October 11, 2025

BIT MANIPULATION FOUNDATIONS

* Bit Manipulation
→ Foundations
→ Masking
→ Application (5-6 Types)

* Foundations

1. Operators

&, <<, >>, ~, ^, |, −

Example:

→ cout << (1 << 2+3) << endl;

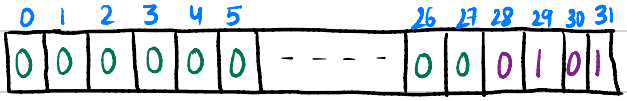
Output = 32

When we write a statement like:

int x = 5

Its binary representation is stored in the memory in the form of 32 bits (4 Bytes)

$X = 5 = (0101)_2$



So, we can use operators to work on these individual bits instead of using the 'x' or its value.

int x = 5 = 0101

int y = 9 = 1001

Now, when we use operators:

x | y (x or y)

Its computation is done like -

x : 0 1 0 1

y : 1 0 0 1

x|y 1 1 0 1

x	y	x|y
0	0	0
0	1	1
1	0	1
1	1	1

We apply the OR operation on each individual bits of x with respective bits of y calculating the result.

x : 0 1 0 1

y : 1 0 0 1

x & y : 0 0 0 1

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

So, that's how computation happens in the machine. The binary output is then converted back to decimal for human understanding

→ What is the output of ~x ?

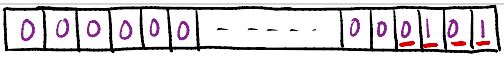
Flip every bit.

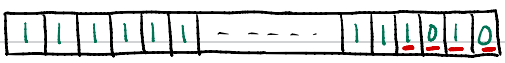
So, x = 5

= 0101

~x = 1010 $= (-6)_{10}$

$(1010)_2 = (10)_{10}$ But the ans will be calculated on the basis of ALL OF 32 BITS. ALL 32 BITS WILL FLIP.

X: 

~X: 

→ XOR (^)

Exclusive OR. If there are even

number of 1 in a column, it will become 0. If there are odd no. of 1's = 1.

```
  0 1 0 1
^ 1 0 0 1
---------
 (1 1 0 0)2 = (12)10
```

→ Shift operators

- x << y (leftshift)

2<<2

So you write this 2 in binary form and shift the bits by 2 towards the left.

$= (0\,0\,1\,0)_2$ (2 ← 2)

Shift this 1 by 2.

$2<<2 = (1\,0\,0\,0)_2 = (8)_{10}$

- 5 << 3

Shift the bits of 5 by 3 towards the left.

$5 = (0\,0\,0\,1\,0\,1)_2$ (3 2 1)

Shift this by 3.

$5<<3 = (1\,0\,1\,0\,0\,0)_2 = (40)_{10}$

→ Rightshift

(101) >>2

Shift this by 2 towards right

⇒ (00 1) 01 → vanishes as 32 bits are full

$= (001)_2$ Ans

When we do x << y

⇒ $x * 2^y$

and x >> y

⇒ $\frac{x}{2^y}$

Ques. Give output of the following :

1. cout << 1 << 2+3 << endl;

Binary representation of 1

$(1)_{10} \Rightarrow (0001)_2$

Now, we have to perform leftshift

1 << (2+3)

= 1 << 5

00(0001) (5 ←)

shift this 1 by 5 towards left.

$\Rightarrow (100000)_2$

Bin→Dec : $2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$= (32)_{10}$

Bit operator has lower precedence. So, put explicit brackets.

2. 10 >> 2

$10 = (1010)_2$

10>>2 = 1010 (→2)

$= (0010)10$ → vanishes

$10>>2 = (0010)_2$ Ans ($2^1\ 2^0$)

$10>>2 = (2)_{10}$

3. $5 << 4$

$5 = (0101)_2$

$= 000(0101)_2$

$= (01010000)_2$ Ans

Bin → Dec $2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$5 << 4 = (80)_{10}$

4. $10 \& 4 | 2$

→ $10 \& 4$

$10 = (1010)_2$

$4 = (0100)_2$

$10 \& 4 = (0000)_2$

$2 = (0010)$

$10 \& 4 | 2 = (0010)_2$ Ans

$= (2)_{10}$

→ 2's complement

$x = 5 = (0101)$

$\sim x = (1010)$ 1's complement of x

$+1$

$= (1011)$ 2's complement of x

$-x$ = 2's complement of x as it is evaluated like $-x = (\sim x) + 1$

Let's say our numbers are made up of just 8 bits. So,

$(11)_{10} = 00001011$

↳ decides if a number is positive or negative. If the first bit is 0 then positive number. If the first bit is 1 then negative number.

Representation of -11

$-11 = (\sim 11) + 1$

$\sim 11 = 11110100$

$\sim 11 + 1 = \quad\quad\quad + 1$

$-11 = 11110101$

Now,

$11 + (-11) = 0$

$-11 \quad 11110101$

$11 \quad 00001011$

$1\ (00000000)$

out of bounds

Ques. What is the decimal value of 11110000 considering only 8 bits.

$x \rightarrow (1)1110000$

↑ indicates negative number

$\sim x \rightarrow 00001111$

$+1 \downarrow$

$\sim x + 1 = 00010000$

$-x = -16$ Ans

Ques. Find the value of x.

$x = 1 << 31$

Initial state:

00000...00001

32 bits

Shift 31 places towards left

x: 10000...0000

32 bits

negative number. Find out the 2's complement.

~x: 0111111...1111

32 bits ↓ +1

~x+1: 1000000...0000

2^{31} (Now put -ve sign)

$-x = ~x+1$

$x = -2^{31}$ (Ans)

```cpp
#include<bits/stdc++.h>
using namespace std;

void solve(){
    cout<< -(1<<31) << endl;
}

signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int t=1;
    //cin>>t;
    while(t--){
        solve();
    }
}
```

Test Case 1 | Input: Enter input | Output: -2147483648 -2^{31} | Desired Output (Optional): 2147483648 2^{31}

```cpp
#include<bits/stdc++.h>
using namespace std;

void solve(){
    cout<< (1<<31) << endl;
}

signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int t=1;
    //cin>>t;
    while(t--){
        solve();
    }
}
```

Test Case 1 | Input: Enter input | Output: -2147483648 (indicates Overflow) | Desired Output (Opti...): 2147483648

```cpp
#include<bits/stdc++.h>
using namespace std;

void solve(){
    cout<< INT_MIN<< " " << INT_MAX << endl;
}

signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int t=1;
```

Test Case 1 | Input: Enter input | Output: -2147483648 2147483647

INT_MAX+1 → overflows to the negative side (-214--)

= INT_MIN

```cpp
#include<bits/stdc++.h>
using namespace std;

void solve(){
    cout<< INT_MIN<< " " << INT_MAX+1 << endl;
}

signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int t=1;
```

Test Case 1 | Input: Enter input | Output: -2147483648 -2147483648

INT_MIN − 1 = INT_MAX

overflows to +ve side

```cpp
#include<bits/stdc++.h>
using namespace std;

void solve(){
    cout<< INT_MIN-1<< " " << INT_MAX << endl;
}

signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int t=1;
    //cin>>t;
    while(t--){
        solve();
    }
}
```

Test Case 1 | Input: Enter input | Output: 2147483647 2147483647 | Desired Output (Optional to C...): 2147483648

Value of INT_MAX + INT_MAX

= (INT_MIN − 1) + INT_MAX

= (INT_MAX + INT_MIN) − 1

= −1−1 = −2 Ans

```cpp
#include<bits/stdc++.h>
using namespace std;

void solve(){
    cout << INT_MIN << " " << INT_MAX <<endl;

    cout<< INT_MIN - 1 + INT_MAX << endl;
}

signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int t=1;
    //cin>>t;
```

Test Case 1 | Input: Enter input | Output: -2147483648 2147483647 -2 | Desired Ou...

AlgoZenith

* Masking

gives us a way to compress info.

Concept of Boolean Array:

An array consisting of 0's and 1's

bool arr[ ] = | 0 | 1 | 1 | 0 | 1 | 1 |

Now, we can see boolean representation of a number in terms of boolean array (32-sized boolean array)

Example of Masking & using bool arr

Let's say we have a set :

$\{2, 5, 7, 8\}$
$0\ 1\ 2\ 3$

Subset: $\{2, 5, 8\}$

position: $(0, 1, 3)$

So we can store this info about position (index) of the subset in a boolean array where the index with the value 0, 1, 3 will have 1 and other 0.

| 1 | 0 | 1 | 1 | $= (1011)_2$
3 2 1 0

Now this just became an integer. An integer can lead us to the subset $\{2, 5, 8\}$

What number will we store in x

if we want to build subset $\{5, 7\}$

$\{5, 7\} = (1, 2)$

Boolean Array = | 0 | 1 | 1 | 0 |
3 2 1 0

$X = (0110)_2$

$X = (6)_{10}$

Ques. What is the value of :

$\{5, 7\}$ OR $\{2, 5, 8\}$

$\Rightarrow \{2, 5, 7, 8\}$

Any set & set operations can be converted to Binary & Binary operations. We can save sets & subsets using mask.

* Masks

$\{2, 5, 7, 8\}$ Mask = 6
$0\ 1\ 2\ 3$

Mask = 6

= 0 1 1 0
0 1 2 3

$\Rightarrow \{5, 7\}$ Insert 8 here

8 has position 3 so set idx[3] to 1 in Masked array.

= 0 1 1 0
↳ 1 (flipping 0)

Given an x, if we want to flip the 3rd bit, we can take (1<<3) 1 shifted by 3. and take an OR b/w (1<<3) and x.

Masked array, x = 0 1 1 0

1<<3 = 1 0 0 0

OR = 1 1 1 0

inserted 8

→ Now, let's say there is a set if 8 is present, remove it else insert it.

Change the above OR operation to XOR (^). It basically flips the i-th bit handling both cases of insertion/deletion.

In this case, it will flip the 3rd bit

→ Generate all subsets of a set.

{2, 5, 7} (2 → 0, 5 → 1, 7 → 2)

Masks		All subsets
0 :	000	→ { } empty set
1 :	001	→ 0th position = 2
2 :	010	→ 1st pos → 5
3 :	011	→ 1st & 0th both = {2,5}
4 :	100	→ 2nd position = 7
5 :	101	→ (0,2) → {2,7}
6 :	110	→ (1,2) → {5,7}
7 :	111	→ (0,1,2) → {2,5,7}

We are generating all possible masks of the subsets. For each mask, we are going through its bits one by one (inside loop) and printing the element at index i where the value is 1.

```cpp
#include<bits/stdc++.h>
using namespace std;

void solve(){
    int arr[] = {2,5,7};
    for(int mask=0;mask<(1<<3);mask++){
        for(int i=0;i<3;i++){
            if(mask&(1<<i)){
                cout<<arr[i]<<",";
            }
        }
        cout<<endl;
    }
}

signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
```

Test Case 1

Input

Enter input

Output

```
2,
5,
2,5,
7,
2,7,
5,7,
2,5,7,
```