

# Operator In C Programming:

An Operators is used to perform an operation on Operand of any type. An Operand is the input values which are used Operator to create an resultant output.

**A + B = C**

A -> Operand  
B -> Operand  
C -> Resultant Output  
+ -> Arithmetic Operator  
= -> Assignment Operator

## Types Of Operator In C Programming:

### Operators In C Language

#### Unary Operator

Unary Positive  
Unary Negative  
Increment  
    Pre Increment  
    Post Increment  
Decrement  
    Pre Decrement  
    Post Decrement  
Logical Not

#### Binary Operator

Arithmetic (+ - \* / %)  
Relational (< > == >= <= !=)  
Logical (&& || !)  
Bitwise (& | ~ ^ >> << >>> <<< >>>>)  
Assignment (= += -= \*= /= %=)

#### Ternary Operator

Conditional  
(exp1)? exp2: exp3;

#### Special Operator

Sizeof()

## Unary Operator:

### Unary Positive(+) & Unary Negative(-):

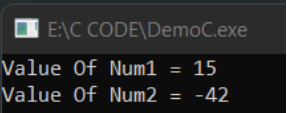
Unary Positive means we give a positive value. For positive value we need to give (+) sign before the number. Generally in C language if we give only the number by default the number is taken as positive number. Unary Negative means we give a negative value. For negative value we need to give (-) sign before the number.

```
#include <stdio.h>
#include <conio.h>

int main()
{
    int num1 = +15; // unary positive
    int num2 = -42; // unary negative

    printf("Value Of Num1 = %d \n", num1);
    printf("Value Of Num2 = %d", num2);

    getch();
}
```



E:\C CODE\DemoC.exe  
Value Of Num1 = 15  
Value Of Num2 = -42

## Increment Operator (++):

Increment operator is used for to increase the value by 1. Increment Operator is two types Pre Increment(++x) & Post Increment(x++).

Pre Increment(++x) means first perform the increment operation then any other operation. Post Increment(x++) means first perform other operation then perform the increment operation.

```
#include <stdio.h>
#include <conio.h>

int main()
{
    int num1 = 50;
    printf("Value of num1 = %d\n", num1);
    printf("Pre increment value of num1 = %d\n", ++num1);
    printf("new value of num1 = %d\n", num1);

    printf("-----\n");

    int num2 = 60;
    printf("Value of num2 = %d\n", num2);
    printf("post increment value of num2 = %d\n", num2++);
    printf("new value of num2 = %d\n", num2);
    getch();
}
```

E:\C CODE\DemoC.exe  
Value of num1 = 50  
Pre increment value of num1 = 51  
new value of num1 = 51  
-----  
Value of num2 = 60  
post increment value of num2 = 60  
new value of num2 = 61

## Decrement Operator:

Decrement operator is used for to decrease the value by 1. Decrement Operator is two types. Pre Decrement(--x) & Post Decrement(x--).

Pre Decrement(--x) means first perform the decrement operation then any other operation. Post Decrement(x--) means first perform other operation then perform the decrement operation.

```
#include <stdio.h>
#include <conio.h>

int main()
{
    int num1 = 50;
    printf("Value of num1 = %d\n", num1);
    printf("Pre decrement value of num1 = %d\n", --num1);
    printf("new value of num1 = %d\n", num1);

    printf("-----\n");

    int num2 = 60;
    printf("Value of num2 = %d\n", num2);
    printf("post decrement value of num2 = %d\n", num2--);
    printf("new value of num2 = %d\n", num2);
    getch();
}
```

E:\C CODE\DemoC.exe  
Value of num1 = 50  
Pre decrement value of num1 = 49  
new value of num1 = 49  
-----  
Value of num2 = 60  
post decrement value of num2 = 60  
new value of num2 = 59

# Binary Operator:

## Arithmetic Operator:

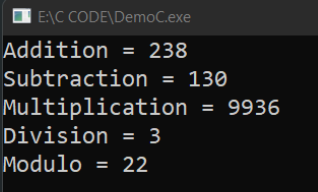
Arithmetic Operator perform the basic mathematical arithmetic operation between two operand. The followings are used in arithmetic operator Addition(+), Subtraction(-), Multiplication(\*), Division(/), Modulo(%).

```
#include <stdio.h>
#include <conio.h>

int main()
{
    int num1 = 184, num2 = 54;

    printf("Addition = %d\n", (num1 + num2));
    printf("Subtraction = %d\n", (num1 - num2));
    printf("Multiplication = %d\n", (num1 * num2));
    printf("Division = %d\n", (num1 / num2));
    printf("Modulo = %d\n", (num1 % num2));

    getch();
}
```



## Relational Operator:

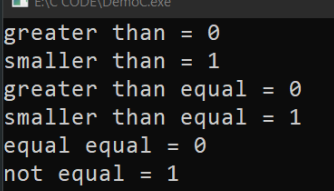
Relational Operator perform relation operation between two operand and represent the resultant output relation between two operand. The followings are used in Relational Operator greater than(>), smaller than(<), greater than equal(>=), smaller than equal(<=), equal equal(==), not equal(!=).

```
#include <stdio.h>
#include <conio.h>

int main()
{
    int num1 = 184, num2 = 254;

    printf("greater than = %d\n", (num1 > num2));
    printf("smaller than = %d\n", (num1 < num2));
    printf("greater than equal = %d\n", (num1 >= num2));
    printf("smaller than equal = %d\n", (num1 <= num2));
    printf("equal equal = %d\n", (num1 == num2));
    printf("not equal = %d\n", (num1 != num2));

    getch();
}
```



Here in output it represent the output as "1" or "0". It means if "1" it means true, if "0" it means false.

# Logical Operator:

Logical Operator is also known as short-circuit operator. In this operator there are two operators used short-circuit And (&&), short-circuit Or(||). These operators are used for checking expressions and conditions. In Logical Operator there is a special property. The special property is if they can determine all they need to know by evaluating the left operand, they won't evaluate the right operand.

```
#include <stdio.h>
#include <conio.h>
#include <stdbool.h>

int main()
{
    _Bool T = true;
    _Bool F = false;

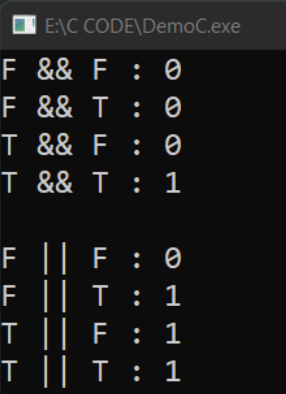
    printf("F && F : %d \n", (F && F));
    printf("F && T : %d \n", (F && T));
    printf("T && F : %d \n", (T && F));
    printf("T && T : %d \n", (T && T));

    printf("\n");

    bool t = true;
    bool f = false;

    printf("F || F : %d \n", (F || F));
    printf("F || T : %d \n", (F || T));
    printf("T || F : %d \n", (T || F));
    printf("T || T : %d \n", (T || T));

    getch();
}
```

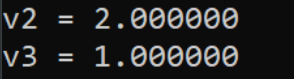


```
#include <stdio.h>
#include <conio.h>
#include <stdbool.h>
int main()
{
    double v1 = 0, v2 = 1, v3 = 1;

    if (v1 != 0 & (v2 = 2) == 1)
    {
    }
    printf("v2 = %lf \n", v2);

    if (v1 != 0 && (v3 = 2) == 1)
    {
    }
    printf("v3 = %lf \n", v3);

    getch();
}
```



## Bit Wise Operator:

Bit Wise Operator is type of operator which perform the operation bit by bit. The bit wise operator is used in both numerical values and boolean values. For numerical values we used Bit Wise Not( $\sim$ ), Bit Wise Or( $|$ ), Bit Wise And( $\&$ ), Bit Wise Xor( $\wedge$ ). For boolean values we use Bit Wise Logical Not( $!$ ), Bit Wise Logical Or( $|$ ), Bit Wise Logical And( $\&$ ), Bit Wise Logical Xor( $\wedge$ ).

Not Operation  True : 1 False : 0  !True : 0 !False : 1	Or Operation	And Operation	XOR Operation
	f   f : 0	f & f : 0	f ^ f : 0
	f   t : 1	f & t : 0	f ^ t : 1
	t   f : 1	t & f : 0	t ^ f : 1
	t   t : 1	t & t : 1	t ^ t : 0
	Nor Operation	Nand Operation	X-NOR Operation
	!(f   f) : 1	!(f & f) : 1	!(f ^ f) : 1
	!(f   t) : 0	!(f & t) : 1	!(f ^ t) : 0
	!(t   f) : 0	!(t & f) : 1	!(t ^ f) : 0
	!(t   t) : 0	!(t & t) : 0	!(t ^ t) : 1

num1 : 157 num2 : 203  Not Operator  ~num1 : -158 ~num2 : -204  And and Nand Operator  num1 & num2 : 137 ~(num1 & num2) : -138	Or and Nor Operation  num1   num2 : 223 ~(num1   num2) : -224  XOR and X-NOR Operation  num1 ^ num2 : 86 ~(num1 ^ num2) : -87
---	---

## Shift Operator:

Shift operator means each bit will shift either right or left by one step. There three types of shift operator is there "Right Shift", "Left Shift", "Unsigned Right shift".

In Right Shift Operator All bits are shifted to their right and the leftmost blank space should be filled up by MSB. Whenever we perform right shift divide the value by 2. (except where 1 goes out of boundary).

In Left Shift Operator All bits are shifted to their left and the rightmost blank space should be filled up by 0. Whenever we perform left shift multiply the value by 2. (except where 1 goes out of boundary).

In Unsigned Right Shift Operator All bits are shifted to their right and the leftmost blank space should be filled up by 0. Whenever we perform right shift divide the value by 2. (except where 1 goes out of boundary).

```
num1 : 157

Left Shift Operator
num1 << 5 : 5024

Right Shift Operator
num1 >> 5 : 4
```

## Assignment Operator:

Assignment Operator is the type of operator which is used for assigning a value or statement to a variable. These are the following procedure to use assignment operators. (=, +=, -=, \*=, /=, %=, ....., etc).

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int num1 = 10;

    printf(" num1 : %d \n", num1);
    printf(" Addition: %d \n", num1 += 15);
    printf(" Subtraction : %d \n", num1 -= 5);
    printf(" Multiplication : %d \n", num1 *= 135);
    printf(" Division : %d \n", num1 /= 20);
    printf(" Modulo : %d \n", num1 %= 10);
    printf(" num1 : %d \n", num1);

    getch();
}
```

E:\C CODE\DemoC.exe

```
num1 : 10
Addition: 25
Subtraction : 20
Multiplication : 2700
Division : 135
Modulo : 5
num1 : 5
```


## Ternary Operator:

In ternary operator we use three operand for perform operation. This is a conditional type operation where in first operand is for relational operation and other two operands are for giving output of resultant operation. The writing procedure of ternary conditional operator is as follows [(Expression1)? Expression2: Expression3;]

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int num1 = 112;
    int num2 = 118;

    (num1 > num2) ? printf(" %d is big", num1) : printf(" %d is big", num2);

    getch();
}
```

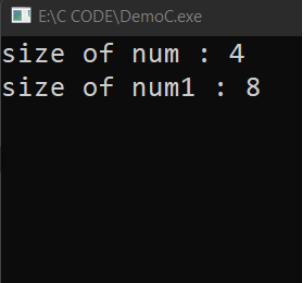


## sizeof() Operator:

Sizeof() operator is a special operator which is used for to find the size of the variable in bytes.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int num;
    printf("size of num : %d\n", sizeof(num));
    double num1;
    printf("size of num1 : %d\n", sizeof(num1));

    getch();
}
```



## Typedef Operator:

The typedef is a keyword used in C programming to provide some meaningful names to the already existing variable in the C program. It behaves similarly as we define the alias for the commands. In short, we can say that this keyword is used to redefine the name of an already existing variable. typedef is used for to create synonyms for data types

```
#include <stdio.h>

int main()
{
    typedef int abc;

    abc i = 10;
    abc j = 50;

    printf("i = %d\n", i);
    printf("j = %d", j);
}
```

