C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.

An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition implementation.

When you call an overloaded function or operator, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called overload resolution.

### Function overloading in C++:

You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You can not overload function declarations that differ only by return type.

Following is the example where same function `print` is being used to print different data types:

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

class printData {
public:
    void print(int i) {
        cout << "Printing int: " << i << endl;
    }

    void print(double f) {
        cout << "Printing float: " << f << endl;
    }

    void print(char* c) {
        cout << "Printing character: " << c << endl;
    }
};

int main(void) {
    printData pd;
    // Call print to print integer
    pd.print(5);
    // Call print to print float
    pd.print(500.263);
    // Call print to print character
    pd.print("Hello C++");
    return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Printing int: 5
Printing float: 500.263
Printing character: Hello C++
```

### Operators overloading in C++:

Operators can also be overloaded in C++ to perform different operations. The syntax for operator overloading is similar to function overloading.
You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

```
Box operator+(const Box&);
```

declares the addition operator that can be used to **add** two Box objects and returns final Box object. Most overloaded operators may be defined as ordinary non-member functions or as class member functions. In case we define above function as non-member function of a class then we would have to pass two arguments for each operand as follows:

```
Box operator+(const Box&, const Box&);
```

Following is the example to show the concept of operator overloading using a member function. Here an object is passed as an argument whose properties will be accessed using this object, the object which will call this operator can be accessed using **this** operator as explained below:

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

class Box {
    public:
        double getVolume(void) {
            return length * breadth * height;
        }
        void setLength( double len ) {
            length = len;
        }
        void setBreadth( double bre ) {
            breadth = bre;
        }
        void setHeight( double hei ) {
            height = hei;
        }
        // Overload + operator to add two Box objects.
        Box operator+(const Box& b) {
            Box box;
            box.length = this->length + b.length;
            box.breadth = this->breadth + b.breadth;
            box.height = this->height + b.height;
            return box;
        }
    private:
        double length;       // Length of a box
        double breadth;      // Breadth of a box
        double height;       // Height of a box
};

// Main function for the program
int main( ) {
    Box Box1;       // Declare Box1 of type Box
    Box Box2;       // Declare Box2 of type Box
    Box Box3;       // Declare Box3 of type Box
    double volume = 0.0;       // Store the volume of a box here
```
// box 1 specification
Box1.setLength(6.0);
Box1.setBreadth(7.0);
Box1.setHeight(5.0);

// box 2 specification
Box2.setLength(12.0);
Box2.setBreadth(13.0);
Box2.setHeight(10.0);

// volume of box 1
volume = Box1.getVolume();
cout << "Volume of Box1 : " << volume << endl;

// volume of box 2
volume = Box2.getVolume();
cout << "Volume of Box2 : " << volume << endl;

// Add two object as follows:
Box3 = Box1 + Box2;

// volume of box 3
volume = Box3.getVolume();
cout << "Volume of Box3 : " << volume << endl;

return 0;
}

When the above code is compiled and executed, it produces the following result:

Volume of Box1 : 210
Volume of Box2 : 1560
Volume of Box3 : 5400

Overloadable/Non-overloadable Operators:
Following is the list of operators which can be overloaded:

+   -   *   /   %   ^
&   |   ~   !   ,   =
<   >   <=  >=  ++  --
<<  >>  ==  !=  &&  ||
+=  -=  /=  %=  ^=  &=
|=  *=  <<=  >>=  []
-> -*   new   new []  delete  delete []

Following is the list of operators, which can not be overloaded:

::  .*  .  ?:  

Operator Overloading Examples:
Here are various operator overloading examples to help you in understanding the concept.
Unary operators overloading

Binary operators overloading

Relational operators overloading

Input/Output operators overloading

++ and -- operators overloading

Assignment operators overloading

Function call operator overloading

Subscripting [] operator overloading

Class member access operator -> overloading