Difference Between Compile Time And Run time Polymorphism In C++

Polymorphism means ability to take more than one form. Polymorphism is considered as one of the important features of Object Oriented Programming. In C++ polymorphism is mainly categorized into two types, Compile time polymorphism (Static) or (Dynamic) Runtime polymorphism. In dynamic polymorphism, the response to a function is determined at the run-time whereas in static polymorphism, the response to a function is determined at compile time.

Compile time polymorphism also referred to as **static polymorphism**, is achieved by functional overloading or operator overloading.

Functional 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 or type of arguments.

```
// Program to compute absolute value
// Works both for integer and float

#include <iostream>
using namespace std;

int absolute(int);
float absolute(float);

int main() {
     int a = -5;
     float b = 5.5;

     cout << "Absolute value of " << a << " = " << absolute(a) << endl;
     cout << "Absolute value of " << b << " = " << absolute(b);
     return 0;
}

int absolute(int var) {
       if (var < 0)
            var = -var;
     return var;
}

float absolute(float var){
     if (var < 0.0)
          var = -var;
     return var;
}
```

Operator Overloading

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

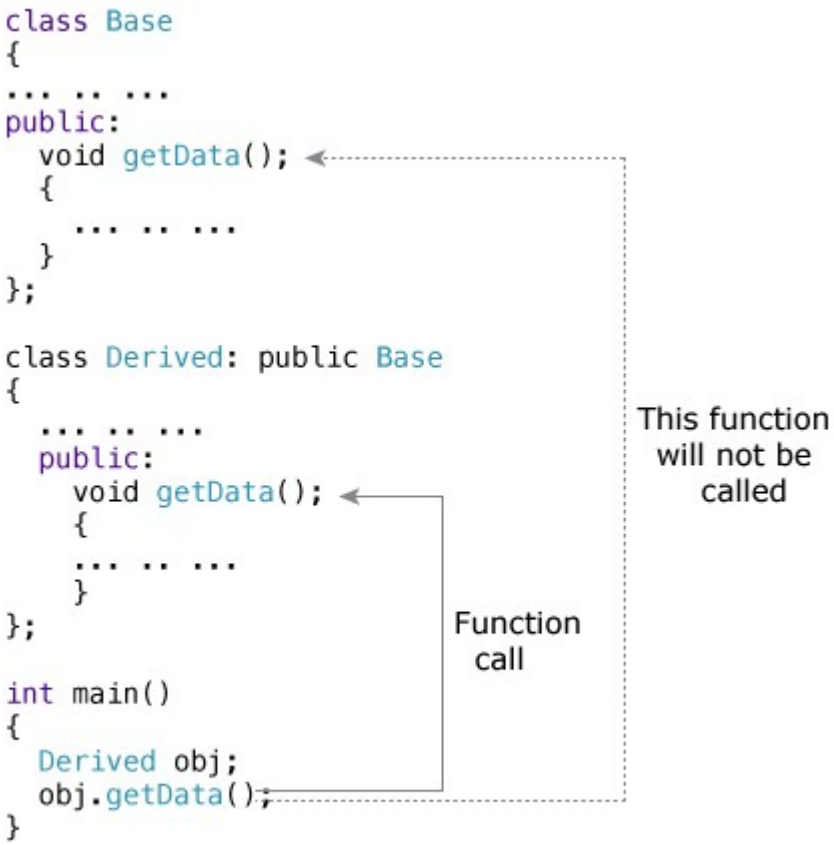
```
#include <iostream>
using namespace std;

class Test
{
    private:
        int count;

    public:
         Test(): count(5){}

         void operator ++()
         {
             count = count+1;
         }
         void Display() { cout<<"Count: "<<count; }
};

int main()
{
     Test t;
     // this calls "function void operator ++()" function
     ++t;
     t.Display();
     return 0;
}
```

What You Need To Know About Compile Time

- The binding of functional call and choosing the correct function declaration is done by compiler at the compile time.
- Compile time polymorphism is achieved by function overloading and operator overloading.
- The properties and behavior of compile time polymorphism are classic examples of static binding /static resolution.
- In compile time polymorphism, call is resolved by compiler.
- Overloading is compile time polymorphism where more than one methods share the same name with different parameters or signature and different return type.
- Compile time polymorphism is less flexible as all things execute at compile time.
- It provides fast execution because it is known early at compile time.

Runtime Polymorphism (Dynamic Polymorphism)

Runtime polymorphism also referred to as dynamic polymorphism is usually achieved by function overriding. Function overriding occurs when a derived class has a definition for one of the member functions of the base class. The base function is said to be **overridden.**



What You Need To Know About Run time Polymorphism

- The binding happens at run time.
- Run time polymorphism is achieved by virtual functions and pointers.
- The properties and behavior of run time polymorphism are classic examples dynamic binding and late binding.
- In run time polymorphism, call is not resolved by the compiler.
- Overriding is run time polymorphism having same method with same parameters or signature, but associated in a class & its subclass.
- Run time polymorphism is more flexible as all things execute at run time.
- It provides slow execution as compared to early binding because it is known at run time.

Difference Between Compile Time And Run Time Polymorphism In Tabular Form

BASIS OF COMPARISON	COMPILE TIME POLYMORPHISM	RUN TIME POLYMORPHISM
Binding	The binding of functional call and choosing the correct function declaration is done by compiler at the compile time.	The binding happens at run time.
How It Is Achieved	Compile time polymorphism is achieved by function overloading and operator overloading.	Run time polymorphism is achieved by virtual functions and pointers.
Properties And Behavior	The properties and behavior of compile time polymorphism are classic examples of static binding /static resolution.	The properties and behavior of run time polymorphism are classic examples dynamic binding and late binding.
Call	In compile time polymorphism, call is resolved by compiler.	In run time polymorphism, call is not resolved by the compiler.
Overriding	Overloading is compile time polymorphism where more than one methods share the same name with different parameters or signature and different return type.	Overriding is run time polymorphism having same method with same parameters or signature, but associated in a class & its subclass.
Flexibility	Compile time polymorphism is less flexible as all things execute at compile time.	Run time polymorphism is more flexible as all things execute at run time.
Execution Speed	It provides fast execution because it is known early at compile time.	It provides slow execution as compared to early binding because it is known at run time.