

C Programming Operators

An operator is a symbol that operates on a value or a variable. For example: + is an operator to perform addition.

C has a wide range of operators to perform various operations.

C Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Example 1: Arithmetic Operators

```
// Working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;
```

```

c = a+b;
printf("a+b = %d \n",c);
c = a-b;
printf("a-b = %d \n",c);
c = a*b;
printf("a*b = %d \n",c);
c = a/b;
printf("a/b = %d \n",c);
c = a%b;
printf("Remainder when a divided by b = %d \n",c);

return 0;
}

```

Output

```

a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1

```

The operators +, - and * computes addition, subtraction, and multiplication respectively as you might have expected.

In normal calculation, $9/4 = 2.25$. However, the output is 2 in the program.

It is because both the variables a and b are integers. Hence, the output is also an integer. The compiler neglects the term after the decimal point and shows answer 2 instead of 2.25.

The modulo operator % computes the remainder. When a=9 is divided by b=4, the remainder is 1. The % operator can only be used with integers.

Suppose a = 5.0, b = 2.0, c = 5 and d = 2. Then in C programming,

```

// Either one of the operands is a floating-point number

a/b = 2.5

a/d = 2.5

c/b = 2.5

```

```
// Both operands are integers
```

```
c/d = 2
```

C Increment and Decrement Operators

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

Example 2: Increment and Decrement Operators

```
// Working of increment and decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;

    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);

    return 0;
}
```

Output

```
++a = 11
--b = 99
```

```
++c = 11.500000
--d = 99.500000
```

Here, the operators ++ and -- are used as prefixes. These two operators can also be used as postfixes like a++ and a--. Visit this page to learn more about how [increment and decrement operators work when used as postfix](#).

C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Example 3: Assignment Operators

```
// Working of assignment operators
#include <stdio.h>
int main()
{
    int a = 5, c;

    c = a;    // c is 5
```

```
printf("c = %d\n", c);
c += a; // c is 10
printf("c = %d\n", c);
c -= a; // c is 5
printf("c = %d\n", c);
c *= a; // c is 25
printf("c = %d\n", c);
c /= a; // c is 5
printf("c = %d\n", c);
c %= a; // c = 0
printf("c = %d\n", c);

return 0;
}
```

Output

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

C Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in [decision making](#) and [loops](#).

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1

Operator	Meaning of Operator	Example
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

Example 4: Relational Operators

```
// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

Output

```
5 == 5 is 1
5 == 10 is 0
5 > 5 is 0
5 > 10 is 0
5 < 5 is 0
```

```
5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
5 >= 5 is 1
5 >= 10 is 0
5 <= 5 is 1
5 <= 10 is 1
```

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in [decision making in C programming](#).

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

Example 5: Logical Operators

```
// Working of logical operators

#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);
}
```

```

result = (a == b) && (c < b);
printf("(a == b) && (c < b) is %d \n", result);

result = (a == b) || (c < b);
printf("(a == b) || (c < b) is %d \n", result);

result = (a != b) || (c < b);
printf("(a != b) || (c < b) is %d \n", result);

result = !(a != b);
printf("!(a != b) is %d \n", result);

result = !(a == b);
printf("!(a == b) is %d \n", result);

return 0;
}

```

Output

```

(a == b) && (c > b) is 1
(a == b) && (c < b) is 0
(a == b) || (c < b) is 1
(a != b) || (c < b) is 0
!(a != b) is 1
!(a == b) is 0

```

Explanation of logical operator program

- $(a == b) \ \&\& \ (c > 5)$ evaluates to 1 because both operands $(a == b)$ and $(c > 5)$ is 1 (true).
- $(a == b) \ \&\& \ (c < b)$ evaluates to 0 because operand $(c < b)$ is 0 (false).
- $(a == b) \ || \ (c < b)$ evaluates to 1 because $(a == b)$ is 1 (true).
- $(a != b) \ || \ (c < b)$ evaluates to 0 because both operand $(a != b)$ and $(c < b)$ are 0 (false).
- $!(a != b)$ evaluates to 1 because operand $(a != b)$ is 0 (false). Hence, $!(a != b)$ is 1 (true).
- $!(a == b)$ evaluates to 0 because $(a == b)$ is 1 (true). Hence, $!(a == b)$ is 0 (false).

C Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Visit [bitwise operator in C](#) to learn more.

Other Operators

Comma Operator

Comma operators are used to link related expressions together. For example:

```
int a, c = 5, d;
```

The sizeof operator

The `sizeof` is a unary operator that returns the size of data (constants, variables, array, structure, etc).

Example 6: sizeof Operator

```
#include <stdio.h>
int main()
{
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=%lu bytes\n",sizeof(a));
    printf("Size of float=%lu bytes\n",sizeof(b));
    printf("Size of double=%lu bytes\n",sizeof(c));
    printf("Size of char=%lu byte\n",sizeof(d));

    return 0;
}
```

Output

```
Size of int = 4 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte
```