Introduction:

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.Real life example of polymorphism, a person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism.

Polymorphism is considered as one of the important features of Object Oriented Programming.

In C++ polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism

Compile time polymorphism: This type of polymorphism is achieved by function overloading or operator overloading.

- **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 type of arguments.

- **Operator Overloading**: C++ also provide option to overload operators. For example, we can make the operator (‘+’) for string class to concatenate two strings. We know that this is the addition operator whose task is to add two operands. So a single operator ‘+’ when placed between integer operands , adds them and when placed between string operands, concatenates them.

Runtime polymorphism: This type of polymorphism is achieved by Function Overriding.

- **Function Overriding**: Function overriding is a feature that allows us to have a same function in child class which is already present in the parent class. A child class inherits the data members and member functions of parent class, but when you want to override a functionality in the child class then you can use function overriding. It is like creating a new version of an old function, in the child class.

Function Overriding Example

To override a function you must have the same signature in child class. By signature it means the data type and sequence of parameters. Here we don't have any parameter in the parent function so we didn't use any parameter in the child function.

```
#include <iostream>
using namespace std;
class BaseClass {
public:
   void disp(){
      cout<<"Function of Parent Class";
   }
};
class DerivedClass: public BaseClass{
public:
   void disp() {
      cout<<"Function of Child Class";
   }
};
int main() {
   DerivedClass obj = DerivedClass();
   obj.disp();
   return 0;
}
```

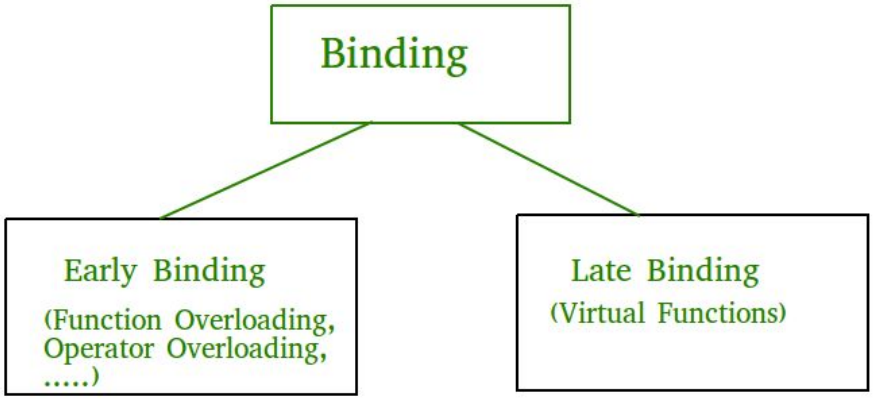
Output:

```
Function of Child Class
```

Note: In function overriding, the function in the parent class is called the overridden function and function in child class is called overriding function.

Early binding and Late binding in C++

The word **binding** means the mechanism which the compiler uses to decide which method should be executed on which call.



Early Binding:

In early binding, the compiler matches the function call with the correct function definition at compile time. It is also known as **Static Binding** or **Compile-time Binding**. By default, the compiler goes to the function definition which has been called during compile time. So, all the function calls we have studied till now are due to early binding.

Example:

```cpp
#include <iostream>

using namespace std;

class Animals
{
    public:
        void sound()
        {
            cout << "This is parent class" << endl;
        }
};

class Dogs : public Animals
{
    public:
        void sound()
        {
            cout << "Dogs bark" << endl;
        }
};

int main()
{
    Animals *a;
    Dogs d;
    a= &d;
    a -> sound();   //  early binding
    return 0;
}
```

Output:

This is parent class

Late Binding:

In the case of late binding, the compiler matches the function call with the correct function definition at runtime. It is also known as **Dynamic Binding** or **Runtime Binding**.

In late binding, the compiler identifies the type of object at runtime and then matches the function call with the correct function definition.This can be achieved by declaring a **virtual function**. Virtual Function is a member function of the base class which is overridden in the derived class. The compiler performs **late binding** on this function.

Example:

```
using namespace std;

class Animals
{
    public:
        virtual void sound()
        {
            cout << "This is parent class" << endl;
        }
};

class Dogs : public Animals
{
    public:
        void sound()
        {
            cout << "Dogs bark" << endl;
        }
};

int main()
{
    Animals *a;
    Dogs d;
    a= &d;
    a -> sound();
    return 0;
}
```

Output:

Dogs bark