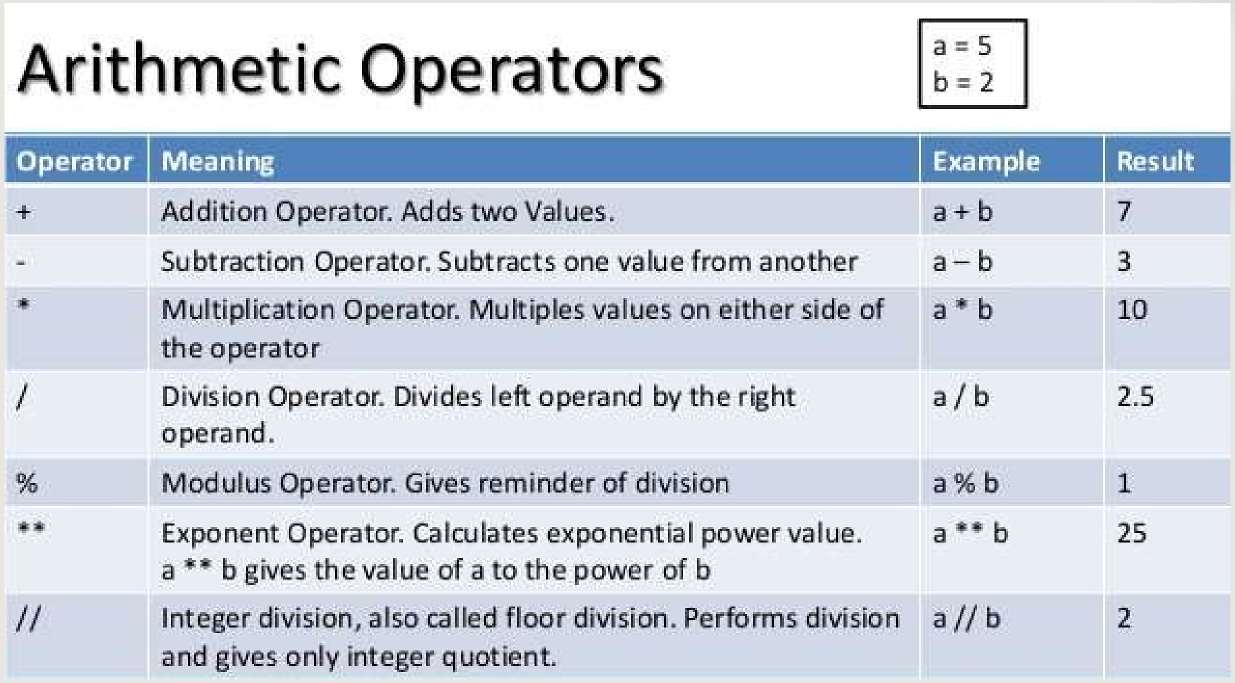
Operators

Operators

- An operator is a symbol that directs the computer to perform certain mathematical or logical manipulations and is usually used to manipulate data and variables.
- For example: +, *, -, /, %.
- 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

Arithmetic operators

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



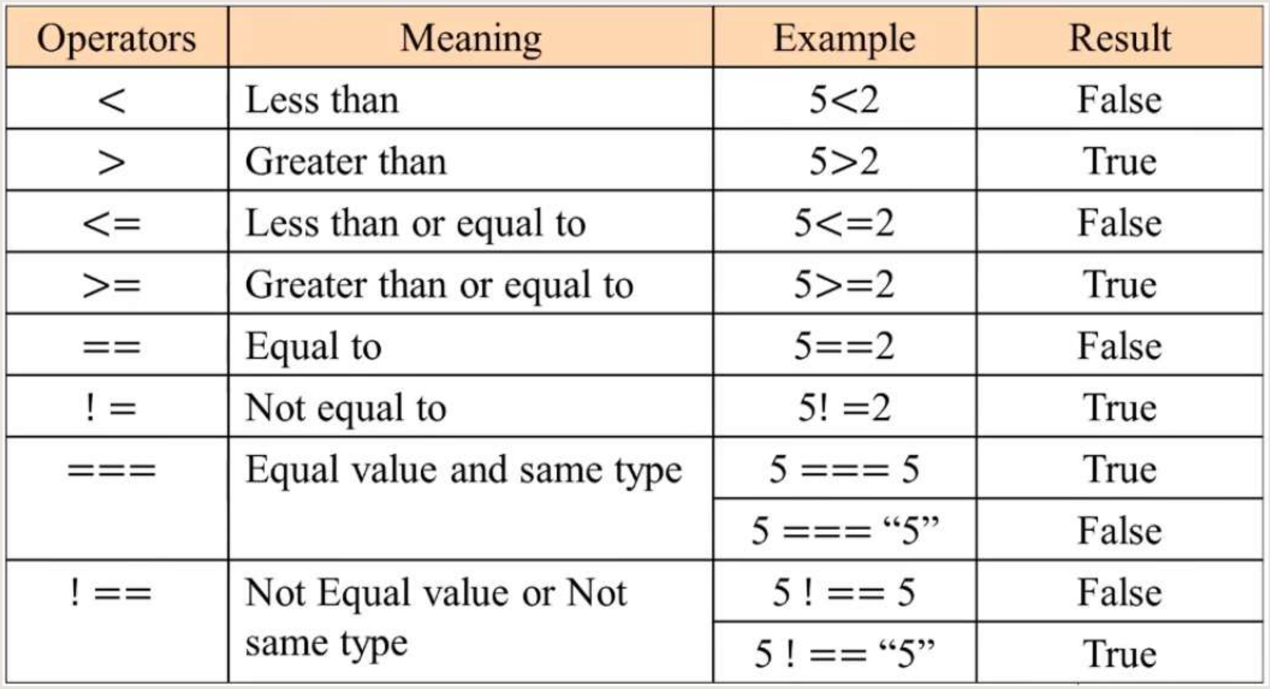
Arithmetic Operators

a = 5
b = 2

Operator	Meaning	Example	Result
+	Addition Operator. Adds two Values.	a + b	7
-	Subtraction Operator. Subtracts one value from another	a – b	3
*	Multiplication Operator. Multiples values on either side of the operator	a * b	10
/	Division Operator. Divides left operand by the right operand.	a / b	2.5
%	Modulus Operator. Gives reminder of division	a % b	1
**	Exponent Operator. Calculates exponential power value. a ** b gives the value of a to the power of b	a ** b	25
//	Integer division, also called floor division. Performs division and gives only integer quotient.	a // b	2

Relational operators

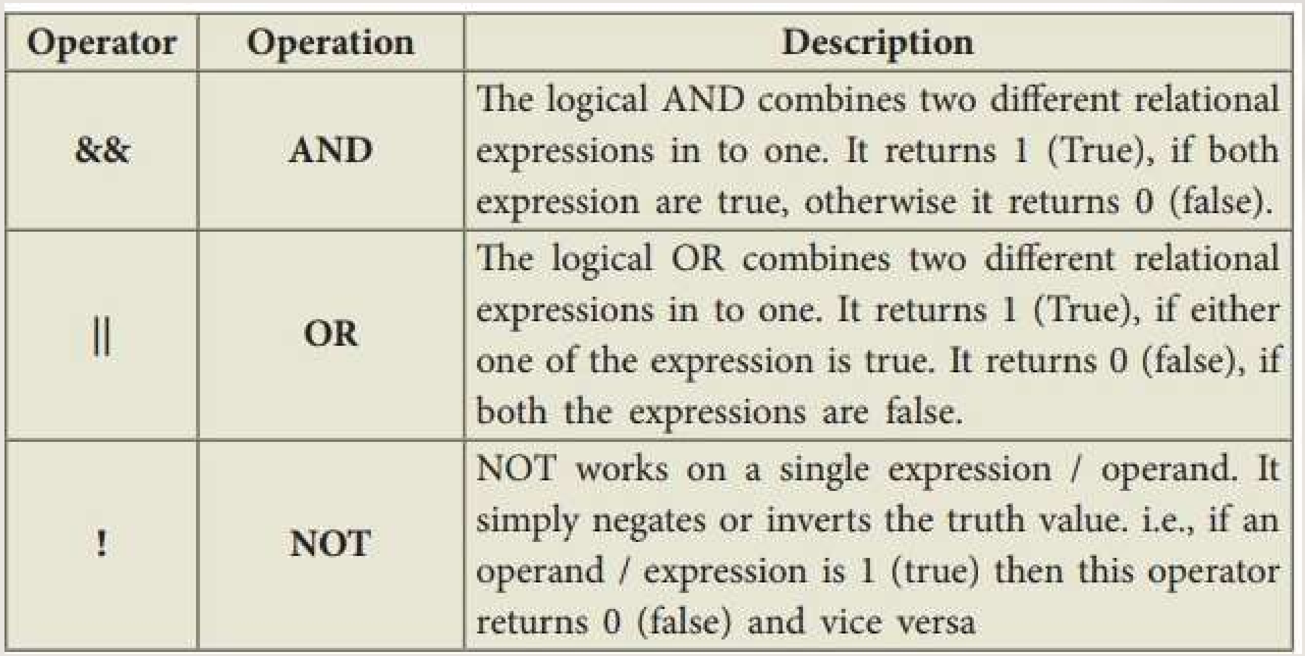
- A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.



Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === “5”	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == “5”	True

Logical operators

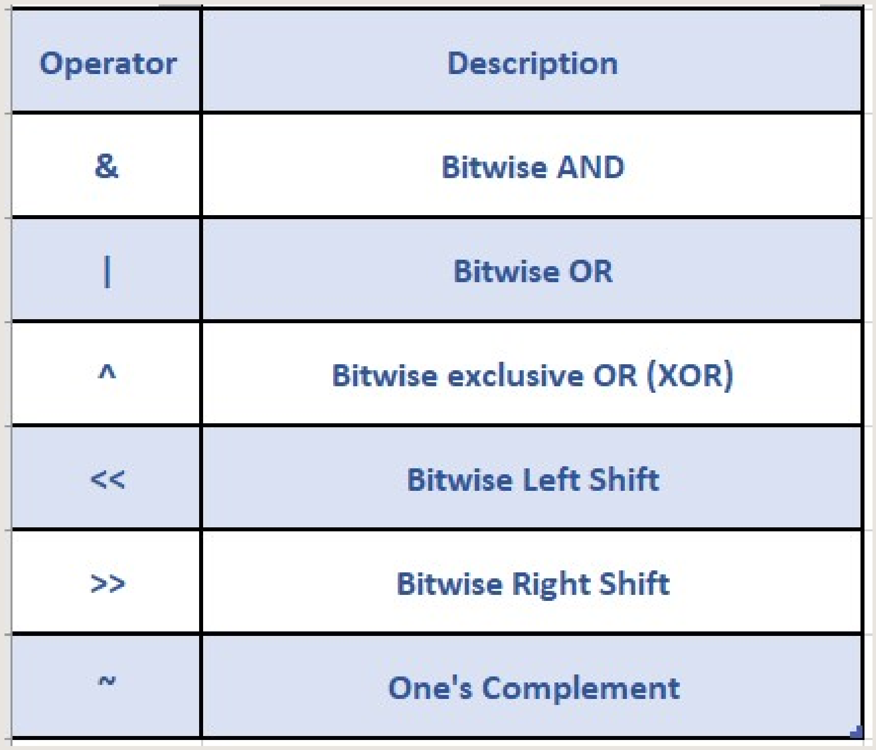
- An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false.



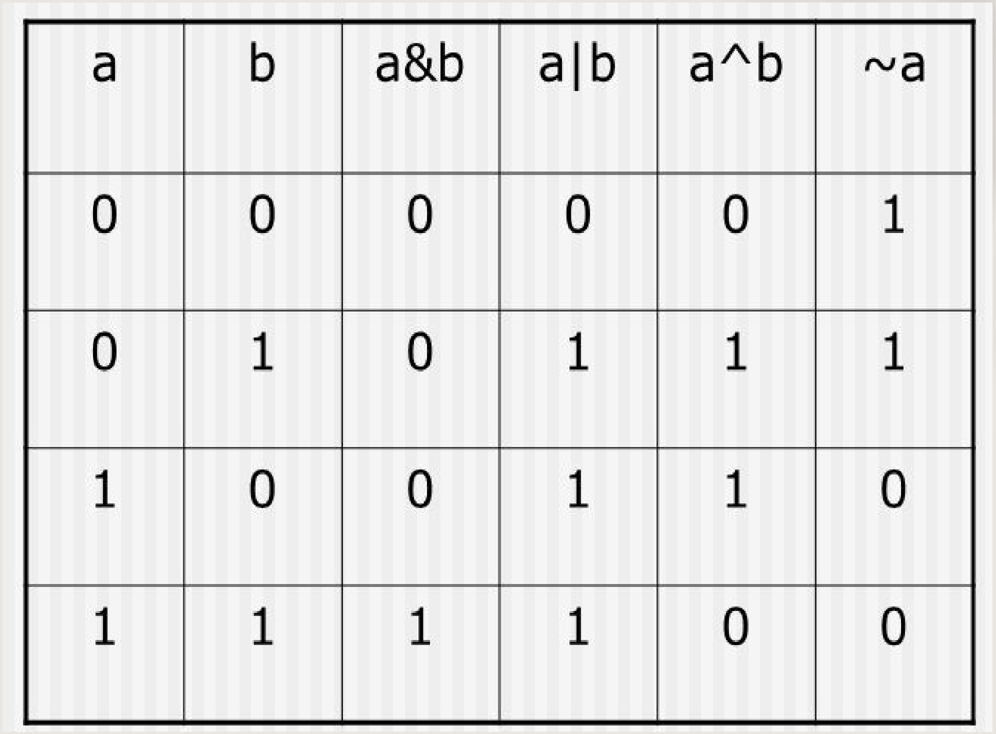
Operator	Operation	Description
&&	AND	The logical AND combines two different relational expressions in to one. It returns 1 (True), if both expression are true, otherwise it returns 0 (false).
||	OR	The logical OR combines two different relational expressions in to one. It returns 1 (True), if either one of the expression is true. It returns 0 (false), if both the expressions are false.
!	NOT	NOT works on a single expression / operand. It simply negates or inverts the truth value. i.e., if an operand / expression is 1 (true) then this operator returns 0 (false) and vice versa

Bitwise operators

Bitwise operator works on bits and perform bit-by-bit operation.



Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR (XOR)
<<	Bitwise Left Shift
>>	Bitwise Right Shift
~	One's Complement



a	b	a&b	a|b	a^b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Assume A = 60 and B = 13 in binary format, they will be as follows:
A = 0011 1100   B = 0000 1101

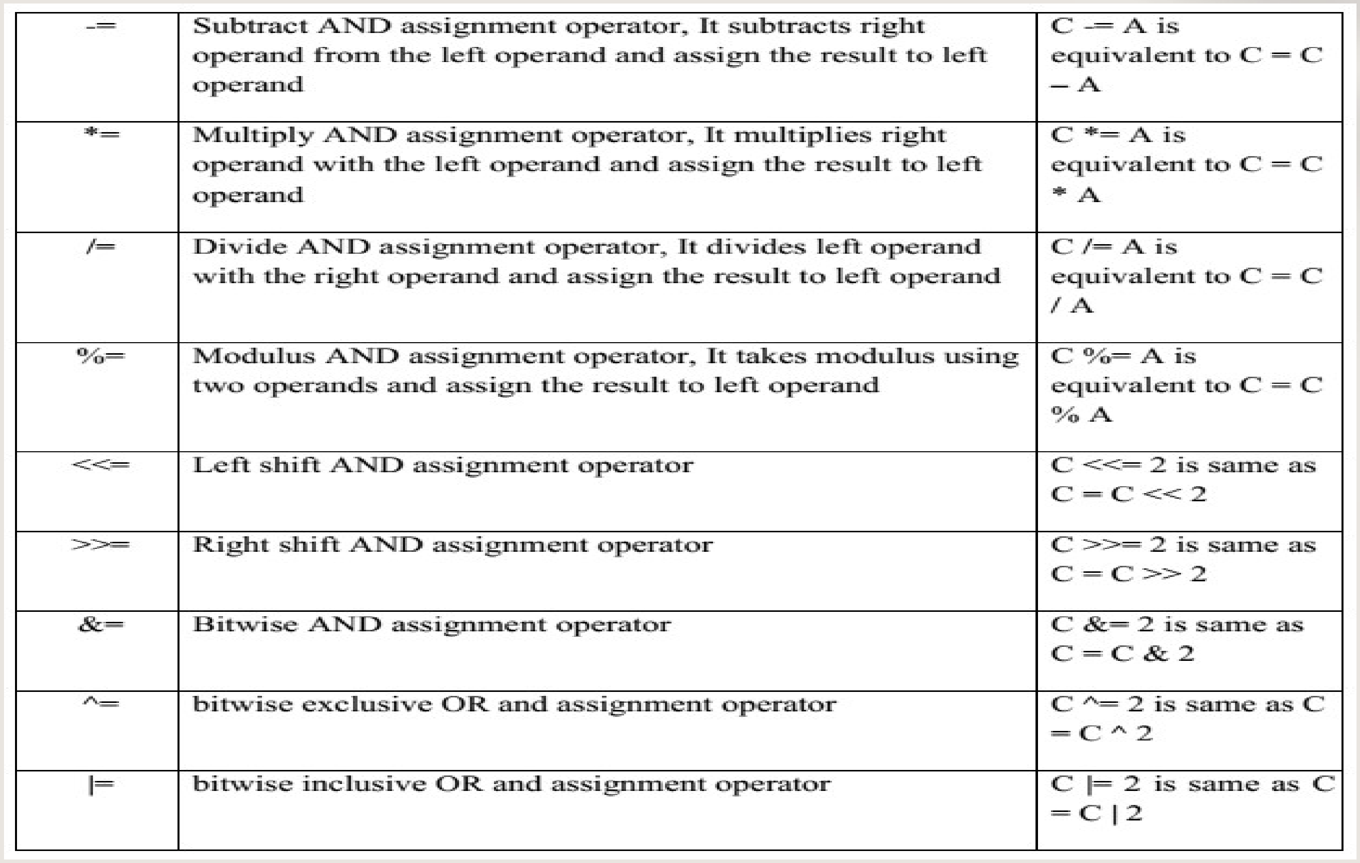
Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(0011 1100 11000011 0000 1100=12
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand 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= 1...0001<<1
		A << 1 =i .e.,0010.....2
		i.e=1*2=2

>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand	A=4....0100>>1 A >> = i.e., 0010......2

Assignment operator

- An assignment operator is used for assigning a value to a variable.



-=	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
*=	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
/=	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
%=	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
<<=	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

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 2.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable. It can check value at address	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Operator 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: x = 7 + 3 * 2; here, x 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

Highest Priority

↑

Lowest Priority

Increment and Decrement operators

- C has two operators to change the value of the operand by 1. These operators are:
  - Increment operator (++a): it increments the value of operand by 1
  - Decrement operator (--): it decreases the value of the operand by 1

Pre-increment

- A pre-increment operator is used to increment the value of a variable before using it in a expression that is first value is incremented and then used inside the expression.
- Syntax: a = ++x;
- Example: if x = 10;

  a = ++x;

  then a = 11

Because the value of ‘x’ is modified before assigning it to ‘a’

Post-increment

- A post increment operator is used to increment the value of the variable after executing expression completely in which post increment is used.
- Syntax: a = x++;
- Example: if x = 10;

                a = x++;

              then a = 10

X=11

a=X

Because the value of ‘x’ is modified after assigning it to ‘a’

Pre decrement and Post decrement

- Pre decrement and post decrement works same as the pre increment and post increment respectively.
- The only difference is that it is used to decrease the value of the operand by 1.