▶ Bitwise operators and loops.

→ Bitwise operators. → (Bit level)

① Bitwise - AND (&)

② Bitwise - OR (|)

③ Bitwise - NOT (~)

④ Bitwise - XOR ~~[illegible]~~ (^)

These operators perform operations on Bit level.

TRUTH TABLE

AND (&)

a	b	o/p
0	0	0
0	1	0
1	0	0
1	1	1

OR (|)

a	b	o/p
0	0	0
0	1	1
1	0	1
1	1	1

NOT (~)

a	o/p
1	0
0	1

XOR (^)

a	b	o/p
0	0	0
0	1	1
1	0	1
1	1	0

same = 0

different = 1.

eg:

i/p : $\cancel{2} \wedge \cancel{3} \wedge \cancel{4} \wedge 5 \wedge \cancel{2} \wedge \cancel{3} \wedge \cancel{4}$

o/p → 5

Let's check with code snippet.

```
#include<iostream>
 using namespace std;
  int   main(){
       int  a=2;
       int  b = 7;

       cout<<(a&b)<<endl;
       cout<<(a|b)<<endl;
       cout<<(~b)<<endl;
       cout<<(a^b)<<endl;

 return 0;
  }
```

o/p
2
7
-8
5

(i) 2 & 7

2 → 010

7 → 111

~~101~~

010 → 2

(ii) 2 | 7

2 → 010

7 → 111

111 → 7.

(iii) ~7 → 000 ---- 0101

① 111 ---- 010

↓

negative

2's complement

0000 ---- 00101

+ 1

000 ---- 01000.

8

= -8

(iv) 2 ^ 7

010

111

101 - 5

One more snippet:

```
int a=5;
int b = -5;
cout (a^b) <<endl;
```

O/P -2.

Explanation:

5 → 0000------0101

-5 → 1111 — — -1011

①1111----1110

5 = 00--·101

1's comp → 111--·010

2's comp +1

①11--·011

negative → 5

negative 2.

= -2.

How (~a) vs ~(a)

Here, using parenthesis doesn't change the outcome because the bitwise NOT operator has higher precedence than the bitwise shift (<<) operator, is applied first. Therefore, '~num', '(~num)' and '~(num)' all result in the same value, which is -2.

→ Left and Right shift operator.

→ left shift operator → "<<"

→ right shift operator → ">>"

int a=2   left shift "<<"

~~0~~000 . . . 010

syntax

Lost   a<<1   → a ko left shift karo 1 bit.

Now value is

0000 - . . . 0100   (Added zero at last)

Now decimal value = 4

basically when we left shift any value it will be multiplied by 2 in decimal.

eg: int a=5   → 000 - . . 0101.

a<<1   → 000 - - . 1010

O/P → 10   (multiplied by 2)

agi again left shift 5 (twice)

a << 2

↳ 0000 . . . 010100

O/P 2.

Derived formula of left shift

$a << n$

↳ $a * 2^n$.

Right shift ">>"

$5 >> 1$

0 0 0 0 0 . . . 0 0 1 0 1

↳

0000 - - - . 0 0 1 0

Hence
Zero
comes. if any number is being right shifted n times it means that no. is dividing by 2^n

eg: $10 >> 1$

↳ $\frac{10}{2^1} = \frac{10}{2} = 5$ ← o/p

$n > K$

↳ $\frac{\cancel{a}\, n}{2^K}$ (formulas.

eg: $20 >> 2$

o/p 5

But Here's one more catch, what if When we enter negative no

let say -5

1111 11011

do >> 1

↳ 0111 1101

↳ now it's positive.

Note :- If negative number or signed integer the compiler will handle on his own.

and in case of unsigned ~~not~~ we'll get a large int number.

Garbage value concept.

int n=10 → shifted by negative no.

cout<<(n<<-1);

O/P //Garbage value.

pre / post Increment/Deerement operator :

pre increment → ++a

i- pehle increment karo fir use karo

post increment → a++

i- pehle use karo fir increment karo

pre-decrement → --a

i- pehle deerement karo fir use karo.

post-decrement → a--

i- pehle use karo fir decrement karo

inc & dec opr.

eg:
```
int a=4;
cout<<++a<<endl;
cout<<a++<<endl;
cout<<a<<endl;
```

o/p
```
5
5
6
```

```
int b=4;
cout<<--b<<endl;
cout<<b--<<endl;
cout<<b<<endl;
```

O/P 3
3
2.

problem (MCQ) on this operation.

```
main(){
        int a=10;
        Cout<< (++a) * (a++);

}.
```

O/P
121 — replit
132 — vs code.

also for (a++) * (++a) → 120 in replit.
188 in vs code.

Break and Continue.

Break.

```
→ for (int i=0; i<=5; i++){
      if(i==2)
         break;
  }
```

loop se bahar nikal deta hai.

Continue

→ just like tauji who skip the fun part in every shaadi

so, continue skips iterations in loops.

eg:
```
for(int i=0; i<=5; i++){
    if(i==1)          → when condition
                          satisfies
    continue;
    cout<<i;  } ⟷ this part does not
                    work.
}
```

o/p
0
2
3
4
5

```
→ for (int i=0 ; i<=5 ; i++){

        cout <<"Hii folks!!"; cout<<endl;
        cout << "How are you ?";
        continue;
```

o/p 6 bar dono statement print hoga.

```
→ for (int i=0 ; i<=5 ; i++){
        continue;
        cout<<" Hii guys!!" ;
        cout <<endl<<" I am not visible" ;
  }
```

o/p Kuchh bi print nahi hoga.
baat khatam!!.

Variable scoping.

→ Local Variable.

→ Global variable.

→ //we can initialize here global variable defined.

```
main () {  → this is the scope ^ function

      int a=2;
      if(1){
          int a = 5;
          cout << a;
      }
      cout<<endl<< a;
}
```

inner a

o/p 5
2

outer a

Operator precedence

operator	Type.	Associativity.
() [] . ->		left to right.
++, --, !, ~, (type), *, &, sizeof.	Unary operator	right to left.
* / %	Arithmetic operator.	left to right.
+ -	Arithmetic operator.	left to right.
<< >>	Shift operator	left to right.
< <= > >=	Relational operator.	left to right.
== !=	Relational operator	left to right.
&	Bitwise AND operator	left to right.
^	" x-OR operator	left to right.
|	" ~~AND~~ OR operator	left to Right.
&&	Logical AND operater.	left to right.
||	Logical OR operator	left to right.
?:	Ternary opr, conditional	right to left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment operator.	right to left
,	Comma	left to right.

The major thing that we can resolve any operator evaluations using parenthesis (brackets)

eg: $((2*3) + (5/10)) - 2$

$\downarrow$

$6 + 0.5$

$\downarrow$

$6.5 - 2 = 4.5$