Operators

08 June 2023 21:06

Arithmetic (Inc / Decrement) operators.

2 types → pre & post.

pre means → pehle (Inc / Dec) hoga fir task hoga.

post ——— → pehle (task) hoga fir (Inc / Dec) hoga.

```
int a = 10
int b = ++10;
int c = b--;
```

// a: ~~10~~ 11, b: ~~11~~ 10, c: 11

11, 10, 11

Bitwise Operators. : They are very fast as in their binary form

op1 B.O op2 → o/p.

{ '&', '|', '~', '^', '<<', '>>' }

① Bitwise AND '&'

similar to logical AND.

If All 1 → 1

else → 0

5 & 7 → Binary form

① long Division.

5 = ( )₂ → base of number system.

longer

```
2 | 5
2 | 2   1
    1   0
```

5 = $(101)_2$

② shortcut

0	0	1	0	1
16	8	4	2	1

5 - 4 = 1

7 = ( )₂

```
2 | 7   1
2 | 3   1
    1
```

(111)

2 | 3 | 1 | ...

→ (12, 14, 17, 19, 33)

with long division
also with shortcut method.

5 & 7 ⇒ (5)

5 → 101
7 → 111
res 101

13 & 7 & 17 = (1)

13 → 01101
7 → 00111
17 → 10001
res 00001 = (1).
10011

BITWISE OR '|' → very similar to logical OR.

→ Any 1 → 1,
→ All 0 → 0, else 1.

5 | 7 = 7.

101
111
res 111

13 | 7 | 17 | 19
= (31)

01101
00111
10001
10011
11111

'~' Bitwise NOT → similar to logical NOT

0 → 1
1 → 0.

~ 101
010.

'^' Bitwise XOR operator

if odd no. of 1's are present
→ 1. (xor of every no. with itself is always 0)
else 0.

$5 \wedge 7 = 2$

```
 1 0 1
 1 1 1
-------
 0 1 0  → (2)
-------
```

$13 \wedge 7 \wedge 17$

```
 0 1 1 0 1
 0 0 1 1 1
 1 0 0 0 1
-----------
 1 1 0 1 1  → (27)
```

$5 \wedge 5$

```
 1 0 1
 1 0 1
-------
 0 0 0
-------
```

$7 \wedge 7$

```
 1 1 1
 1 1 1
-------
 0 0 0
-------
```

Left shift '<<'

op1 << op2 — leftwards

shift every bit of op1 ^, op2 times

$5 << 2$

$5 \times 2 \times 2$

```
 1 0 1 0 0
16 8 4 2 1
```

(20)

$7 << 3$

$7 \times 2 \times 2 \times 2$

(56)

```
 1 1 1 0 0 0
32 16 8 4 2 1
```

for left shift → if $x << y$.

then $\Rightarrow x = x \times 2^y$.

Right shift >>

... rightwards every bit

Right shift //

shift rightwards every bit

op1 >> op2

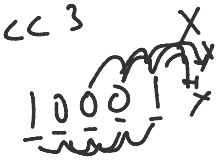
shift every bit of op1 rightwards, op2 times.

101 -> 5 >> 2.

0 0 1

(1)

17 << 3



0 0 1 0 0

(2)

if x >> y

$$x = \frac{x}{2^y}$$

(~13) → (-14)

int → 4 bytes. → 32 bits.

~ (1101)

(0010)

~ (0 0 0 .... 1 1 0 1)

32 bits.

signed int

-ive numbers (1 1 1 ..... 0 0 1 0)

How are -ive nos. stored.

→ 2's complement of their +ive counterparts binary representation

0101 = (5)$_2$

(----)

(-5) → 2's complement of (6101)

(-5) → 2's complement of (6101)

Method for 2's complement:

→ first we find out 1's complement
negation of original no.

0101 → 1's complement → 1010

1010 → 1's complement of (5).

Now for 2's complement we add 1 to it.

1010 + 1 → 1011 → (-5)

signed int

first bit represents sign (+ive or -ive)

first bit (extreme left bit)
Most Significant Bit (MSB)

for +ive MSB = 0.
— -ive — = 1

---

range of int (4 bytes)

-2---8 to 2---7

→ [-2,147,483,648 to 2,147,483,647] → range of (int) (4 bytes)

not this: [-2,147,483,647, 2,147,483,648]

~13 → -14

~ 0 0 0 0 1 1 0 1

1 1 1 1 0 0 1 0

-ive

(-14)

-128 = 10000000

128 is 01111111 + 1

10000000

01111111 + 1

-ive

(-1)

cout << (~1) → (-2)

~2 → -3

~3 → -4

Qus. You are given N numbers & in those N numbers each no. appears twice, except 1 no. which appears only once. Find out that no.

{1, 2, 3, 1, 2}

1^1 = 0

^2 = 0

3^0 = 3

1 1
0 0
(1)

Any no. xored with 0 is no. itself.

x^x^3^x^x → 3^0 → (3)

0 0

You are given a number & you have to tell how many 1s are present in Binary Rep. of that no.

13 = 1101 → (3)

7 = 111 → (3)

8 = 1000 → (1).

13 & 1 →

1101
0001
001 → (1)

1000
1
0000 → 0.

c = 0

13 & 1 → non zero → last bit is set.

c++;

13 >> 1

Compound Arith operators.

a = 10;

(1) a++

-1

-ive no.

128 → 25

-ive

-[1 1 1]

(-8 + 0 7)

-128 to 127 (7)

1000 ~~a++~~ 1000 times

same: a = a + 1000;
a += 1000

u can use this syntax with Arithmetic & Bitwise operators.

You are given a number & you have to set the last bit off (that is ON) from RHS.

13 → 1101 → 1100 → 12
1000 → 8.
0000
0

10001
17 → 18

13 → 12
12 → 8

10:55 PM

c += b / 2

1100
1

1100
1011
1000
0

11
1

11
1

nzero

10 << 2
1000

1

11
01
10

Bitmasking

(2) **Bit Masking**

1100.

we can find out the set bit count from right.

ct = 2

mask = 1 << ct.

1 << 2.

1 _ _ 1

(100)

num & mask → (num)

(3) 1100 → 12

12 & 11 = (12) ↓

1 0 1 1 → (11)

1 1 0 0

1 0 0 0

num = num & (num - 1)

13

1101

1100

1100 → (12)

num = 123 ,

rev = 321

num = 1 2 3

Hundredth. Tens Ones

extract last num.

add it the curr value.

→ mul. curr value by 10

→ div num 10

rev = 0. 3 32 +2

320 = 321 0

123 / 10 = 12

12 % 10 = 2

123 % 10 = 3

12 / 10 = 1

cout << rev / 10

1 % 10 =

1 / 10 = 0. X

---

while (true)
{
task
}

→ infinitely

for (; ;)
{ task;
}

→ infinite loop.

Invalid X

Valid ✓