BITWISE OPERATOR:

In arithmetic-logic unit (which is within the CPU), mathematical operations like: addition, subtraction, multiplication and division are done in bit-level. To perform bit-level operations in C programming, bitwise operators are used.

Bitwise AND operator &
The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Let us suppose the bitwise AND operation of two integers 12 and 25.

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

Bit Operation of 12 and 25

```
  00001100
& 00011001
  ________
  00001000  = 8 (In decimal)
```

Example #1: Bitwise AND

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

Output

```
Output = 8
```

Bitwise OR operator |
The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

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

Bitwise OR Operation of 12 and 25

```
  00001100
| 00011001
  ________
  00011101  = 29 (In decimal)
```

Example #2: Bitwise OR

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

Output

```
Output = 29
```

Bitwise XOR (exclusive OR) operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

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

Bitwise XOR Operation of 12 and 25
  00001100
^ 00011001
  ________
  00010101  = 21 (In decimal)
```

Example #3: Bitwise XOR

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

Output

```
Output = 21
```

Bitwise complement operator ~

Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

```
35 = 00100011 (In Binary)

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

Twist in bitwise complement operator in C Programming

The bitwise complement of 35 (~35) is -36 instead of 220, but why?

For any integer n, bitwise complement of n will be -(n+1). To understand this, you should have the knowledge of 2's complement.

2's Complement

Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1. For example:

Decimal	Binary	2's complement
0	00000000	-(11111111+1) = -00000000 = -0(decimal)
1	00000001	-(11111110+1) = -11111111 = -256(decimal)
12	00001100	-(11110011+1) = -11110100 = -244(decimal)
220	11011100	-(00100011+1) = -00100100 = -36(decimal)

Note: Overflow is ignored while computing 2's complement.

The bitwise complement of 35 is 220 (in decimal). The 2's complement of 220 is -36. Hence, the output is -36 instead of 220.

Bitwise complement of any number N is -(N+1). Here's how:
bitwise complement of N = ~N (represented in 2's complement form)
2'complement of ~N= -(~(~N)+1) = -(N+1)
Example #4: Bitwise complement

```
#include <stdio.h>
int main()
{
    printf("Output = %d\n",~35);
    printf("Output = %d\n",~-12);
    return 0;
}
```

Output

```
Output = -36
Output = 11
```

Shift Operators in C programming
There are two shift operators in C programming:

Right shift operator
Left shift operator.
Right Shift Operator
Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

```
212 = 11010100 (In binary)
212>>2 = 00110101 (In binary) [Right shift by two bits]
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)
```

Left Shift Operator
Left shift operator shifts all bits towards left by certain number of specified bits. It is denoted by <<.

```
212 = 11010100 (In binary)
212<<1 = 110101000 (In binary) [Left shift by one bit]
212<<0 =11010100 (Shift by 0)
212<<4 = 110101000000 (In binary) =3392(In decimal)
```

Example #5: Shift Operators

```
#include <stdio.h>
int main()
{
    int num=212, i;
    for (i=0; i<=2; ++i)
        printf("Right shift by %d: %d\n", i, num>>i);

     printf("\n");

     for (i=0; i<=2; ++i)
        printf("Left shift by %d: %d\n", i, num<<i);

     return 0;
}
```

```
Right Shift by 0: 212
Right Shift by 1: 106
Right Shift by 2: 53

Left Shift by 0: 212
Left Shift by 1: 424
Left Shift by 2: 848
```

TERNARY OPERATOR:
C includes a decision-making operator ?: which is called the conditional operator or ternary operator. It is the short form of the if else conditions.

Syntax:
condition ? statement 1 : statement 2
The ternary operator starts with a boolean condition. If this condition evaluates to true then it will execute the first statement after ?, otherwise the second statement after : will be executed.

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

{
int x = 20, y = 10;
var result = x > y ? "x is greater than y" : "x is less than y";

return 0;
}
```

output: x is greater than y