Increment and Decrement

To add or subtract one from a variable use increment (`++`) and decrement (`--`) operators. These are **unary operators** that can **only** be applied to a variable (*`lvalue`*).

```
int a = 5, b = 10;
a++;        // a is changed to 6
--b;        // b is changed to 9
```

In addition to the side effect (changing the variable), expressions using these operators produce a value. When placed **before** a variable, it is called **pre-increment** (or decrement); when placed **after**, it is called a **post-increment** (or decrement) expression. The side effect is the same for both: the variable is left with a value one greater (or less) than it was before.

The **expression value** (result) produced depends on whether the expression uses post or pre-increment.

```
int a = 5, b = 10, c, d;
c = a++;        // a is changed to 6; c is assigned 5
d = --b;        // b is changed to 9 and so is d
```

With **pre-increment**, the variable is **first modified** and the **modified variable** is returned as the value. A prefix expression is thus an *`lvalue`*, so the expression `++++a` is legal.

With **post-increment**, the original value is saved to a **temporary** location. Then, the variable is changed. Finally, the temporary value is returned from the expression. That's why `c` in the example above is given the value `5` and not `6`. A postfix expression is an *`rvalue`*, so the expression `a++++` is **illegal**.

A Side-effect Pitfall

Don't ever use any side-effect operator twice on the same variable in the same expression. These expressions all result in **undefined behavior**, as you'll see if you run the code yourself in g++ , visual c++ or in clang++ .

```
int n = 6;
print(n, ++n);    // passing 6,7 or 7,7? Can't tell!
int a = n * n++;
n = n++;
cout << n++ << n++ << n++ << endl;
```