

OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. 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
- Increment and decrement operators
- Conditional operators
- Misc Operators

Arithmetic operator:

These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus.

Following table shows all the arithmetic operators supported by C language. Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A – B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2

%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increments operator increases integer value by one	A++ will give 11
--	Decrements operator decreases integer value by one	A--will give 9

Relational Operators:

These operators are used to compare the value of two variables.

Following table shows all the relational operators supported by C language. Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then 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 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 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 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 condition becomes true.	(A <= B) is true.

Logical Operators:

These operators are used to perform logical operations on the given two variables.

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

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

Bitwise Operators

Bitwise operator works on bits and performs bit-by-bit operation. Bitwise operators are used in bit level programming. These operators can operate upon *int* and *char* but not on *float* and *double*.

Showbits() function can be used to display the binary representation of any integer or character value.

Bit wise operators in C language are; & (bitwise AND), | (bitwise OR), ~ (bitwise OR), ^ (XOR), << (left shift) and >> (right shift).

The truth tables for &, |, and ^ are as follows:

<i>p</i>	<i>q</i>	<i>p & q</i>	<i>p q</i>	<i>p ^ q</i>
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

The Bitwise operators supported by C language are explained in the following table. Assume variable A holds 60 (00111100) and variable B holds 13 (00001101), then:

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

Assignment Operators:

In C programs, values for the variables are assigned using assignment operators.

There are following assignment operators supported by C language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A

<code>-=</code>	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	$C -= A$ is equivalent to $C = C - A$
<code>*=</code>	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	$C *= A$ is equivalent to $C = C * A$
<code>/=</code>	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	$C /= A$ is equivalent to $C = C / A$
<code>%=</code>	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	$C \% = A$ is equivalent to $C = C \% A$
<code><<=</code>	Left shift AND assignment operator	$C <<= 2$ is same as $C = C << 2$
<code>>>=</code>	Right shift AND assignment operator	$C >>= 2$ is same as $C = C >> 2$
<code>&=</code>	Bitwise AND assignment operator	$C \&= 2$ is same as $C = C \& 2$
<code>^=</code>	bitwise exclusive OR and assignment operator	$C \wedge= 2$ is same as $C = C \wedge 2$
<code> =</code>	bitwise inclusive OR and assignment operator	$C = 2$ is same as $C = C 2$