Bitwise Operators in C

Bitwise operators in C allow low-level manipulation of data stored in computer's memory.

Bitwise operators contrast with logical operators in C. For example, the logical AND operator (**&&**) performs AND operation on two Boolean expressions, while the bitwise AND operator (**&**) performs the AND operation on each corresponding bit of the two operands.

For the three logical operators **&&**, **||**, and **!**, the corresponding bitwise operators in C are **&**, **|** and **~**.

Additionally, the symbols **^** (XOR), **<<** (left shift) and **>>** (right shift) are the other bitwise operators.

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B)
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B)
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B)
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2

Even though these operators work on individual bits, they need the operands in the form C data types or variables only, as a variable occupies a specific number of bytes in the memory.

Bitwise AND Operator (&) in C

The bitwise AND (&) operator performs as per the following truth table –

bit a	bit b	a & b
0	0	0

0	1	0
1	0	0
1	1	1

Bitwise binary AND performs logical operation on the bits in each position of a number in its binary form.

Assuming that the two int variables "a" and "b" have the values 60 (equivalent to 0011 1100 in binary) and 13 (equivalent to 0000 1101 in binary), the "a & b" operation results in 13, as per the bitwise ANDing of their corresponding bits illustrated below −

```
  0011 1100
& 0000 1101
  ---------
= 0000 1100
```

The binary number 00001100 corresponds to 12 in decimal.

Bitwise OR (|) Operator

The bitwise OR (|) operator performs as per the following truth table −

bit a	**bit b**	**a | b**
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise binary OR performs logical operation on the bits in each position of a number in its binary form.

Assuming that the two int variables "a" and "b" have the values 60 (equivalent to 0011 1100 in binary) and 13 (equivalent to 0000 1101 in binary), then "**a | b**" results in 61, as per the bitwise OR of their corresponding bits illustrated below −

```
  0011 1100
| 0000 1101
  ---------
= 0011 1101
```

The binary number 00111101 corresponds to 61 in decimal.

Bitwise XOR (^) Operator

The bitwise XOR (^) operator performs as per the following truth table −

bit a	bit b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise binary XOR performs logical operation on the bits in each position of a number in its binary form. The XOR operation is called "exclusive OR".

Note: The result of XOR is 1 if and only if one of the operands is 1. Unlike OR, if both bits are 1, XOR results in 0.

Assuming that the two int variables "a" and "b" have the values 60 (equivalent to 0011 1100 in binary) and 13 (equivalent to 0000 1101 in binary), the "**a ^ b**" operation results in 49, as per the bitwise XOR of their corresponding bits illustrated below −

```
  0011 1100
^ 0000 1101
  ---------
= 0011 0001
```

The binary number 00110001 corresponds to 49 in decimal.

The Left Shift (<<) Operator

The left shift operator is represented by the **<<** symbol. It shifts each bit in its left-hand operand to the left by the number of positions indicated by the right-hand operand. Any blank spaces generated while shifting are filled up by zeroes.

Assuming that the int variable "a" has the value 60 (equivalent to 0011 1100 in binary), the "**a << 2**" operation results in 240, as per the bitwise left-shift of its corresponding bits illustrated below −

```
0011 1100 << 2 = 1111 0000
```

The binary number 11110000 corresponds to 240 in decimal.

The Right Shift (>>) Operator

The right shift operator is represented by the **>>** symbol. It shifts each bit in its left-hand operand to the right by the number of positions indicated by the right-hand operand. Any blank spaces generated while shifting are filled up by zeroes.

Assuming that the int variable a has the value 60 (equivalent to 0011 1100 in binary), the "**a >> 2**" operation results in 15, as per the bitwise right-shift of its corresponding bits illustrated below –

```
0011 1100 >> 2 = 0000 1111
```

The binary number 00001111 corresponds to 15 in decimal.

The 1's Complement (~) Operator

The 1's compliment operator (~) in C is a unary operator, needing just one operand. It has the effect of "flipping" the bits, which means 1s are replaced by 0s and vice versa.

a	~a
0	1
1	0

Assuming that the int variable "a" has the value 60 (equivalent to 0011 1100 in binary), then the "**~a**" operation results in -61 in 2's complement form, as per the bitwise right-shift of its corresponding bits illustrated below –

```
~ 0011 1100 = 1100 0011
```

The binary number 1100 0011 corresponds to -61 in decimal.

Example

In this example, we have highlighted the operation of all the bitwise operators:

</> Open Compiler

```c
#include <stdio.h>

int main(){

   unsigned int a = 60; /* 60 = 0011 1100 */
   unsigned int b = 13; /* 13 = 0000 1101 */
   int c = 0;

   c = a & b;        /* 12 = 0000 1100 */
   printf("Line 1 - Value of c is %d\n", c );

   c = a | b;        /* 61 = 0011 1101 */
   printf("Line 2 - Value of c is %d\n", c );

   c = a ^ b;        /* 49 = 0011 0001 */
   printf("Line 3 - Value of c is %d\n", c );

   c = ~a;           /*-61 = 1100 0011 */
   printf("Line 4 - Value of c is %d\n", c );

   c = a << 2;     /* 240 = 1111 0000 */
   printf("Line 5 - Value of c is %d\n", c );

   c = a >> 2;     /* 15 = 0000 1111 */
   printf("Line 6 - Value of c is %d\n", c );

   return 0;
}
```

Output

When you run this code, it will produce the following output –

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```