Bitwise Operators in C

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)
```

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

}
```

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)
```

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

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)
```

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

Bitwise complement operator ~: Bitwise compliment operator is a 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)
```

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)

Bitwise Operators in C

Decimal	Binary	2's complement
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): bitwise complement of N = ~N (represented in 2's complement form)**.** 2'complement of ~N= -(~(~N)+1) = -(N+1)

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

}
```

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)

```
#include <stdio.h>
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);
}
```