Expression using Operators in C

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

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

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

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 −

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 −

Operator	Description	Example
&	Binary AND **It takes 1 if both operands has value 1.**	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operan **The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1.**	(A | B) = 61, i.e., 0011 1101
^	Binary XOR **1 if the corresponding bits of two operands are opposite**	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement 'flipping' bits- 0 changed to 1and 1 changed to 0	(~A) = -60, i.e,. 1100 0100
<<	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

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

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

Operators Precedence in C

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.

Table showing highest precedence to lowest precedence

Category	Operator	Associativity
Postfix	( ) [ ] -> . ++ - -	Left to right
Unary	Unary +,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

Expression

Expression is a combination of variables(like a,b,m,n..), constants(3,2,1) and operators(+,/*).

Eg : c+d

x/y+b+a*a*a

3.14 *r *r

Algebraic Expression	C Expression
ab-c	a*b-c
(m+n)(k+j)	(m+n)*(k+j)
(ab/c)	a*b/c
$3x^2+2x+1$	3*x^2+2*x+1

Example Program

```
#include<stdio.h>
```

Program

```
int main( )
{
int x=2,y=3,result;
result=x*5+y*7;
printf("result =:%d",result);
return 0;
}
```

Expression evaluation

```
result=x*5 + y*7;
result=2*5 + 3*7;
result=2*5 + 3*7;
result=10 + 3*7;
result=10 + 21;
result=31;
```

Example program –find greatest of 3 numbers

Example of logical(&& logical AND) and relational operators(>)

```
#include<stdio.h>
int main()
{
   int num1,num2,num3;

   printf("\nEnter value of a, b and c:");

   scanf("%d %d %d",&a,&b,&c);

   if((a>b)&&(a>c))
      printf("\n %d is greatest",a);
   else if(b>c)
      printf("\n %d is greatest",b ");
   else
      printf("\n %d is greatest",c);
   return 0;
}
```

Example program –find odd or even number

Example of Arithmetic(% mod) and relational operators(==)

```
#include<stdio.h>
int main()
{    int num,result;
   if(num%2==0)
   printf("even number \n");
   else
   printf("odd number \n");
   return 0;
```

Bitwise XOR

```
#include
<stdio.h
>int
main()
{
    int a = 12, b =
    25;
    printf("Output =
    %d", a^b);return
    0;
}
```

Output = 21

```
Explanation

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
  00001100
  00011001

  00010101  = 21 (In decimal)
  ________
```

Bitwise complement 1's compliment

```
#include
<stdio.h
>int
main()
{
  printf("complement =
  %d\n",~35);return 0;
}
```

OutPut:

complement = 220

```
Explanation

35 = 00100011 (In Binary)

Bitwise complement Operation of 35
~ 00100011
  ________
  11011100  = 220 (In decimal)
```

Bitwise AND and OR operator

```
#include
<stdio.h
>int
main()
{
 int a = 12, b = 25;
 printf("OutputAND =
 %d", a&b);
 printf("OutputOR =
 %d", a|b); return 0;
}
```

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bitwise AND Operation of 12 and 25
  00001100
& 00011001
  ________
  00001000  = 8 (In decimal)
_____________________________________
Bitwise OR Operation of 12 and 25
  00001100
| 00011001
  ________
  00011101  = 29 (In decimal)
```

```
OutputAND = 8
OutputOR = 29
```