Incrementing and decrementing - it has its ups and downs ☺

C provides the programmer with many abbreviations for common programming statements. Sometimes these abbreviations are useful, and sometimes they are simply too complex to understand. When they are too complex, readability is reduced, simply because the programmer didn't want to do much typing. Most of these abbreviations do not cause the code to be any more efficient, they simply make the programmer’s job easier.

The reason I carry on about this is twofold. First of all, I love to type, and just thought I can get lots of practice with this class. ;-) Actually, first of all, when I introduce an abbreviation, it is because I have found it to be commonly used in industry. Second of all, I want to point out that if you choose never to use them, that is perfectly fine. However, you need to have some exposure to them, as you will see them along the way.

One of the most common abbreviations used is the one to increment or decrement a variable's value by 1. I use this one myself, as it is really nice, and I don't believe that it reduces the readability of the code.

Incrementing a variable's value by 1

To increment a variable's value by 1, you have 3 options. The first option below is the original "long hand" method. The second and third options are the abbreviated versions. Assume the following variable declaration, which declares variable **num** to be of type **int**, and initializes it to the value **25** during declaration. All of the options below increment variable **num** by **1**. So, assuming **num** starts off at **25** for each option, after the statements are executed, variable **num** is equal to **26**.

int num = 25;

Option 1: **num = num + 1;** /* increment variable num1 by 1. */

Option 1 above shows the long-hand version of an increment.
We have seen this in the **for** loops covered so far.

Option 2: **num++;** **/*** increment variable num by 1. ***/**

Option 2 above shows an abbreviated increment statement. You put the variable name followed by the increment operator **++** (2 plus signs immediate following one another, with no spaces between). This causes variable **num** to be incremented by 1. You can instead place the **++** sign *before* the variable as follows: **++num;** When the increment operator is placed *before* the variable, it is known as a **pre increment**. When the increment operator is placed *after* the variable, it is known as a **post increment**. The difference between pre and post increments will be discussed later. The increment operator can only be used to increment variables by 1. If you want to increment variables by anything other than 1 (2 for example), you need to either use the long version (option 1), or use option 3 below.

Option 3: **num += 1;** /* increment variable num by 1.*/

Option 3 above is another abbreviated form of the increment. You put the variable name followed by a space (not required), followed by a plus-sign then and equal sign, with no spaces between the plus and equal signs. Then you put the value that you want to add to the variable. In this case, we used the number **1**. So, we will be incrementing variable **num** by **1**. If we wanted to use this option to increment variable **num** by **2**, we could, simply by typing: **num += 2;**

So, in summary, regarding the increment, if you need to increment your variable's value by 1, you can pick any one of the three options above. If you want to increment your variable's value by anything other than 1, you can choose only options 1 and 3. Option 2 is solely reserved for incrementing by 1, and it is the most commonly used increment in for loops. (Recall, the 3rd segment of a for loop, "the increment or decrement".) Using the increment in option 2 a sample **for** loop statement would look like:

```
for (x = 1;  x <= 5;  x++)
                                  ^ notice the increment operator here.
```

<u>**Decrementing a variable's value by 1**</u>

Like incrementing, decrementing a variable's value by 1 can also be abbreviated. Again, you have 3 options. They follow the exact same format as the increment, with a slight modification. I think you will get the picture, even if I summarize below. All of the options below decrement variable **num** by **1**. So, assuming **num** starts off at **25** for each option, after the statements are executed, variable **num** is equal to **24.**

int num = 25;

Option 1: **num = num - 1;** /* long hand decrement of variable num1 by 1.*/

Option 2:   **num--;**   **/*** abbreviated decrement of variable num by 1. ***/**
/* could have been: **--num;** instead */

Option 3:   **num -= 1;**   /* abbreviated decrement of variable num by 1 */

Again, in summary, regarding the decrement, if you need to decrement your variable's value by 1, you can pick any one of the three options above. If you want to decrement your variable's value by anything other than 1, you can choose only options 1 and 3. Option 2 is solely reserved for decrementing by 1, and it is the most commonly used decrement in **for** loops. (Recall, the 3rd segment of a **for** loop, "the increment or decrement".) Using the decrement in option 2 a sample for loop statement would look like:

```
for (x = 5;  x >=1 5;  x--)
                                ^ notice the decrement operator here.
```

One final note regarding Option 3 above. You can use the abbreviation for other mathematical operations.

For example, if you wanted to *multiply* variable num by 3, you could use:

```
num = num * 3;          /* Option 1.*/
```

Or, you could use:

```
num *= 3;               /*  Option 3 */
```

If you want to *divide* variable num by 5, you could use:

```
num = num / 5;          /*  Option 1*/
```

Or you could use:

```
num /= 5;               /* Option 3.*/
```

That's about all there is regarding incrementing and decrementing. Next we learn to count. ☺