v.v.i

6. Bitwise Operator.

C supports 6 bitwise operators. They are

1. ~ (tild) one's complement operator

2. >> Right shift operator

3. << Left shift operator

4. & Bitwise AND.

5. ¦ Bitwise OR.

6. ∧ (cap) Bitwise XOR.

All the above operators converts the given decimal number into binary and then the operation is performed. As the operations are performed on the bits (binary), they are called bitwise operators

The following table gives the conversion from decimal to binary for 4 bit representation.

2^3	2^2	2^1	2^0	
8	4	2	1	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

1 0 0 1	9
1 0 1 0	10
1 0 1 1	11
1 1 0 0	12
1 1 0 1	13
1 1 1 0	14
1 1 1 1	15

i) Bitwise one's complement operator (~)

After converting decimal to binary, all one's present in the number are changed to zero and all zeroes to one. Sign width will not be affected.

eg.

Sign bit

Given -	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	=> 7
=> one's complement result	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	=> 32760.

```
main ( )
{
  int x = 7, y;
  y = ~(x);
  printf ("x = %d\n", x);
  printf ("y = %d\n", y);
}
```

```
main ( )
{
  int x = 7, y
  y = ~(7);
```

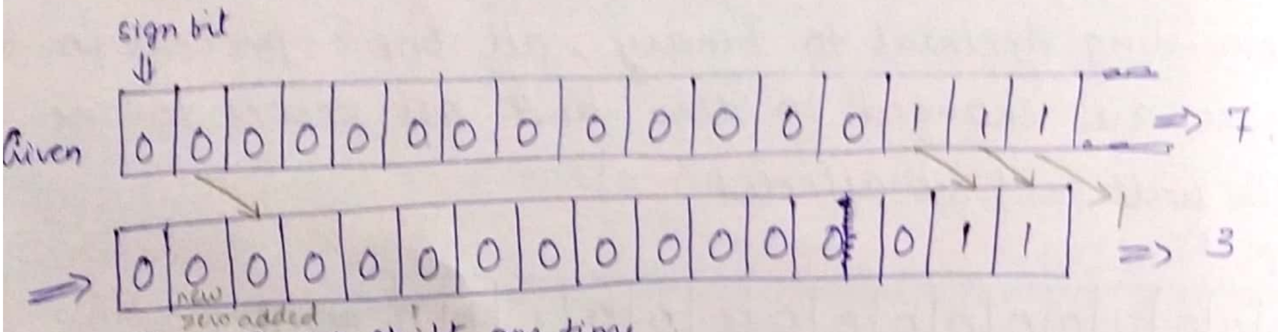
Output

```
x = 7
y = 32760
```

ii) Right shift operator (>>)

It shifts each bit in the operand to the right. The number of times, the bits are shifted depends on the number following the operator.

eg. x >> 3, will shift the bits three places to the right side. As the bits are shifted, blank space is created on left hand side which is filled with zero. Right shift by one place is equivalent to integer division by 2.



Result of Right shift one time

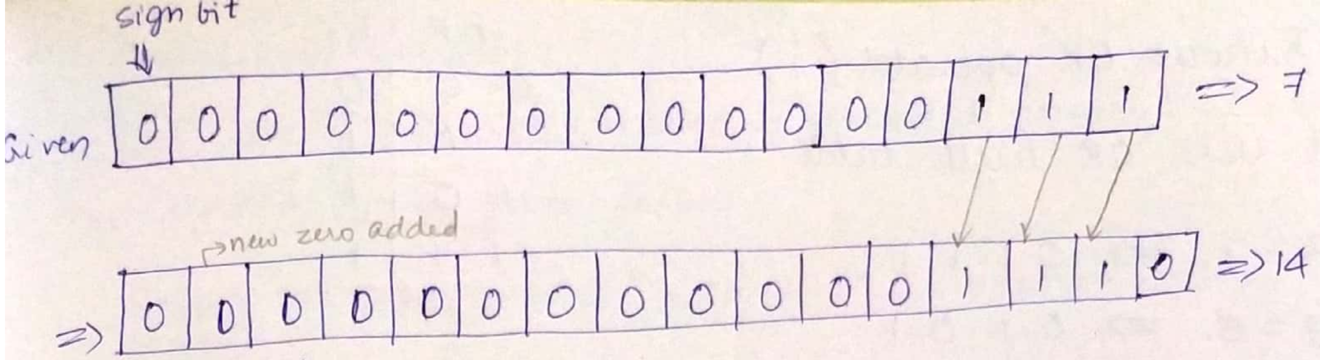
```
main ()
{
  int x = 7, y;
  y = x>>1;
  printf(" x= %d\n", x);
  printf(" y= %d\n", y);
}
```

Output

```
x=7
y=3
```

iii) Left shift operator (<<)

It is similar to right shift operator. Only difference is bits are shifted to the left side and zeroes are added to the right side. Left shift operator is equal to multiplication by 2.



Result of ~~the~~ left shift one time

```
main ()
{
  int x = 7, y;
  y = x << 1;
  printf (" x = %d\n", x);
  printf (" y = %d\n", y);

}
```

output
x = 7
y = 14

iv) Bitwise AND operator (&)

It uses AND truth table.

0	0	0
0	1	0
1	0	0
1	1	1

```
x = 7     =>  0111
y = 5     =>  0101
              ----
z = Result    0101  = 5
```

```
main ()
{
  int x = 7, y = 5, z;
  z = x & y;
  printf (" Bitwise AND result = %d\n", z);

}
```

0111
0101
0101

Output

Bitwise AND result = 5

v) Bitwise OR operator (¦)

It uses OR truth table

OR TT

0	0	0
0	1	1
1	0	1
1	1	1

```
x = 7    =>  0 1 1 1
y = 5    =>  0 1 0 1
             -------
z = Result   0 1 1 1   = 7.
             -------
```

```
main ( )
{
 int x = 7, y = 5, z ;
  z = x ¦ y ;
  printf(" Bitwise OR result = %d\n", z);
}
```

Output

Bitwise OR result = 7.

vi) Bitwise XOR operator (^)

It uses XOR truth table.

XOR TT

0	0	0
0	1	1
1	0	1
1	1	0

```
x = 7    =>  0 1 1 1
y = 5    =>  0 1 0 1
             -------
z = Result   0 0 1 0   = 2
             -------
```

```
main ( )
{
 int x = 7, y = 5, z;
 z = x ^ y;
printf("Bitwise XOR result = %d\n", z);
}
```

Output - Bitwise XOR Result = 2

(ii) Conditional Operator (?:)

It uses 3 operands, hence it is called ternary operator.

The syntax is as given below.

```
Result = expression 1 ? expression 2 : expression 3 ;
                ⇓
            condition
            True case              False case
```

If expression 1 is true, then expression 2 will be evaluated as result, otherwise expression 3 will be the result.

eg

```
main ()
{
  int x=10, y=20, r;
   r= (x>y) ? 500 : 1000;
   printf (" r= %d\n", r);
 }
```

Output

```
r = 1000.
```

```
Q2. /* To perform basic arithmatic operation */
    # include <stdio.h>
    main ( )
    {
      int a = 4 , b = 5, add, sub, mul, div, modu;
      printf
       add = a+b;
       sub = a-b;
       mul = a*b;
      div = a/b;
      modu = a % b;
      printf ("Sum of %d and %d is %d\n", a, b, add);
      printf (" Difference of %d and %d is %d\n", a, b, sub);
      printf (" Product of %d and %d is %d\n", a, b, mul);
      printf (" Division result of %d and %d is %d\n", a. b, div);
      printf ("Modulo division result of %d and %d is %d \n",
                                                     a. b, modu);

    }
```

Output :

```
Sum of 4 and 5 is 9
Difference of 4 and 5 is -1
Product of 4 and 5 is 20
Division result of 4 and 5 is 0
Modulo division of 4 and 5 is 0.
```