# Operators

- **+ - * / %** (Arithmetic)
- **=** (Assignment)
- **++ --** (Pre and Post increment/decrement)
- **+= -= *= /= %=** (compound assignment)
- **< <= > >=** (Relational )
- **== !=** (Equality)
- **&& || !** (logical )
- **? :** (ternary or conditional)

# Bitwise Operators

- **~** (One's complement)
- **>>** (Right Shift)
- **<<** (Left Shift)
- >>> (Unsigned right shift)
- **&** (Bitwise AND)
- **|** (Bitwise OR)
- **^** (Bitwise XOR [Exclusive OR])

# One’s Complement Operator

**~**

On taking one’s complement of a number, all 1’s present in the number are changed to 0’s and all 0’s are changed to 1’s. e.g. one’s compliment of 1010 is 0101.

# Right Shift Operator

>>

It requires two operands.It shifts each bit in its left operand to the right. The number of  places the bits are shifted depends on the number I.e. its right operand.

As the bits are shifted to the right ,blanks are created on the left. These blanks must be filled  somehow. They are always filled with **zeros.**

# Right Shift Operator

64 >> 1  gives 32

64 >> 2 gives 16

i.e.

64 >> 1 is equivalent of

$64/2^1$

64 >> 2 is equivalent of

$64/2^2$

# Left Shift Operator

**<<**

Bits are shifted to the left and for each bit shifted a **zero** is added to the right of a number

64 << 1 gives 128

i.e.

64 << 1 is

$64 * 2^1$

# >>> (Unsigned right shift)

It is the same as the signed right shift, but the vacant leftmost position is filled with 0 instead of the sign bit.

# Bitwise AND Operator

**&**

It operates on two operands.While operating upon these two operands they are compared on a bit-by-bit basis.

| First bit | Second bit | Result |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

# Bitwise OR Operator

|

| First bit | Second bit | Result |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

# Bitwise XOR Operator

^

| First bit | Second bit | Result |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |