

# C - Operators

---

[Previous Page](#)

[Next Page](#)

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

We will, in this chapter, look into the way each operator works.

## Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then –

[Show Examples](#)

Operator	Description	Example
+	Adds two operands.	$A + B = 30$
-	Subtracts second operand from the first.	$A - B = -10$
*	Multiplies both operands.	$A * B = 200$
/	Divides numerator by de-numerator.	$B / A = 2$
%	Modulus Operator and remainder of after an integer division.	$B \% A = 0$
++	Increment operator increases the integer value by one.	$A++ = 11$

--	Decrement operator decreases the integer value by one.	A-- = 9
----	--	---------

## Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then –

[Show Examples](#)

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

## Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then –

[Show Examples](#)

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A    B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

## Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows –

p	q	p & q	p   q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows –

A = 0011 1100

B = 0000 1101

-----

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

[Show Examples](#)

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	(A   B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e., - 0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

## Assignment Operators

The following table lists the assignment operators supported by the C language –

[Show Examples](#)

Operator	Description	Example
----------	-------------	---------

=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same

		as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
=	Bitwise inclusive OR and assignment operator.	C  = 2 is same as C = C   2

## Misc Operators ↪ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Show Examples

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y