Bitwise Operator in C

The bitwise operators are the operators used to perform the operations on the data at the bit-level. When we perform the bitwise operations, then it is also known as bit-level programming. It consists of two digits, either 0 or 1. It is mainly used in numerical computations to make the calculations faster.

We have different types of bitwise operators in the C programming language. The following is the list of the bitwise operators:

Operator	Meaning of operator
&	Bitwise AND operator
|	Bitwise OR operator
^	Bitwise exclusive OR operator
~	One's complement operator (unary operator)
<<	Left shift operator
>>	Right shift operator

Let's look at the truth table of the bitwise operators.

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	1

Bitwise AND operator

Bitwise AND operator is denoted by the single ampersand sign (&). Two integer operands are written on both sides of the (&) operator. If the corresponding bits of both

the operands are 1, then the output of the bitwise AND operation is 1; otherwise, the output would be 0.

For example,

1. We have two variables a and b.
2. a =6;
3. b=4;
4. The binary representation of the above two variables are given below:
5. a = 0110
6. b = 0100
7. When we apply the bitwise AND operation in the above two variables, i.e., a&b, the outp ut would be:
8. Result = 0100

As we can observe from the above result that bits of both the variables are compared one by one. If the bit of both the variables is 1 then the output would be 1, otherwise 0.

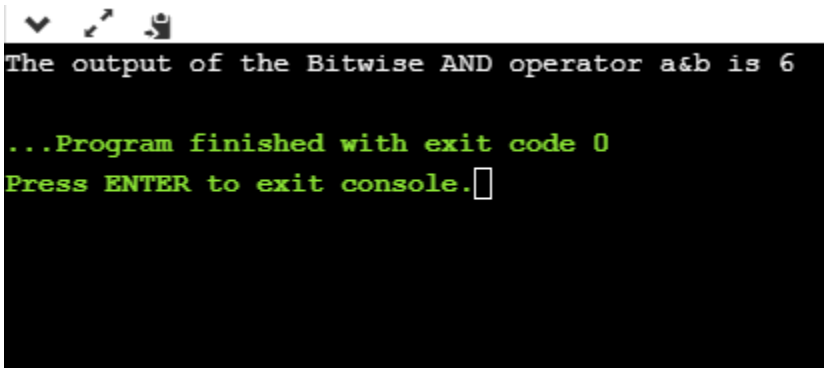
Let's understand the bitwise AND operator through the program.

```
1. #include <stdio.h>
2. int main()
3. {
4.    int a=6, b=14;  // variable declarations
5.    printf("The output of the Bitwise AND operator a&b is %d",a&b);
6.    return 0;
7. }
```

In the above code, we have created two variables, i.e., 'a' and 'b'. The values of 'a' and 'b' are 6 and 14 respectively. The binary value of 'a' and 'b' are 0110 and 1110, respectively. When we apply the AND operator between these two variables,

a AND b = 0110 && 1110 = 0110

Output



Bitwise OR operator

The bitwise OR operator is represented by a single vertical sign (|). Two integer operands are written on both sides of the (|) symbol. If the bit value of any of the operand is 1, then the output would be 1, otherwise 0.

For example,

1. We consider two variables,
2. a = 23;
3. b = 10;
4. The binary representation of the above two variables would be:
5. a = 0001 0111
6. b = 0000 1010
7. When we apply the bitwise OR operator in the above two variables, i.e., a|b , then the ou tput would be:
8. Result = 0001 1111

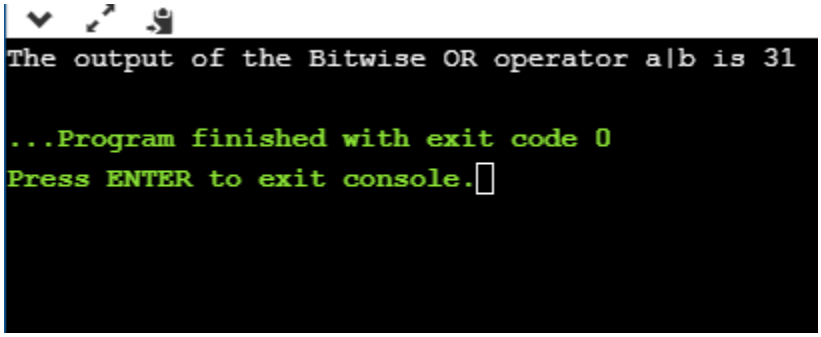
As we can observe from the above result that the bits of both the operands are compared one by one; if the value of either bit is 1, then the output would be 1 otherwise 0.

Let's understand the bitwise OR operator through a program.

```
1. #include <stdio.h>
2. int main()
3. {
4.    int a=23,b=10;  // variable declarations
5.    printf("The output of the Bitwise OR operator a|b is %d",a|b);
```

```
6.    return 0;
7. }
```

Output



Bitwise exclusive OR operator

Bitwise exclusive OR operator is denoted by (^) symbol. Two operands are written on both sides of the exclusive OR operator. If the corresponding bit of any of the operand is 1 then the output would be 1, otherwise 0.

For example,

1. We consider two variables a and b,
2. a = 12;
3. b = 10;
4. The binary representation of the above two variables would be:
5. a = 0000 1100
6. b = 0000 1010
7. When we apply the bitwise exclusive OR operator in the above two variables (a^b), then the result would be:
8. Result = 0000 1110

As we can observe from the above result that the bits of both the operands are compared one by one; if the corresponding bit value of any of the operand is 1, then the output would be 1 otherwise 0.

Let's understand the bitwise exclusive OR operator through a program.

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

```
2. int main()
3. {
4.    int a=12,b=10;  // variable declarations
5.    printf("The output of the Bitwise exclusive OR operator a^b is %d",a^b);
6.    return 0;
7. }
```

Output

Bitwise complement operator

The bitwise complement operator is also known as one's complement operator. It is represented by the symbol tilde (~). It takes only one operand or variable and performs complement operation on an operand. When we apply the complement operation on any bits, then 0 becomes 1 and 1 becomes 0.

For example,

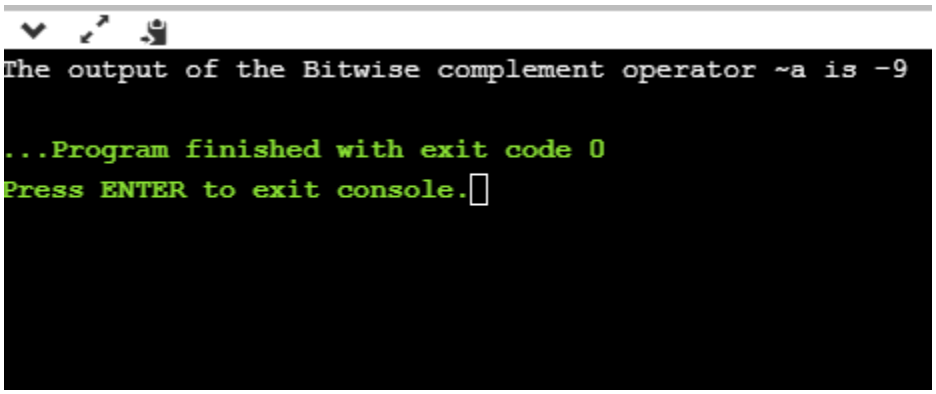
1. If we have a variable named 'a',
2. a = 8;
3. The binary representation of the above variable is given below:
4. a = 1000
5. When we apply the bitwise complement operator to the operand, then the output woul d be:
6. Result = 0111

As we can observe from the above result that if the bit is 1, then it gets changed to 0 else 1.

Let's understand the complement operator through a program.

```
1. #include <stdio.h>
2. int main()
3. {
4.    int a=8;  // variable declarations
5.    printf("The output of the Bitwise complement operator ~a is %d",~a);
6.    return 0;
7. }
```

Output



Bitwise shift operators

Two types of bitwise shift operators exist in C programming. The bitwise shift operators will shift the bits either on the left-side or right-side. Therefore, we can say that the bitwise shift operator is divided into two categories:

- Left-shift operator
- Right-shift operator

Left-shift operator

It is an operator that shifts the number of bits to the left-side.

Syntax of the left-shift operator is given below:

```
1. Operand << n
```

Where,

Operand is an integer expression on which we apply the left-shift operation.

n is the number of bits to be shifted.

In the case of Left-shift operator, 'n' bits will be shifted on the left-side. The 'n' bits on the left side will be popped out, and 'n' bits on the right-side are filled with 0.

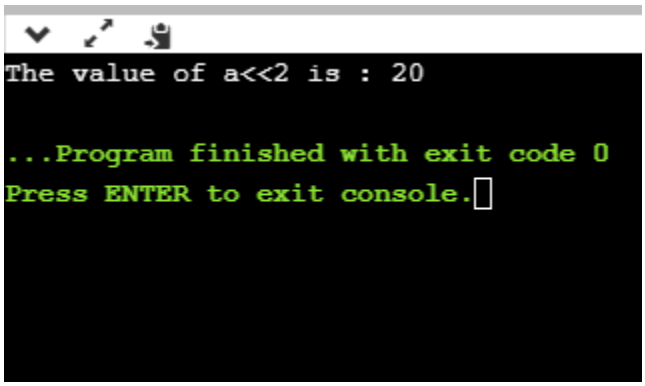
For example,

1. Suppose we have a statement:
2. **int** a = 5;
3. The binary representation of 'a' is given below:
4. a = 0101
5. If we want to left-shift the above representation by 2, then the statement would be:
6. a << 2;
7. 0101<<2 = 00010100

Let's understand through a program.

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a=5; // variable initialization
5.     printf("The value of a<<2 is : %d ", a<<2);
6.     return 0;
7. }
```

Output



Right-shift operator

It is an operator that shifts the number of bits to the right side.

Syntax of the right-shift operator is given below:

1. Operand >> n;

Where,

Operand is an integer expression on which we apply the right-shift operation.

N is the number of bits to be shifted.

In the case of the right-shift operator, 'n' bits will be shifted on the right-side. The 'n' bits on the right-side will be popped out, and 'n' bits on the left-side are filled with 0.

For example,

1. Suppose we have a statement,
2. **int** a = 7;
3. The binary representation of the above variable would be:
4. a = 0111
5. If we want to right-shift the above representation by 2, then the statement would be:
6. a>>2;
7. 0000 0111 >> 2 = 0000 0001

Let's understand through a program.

1. #include <stdio.h>

```
2. int main()
3. {
4.     int a=7; // variable initialization
5.     printf("The value of a>>2 is : %d ", a>>2);
6.     return 0;
7. }
```

Output