Overloadable operators

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

To overload an operator in order to use it with classes we declare *operator functions*, which are regular functions whose names are the `operator` keyword followed by the operator sign that we want to overload. The format is:

```
type operator sign (parameters) { /*...*/ }
```

Although the prototype of a function `operator+` can seem obvious since it takes what is at the right side of the operator as the parameter for the operator member function of the object at its left side, other operators may not be so obvious. Here you have a table with a summary on how the different operator functions have to be declared (replace @ by the operator in each case):

Expression	**Operator**	**Member function**	**Global function**
@a (left)	+ - * & ! ~ ++ --	A::operator@()	operator@(A)
a@ (right)	++ --	A::operator@(int)	operator@(A,int)
a@b	+ - * / % ^ & | < > == != <= >= << >> && || ,	A::operator@ (B)	operator@(A,B)
a@b	= += -= *= /= %= ^= &= |= <<= >>= []	A::operator@ (B)	-
a(b, c...)	()	A::operator() (B, C...)	-
a->x	->	A::operator->()	-

Where `a` is an object of class `A`, `b` is an object of class `B` and `c` is an object of class `C`.

You can see in this panel that there are two ways to overload some class operators: as a member function and as a global function. Its use is indistinct, nevertheless I remind you that functions that are not members of a class cannot access the private or protected members of that class unless the global function is its friend (friendship is explained later).