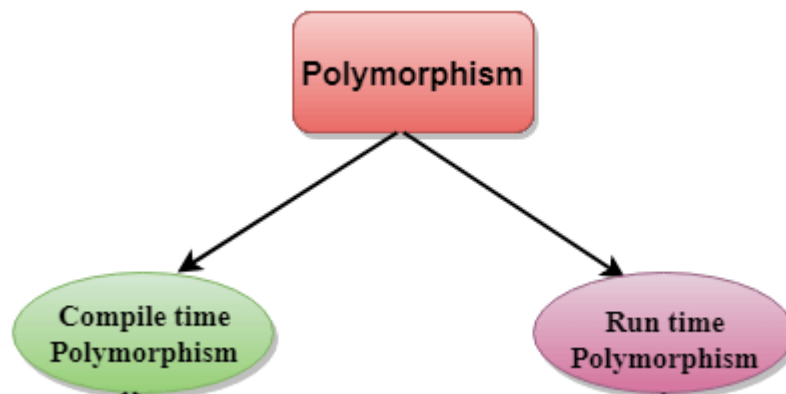


Polymorphism is a feature of OOPs that allows the object to behave differently in different conditions. polymorphism is done, by method overriding, when both super and sub class have member function with same declaration by different definition.

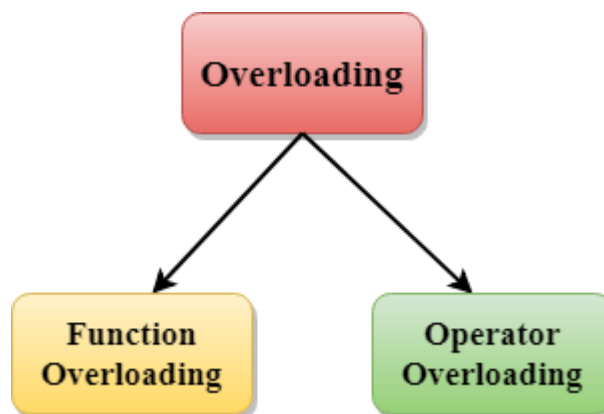
In C++ we have two types of polymorphism:

- 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



Function Overloading:

When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by **change in number of arguments** or/and **change in**

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 class Add {
4 public:
5     int sum(int num1, int num2){
6         return num1+num2;
7     }
8     int sum(int num1, int num2, int num3){
9         return num1+num2+num3;
10    }
11 };
12 int main() {
13     Add obj;
14     //This will call the first function
15     cout<<"Output: "<<obj.sum(10, 20)<<endl;
16     //This will call the second function
17     cout<<"Output: "<<obj.sum(11, 22, 33);
18     return 0;
19 }
```

input

```
Output: 30
Output: 66
```

Operator Overloading:

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type. Operator overloading is used to overload or redefines most of the operators available in C++. It is used to perform the operation on the user-defined data type. For example, C++ provides the ability to add the variables of the user-defined data type that is applied to the built-in data types.

The advantage of Operators overloading is to perform different operations on the same operand.

Operator that cannot be overloaded are as follows:

- Scope operator (::)
- Sizeof
- member selector(.)
- member pointer selector(*)
- ternary operator(?:)

Syntax of Operator Overloading

```
return_type class_name :: operator op(argument_list)
{
```

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>

```
// body of the function.  
}
```

Where the **return type** is the type of value returned by the function.

class_name is the name of the class.

operator op is an operator function where op is the operator being overloaded, and the operator is the keyword.

Rules for Operator Overloading

- Existing operators can only be overloaded, but the new operators cannot be overloaded.
- The overloaded operator contains atleast one operand of the user-defined data type.
- We cannot use friend function to overload certain operators. However, the member function can be used to overload those operators.
- When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.
- When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

C++ Operators Overloading Example

```
main.cpp  
1  #include <iostream>  
2  using namespace std;  
3  class Test  
4  {  
5      private:  
6          int num;  
7      public:  
8          Test(): num(8){}  
9          void operator ++() {  
10             num = num+2;  
11         }  
12         void Print() {  
13             cout<<"The Count is: "<<num;  
14         }  
15     };  
16     int main()  
17     {  
18         Test tt;  
19         ++tt; // calling of a function "void operator ++()"  
20         tt.Print();  
21         return 0;  
22     }
```

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>

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.

```
main.cpp
1  #include <iostream>
2  using namespace std;
3  class A {
4  public:
5      void disp(){
6          cout<<"Super Class Function"<<endl;
7      }
8  };
9  class B: public A{
10 public:
11     void disp(){
12         cout<<"Sub Class Function";
13     }
14 };
15 int main() {
16     //Parent class object
17     A obj;
18     obj.disp();
19     //Child class object
20     B obj2;
21     obj2.disp();
22     return 0;
23 }
```

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

How to call overridden function from the child class

As we have seen above that when we make the call to function (involved in overriding), the child class function (overriding function) gets called. What if you want to call the overridden function by using the object of child class. You can do that by creating the child class object in such a way that the reference of parent class points to it. Let's take an example to understand it.

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>


main.cpp

```
1 #include <iostream>
2 using namespace std;
3 class BaseClass {
4 public:
5     void disp(){
6         cout<<"Super Class Function"<<endl;
7     }
8 };
9 class DerivedClass: public BaseClass{
10 public:
11     void disp(){
12         cout<<"Sub Class Function";
13     }
14 };
15 int main() {
16     /* Reference of base class pointing to
17        * the object of child class.
18        */
19     BaseClass obj = DerivedClass();
20     obj.disp();
21     return 0;
22 }
23
24
25
```


Super Class Function

Way 2 calling

```
15 int main() {
16     /* Reference of base class pointing to
17        * the object of child class.
18        */
19     BaseClass obj = DerivedClass();
20     obj.disp();
21     DerivedClass obj2 = DerivedClass();
22     obj2.disp();
23 }
24
25
26
```


Super Class Function
Sub Class Function

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>