

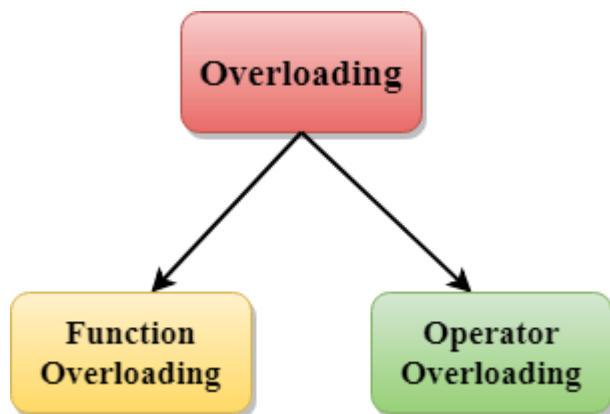
Lecture 17

C++ Overloading (Function and Operator)

If we create two or more members having the same name but different in number or type of parameter, it is known as C++ overloading.

Types of overloading in C++ are:

- Function overloading
- Operator overloading



C++ Function Overloading

Function Overloading is defined as the process of having two or more function with the same name, but different in parameters is known as function overloading in C++. In function overloading, the function is redefined by using either different types of arguments or a different number of arguments. It is only through these differences compiler can differentiate between the functions.

The **advantage** of Function overloading is that it increases the readability of the program because you don't need to use different names for the same action.

C++ Function Overloading Example

Let's see the simple example of function overloading where we are changing number of arguments of add() method.

// program of function overloading when number of arguments vary.

```
#include <iostream>
using namespace std;
class Cal {
public:
static int add(int a,int b){
    return a + b;
}
static int add(int a, int b, int c)
{
    return a + b + c;
}
};
int main(void) {
    Cal C; // class object declaration.
    cout<<C.add(10, 20)<<endl;
    cout<<C.add(12, 20, 23);
    return 0;
}
```

Output:

```
30
55
```

Let's see the simple example when the type of the arguments vary.

// Program of function overloading with different types of arguments.

```
#include<iostream>
using namespace std;
int mul(int,int);
float mul(float,int);

int mul(int a,int b)
{
    return a*b;
}
float mul(double x, int y)
{
    return x*y;
}
int main()
{
    int r1 = mul(6,7);
    float r2 = mul(0.2,3);
    std::cout << "r1 is : " <<r1<< std::endl;
    std::cout <<"r2 is : " <<r2<< std::endl;
    return 0;
}
```

Output:

```
r1 is : 42
r2 is : 0.6
```

C++ Operators 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)
{
    // 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.

C++ Operators Overloading Example

Let's see the simple example of operator overloading in C++. In this example,

void operator ++ () operator function is defined (inside Test class).

// program to overload the unary operator ++.

```
#include <iostream>
using namespace std;
class Test
{
    private:
        int num;
    public:
        Test(): num(8){}
        void operator ++()    {
            num = num+2;
        }
        void Print() {
            cout<<"The Count is: "<<num;
        }
};
int main()
{
    Test tt;
    ++tt; // calling of a function "void operator ++()"
    tt.Print();
    return 0;
}
```

Output:

```
The Count is: 10
```

Let's see a simple example of overloading the binary operators.
// program to overload the binary operators.

```
#include <iostream>
using namespace std;
class A
{
    int x;
    public:
    A(){}
    A(int i)
    {
        x=i;
    }
    void operator+(A);
    void display();
};

void A :: operator+(A a)
{
    int m = x+a.x;
    cout<<"The result of the addition of two objects is : "<<m;

}
int main()
{
    A a1(5);
    A a2(4);
    a1+a2;
    return 0;
}
```

Output:

```
The result of the addition of two objects is : 9
```