Operator overload and Load overlap in C ++

C ++ allows you to define more than one definition for a function name or an operator in the scope, respectively called Load overloading (function overloading) and Load operator overloading in C ++. .

C ++ allows you to define more than one definition for a function name or an operator in the scope, respectively called **Load overloading (function overloading)** and **Load operator overloading** in C ++. .

A overloaded declaration is a declaration that has been declared with the same name as a previously declared declaration in the same scope, except that: both declarations have different parameters and different definitions. .

When you call a overloaded function or a overloaded operator, the compiler decides the most appropriate definition to use by comparing the parameter types you used to call the function or operator with the parameter types. defined in the definition. This process of choosing the best overloading function or overloading operator is called **overloading overload** .

Load overlay in C ++

You can have multiple definitions for the same function name in the same scope. The definition of the function must be different from each other in terms of type and / or number of parameters in the parameter list. You cannot overload function declarations but only in return types.

In the following example, the same **ham** function is used to print different data types:

```
#include using namespace  std ; class  inDuLieu  { public : void  hamIn ( int
```

Compile and run the above C ++ program to see the results:

Overload operator in C ++

You can redefine or overload most of the built-in operators in C ++. Therefore, a programmer can use operators with self-defined type (user-defined).

Operator overloading in C ++ functions with special names: Function name is the operator keyword followed by the symbol of the operator being defined. Like any other function, a overloaded operator has a return type and a parameter list.

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

Declare the + operator to add two Box objects and return the last Box object. Most overloading operators can be defined as: non-member functions or class member functions. In the above case, we define the function as a non-

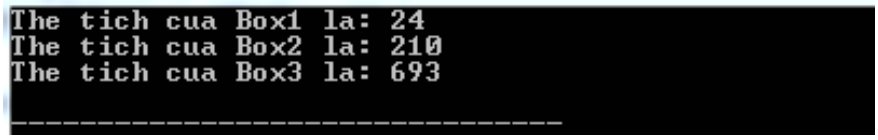
member of a class, then we have to pass two parameters for each operand, as follows:

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

The following example illustrates the concept of operator overloading using a member function. Here, an object is passed as a parameter whose properties will be accessed using this object, the object that will call this operator can be accessed using the **this** operator. as follows:

```
#include using namespace  std ; class Box { public : double  tinhTheTich ( void
```

Compiling and running the above C ++ program will produce the following results:



The operator can overload and cannot overload in C ++

The following table lists the operators that can be overloaded in C ++:

+ - * /% ^ & | ~! , = > => = ++ - >> ==! = && || + = - = / =% = ^ = & = | = * = = >> = [] () -> -> * new new [] delete delete []
Here is a list of operators that cannot be overloaded in C ++:

::. *. ?:

Example of operator overloading in C ++

Here are the various examples illustrating Operator overloading in C ++, thereby helping you understand this concept more deeply. Click on the link to see an example:

STATE DEBT AND Example1
Overload the unary operator in C ++

2
Overload binary operators in C ++

3
Overload relational operator in C ++

4
Load the Input / Output operator stack in C ++

5
Load operator ++ and - in C ++

6

Load the stack of assignment operators in C ++

7
Load the operator stack to call the function () in C ++

8
Overload the [] operator in C ++

9
Overload class member access operator -> in C ++

According to Tutorialspoint

Previous article: Calculating inheritance in C ++

Next lesson: Overloading a one-operator operator in C ++