**C++ Overloading (Method and Operator)**

An overloaded declaration is a declaration that is 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 have 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**.

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

* methods,
* constructors, and
* indexed properties

It is because these members have parameters only.

**Types of overloading in C++ are:**

* **Function overloading**
* **Operators overloading**

C++ Function Overloading

Having two or more function with same name but different in parameters, is known as function overloading in C++.

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

## C++ Operators Overloading

Operator overloading is used to overload or redefine most of the operators available in C++. It is used to perform operation on user define data type.

The advantage of Operators overloading is to perform different operations on the same operand.

|  |
| --- |
| **C++ Function Overloading Example** |
| #include<iostream>using namespace std;struct holybitch{public:static int add(int a, int b){int c = a + b;return c;}static int add(int a, int b, int c){int d = a + b + c;return d;}};int main(){holybitch hb;cout << "with two parameters : " << hb.add(10, 20) << endl;cout << "with three parameters : " << hb.add(10, 20, 30) << endl;return 0;}**/\*with two parameters : 30****with three parameters : 60\*/** |

## Operators Overloading in C++

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 over loading 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 −

|  |  |
| --- | --- |
| #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;  }  /\*Volume of Box1 : 210  Volume of Box2 : 1560  Volume of Box3 : 5400\*/ |

Overloadable/Non-overloadableOperators

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 −

|  |  |  |  |
| --- | --- | --- | --- |
| :: | .\* | . | ?: |

Other kind of overloadings :

<https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm>