```
1.int main()
{
        char a = 255;
        char b = 127;

        b = ~b;
        a = a ^ b;

        printf("\n%d,%d",a,b);

        return 0;
}
```

Bitwise complement(~) of 127 is -128 (o/p of b is -128)
255 ^ -128

Binary equivalent of 255-> 1111 1111
Binary equivalent of 128-> 1000 0000

As the number 128 is negative we perform the 2's complement on 128 to gets its binary equivalent

```
         0111 1111   --->( 1's compliment of 128 )
    +           1

         1000 0000   -- > ( 2's compliment of 128 i.e -128 binary )
```

Know we perform ex-or operations of 255 and -128

```
         1111 1111    ---->  Binary Equivalent of 255
         1000 0000    ---->  Binary Equivalent of -128

       ---------------------------->   ^

       // 0111 1111   --->   127
```

so O/P is 127

```
2.int main()
{
        int x = -1;

        printf("%u,%x,%d",x>>1,x<<4,(unsigned)x>>1);

}
```

As x is negative number to find its equivalent binary we perform two's compliment on it.

0000 0000 0000 0000 0000 0000 0000 0001 (Binary equivalent of 1)

1111 1111 1111 1111 1111 1111 1111 1110 (1's compliment)

1111 1111 1111 1111 1111 1111 1111 1111 (2's compliment)

Know we perform shift operations

1111 1111 1111 1111 1111 1111 1111 1111

x>>1

```
               o/p - maximum range of the integer variable i.e ( 4294967295 for
32 bit integer )
        ( right shifted by 1 is  no change as signed bit is not shifted so decimal
equivalent will be
          the maximum range of the integer variable )


                        x<<4
               1111 1111 1111 1111 1111 1111 1111 0000
            o/p - (%x) Hexadecimal equivalent of above is ffffffff0


                        x>>1
         x is left shifted by 1 considering it as unsigned final shift will be
               0111 1111 1111 1111 1111 1111 1111 1111
        (decimal equivalent of above bits will be 2147483647 )
```