Bitwise Operators in C
[AND, OR, XOR, Shift & Complement]

What are Bitwise Operators?

Bitwise Operators are used for manipulating data at the bit level, also called bit level programming. Bitwise operates on one or more-bit patterns or binary numerals at the level of their individual bits. They are used in numerical computations to make the calculation process faster.
Following is the list of bitwise operators provided by 'C' programming language:

Operator	**Meaning**
&	Bitwise AND operator
|	Bitwise OR operator
^	Bitwise exclusive OR operator
~	Binary One's Complement Operator is a unary operator
<<	Left shift operator
>>	Right shift operator

Bitwise operators cannot be directly applied to primitive data types such as float, double, etc. Always remember one thing that bitwise operators are mostly used with the integer data type because of its compatibility.
The bitwise logical operators work on the data bit by bit, starting from the least significant bit, i.e. LSB bit which is the rightmost bit, working towards the MSB (Most Significant Bit) which is the leftmost bit.

The result of the computation of bitwise logical operators is shown in the table given below.

x	**y**	**x & y**	**x | y**	**x ^ y**
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Content-

Bitwise AND

This is one of the most commonly used logical bitwise operators. It is represented by a single ampersand sign (&). Two integer expressions are written on each side of the (&) operator. The result of the bitwise AND operation is 1 if both the bits have the value as 1; otherwise, the result is always 0.

Let us consider that we have 2 variables op1 and op2 with values as follows:

```
Op1 = 0000 1101
Op2 = 0001 1001
```

The result of the AND operation on variables op1 and op2 will be

```
Result = 0000 1001
```

As we can see, two variables are compared bit by bit. Whenever the value of a bit in both the variables is 1, then the result will be 1 or else 0.

Bitwise OR

It is represented by a single vertical bar sign (|). Two integer expressions are written on each side of the (|) operator.

The result of the bitwise OR operation is 1 if at least one of the expression has the value as 1; otherwise, the result is always 0.

Let us consider that we have 2 variables op1 and op2 with values as follows:

```
Op1 = 0000 1101
Op2 = 0001 1001
```

The result of the OR operation on variables op1 and op2 will be

```
Result = 0001 1101
```

As we can see, two variables are compared bit by bit. Whenever the value of a bit in one of the variables is 1, then the result will be 1 or else 0.

Bitwise Exclusive OR

It is represented by a symbol (^). Two integer expressions are written on each side of the (^) operator.

The result of the bitwise Exclusive-OR operation is 1 if only one of the expression has the value as 1; otherwise, the result is always 0.

Let us consider that we have 2 variables op1 and op2 with values as follows:

```
Op1 = 0000 1101
Op2 = 0001 1001
```

The result of the XOR operation on variables op1 and op2 will be

```
Result = 0001 0100
```

As we can see, two variables are compared bit by bit. Whenever only one variable holds the value 1 then the result is 0 else 0 will be the result.

Let us write a simple program that demonstrates bitwise logical operators.

```
#include <stdio.h>
int main()
{
        int a = 20;     /* 20 = 010100 */
    int b = 21;/* 21 = 010101 */
        int c = 0;

        c = a & b;       /* 20 = 010100 */
        printf("AND - Value of c is %d\n", c );

        c = a | b;       /* 21 = 010101 */
        printf("OR - Value of c is %d\n", c );

        c = a ^ b;       /* 1 = 0001 */
        printf("Exclusive-OR - Value of c is %d\n", c );

        getch();
}
```

Output:

```
AND - Value of c is 20
OR - Value of c is 21
Exclusive-OR - Value of c is 1
```

Bitwise shift operators

The bitwise shift operators are used to move/shift the bit patterns either to the left or right side. Left and right are two shift operators provided by ‘C’ which are represented as follows:

```
Operand << n (Left Shift)
Operand >> n (Right Shift)
```

Here,

- an operand is an integer expression on which we have to perform the shift operation.
- ‘n’ is the total number of bit positions that we have to shift in the integer expression.

The left shift operation will shift the ‘n’ number of bits to the left side. The leftmost bits in the expression will be popped out, and n bits with the value 0 will be filled on the right side.

The right shift operation will shift the ‘n’ number of bits to the right side. The rightmost ‘n’ bits in the expression will be popped out, and the value 0 will be filled on the left side.

Example: x is an integer expression with data 1111. After performing shift operation the result will be:

```
x << 2 (left shift) = 1111<<2 = 1100
x>>2 (right shift) = 1111>>2 = 0011
```

Shifts operators can be combined then it can be used to extract the data from the integer expression. Let us write a program to demonstrate the use of bitwise shift operators.

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

```
int main() {
   int a = 20;  /* 20 = 010100 */
   int c = 0;

   c = a << 2;  /* 80 = 101000 */
   printf("Left shift - Value of c is %d\n", c );

   c = a >> 2;  /*05 = 000101 */
   printf("Right shift - Value of c is %d\n", c );
   return 0;
}
```

Output:

```
Left shift - Value of c is 80
Right shift - Value of c is 5
```

After performing the left shift operation the value will become 80 whose binary equivalent is 101000.

After performing the right shift operation, the value will become 5 whose binary equivalent is 000101.

Bitwise complement operator

The bitwise complement is also called as one's complement operator since it always takes only one value or an operand. It is a unary operator.

When we perform complement on any bits, all the 1's become 0's and vice versa.

If we have an integer expression that contains 0000 1111 then after performing bitwise complement operation the value will become 1111 0000.

Bitwise complement operator is denoted by symbol tilde (~).

Let us write a program that demonstrates the implementation of bitwise complement operator.

```
#include <stdio.h>
int main() {
   int a = 10;  /* 10 = 1010 */
   int c = 0;
   c = ~(a);
   printf("Complement - Value of c is %d\n", c );
   return 0;
}
```

Output:

```
Complement - Value of c is -11
```

Here is another program, with an example of all the operatoes discussed so far:

```
#include <stdio.h>
main() {
   unsigned int x = 48;        /* 48 = 0011 0000 */
   unsigned int y = 13;        /* 13 = 0000 1101 */
   int z = 0;

   z =x & y;       /* 0 = 0000 0000 */
   printf("Bitwise AND Operator - x & y = %d\n", z );

   z = x | y;       /* 61 = 0011 1101 */
   printf("Bitwise OR Operator - x | y = %d\n", z );

   z= x^y;       /* 61 = 0011 1101 */
   printf("Bitwise XOR Operator- x^y= %d\n", z);

   z = ~x;          /*-49 = 11001111 */
   printf("Bitwise One's Complement Operator - ~x = %d\n", z);

   z = x << 2;     /* 192 = 1100 0000 */
   printf("Bitwise Left Shift Operator x << 2= %d\n", z );

   z= x >> 2;     /* 12 = 0000 1100 */
   printf ("Bitwise Right Shift Operator x >> 2= %d\n", z );}
```

After we compile and run the program, it produces the following result:

```
Bitwise AND Operator - x & y = 0
Bitwise OR Operator - x | y = 61
Bitwise XOR Operator- x^y= 61
Bitwise One's Complement Operator - ~x = -49
Bitwise Left Shift Operator x << 2= 192
Bitwise Right Shift Operator x >> 2= 12
```

Summary

- Bitwise operators are special operator set provided by ‘C.’
- They are used in bit level programming.
- These operators are used to manipulate bits of an integer expression.
- Logical, shift and complement are three types of bitwise operators.
- Bitwise complement operator is used to reverse the bits of an expression.

@https://www.guru99.com/c-bitwise-operators.html