Operator Overloading in C++

Operator overloading is giving new functionality to an existing operator. It means the behavior of operators when applied to objects of a class can be redefined. It is similar to overloading functions except the function name is replaced by the keyword operator followed by the operator's symbol. There are 5 operators that are forbidden to overload. They are **:: . .* sizeof ?:**

In the following code fragment, we will overload binary + operator for Complex number class object.

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

class Complex
{
    private :
        double real;
        double imag;
    public:
        Complex () {};
        Complex (double, double);
        Complex operator + (Complex);
        void print();
};

Complex::Complex (double r, double i)
{
    real = r;
    imag = i;
}

Complex Complex::operator+ (Complex param)
{
    Complex temp;
    temp.real = real + param.real;
    temp.imag = imag + param.imag;
    return (temp);
}

void Complex::print()
{
    cout << real << " + i" << imag << endl;
}
```

```
int main ()
{
    Complex c1 (3.1, 1.5);
    Complex c2 (1.2, 2.2);
    Complex c3;

    c3 = c1 + c2; //use overloaded + operator

    c1.print();
    c2.print();
    c3.print();
    return 0;
}
```

Output :
3.1 + i1.5
1.2 + i2.2
4.3 + i3.7

In C++ we can cause an operator to invoke a member function by giving that member function a special name (of the form: operator<symbol>). Hence for the sum operation, the special name is: operator+. So, by naming the member function operator+ we can call the function by statement

```
c3 = c1 + c2
```

That is similiar to

```
c3 = c1.operator+(c2);
```