Operators in C Language

Operators in c language are the symbols that tells the compiler to perform specific **mathematical or logical task**.

C language provides the following types of operators:

1. **Arithmetic Operators**
2. **Relational Operators**
3. **Logical Operators**
4. **Bitwise Operators**
5. **Assignment Operators**
6. **Misc Operators**

Arithmetic Operators:

The following table shows all the arithmetic operators supported by the C language. Assume variable ***var1*** holds **10** and variable ***var2*** holds **20** then:

Operators	Description	Example
+	Adds two operands.	*var1* + *var2* = 30
-	Subtracts second operand from the first.	*var1* - *var2* = -10
*	Multiplies both operands.	*var1* * *var2* = 200
/	Divides numerator by de-numerator.	*var2* / *var1* = 2
%	Modulus Operator and remainder of after an integer division.	*var2* % *var1* = 0
++	Increment operator increases the integer value by one.	*var1* ++ = 11
--	Decrement operator decreases the integer value by one.	*var1* -- = 9

Relational Operators:

The following table shows all the relational operators in C. Assume variable ***var1*** holds ***10*** and variable ***var2*** holds ***20*** then:

Operators	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(***var1*** == ***var2***) 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.	(***var1*** != ***var2***) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(***var1*** > ***var2***) 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.	(***var1*** < ***var2***) 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.	(***var1*** >= ***var2***) 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.	(***var1*** <= ***var2***) is true.

Logical Operators:

Following table shows all the logical operators in C language. Assume variable *var1* holds *1* and variable *var2* holds *0*, then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(***var1*** && ***var2***) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(***var1*** || ***var2***) 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.	!(***var1*** && ***var2***) 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 *var1* = *60* and *var2* = *13* in binary format, they will be as follows:

var1 = 0011 1100

var1 = 0000 1101

----

var1 & *var2* = 0000 1100

var1 | *var2* = 0011 1101

var1 ^ *var2* = 0011 0001

~ *var1* = 1100 0011

The following table lists the bitwise operators in C. Assume variable '*var1*' holds *60* and variable ' *var2*' holds *13*, then:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(*var1*& B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(*var1* | *var2*) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(*var1* ^ *var2*) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~*var1*) = ~(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.	*var1* << 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.	*var1* >> 2 = 15 i.e., 0000 1111

Assignment Operators:

The following table lists the assignment operators in the C language:

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	***var3*** = ***var1*** + ***var2*** will assign the value of ***var1*** + ***var2*** to ***var3***
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	***var3*** += ***var1*** is equivalent to ***var3*** = ***var3*** + ***var1***
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	***var3*** -= ***var1*** is equivalent to ***var3*** = ***var3*** - ***var1***
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	***var3*** *= ***var1*** is equivalent to ***var3*** = ***var3*** * ***var1***
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	***var3*** /= ***var1*** is equivalent to ***var3*** = ***var3*** / ***var1***
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	***var3*** %= ***var1*** is equivalent to ***var3*** = ***var3*** % ***var1***
<<=	Left shift AND assignment operator.	***var3*** <<= 2 is same as ***var3*** = ***var3*** << 2

>>=	Right shift AND assignment operator.	***var3*** >>= 2 is same as ***var3*** = ***var3*** >> 2
&=	Bitwise AND assignment operator.	***var3*** &= 2 is same as ***var3*** = ***var3*** & 2
^=	Bitwise exclusive OR and assignment operator.	***var3*** ^= 2 is same as ***var3*** = ***var3*** ^ 2
|=	Bitwise inclusive OR and assignment operator.	***var3*** |= 2 is same as ***var3*** = ***var3*** | 2

Misc Operators ↦ sizeof & ternary

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

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(***var1***), where ***var1*** is *int*, will return 4.
&	Returns the address of a variable.	&***var1***; returns the actual address of the variable.
*	Pointer to a variable.	****var1***;
? :	Conditional Expression.	If Condition is true ? then value ***exp1*** : otherwise value ***exp2***

Operators Precedence in C:

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, ***var1*** = *7 + 3 * 2*; here, ***var1*** is assigned *13*, not *20* because operator *** has a higher precedence than *+*, so it first gets multiplied with *3*2* and then adds into *7*.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right

Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right