Tutorial Link https://codequotient.com/tutorials/C - Arithmetic Operators/5a1d964152795c1b16c0abef

TUTORIAL

C - Arithmetic Operators

Chapter

1. C - Arithmetic Operators

Lot of operators are defined in C language. Basically, operators are divided into four sections: arithmetic, relational, logical, bitwise. There are other special operators also like assignment operator.

Arithmetic Operators: -

Following are the arithmetic operators in C: -

```
Operator   Description
+          Addition
-          Subtraction
*          Multiplication
/          Division
%          Modulus
--         Decrement
++         Increment
```

+ and – comes in to variants as they can be used as unary operators and binary operators both. In case of unary - acts as returning the negative value, which is used to represent negative values for example,

```
int b = 5;
```

```
int a = -b;   // a will be -5 now.
```

+, -, *, / will act in usual behavior, returning addition, subtraction, multiplication and division (quotient) of two operands. % (modulus) returns the division (remainder) instead quotient. For example,

```c
#include <stdio.h>

int main()
{
  int a,b,c,d;
  a=5;
  b=2;
  c=1;
  d=2;
  printf("a=%d b=%d c=%d d=%d \n\n",a,b,c,d);
  printf("a+b=%d\n", a + b);
  printf("a-b=%d\n", a - b);
  printf("a*b=%d\n", a * b);
  printf("a/b=%d\n", a / b);
  printf("a%%b=%d\n\n", a % b);

  printf("c+d=%d\n", c + d);
  printf("c-d=%d\n", c - d);
  printf("c*d=%d\n", c * d);
  printf("c/d=%d\n", c / d);
  printf("c%%d=%d\n", c % d);

  return 0;
}

```

++ and – are two operators which are required for quick addition and subtraction. If we need unit increment or decrement, we can use these operators. For example,

```
int x=10;
x = x + 1;   // x will be 11 now onwards.

int y=10;
y++;      // y will be 11 now onwards.

int a = 5;
a--;      // a will be 4 now onwards.

int b =5;
b = b - 1;   // b will be 4 now onwards.
```

So, a++ is same as a = a + 1 and a-- is same as a = a – 1.

Both increment and decrement operators can either be prefix to operand or suffix to operand, in those case, there are called as,

```
a++;   // post increment
++a;   // pre increment
a--;   // post decrement
--a;   // pre decrement
```

They are logically do the same thing as stated above, but when they used in expressions they behave differently. For example,

```c
int a=5;
int b=a++;

```

These two lines can be written as below,

```c
int a=5;
int b=a;        // returns the value first.
a = a + 1;    // After returning, do the increment.

```

Below is its counterpart called pre increment.

```c
int a =5;
int b = ++a;

```

This can be rewritten as below: -

```c
int a = 5, b;
a = a + 1;    // Do the increment first.
b = a;        // After increment return the value.

```

Obviously b will get different values in both cases. If post-increment case, b will use the old value of a (i.e. 5), and in pre-increment case b gets the new value of a (i.e. 6). But in both cases a will be 6 after execution. So both affects a in same manner (increment by 1), but they affect the expression in which they are written. Same is the effect of post-decrement and pre-decrement operators.

In C, when we try to mix these operators too much in a single expression, that expression may yield different output on different compilers. For example,

```
int a =10, b = 15;
int c = a++ + --a + --b - b-- - ++a;   // it affects one variable
multiple times in same expression.
```

This kind of statement has no specified order of evaluation for its execution in standard C. So depending on different implementations of C (different compiler of C) we may get different outputs. So in general, use them till they helps you, and it's easy to interpret. Otherwise the potability of your code may lose.