Chapter 52: Side Effects

Section 52.1: Pre/Post Increment/Decrement operators

In C, there are two unary operators - '++' and '--' that are very common source of confusion. The operator `++` is called the *increment operator* and the operator `--` is called the *decrement operator*. Both of them can be used used in either *prefix* form or *postfix* form. The syntax for prefix form for `++` operator is `++operand` and the syntax for postfix form is `operand++`. When used in the prefix form, the operand is incremented first by `1` and the resultant value of the operand is used in the evaluation of the expression. Consider the following example:

```
int n, x = 5;
n = ++x; /* x is incremented  by 1(x=6), and result is assigned to n(6) */
         /* this is a short form for two statements: */
         /* x = x + 1; */
         /* n = x ; */
```

When used in the postfix form, the operand's current value is used in the expression and then the value of the operand is incremented by 1. Consider the following example:

```
int n, x = 5;
n = x++; /* value of x(5) is assigned first to n(5), and then x is incremented by 1; x(6) */
         /* this is a short form for two statements: */
         /* n = x; */
         /* x = x + 1; */
```

The working of the decrement operator `--` can be understood similarly.

The following code demonstrates what each one does

```
    int main()
    {
        int a, b, x = 42;
        a = ++x; /* a and x are 43 */
        b = x++; /* b is 43, x is 44 */
        a = x--; /* a is is 44, x is 43 */
        b = --x; /* b and x are 42 */

        return 0;
    }
```

From the above it is clear that post operators return the current value of a variable and *then* modify it, but pre operators modify the variable and *then* return the modified value.

In all versions of C, the order of evaluation of pre and post operators are not defined, hence the following code can return unexpected outputs:

```
    int main()
    {
        int a, x = 42;
        a = x++ + x; /* wrong */
        a = x + x; /* right */
        ++x;

        int ar[10];
        x = 0;
        ar[x] = x++; /* wrong */
```

```
    ar[x++] = x; /* wrong */
    ar[x] = x; /* right */
    ++x;
    return 0;
}
```

Note that it is also good practice to use pre over post operators when used alone in a statement. Look at the above code for this.

Note also, that when a function is called, all side effects on arguments must take place before the function runs.

```
int foo(int x)
{
    return x;
}

int main()
{
    int a = 42;
    int b = foo(a++);  /* This returns 43, even if it seems like it should return 42 */
    return 0;
}
```