Bitwise operator
AND
OR
NOT
XOR

AND-
The truth table for an AND gate, which performs a logical conjunction operation, is as follows:

Input A	Input B	Output (A AND B)
0	0	0
0	1	0
1	0	0
1	1	1

OR (|)-
The OR gate performs a logical disjunction, outputting `1` if at least one input is `1`.

Input A	Input B	Output (A OR B)
0	0	0
0	1	1
1	0	1
1	1	1

NOT (~)-
The NOT gate is a logical inverter that flips the input value.

Input	Output (NOT Input)
0	1
1	0

Take 2’s complement with the sign

XOR(Exclusive OR)(^)
The XOR (Exclusive OR) gate outputs `1` only if the inputs are different.

Input A	Input B	Output (A XOR B)
0	0	0
0	1	1
1	0	1
1	1	0

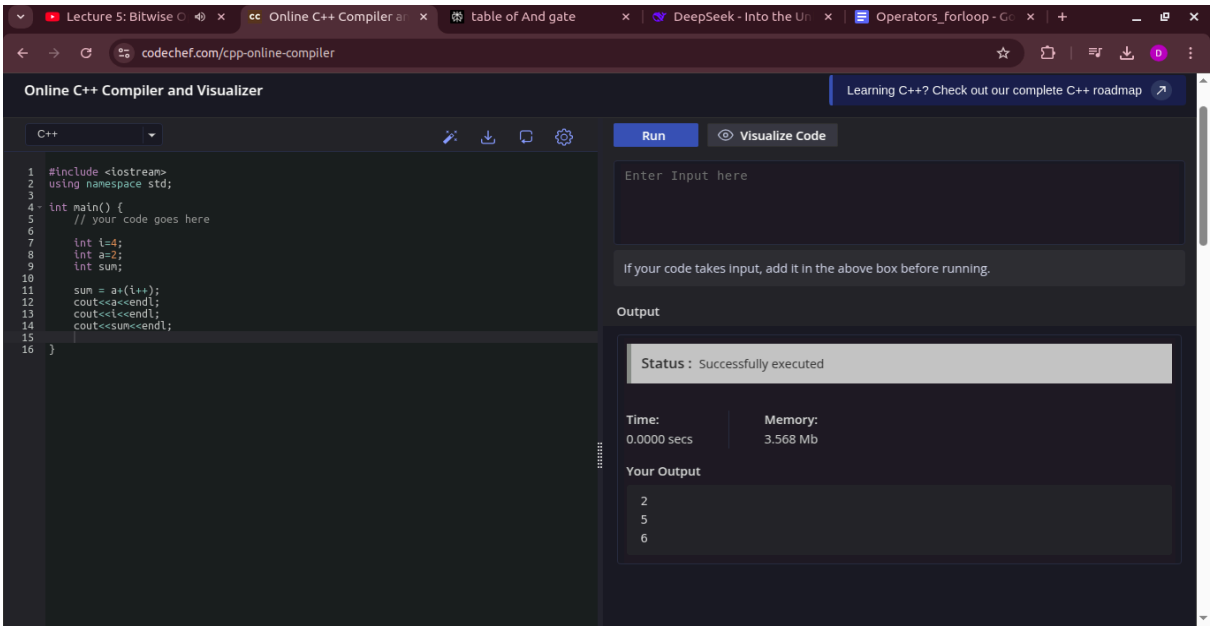
Left Shift and Right shift operator-
Left Shift
5<<1
(5 ko ek baar left shift kr do)
(means we are shifting all the bits by 1 place left)
(Ultimately we are multiplying the number by 2)(**In majority of cases**)
(n*2)

Right Shift-
15>>1
5>>2
(shift the digits by right side)

How to shorten while loop
i=i+1 can be written as i++(post increment) , ++i(pre increment)
i+=1
i = i-1 can be written as i–(post decrement) , –i(pre decrement)
i-=1

Pre/Post/Increment/Decrement-

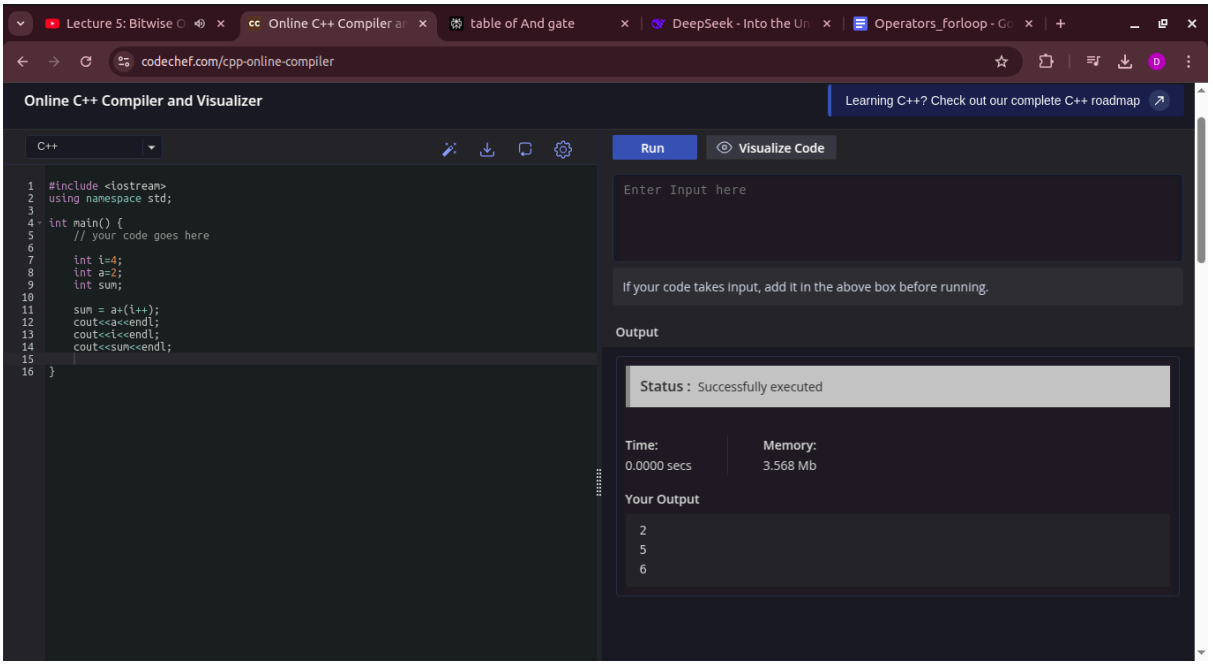
Int i=4;
Int a=i++;
cout<<a<<i<<endl;
4 5
Because a is just used here to increment i , it holds the old value of i
But i has increased by 1
Phle use kro fir value badha do (Post increment)



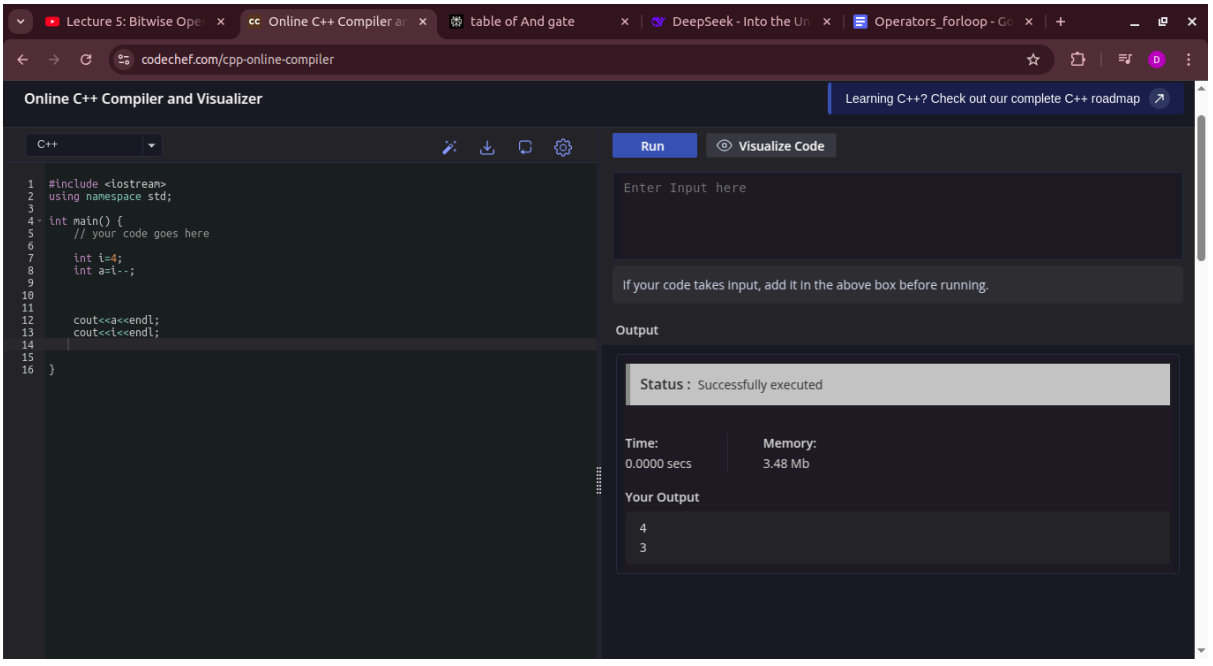

```
int i=4;
   int a=++i;
   int sum;

   sum = a+(++i);
   cout<<a<<endl;
   cout<<i<<endl;
   cout<<sum<<endl;
```

5
6
11



Post increment
Phle use kr lo, fir ghata do



Pre Decrement-
Phle decrement karo fir use karo

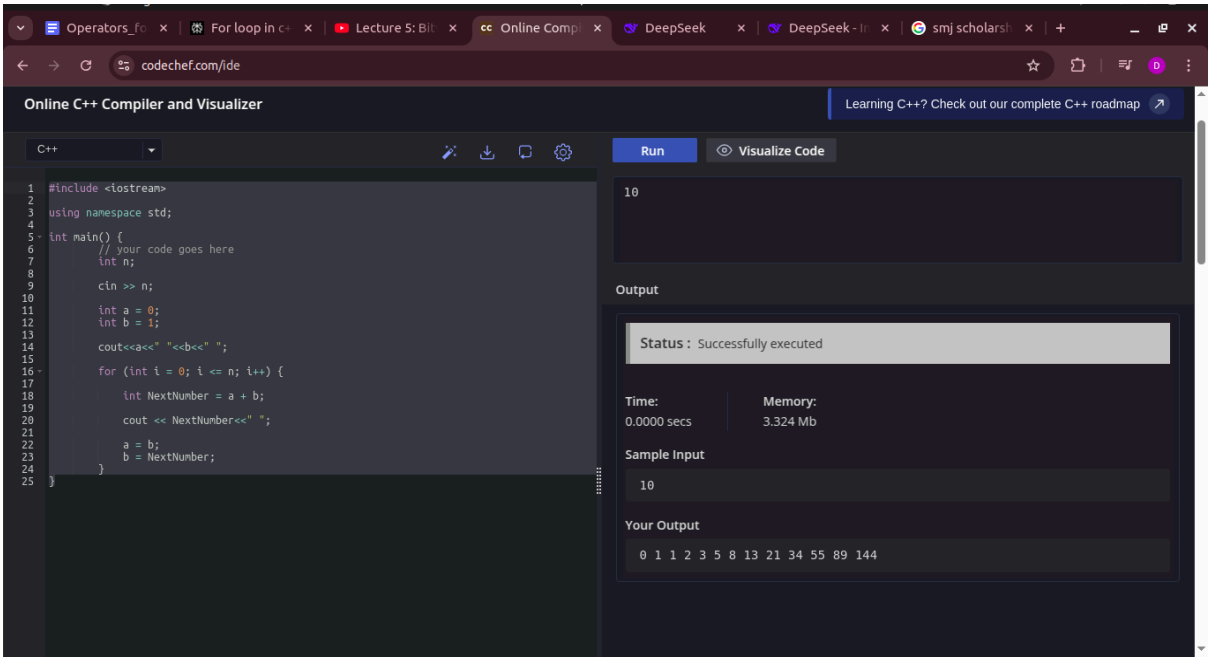
For loop

The basic syntax of a for loop in C++ is:

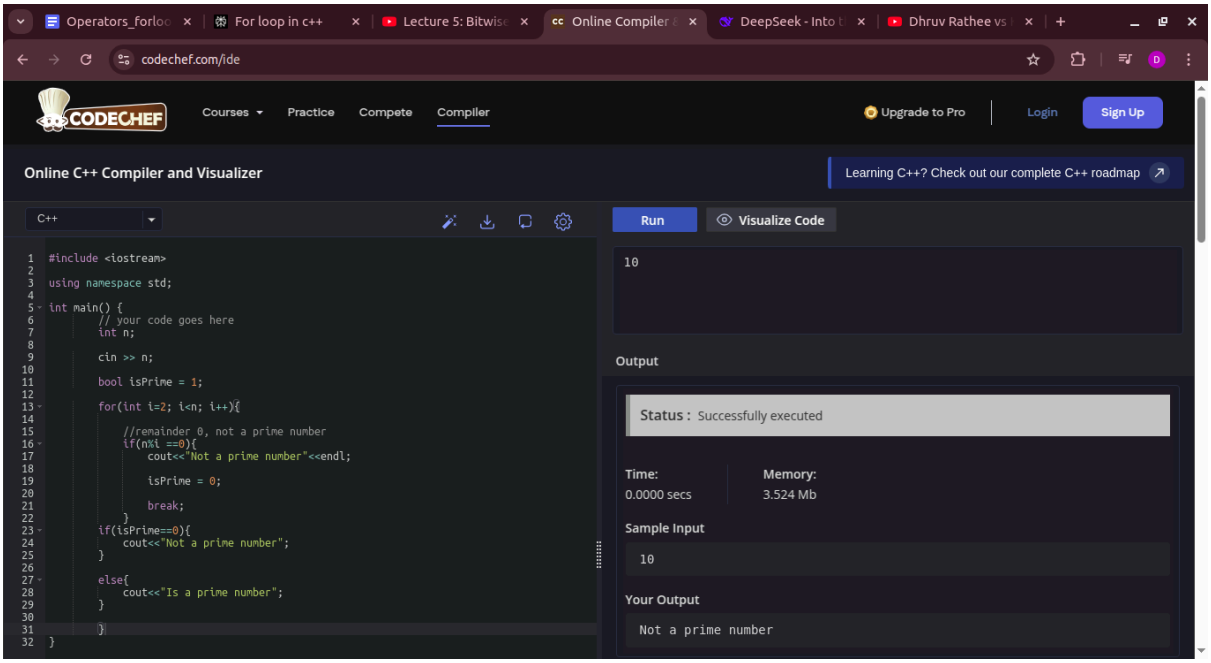
```
for (initialization; condition; update) {
    // code block to be executed
}
```

We can do multiple operations in 1 for loop
Means we can initialize 2 variables, give 2 conditions and update 2 variables

Fibonacci series-



Prime Number or not



Continue keyword

loop control statement that skips the rest of the code in the current iteration and moves directly to the next iteration of the loop.

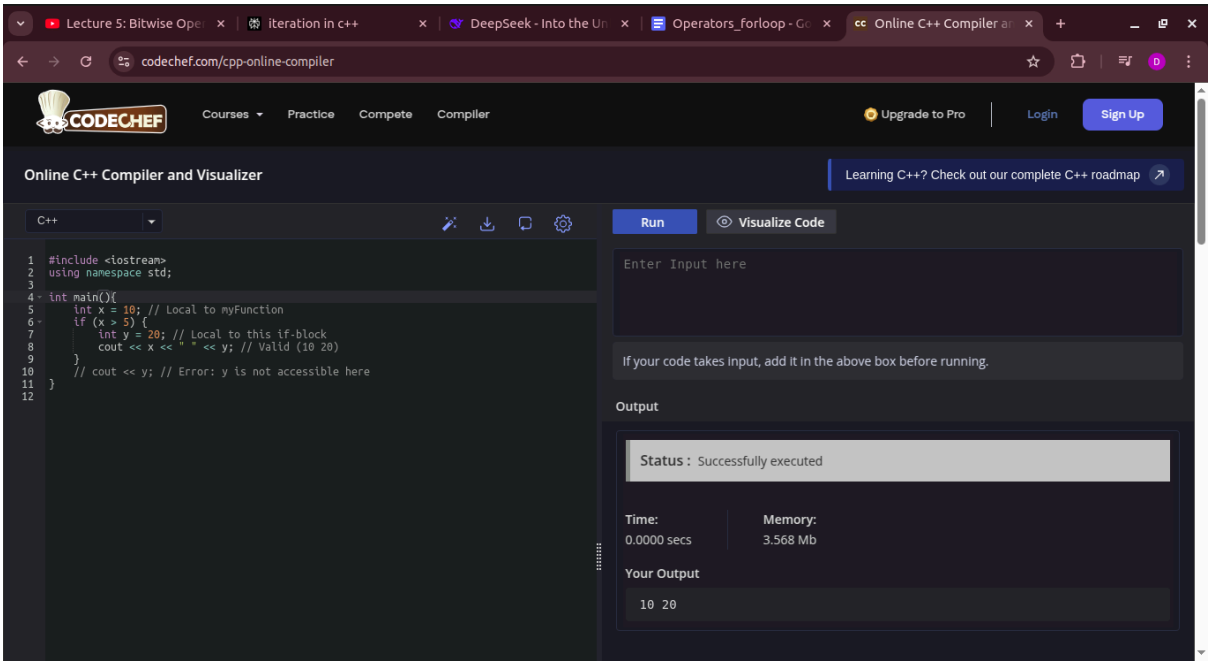
```
if(i & 1)
```

- **What it does**: Checks if the least significant bit (LSB) of `i` is `1`.
- **Purpose**: Tests if `i` is **odd**.
  - If `i` is odd (`1, 3, 5, ...`), `i & 1` evaluates to `1` (true).
  - If `i` is even (`0, 2, 4, ...`), `i & 1` evaluates to `0` (false).

Scope of variables-

1. Local (Block) Scope

A variable declared inside a block (`{}`) is accessible only within that block and its nested blocks.

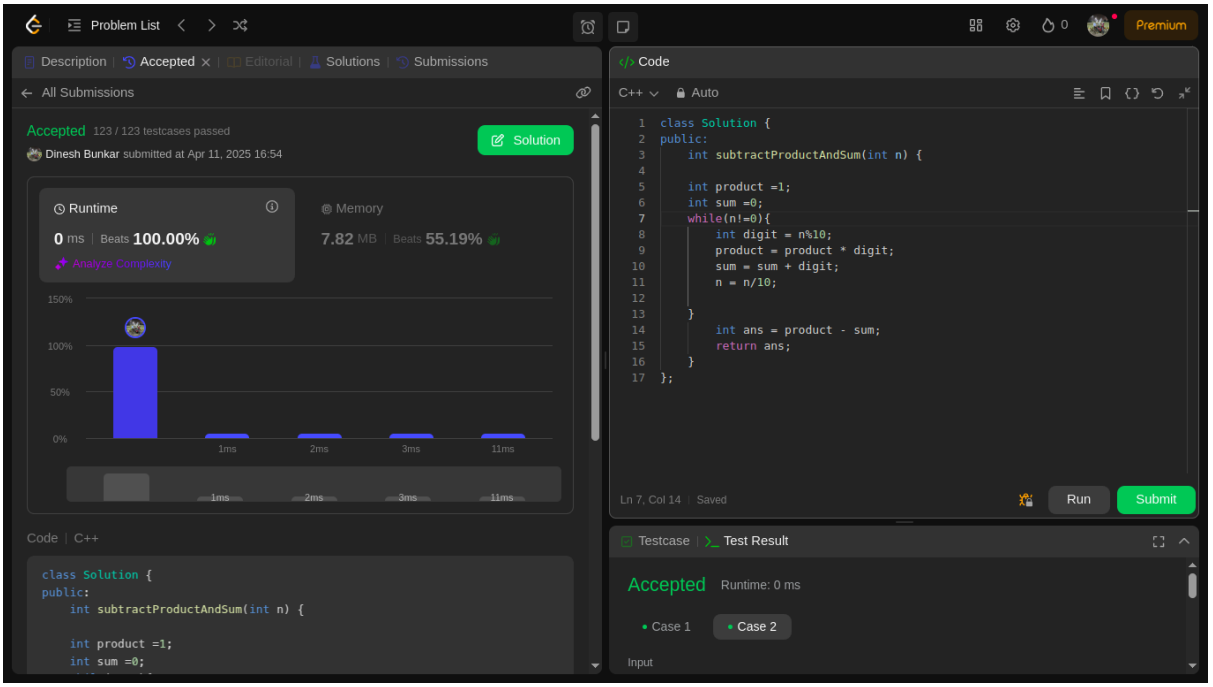


We cannot define 2 variable in single block
If we have another block inside a block then we can use both the variables

Operator Precedence-

It determines the order which operators are evaluated in an expression
It is quite similar to the bodmas rule

Leetcode question
Subtract the product and sum of digits of an integer



No of 1 bits
Solving it by using right shift operator

n>>1 right shifts by 1 digit

- For example, if `n` is `8` (binary `1000`), `n >> 1` becomes `4` (binary `0100`).

Reverse integer

Integer= 123
Reverse integer = 321

123= 3*10^0 +2*10^1+1*10^2

Output should be 321=
3*10^2+2*10^1+1*10^0

Binary and Decimal number system

1. Decimal to Binary

- Divide the number by 2.
- Write down the **remainder**.
- Repeat with the **quotient** until it's 0.
- **Read the remainders in reverse.**

  Example:

  Convert `13` to binary

  13 ÷ 2 = 6 remainder 1

  6 ÷ 2 = 3 remainder 0

  3 ÷ 2 = 1 remainder 1

  1 ÷ 2 = 0 remainder 1

  → **Binary: 1101**

2. Binary to Decimal

- Multiply each bit by `2^position` and sum them.

Storing a number

For eg number is 123
Digits are 1,2,3
We have to save this in same flow

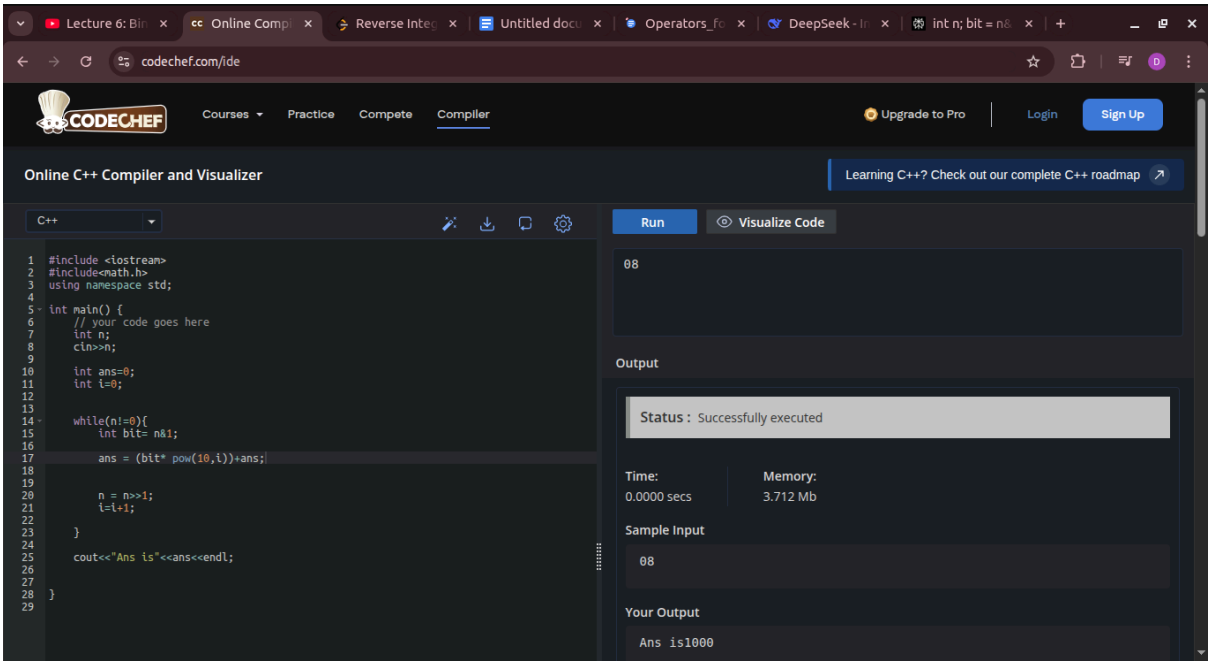
The formula we will use is
ans=0, to intialize it with 0
ans= (ans*10)+digit;
ans= (0*10)+1=1
ans= (1*10)+2=12
ans= (12*10)+3= 123

So by this way we can save the number in same flow

In reverse order

The formula is
ans=0, to intialize it with 0
ans = (digit*10^i)+ans
ans= (1*10^0)+0
ans= 1
ans= (2*10^1)+1=21
ans= (3*10^2)+21= 321

Decimal to binary



Binary of a negative number=

Ignore negative
Convert to binary
Find 2’s complement
(1’s complement +1)