8 Count no of 1 bits.

Input → 0000 0000 0000 0000 0000 0000 0000 1011

output → 3

Ap Case 1 → Convert to binary then count

Ap 2 → Checking the last bit

code

```
class solution {
    public:
        int hammingWeight (uint32_t n) {
            int ans = 0;
            while (n != 0) {
                if (n & 1) {
                    ans ++;
                }
                n = n >> 1; // Right shift
            }
            return ans;
        }
};
```

DECIMAL & BINARY

Decimal to Binary

① 10 → $\frac{10}{2} = 5$, $\frac{5}{2} = 2$, $\frac{2}{2} = 1$, $\frac{1}{2} = 0$

(Remainder) 0 1 0 1

Finish ← Start

2^3 2^2 2^1 2^0

10 → 1 0 1 0

Steps : (i) Divide number by 2
(ii) Store the remainder
(iii) Repeat for the dividend

Division	Remainder
10/2	0
5/2	1
2/2	0
1/2	1
Khatam	

Reverse = 1010

② $n = 5 \longrightarrow$ Binary?

II nd Logic : $n \& 1 = 0$ if n is even
$n \& 1 = 1$ if n is odd

Thus, a bit is 1 if bit & 1 = 1
a bit is 0 if bit & 1 = 0

Thus, (any-bit) & 1 = (any-bit)

∴ Last bit of 5 = 5 & 1 = 1
check 5 = 101 ⇒ 101
& 1
1

Right shift 5 by 1 ⇒ n = 2 (010)
Second last bit of 5 = 2 & 1 = 0
Right shift of 2 by 1 ⇒ n = 1 (001)
First bit of 5 = 1 & 1 = 1
Right shift of 1 by 1 ⇒ n = 0 (000)

Khatam!

code

```
# include <iostream>
# include <math.h>
  using namespace std;
  int main() {
      int n;
      cin >> n;
      int ans = 0;
      int i = 0;
      while (n != 0) {
          int bit = n & 1;          // same as divide by 2
          ans = (bit * pow(10, i)) + ans;
          n = n >> 1;
          i++;
      }
      cout << "Answer is" << ans << endl;
  }
```

Kuch to kadbad hai!

Example n = 6

① ans = 0, i = 0

② bit = 6 & 1 = 0 (even)

③ ans = $10^0 \times 0$ + ans = 0 n = 3, i = 1

④ bit = 3 & 1 = 1 (odd)

⑤ ans = $(10^1 \times 1)$ + ans = 10 n = 1, i = 2

⑥ bit = 1 & 1 = 1 (odd)

⑦ ans = $(10^2 + 1)$ + ans = 100 + 10 = 110 n = 0

ans = 100

⇒ 1, 2, 3

(output 123)

Same flow

$(0 \times 10) + 1 \rightarrow 1$

$(1 \times 10) + 2 \rightarrow 12$

$(12 \times 10) + 3 \rightarrow 123$

ans = (ans × 10) + digit

321 = output

Reverse flow

ans = 0

ans = $(1 \times 10^0) + 0 = 1$

ans = $(2 \times 10^0) + 1 = 21$

ans = $(3 \times 10^2) + 21 = 321$

ans = (digit × 10^i) + ans

Negative decimal to Binary.

$-6 \rightarrow$ Binary ??

$-6 \rightarrow 6 \rightarrow 000\ldots00110 \rightarrow$

```
111...11001
          +1
-----------
111...11010
```

Think what is 111...11010 equal to if it is unsigned?

$\underbrace{111\ldots11}_{29\text{ bits}}010 = 4294967290$

$= 2^{32} - 6$

$\therefore -6 \xrightarrow{\text{Binary}}$ Binary of $(2^{32} - 6) \rightarrow \underbrace{111\ldots11}_{29}010$

we can't express this in any data type

code

```
#include <iostream>
#include <math.h>
using namespace std;

int main(void)
{
    long long int n;
    cin >>n;
    unsigned long long int i=0, ans=0
    if (n<0) {
        n= pow(2,16)+n
    }
    cout << n <<endl;
    while(n){
        int lastBit = n& 1;
        ans = (pow(10,i) * lastBit) + ans;
        n= n>>1;
        i++
        cout << ans <<endl;
    }
    cout << ans <<endl;
    return 0;
}
```

Kuch to kadbad hai!

Binary to Decimal:

$$1\ 0\ 1\ 0\ 1\ 1 = 2^0\times1 + 2^1\times1 + 2^2\times0 + 2^3\times1 + 2^4\times0 + 2^5\times1$$
$$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$$
$$\Rightarrow 1 + 2 + 0 + 8 + 0 + 32 \Rightarrow \boxed{43}$$

Logic: $n \& 1 = \begin{cases} 0, & \text{dont do anything} \\ 1, & \text{multiply with } 2^i \end{cases}$

$i++,\ n >> 1$

- We will give int n = 10010; but it is actually a decimal number so last bit = n%10;

Code

```
int main(){
    int n;
    cin >> n;
    int i=0, ans=0;
    while(n!=0){
        int digit = n%10;
        if(digit==1){
            ans = ans + pow(2,i);
        }
        n = n/10;
        i++;
    }
    cout << ans << endl;
}
```

Question Reverse Integer

case 1 → Normal case → 123 → 321

case 2 → 12 ..... 3 → 3 ..... 1 → out of range of int then return 0

Ex num = 123 → out

123 % 10 = 3

ans = 0

ans = ans × 10 + digit
= 0 × 10 + 3 = 3
= 3 × 10 + 2 = 32
= (32 × 10) + 1 = 321

normal case

```
while (n != 0) {
    int digit = n % 10;
    ans = (ans * 10) + digit
    n = n / 10;
}
```

wrong code

```
int reverse(int x){
    int ans = 0;
    while (x != 0){
        int digit = x % 10;
        ans = (ans * 10) + digit;
        x = x / 10
    }
    return ans;
}
```

* This is not a proper correct code b/c range of int is $[-2^{31}$ to $2^{31}-1]$

Let $n = 2^{31} - 10$ is in range.

if $(2^{31} - 10) * 10$ then this is not in range.

code correction

if (ans > INT_MAX/10) → return

ans x10 > INT_MAX x10
/10

✗ ans x10 > INT_MAX

return 0;

Code

```
int reverse(int x){
    int ans = 0;
    while (x != 0){
        int digit = x%10;
        if ((ans > INT_MAX/10) || (ans < INT_MIN/10)){
            return 0;
        }
        ans = (ans*10) + digit;
        x = x/10
    }
    return ans
}
```

92 Compliment of Base 10 Integer

Ex $n = 5 \rightarrow 101$

1's compliment

010 → (2) → ans

Bits: & | ~ ^ << >>

Approch

$n = 5 \rightarrow 101$

0000 ---- ---- 0101 i/p

ans = 2

0000 ---- ---- 0010 o/p

$\sim n = \sim 5 \Rightarrow$ 1111 ---- ---- 1101 → 10 → ans = 2

operation to ignore this.

Let mask = 00000 ---- 0011

00000 0 ---- 0010 → Ans.

mask = ?

n & mask → ans → formula

$\Rightarrow$ n = 5 → 00000 ---- 0101

mask 00000 0 ---- 0011

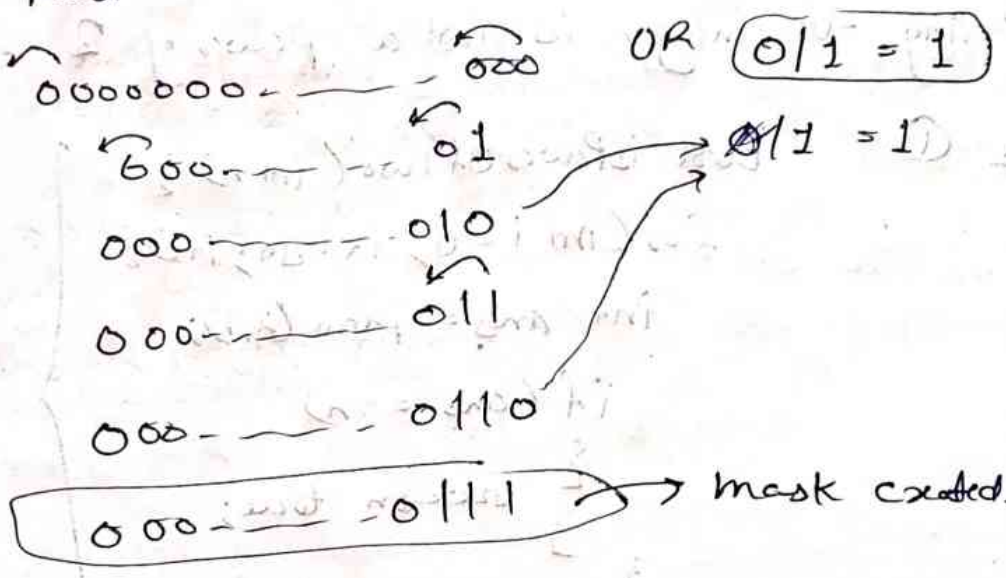
000 ---- 0101

(>>) shift right 1st → 0000 ---- 0010 → != 0

2nd → 0000 ---- 0001 → != 0

3rd → 0000 ---- 0000 → = 0 stop.

mask = 0



Edge case

```
if (n == 0)
    return 1
```

Code

```
class Solution {
public:
    int bitwiseComplement(int n) {
        int m = n;
        int mask = 0;
        // edge case
        if (n == 0)
            return 1;
        while (m != 0) {
            mask = (mask << 1) | 1;
            m = m >> 1;
        }
        int ans = (~n) & mask;
        return ans;
    }
};
```

9 Power of 2

Hint :- any -ve integer is not a power of 2

code ①

```
bool isPowerOfTwo(int n){
    for(int i=0; i<=30; i++){
        int ans = pow(2,i);
        if(ans == n)
        {
            return true;
        }
    }
    return false;
}
```

Brute Force

cod②

```
bool isPowerofTwo(int n){
    int temp = 1;
    while(n != temp && temp < INT_MAX/2) {
        temp *= 2;
    }
    return n == temp;
}
```

OR

```
bool isPowerOfTwo(int n) {
    int ans = 1;
    for(int i = 0; i<=30; i++) {
    if(ans == n)
    {   return true;
    }
    if(ans < INT_MAX/2)
        ans = ans * 2;
    }
    return false;
}
```

code ③ (most Efficient)

```
bool isPowerOfTwo(int n) {
    if (n<=0)
        return false;
    return (n & (n-1)) == 0;
}
```

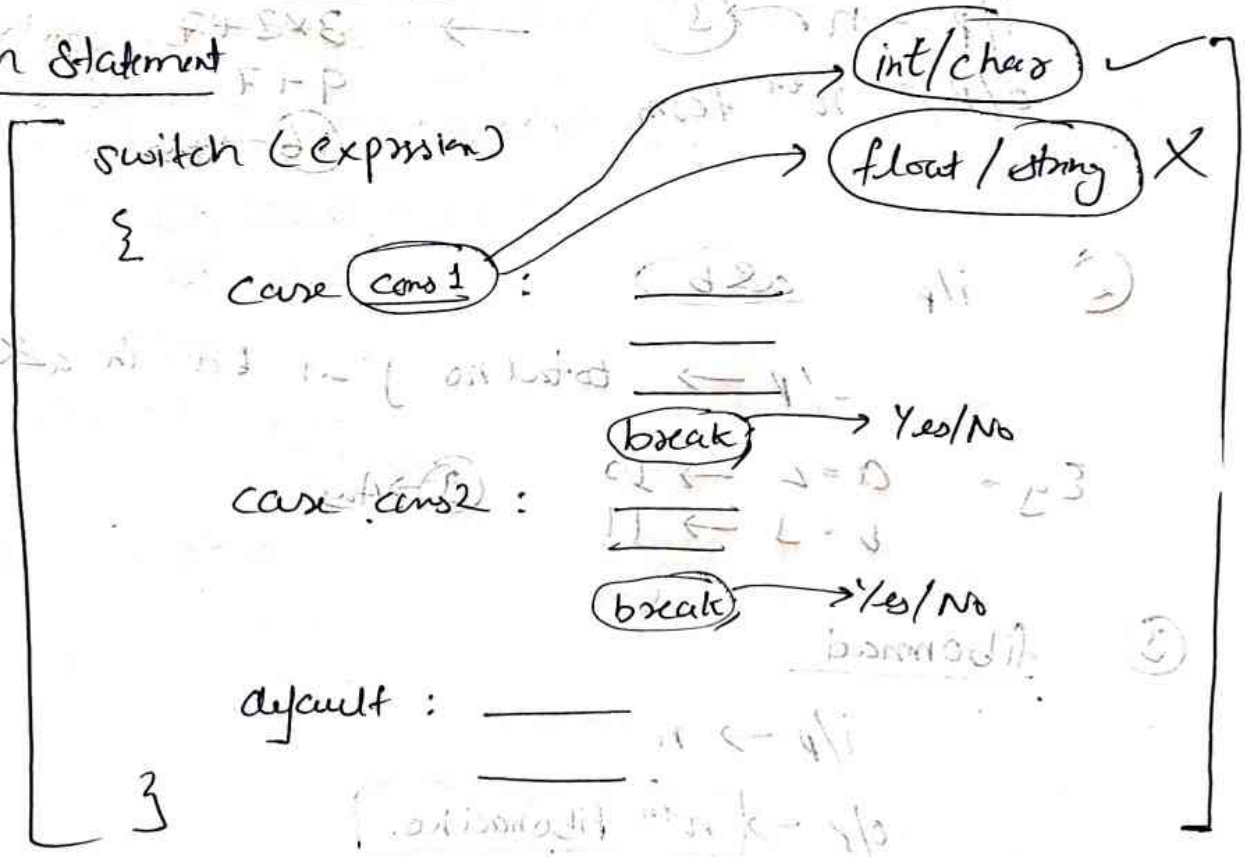
Hint ⇒ n & (n-1) flips the rightmost SET bit

Eg: n = 18 ⇒ 000 — 010010

18 & (17) ⇒ 000 — 010010 (18)
000 010001 (17)
000 — 010000

★ Switch Statement & Function ★

Switch Statement



NOTE ⇒ here break is used to go out of the loop without printing invalid case.

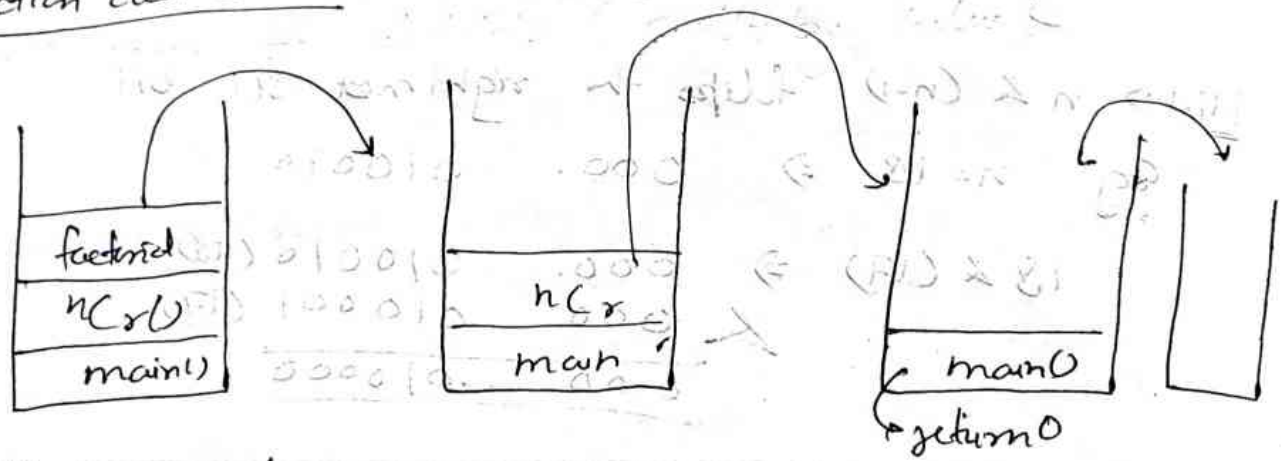
- Continue is not used in case of switch (Try)

b/c As the case constant matches with the users i/p the flow will come down and try to execute but b/c of continue it will skip that part and iteration will go on and on. that is why we use break

Functions

→ It is a well defined task that is used in many time without repeating the code.

Function call stack



H/W ① A.P = (3 * n + 7)

i/p = n (3) → 3×3+7

o/p = n^{th} term → 9+7

(16) → Ans

② i/p (a & b)

o/p → total no. of set bits in a & b

Eg - a = 2 → 10

b = 3 → 11

(3) → Ans

③ fibonnaci

i/p → n

o/p → n^{th} Fibonacci no.

Eg ⇒ 0, 1, 1, 2, 3, 5, 8, 13

n = 5 o/p ⇒ 3