BITWISE OPERATORS & LOOPS

Page No.
Date

* If we want to perform any operation in bit-level, then we can use bitwise operators.

AND → & OR → | NOT → ~ XOR → ^

Truth Table

Input a	Input b	Output
0	0	0
0	1	0
1	0	0
1	1	1

Input a	Input b	Output
0	0	0
0	1	1
1	0	1
1	1	1

Input a	Output
0	1
1	0

Input a	Input b	Output
0	0	0
0	1	1
1	0	1
1	1	0

Trick
→ Same i/p bits → 0
→ Diff i/p bits → 1

* To cancel out two numbers, [duplicate no's] just use XOR (^).
  Ex:- 5 ^ 5 = 0

Imp. Q. Find unique element.
i/p → 2, 3, 4, 5, 2, 4, 3
2^3^4^5^2^4^3 (it cancel out same inputs)
↳ o/p = 5

↳ 5 ^ 5
binary of 5 = 00000000 00000000 00000000 00000101
5 ^ 5 = 00000000 00000000 00000000 00000101
0
00000
0
(That's why ans is 0).

↳ 5 ^ -5

Binary of 5 = 00000000 00000000 00000000 00000101

Binary of -5 = 11111111 11111111 11111111 11111011

11111111 11111111 11111111 11111010

-ve no.

Binary of this value

↳ 2's comp

↳ 1's comp

00000000 00000000 00000000 00000001

+1 → +1

00000000 00000000 00000000 00000010

It's the binary of 2

Ans = -2

Left shift & Right Shift Operators.

"<<" → LeftShift

∴ Shift the value by 1 bit towards left.

">>" → Right shift

∴ Shift the value by 1 bit towards Right.

Leftshift:-

Ex:- int a=2. 00000000 00000000 00000000 00000010

a << 1

Lost Bit

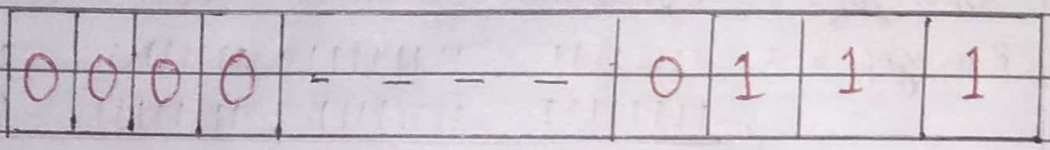
∴ Each bit will shift towards left by 1 bit.

empty space filled with "0"

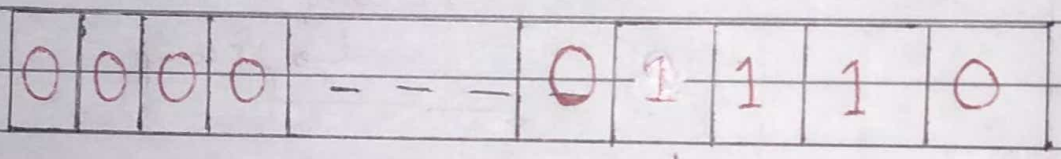
00000000 00000000 00000000 00000100

a<<1 = 4

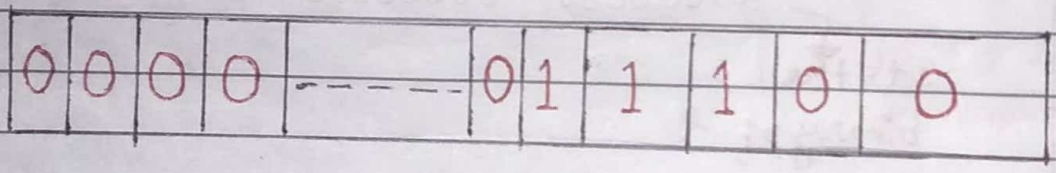
Ex:- a=7, a<<1

7 → 

<<1

14 → 

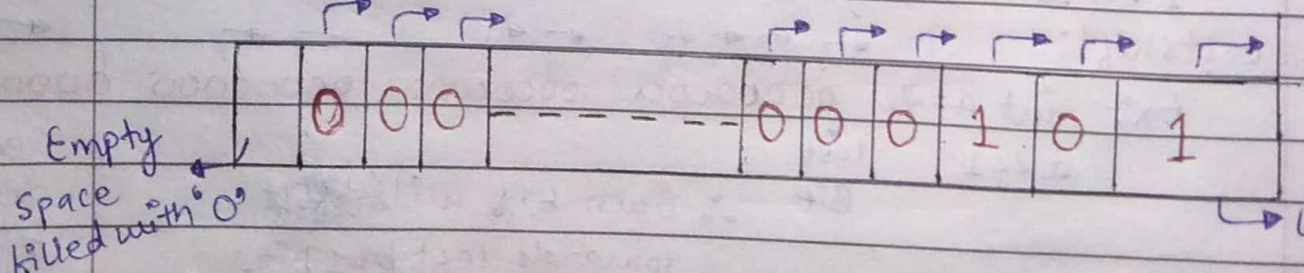
<<1

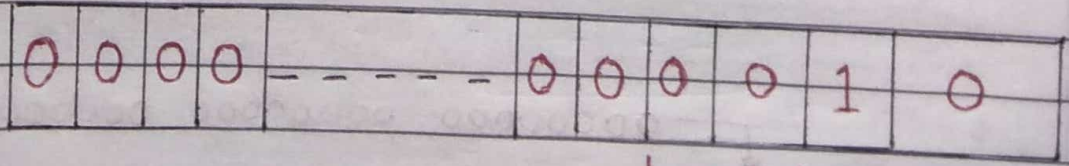
28 → 

General formula for left shift = $(a << n) = a * 2^n$

Right shift:-

Ex:- a=5; a>>1





5>>1=2

Page No.
Date

Ex:- a = 50; (a>>1) = 25

a = 100; (a>>1) = 100/2 = 50

∵ Jab ek baar Right shift kiya hai by 1 bit → Tab divide hoga 2^1 se.

(n>>1) → Divide hoga by 2^1 se

∵ Jab Right shift hoga by 2 bit → Divide hoga 2^2 se

(n>>2) → Divide by 2^2

Ex:- n = 100

(n>>2) = $100/2^2$ = 100/4 = 25

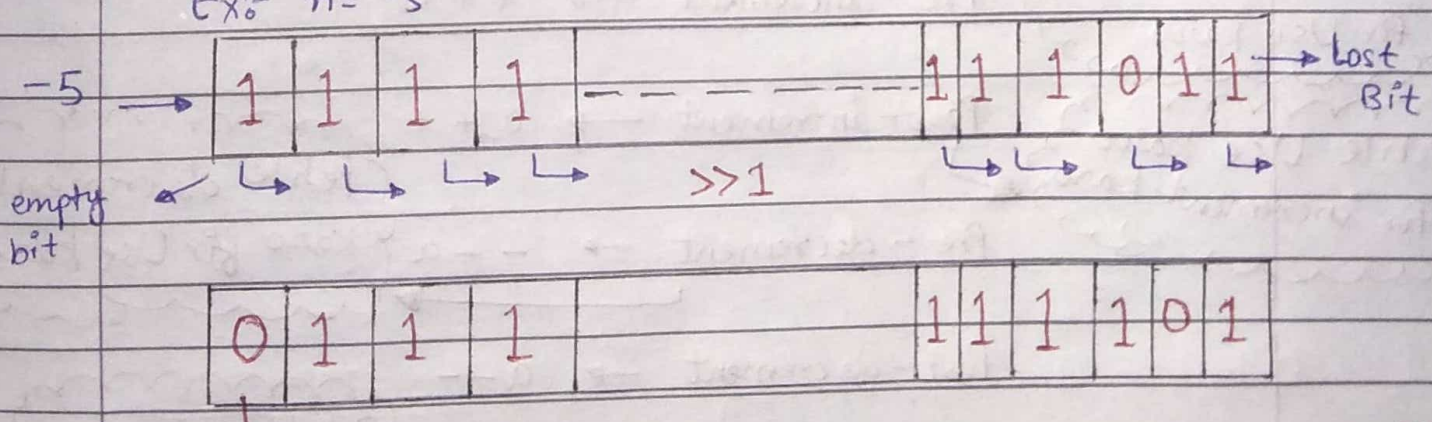
★ It's not generalise statement. It contains a catch

[∵ Kisi bhi no. ko n times right shift krange means the no. is divided by 2^n (compiler dependent statement)]

Ex:- n = -100;

(n>>1) = -50 [∵ But it's different in behind the scene]

Ex:- n = -5



[∵ Sab -ve no. is converted into a +ve large no]

∴ Signed integer mein agar -ve value insert karke usko right shift karte hain to 'compiler handle kar paa raha hai'

∴ Unsigned integer mein -ve value insert karke usko right shift karte hain to 'ek positive large no. mil raha hai'

Ex:- unsigned int n = -100;
(n>>1) → 2147483598 (output)
↳ +ve large no.

Imp:

Garbage value:-

∴ Agar kisi bhi number ko left shift kar raha hu by a -ve value of bit ~~shift~~, to us case mein 'garbage value milti hai.'

Ex:- int n = 10;
cout<< (n<<-1); → Get garbage value in output

∴ Also gives a 'warning' but never produce error.

Pre/Post Increment/Decrement operator

Pehle increment karo fir Use karo

Pre - increment → ++a

Pehle Use karo fir increment karo

Post - increment → a++

Pre - decrement → --a

Pehle decrement karo fir Use karo

Post - decrement → a--

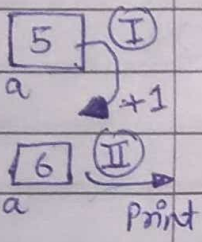
Pehle Use karo fir decrement karo

Page No.
Date

Ex:-

```
main()
{
  int a=5;
  cout << (++a);
}
```

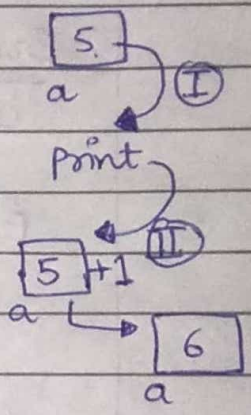
output:- 6



Ex:-

```
main()
{
  int a=5
  cout<<(a++);
}
```

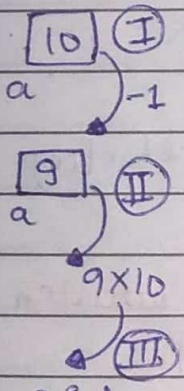
output = 5



Ex:-

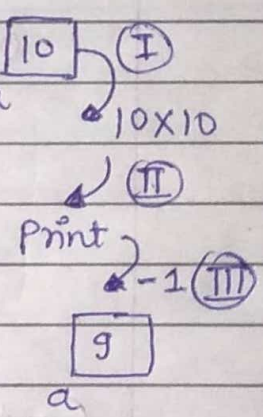
```
main()
{
  int a=10;
  cout<<(--a)*10;
}
```

output = 90



Ex:-

```
main()
{
  int a=10;
  cout<<(a--)*10;
}
```



Q

```
int a = 10;
(++a) * (a++) ⟶ 121 / 132
(a++) * (++a) ⟶ 120
```

} why?

Page No.
Date

Break & Continue Keywords :-

Break :- To stop the continuous flow of any loop, we can use break keyword.

Ex :-
```
for (int i=0; i<3; i++){
    if (i==2){
        break;
    }
    cout<< i <<endl;
}
cout<<"Outside the for loop";
```

Output :-
```
0
1
Outside the for loop
```

Dry Run

i=0, i<3 (T)
i==2 (F)
cout → 0
i++ →
i=1, i<3 (T)
i==2 (F)
cout → 1
i++
i=2, i<3 (T)
i==2 (T)
→ break
cout → Outside the for loop

Continue :- It will skip the further execution written after continue keyword and moves towards the next iteration.

Ex :-
```
for (int i=0; i<5; i++){
    if (i==1){
        continue;
    }
    cout<< i;
}
```

Output :-
```
0
2
3
4
```

Dry Run

i=0, i<5 (T)
i==1 (F)
cout → 0
i++
i=1, i<5 (T)
i==1 (T)
continue
i++
i=2, i<5 (T)
i==1 (F)
cout → 2
i++
i=3
i<5 (T)
i==1 (F)
cout → 3
i++
i=4, 4<5 (T)
i==1 (F)
cout → 4
i++
i=5, 5<5 (F)
loop end

Variable Scoping :-

(i) Local Variable :- Only accesible within the scope it's defined. We can not access any local variable from outside of its scope.

Ex :-
```
for( int i=1; i<3; i++){
    cout<<i;
}
```
∴ Here, 'i' is local variable of that for loop and we can't access it outside the loop.

* We cannot re-define a variable with same name.

Ex :- (I)
```
if(true){
    int a=20;
    if(true){
        cout<<a;
    }
}
```
Output :- 20

Ex :- (II)
```
if(true){
    int a=20;
    if(true){
        int a=30;
        cout<<a;
    }
}
```
Output :- 30

* We can re-define a variable in nested code blocks.

* Every time, compiler first check the nearest scope to get the variable just like Ex (II).

* If compiler can't able to find a variable in its own scop, then it will find the variable outside to its parent scope just like Ex (I).

In Ex (II), both 'a' are different memory allocations.

(ii) Global Variable :-

They are declared at the top of the program outside of all the functions or blocks. Then we can use them anywhere inside the program.

HW 8 Why Global variables are considered as a bad practice?

Ans Their value can be changed by any function which results in the reduce modularity and flexibility of the program.

Ex:- If 2 modules share a global variable, we can't modify one without considering how that affects the other.