Unit 4 .Function Overloading

4.1 Concept of Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance. Polymorphism is considered one of the important features of Object-Oriented Programming.

Consider this example:

The " +" operator in c++ can perform two specific functions at two different scenarios i.e when the "+" operator is used in numbers, it performs addition.

```
int a = 6;
int b = 6;
int sum = a + b;  // sum =12
```

And the same "+" operator is used in the string, it performs concatenation.

```
string firstName = "Great ";
string lastName = "Learning";
// name = "Great Learning "
string name = firstName + lastName;
```

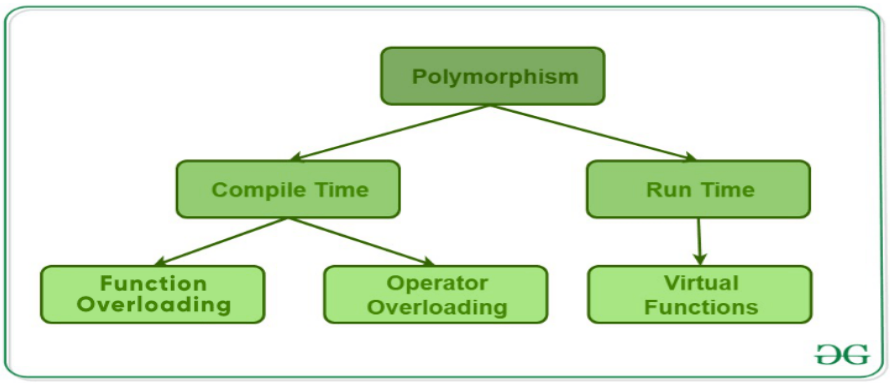
Ex.

```
#include <iostream>
using namespace std;
class Addition {
public:
   int ADD(int X,int Y)   // Function with parameter
   {
      return X+Y;     // this function is performing addition of  two Integer value
   }
   int ADD() {             // Function with same name but without parameter
      string a= "HELLO";
      string b="SAM";   // in this function concatenation is performed
     string c= a+b;
     cout<<c<<endl;
```

```
    }
};
int main(void) {
    Addition obj;   // Object is created
    cout<<obj.ADD(128, 15)<<endl; //first method is called
    obj.ADD();  // second method is called
    return 0;
}
```

- ## Types of Polymorphism –

- **Compile-time Polymorphism**
- **Runtime Polymorphism**



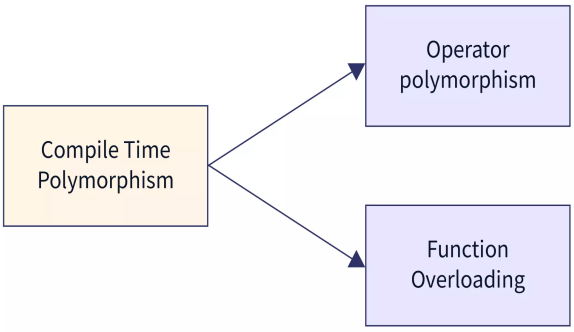
1. Compile-Time Polymorphism

Compile-time polymorphism is done by overloading an **operator or function.** It is also known as "static" or "early binding".

Why is it called compile-time polymorphism?

Overloaded functions are called by comparing the data types and number of parameters. This type of information is available to the compiler at the compile time. Thus, the suitable function to be called will be chosen by the C++ compiler at compilation time.

There are the following types of compile-time polymorphism in C++ :



- **Function Overloading**

When we have two functions with the same name but different parameters, different functions are called depending on the number and data types of parameters. This is known as function overloading

Function Overloading can be achieved through the following two cases:

- The names of the functions and return types are the same but differ in the type of arguments.
- The name of the functions and return types are the same, but they differ in the number of arguments.

Let's understand with an example:

```
#include <iostream.h>
using namespace std;
class Temp
{
   private:
      int x = 10;
      double x1 = 10.1;
   public:
      void add(int y)
      {
        cout << "Value of x + y is: " << x + y << endl;        }
      // Differ in the type of argument.
      void add(double d)
```

```
        {
          cout << "Value of x1 + d is: " << x1 + d << endl;
        }
        // Differ in the number of arguments.
        void add(int y, int z)
        {
          cout << "Value of x + y + z is: " << x + y + z << endl;
        }
};

int main() {
    Temp t1;
    t1.add(10);
    t1.add(11.1);
    t1.add(12,13);

    return 0;
}
```

Example 1: Function Overloading Using Different Types of Parameter

A function can be overloaded in different ways, one of them is by using different types of parameters. Let us understand the situation by having a look at the example:

```
#include <iostream>
using namespace std;
class printmethods {
  public:
     void print(int i) {
       cout << "integer value: " << i << endl;
     }
     void print(double f) {
       cout << "floating value: " << f << endl;
     }
     void print(char* c) {
       cout << "character value: " << c << endl;
```

```
    }
};
int main(void) {
  printmethods obj;
  obj.print(512);
  obj.print(90.75);
  obj.print("Podium");
  return 0;
}
```

Output:
integer value: 512
floating value: 90.75
character value: Podium

The above program has three methods with the same name (print) but with different execution behavior. It takes in different types of input of different types and prints them.

Example 2: Overloading Using Different Number of Parameters

A function can be overloaded in two different ways: one of them is by using a different number of parameters. Let us understand the situation by having a look at the example:

```
#include <iostream>
using namespace std;
class printmethods {
  public:
    void add(int i, int j) {
      cout << "integer sum: " << i+j << endl;
    }
    void add(double f) {
      cout << "floating value: " << f << endl;
    }
};

int main(void) {
```

```
  printmethods obj;
  obj.add(5,3);
  obj.add(90.75);
  return 0;
}
```

Output:
integer sum: 8
floating value: 90.75

The above program has two methods with the same name (add) but with different execution behavior and different set of parameters.

2. Runtime Polymorphism

This type of polymorphism is achieved by **Function Overriding**. Late binding and dynamic polymorphism are other names for runtime polymorphism. The function call is resolved at runtime in runtime polymorphism.

4.3 Scoping rules & features of function overloading.

- . **Rules of Function Overloading in C++**

There are certain rules to be followed while overloading a function in C++. Let us have a look at some of them:

1. The functions must have the same name
2. The functions must have different types of parameters.
3. The functions must have a different set of parameters.
4. The functions must have a different sequence of parameters.