Bitwise operaters

Miscellanuous Concepts :-

→ They perform operations on the binary form of the numbers.

① → Bitwise & (AND)

$a = 4$ $b = 8$

100 1000

0100

&1000

0000 $= (0)_{10}$

0 & 1 = 0

0 & 0 = 0

1 & 0 = 0

1 & 1 = 1

Can be verified in the code as well

② Bitwise | (OR)

$a = 4$ $b = 8$

100 1000

0100

1000

$(1100)_2 = (12)_{10}$

0 | 1 = 1

0 | 0 = 0

1 | 0 = 1

1 | 1 = 1

8+4 = 12

③ Bitwise ^ (XOR)

[Exclusive OR]

if bits same = 0

If bits different = 1

$0 \wedge 0 = 0$ $\quad$ $0 \wedge 1 = 1$

$1 \wedge 1 = 0$ $\quad$ $1 \wedge 0 = 1$

$a = 4$ $\quad$ $b = 8$

100 $\quad$ 1000

$$\begin{array}{r} 0100 \\ 1000 \\ \hline (1100)_2 \end{array} = (12)_{10}$$

④ Bitwise $<<$

leftshift

Actually it shifts the binary form by i times

$a << b$

$ans = a * 2^b$

$8 << 1$

1000

$10000 = (16)_{10}$

$= 8 \cdot 2^1$

$= 8 \cdot 2$

$= 16$

$n = 4 \; (100)_2$

$n << i$

$n << 1$

~~1000~~ 100

$(1000) = (8)_{10}$

New zero added

⑤ $>>$ Bitwise

Rightshift

$a >> b$

$ans = a / 2^b$

$ans = \frac{a}{2^b}$

$10 << 1$

$\frac{10}{2^1} = \frac{10}{2} = (5)$

Hence proved

$10 << 1$

1010

1010

$(0101) = (5)_{10}$

$8 >> 2$

1000

$0010 \; (2)_{10}$

Operater Precedence

operaters		Precedence
!, +, -	Unary operaters first	R to L
*, /, %	second	L-R
+, -	third	L-R
<, <=, >=, >	fourth	L-R
==, !=	fifth	L-R
&&	sixth	L-R
||	seventh	L-R
=	last	R-L

Here → Bitwise operaters exists

we can overwrite all these operations using the bracket.

If operaters of same precedence comes the the associative property applies

$4 * 5 \% 2$

$\longrightarrow$

$*$ $\%$

first second

$L \rightarrow R$

Scope

Area of accessibility/usability of the variables

① Local

② Scope (Global)

① Local
→ if-else
→ functions
→ block of code { }
→ Loops

② Global → accessible everywhere

Data type modifiers

Change meaning of datatypes

long

short

long long

signed

unsigned

Suppose

int x = 2 (4 bytes)

{11011011 _ _ _ _ _ _ _ upto 32

MS

0 → +ve, 1 → −ve

31 bits

Combinations = 2^{31}

0 to $2^{31}-1$ | -2^{31}

If 2^{32} comes int does not have capacity to store the value because it have the capacity of

-2^{31} to $2^{31}-1$

So we can change the capacity

→ lon

ld ouble

lon ives extra 4bytes

int 64 bits -2^{63} to $2^{63}-1$

→ sh 2 bytes

wil ow the capacity to 2bytes.

→ long 8 bytes

→ si

is signed by default

si means can save +ve values / -ve values

→ ed

only Positive values

(MS) - - - - - - - - - - to 32 bits

need of

st significant

So size increases upto

$[-2^{32}$ to $2^{32}-1]$ which is

of

$[-2^{31}$ to $2^{31}-1]$