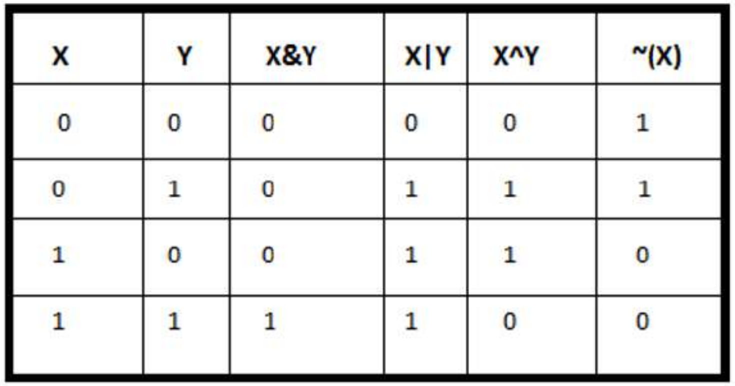
Bit Manipulation

Table of Contents

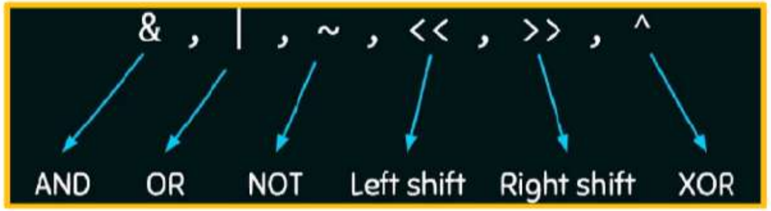
TRUTH TABLE



X	Y	X&Y	X|Y	X^Y	~(X)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

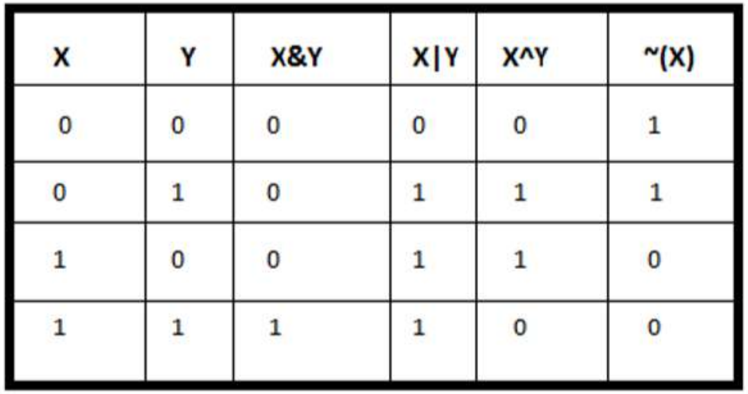
1. Bitwise operators

Bitwise Operators : These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of Binary indexed tree.



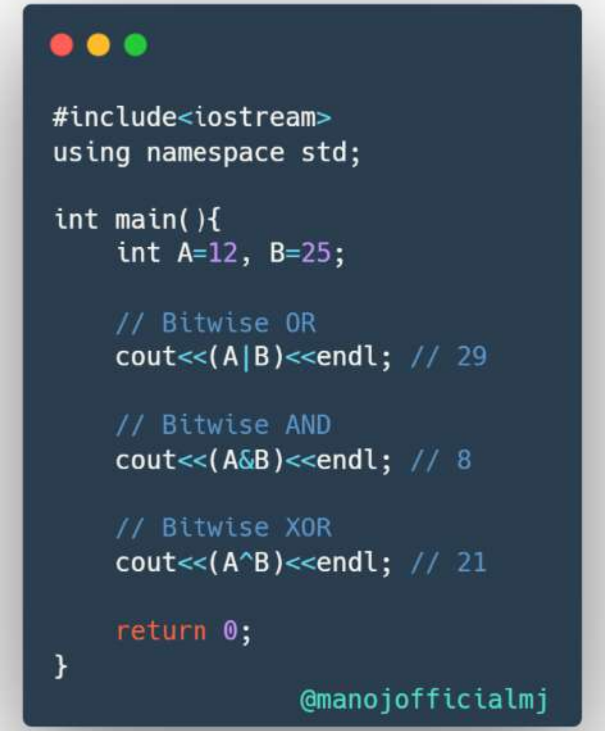
Why Use : in sort, works at bit level

TRUTH TABLE



X	Y	X&Y	X|Y	X^Y	~(X)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Program of Bitwise Operators :



```
#include<iostream>
using namespace std;

int main( ){
    int A=12, B=25;

    // Bitwise OR
    cout<<(A|B)<<endl; // 29

    // Bitwise AND
    cout<<(A&B)<<endl; // 8

    // Bitwise XOR
    cout<<(A^B)<<endl; // 21

    return 0;
}
```

@manojofficialmj

```
12= 00001100(In Binary)
25= 00011001(In Binary)
```

Bitwise OR Operation of 12 and 25

00001100 | 00011001
00011101= 29 (In Decimal)

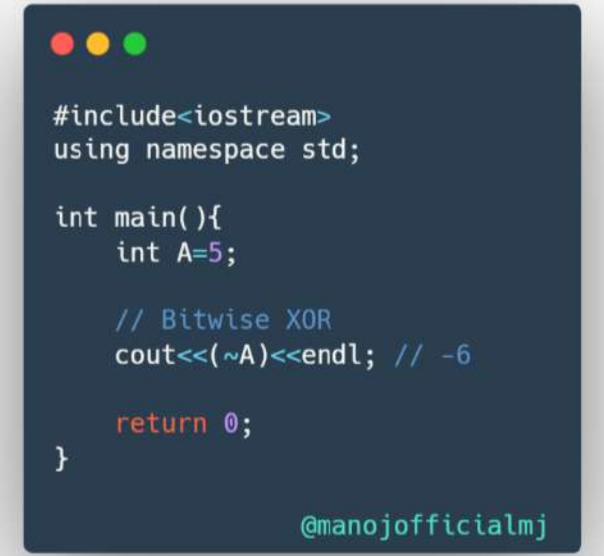
Bitwise AND Operation of 12 and 25

00001100 & 00011001
00001000= 8 (In Decimal)

Bitwise XOR Operation of 12 and 25

00001100 ^ 00011001
00010101= 21 (In Decimal)

Program of Bitwise not/complement :



```
#include<iostream>
using namespace std;

int main(){
    int A=5;

    // Bitwise XOR
    cout<<(~A)<<endl; // -6

    return 0;
}

                        @manojofficialmj
```

It is important to note that the bitwise complement of any integer N is equal to - (N + 1).

WHY????

```
a = 5 => 0101 (In Binary)
Bitwise Complement Operation of 5

~ 0101
 ________
  1010   = 10 (In decimal)
```

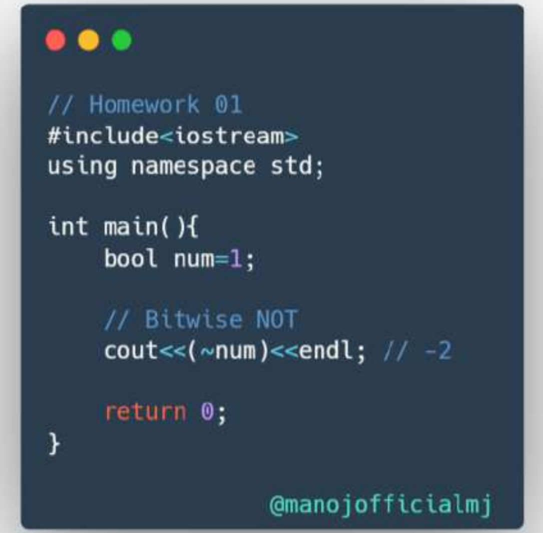
1s com 0101

+1

2's com 1010 =(-6)

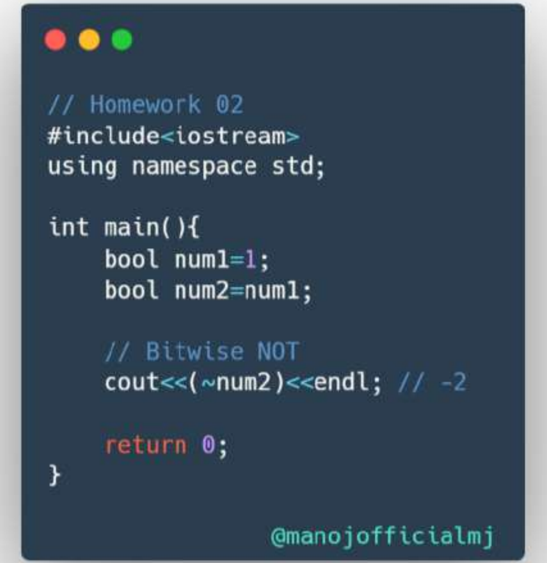
BECAUSE: Compiler will give 2's complement of that number, i.e., 2's complement of 10 will be -6.

Homework programs:



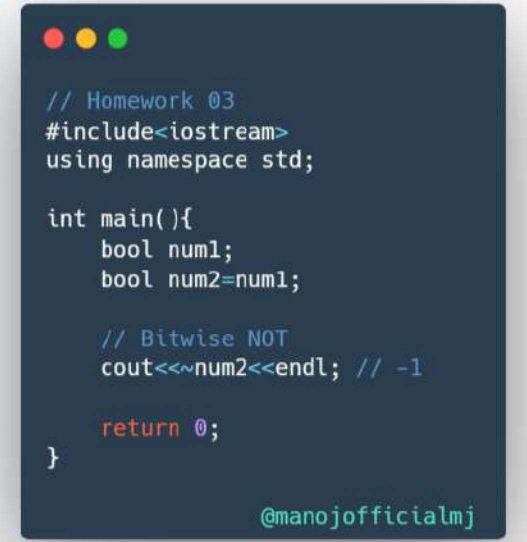
```
// Homework 01
#include<iostream>
using namespace std;

int main(){
    bool num=1;

    // Bitwise NOT
    cout<<(~num)<<endl; // -2

    return 0;
}
```

@manojofficialmj



```
// Homework 02
#include<iostream>
using namespace std;

int main(){
    bool num1=1;
    bool num2=num1;

    // Bitwise NOT
    cout<<(~num2)<<endl; // -2

    return 0;
}
```

@manojofficialmj



```
// Homework 03
#include<iostream>
using namespace std;

int main(){
    bool num1;
    bool num2=num1;

    // Bitwise NOT
    cout<<~num2<<endl; // -1

    return 0;
}
```

@manojofficialmj

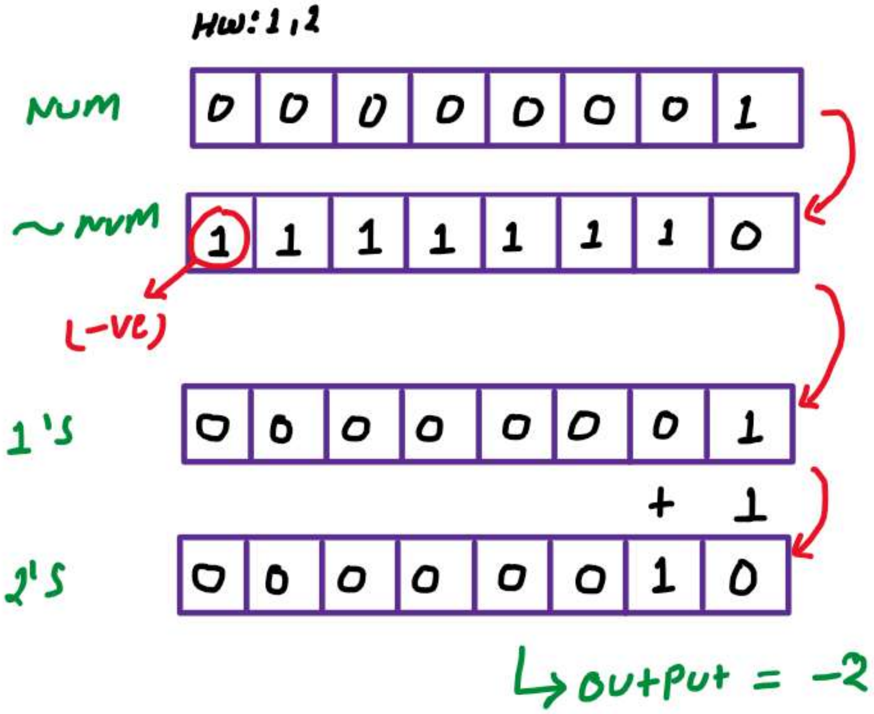
Note

num=1
↓
True (1)

Num=0
↓
False (0)

num=2
↓
True (1)

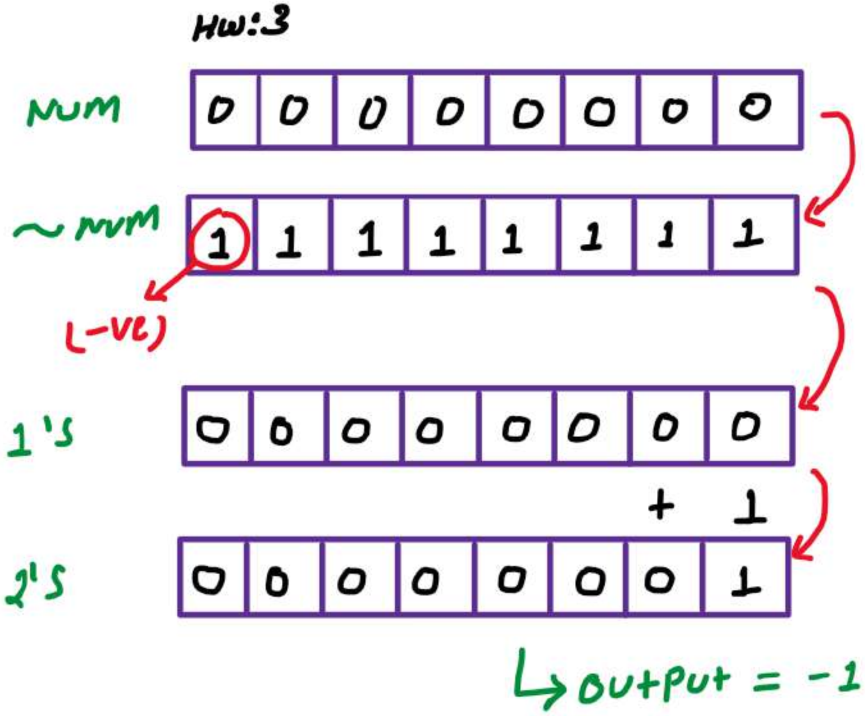
- num contains greater than 0 or less then 0 then ~num always produces the output equal to -2 because true means 1.
- and num contains zero or nothing then ~num always produces the output equal to -1 because false means 0.



HW:1,2

NUM	0	0	0	0	0	0	0	1
~NUM	1	1	1	1	1	1	1	0
1'S	0	0	0	0	0	0	0	1
							+	1
2'S	0	0	0	0	0	0	1	0

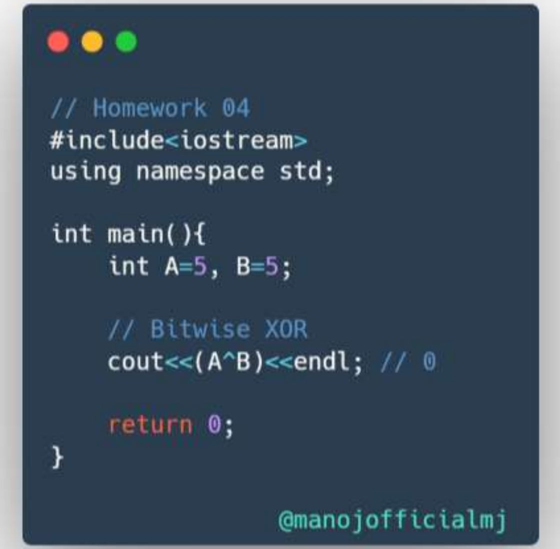
↳ output = -2



HW:3

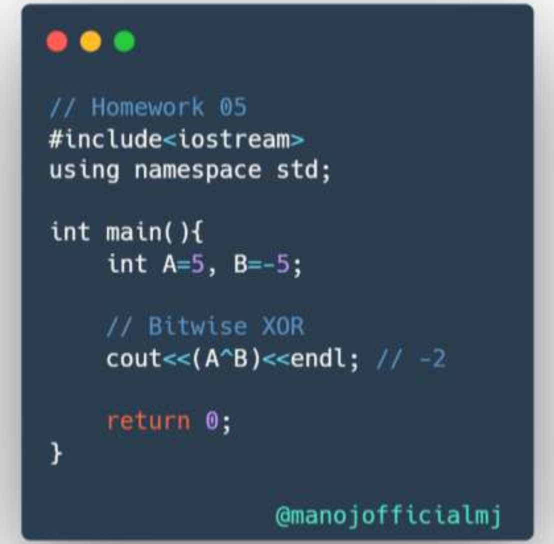
NUM	0	0	0	0	0	0	0	0
~NUM	1	1	1	1	1	1	1	1
1'S	0	0	0	0	0	0	0	0
							+	1
2'S	0	0	0	0	0	0	0	1

↳ output = -1



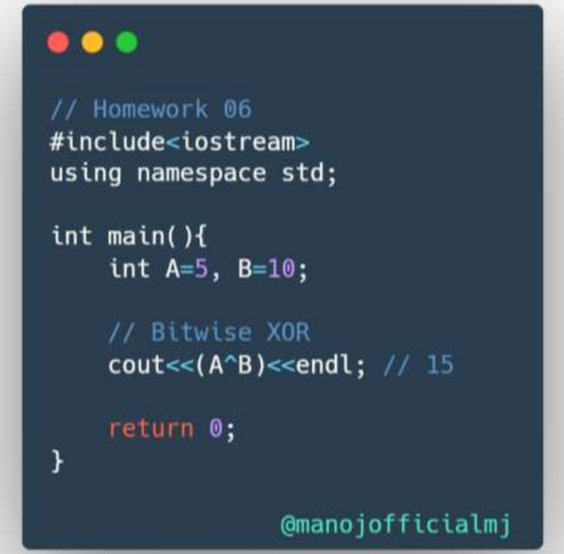
```
// Homework 04
#include<iostream>
using namespace std;

int main(){
    int A=5, B=5;

    // Bitwise XOR
    cout<<(A^B)<<endl; // 0

    return 0;
}
```

@manojofficialmj



```
// Homework 05
#include<iostream>
using namespace std;

int main(){
    int A=5, B=-5;

    // Bitwise XOR
    cout<<(A^B)<<endl; // -2

    return 0;
}
```

@manojofficialmj



```
// Homework 06
#include<iostream>
using namespace std;

int main(){
    int A=5, B=10;

    // Bitwise XOR
    cout<<(A^B)<<endl; // 15

    return 0;
}
```

@manojofficialmj

Hw: 4

A	0	0	.....	0	1	0	1
B	0	0	.....	0	1	0	1
A^B	0	0	.....	0	0	0	0

↳ output = 0

Hw: 6

A	0	0	.....	0	1	0	1
B	0	0	.....	1	0	1	0
A^B	0	0	.....	1	1	1	1

↳ output = 15

Hw: 5

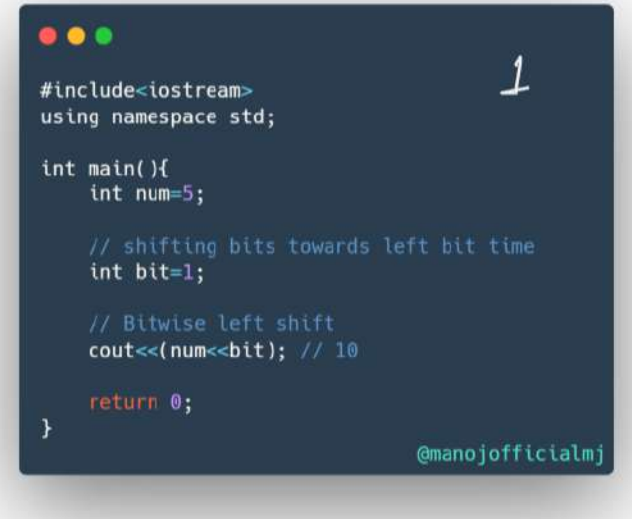
A	0	0	.....	0	1	0	1
B	1	1	.....	1	0	1	1
A^B	1	1	.....	1	1	1	0

-ve

1's ⇒	0	0	.....	0	0	0	1
						+	1
2's ⇒	0	0	.....	0	0	1	0

↳ output = -2

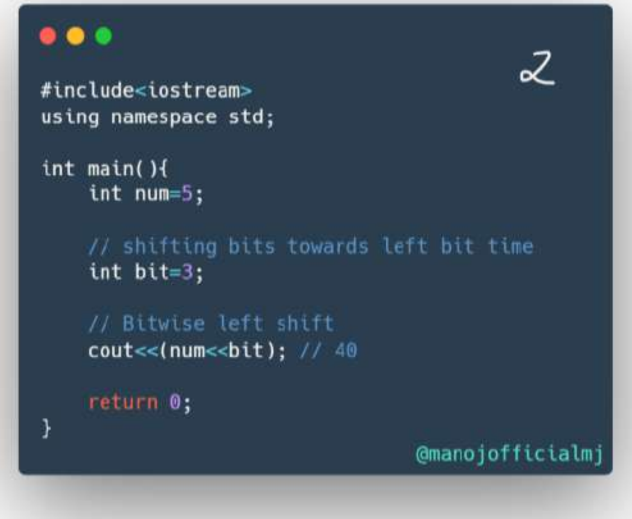
Bitwise left and right shift operators :



1

```
#include<iostream>
using namespace std;

int main(){
    int num=5;

    // shifting bits towards left bit time
    int bit=1;

    // Bitwise left shift
    cout<<(num<<bit); // 10

    return 0;
}
```

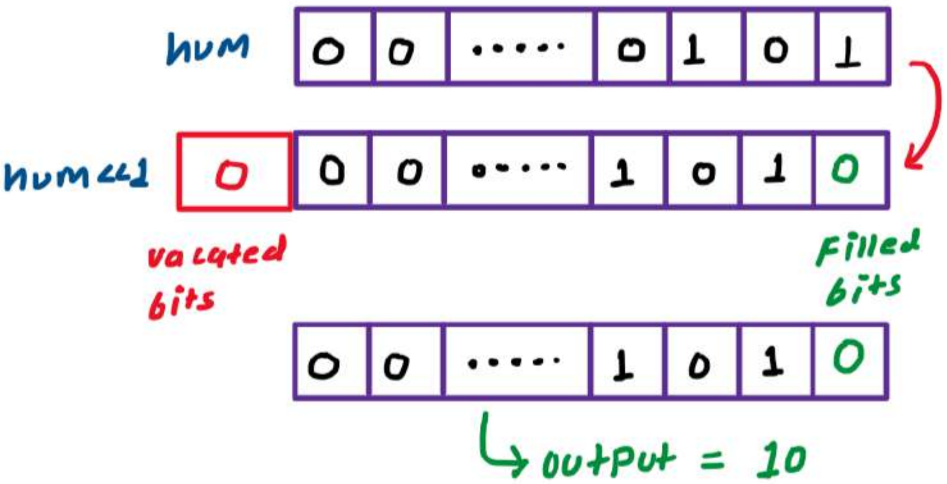
@manojofficialmj

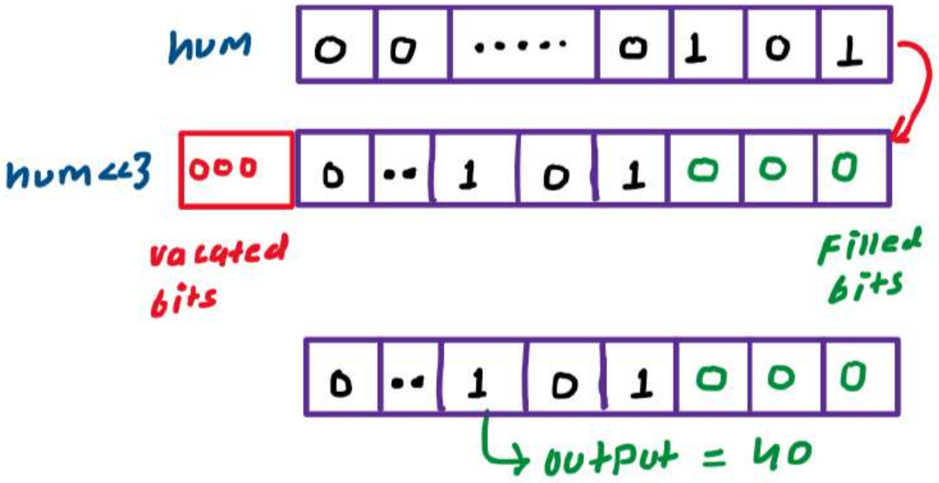


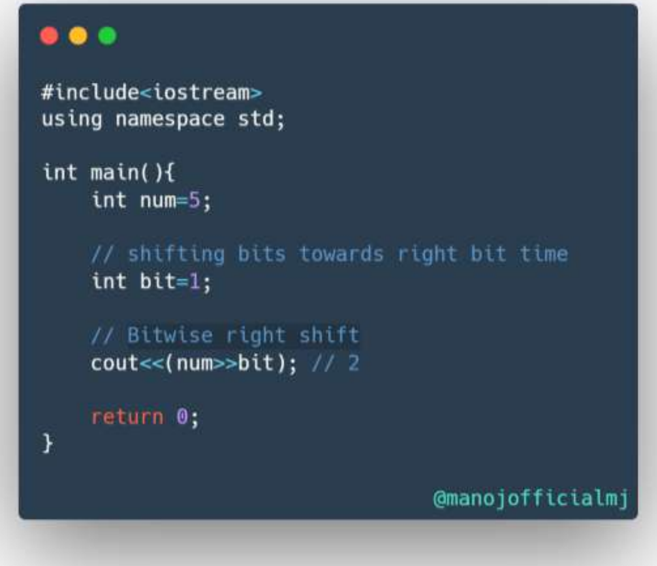
2

```
#include<iostream>
using namespace std;

int main(){
    int num=5;

    // shifting bits towards left bit time
    int bit=3;

    // Bitwise left shift
    cout<<(num<<bit); // 40

    return 0;
}
```

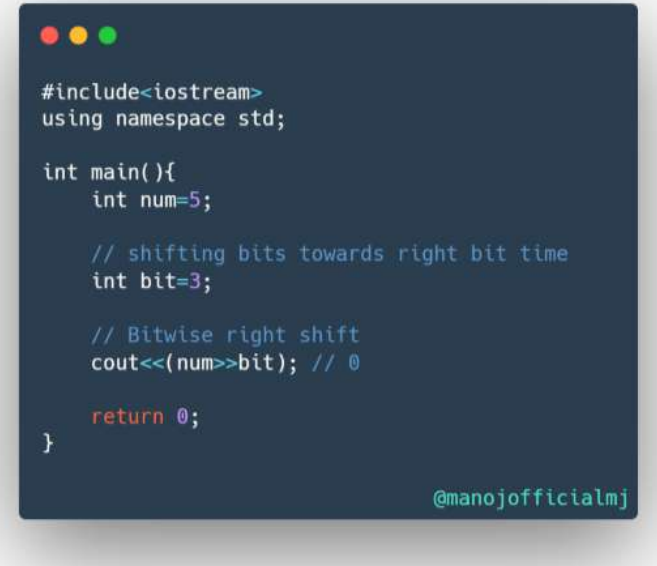
@manojofficialmj


num
0 0 ····· 0 1 0 1
num<<1
0
0 0 ····· 1 0 1 0
vacated bits
Filled bits
0 0 ····· 1 0 1 0
output = 10

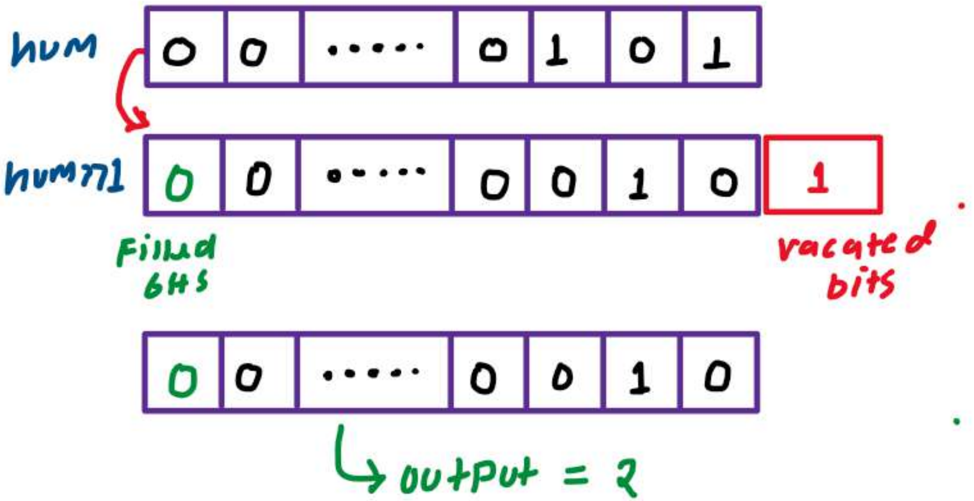

num
0 0 ····· 0 1 0 1
num<<3
000
0 ·· 1 0 1 0 0 0
vacated bits
Filled bits
0 ·· 1 0 1 0 0 0
output = 40

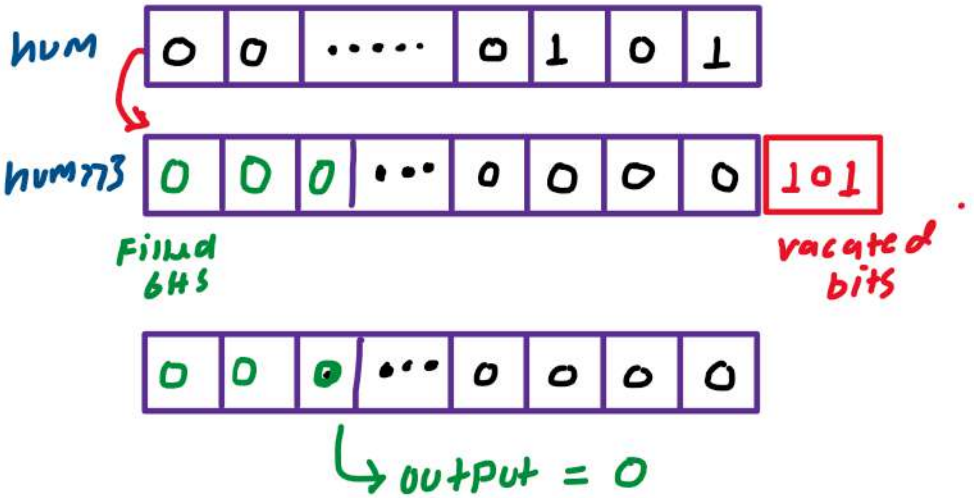


```
#include<iostream>
using namespace std;

int main(){
    int num=5;

    // shifting bits towards right bit time
    int bit=1;

    // Bitwise right shift
    cout<<(num>>bit); // 2

    return 0;
}
```



```
#include<iostream>
using namespace std;

int main(){
    int num=5;

    // shifting bits towards right bit time
    int bit=3;

    // Bitwise right shift
    cout<<(num>>bit); // 0

    return 0;
}
```

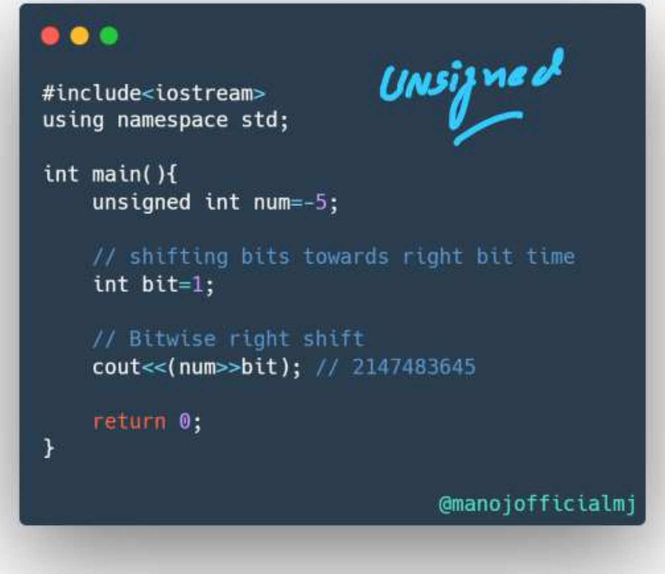

num
0 0 ····· 0 1 0 1
num>>1
0 0 ····· 0 0 1 0
1
Filled bits
vacated bits
0 0 ····· 0 0 1 0
output = 2


num
0 0 ····· 0 1 0 1
num>>3
0 0 0 ··· 0 0 0 0
101
Filled bits
vacated bits
0 0 0 ··· 0 0 0 0
output = 0

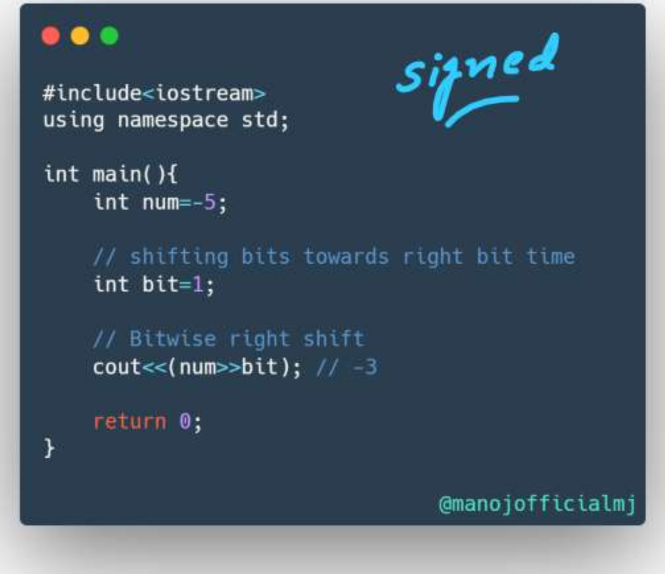
Always remember notes:

If there is a negative signed integer, then this will be handled by the compiler.

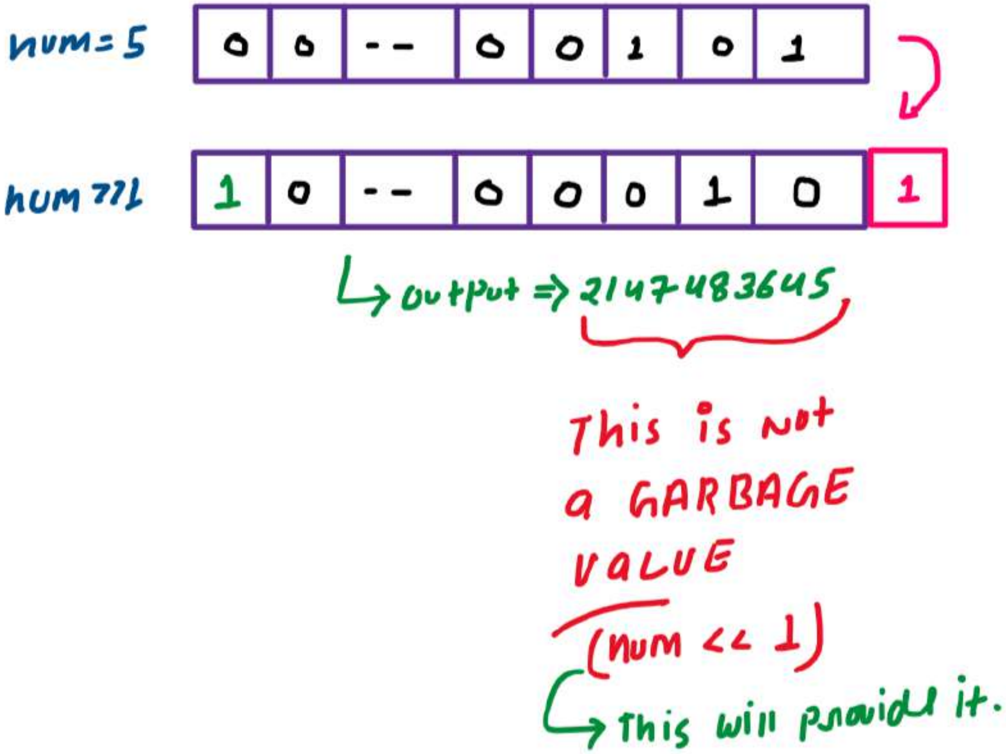
If there is a negative unsigned integer, then this will not be handled by the compiler. Most significant bit gets right shifted and the bit becomes zero.



```cpp
#include<iostream>
using namespace std;

int main( ){
    unsigned int num=-5;

    // shifting bits towards right bit time
    int bit=1;

    // Bitwise right shift
    cout<<(num>>bit); // 2147483645

    return 0;
}
```



```cpp
#include<iostream>
using namespace std;

int main( ){
    int num=-5;

    // shifting bits towards right bit time
    int bit=1;

    // Bitwise right shift
    cout<<(num>>bit); // -3

    return 0;
}
```



num=5 | 0 | 0 | -- | 0 | 0 | 1 | 0 | 1

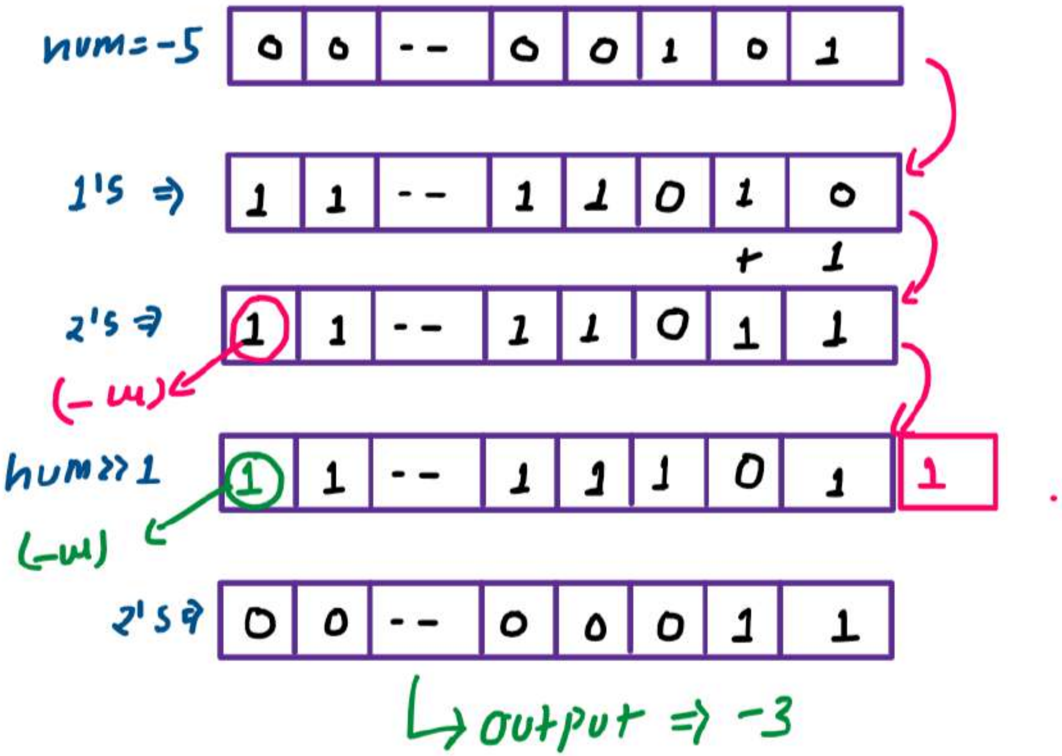
num>>1 | 1 | 0 | -- | 0 | 0 | 0 | 1 | 0 | 1

↳ output ⇒ 2147483645

This is not a GARBAGE VALUE

(num << 1)

↳ This will provide it.



num=-5 | 0 | 0 | -- | 0 | 0 | 1 | 0 | 1

1's ⇒ | 1 | 1 | -- | 1 | 1 | 0 | 1 | 0

+ 1

2's ⇒ | 1 | 1 | -- | 1 | 1 | 0 | 1 | 1

(-ve)

num>>1 | 1 | 1 | -- | 1 | 1 | 1 | 0 | 1 | 1

(-ve)

2's ⇒ | 0 | 0 | -- | 0 | 0 | 0 | 1 | 1

↳ output ⇒ -3

2. Check even or odd number

Even number

2	0000 0010
4	0000 0100
10	0000 1010

Right most bit = 0

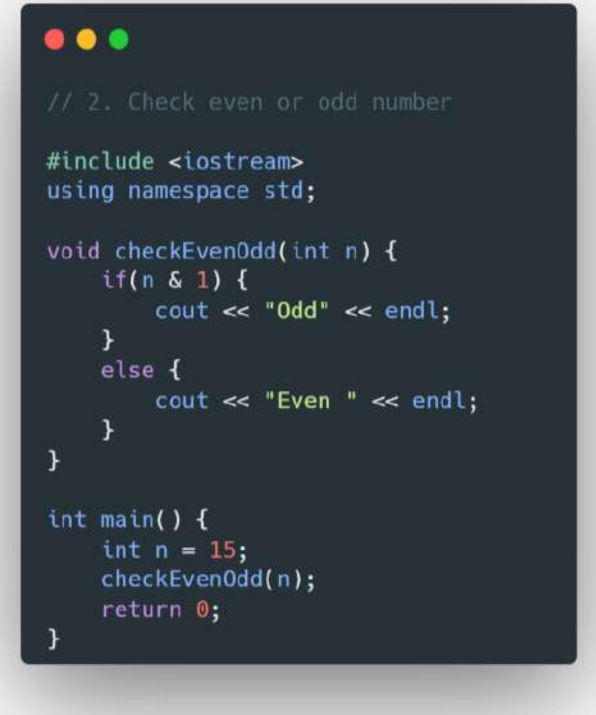
if (x & 1 == 0)
↳ Even

if (x & 1 == 1)
↳ odd

Right most bit = 1

Odd number

3	0000 0011
5	0000 0101
11	0000 1011



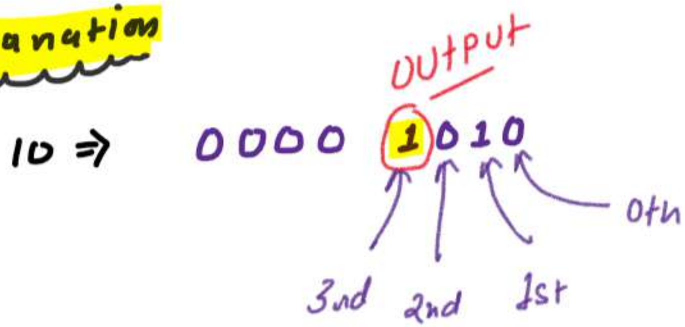
```cpp
// 2. Check even or odd number

#include <iostream>
using namespace std;

void checkEvenOdd(int n) {
    if(n & 1) {
        cout << "Odd" << endl;
    }
    else {
        cout << "Even " << endl;
    }
}

int main() {
    int n = 15;
    checkEvenOdd(n);
    return 0;
}
```

3. Get Ith bit from right side

Input N = 10 and i = 3

Output 1

Explanation



Logic

```
 10  ⇒  0000 1010
Mask ⇒ ~~0~~0000 1000
       ___________
 NUM ⇒  0000 1000
       ___________
```

How to create this mask?

Num != 0
↳ 1 bit

Num == 0
↳ 0 bit

CREATE MASK

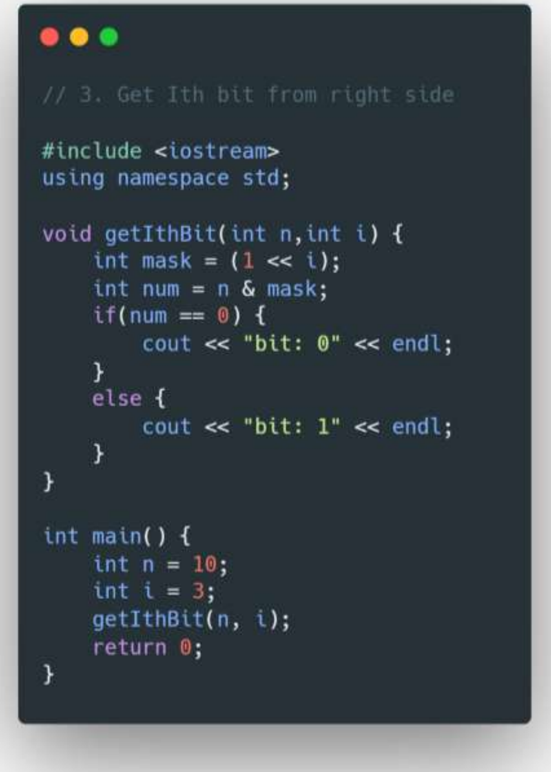
STEP1 1 ⇒ 0000 0001

STEP2 $1 << i$ (left shift)

$1 << 3 \Rightarrow$ 000 0000 1000
xxx

Mask

$$\boxed{\text{Mask} = 1 << i}$$



```
// 3. Get Ith bit from right side

#include <iostream>
using namespace std;

void getIthBit(int n,int i) {
    int mask = (1 << i);
    int num = n & mask;
    if(num == 0) {
        cout << "bit: 0" << endl;
    }
    else {
        cout << "bit: 1" << endl;
    }
}

int main() {
    int n = 10;
    int i = 3;
    getIthBit(n, i);
    return 0;
}
```

4. Set Ith bit from right side

Input N= 10 and i = 2

Output 14

Explanation

2nd bit

10 ⇒ 0000 1010

Ans ⇒ 0000 1110

Set = 1
clean = 0

Logic

10 ⇒ 0000 1010

Mask ⇒ 0000 0100

N | mask ⇒ 0000 1110 ← Ans

How To cReate this mask?

CREATE MASK

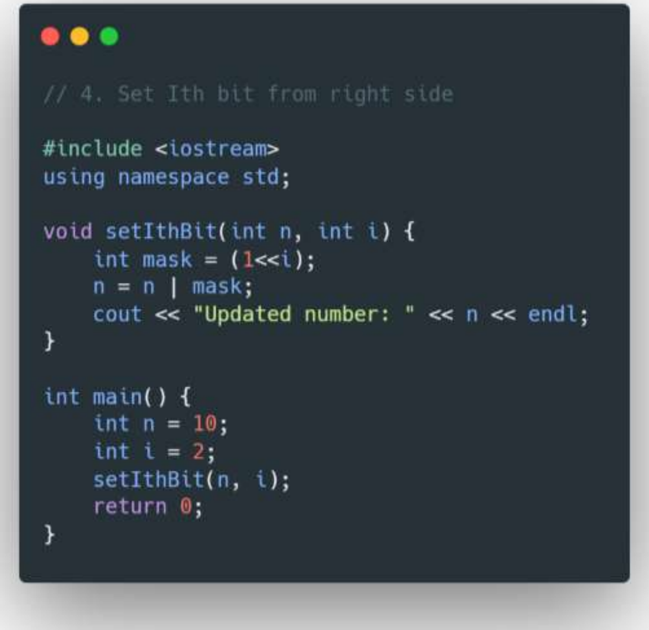
STEP1 1 ⇒ 0000 0001

STEP2 1<<i (left shift)

1<<2 ⇒ $\underset{xx}{00}$ $\boxed{0000\ 0100}$

mask

$\boxed{\text{Mask} = 1 << i}$



```
// 4. Set Ith bit from right side

#include <iostream>
using namespace std;

void setIthBit(int n, int i) {
    int mask = (1<<i);
    n = n | mask;
    cout << "Updated number: " << n << endl;
}

int main() {
    int n = 10;
    int i = 2;
    setIthBit(n, i);
    return 0;
}
```

5. Clear Ith bit from right side

Input N = 10 and i = 1

Output 8

Explanation

1st bit ↓

10 ⇒ 0000 1010

Ans ⇒ 0000 1000

Set = 1

Clear = 0

Logic

10 ⇒ 0000 1010

mask ⇒ 1111 1101

N & mask ⇒ 0000 1000 ← output

How To create this mask?

CREATE mask

STEP1 1 ⇒ 0000 0001

STEP2 1<<i (left shift)

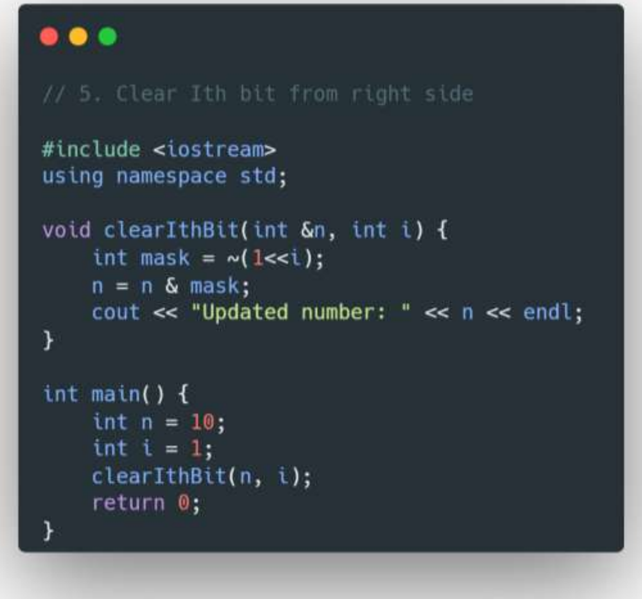
0
x 0000 0010

STEP3 Take 1's complement of STEP2

~(1<<i) 1111 1101

mask

mask = ~(1 << i)



```
// 5. Clear Ith bit from right side

#include <iostream>
using namespace std;

void clearIthBit(int &n, int i) {
    int mask = ~(1<<i);
    n = n & mask;
    cout << "Updated number: " << n << endl;
}

int main() {
    int n = 10;
    int i = 1;
    clearIthBit(n, i);
    return 0;
}
```

6. Update Ith bit from right side

input N=10 , i=3 , target =0

output 2

☺ Target will be 0 or 1 only

Explaination

3rd bit ↓

10 ⇒ 0000 1010

Ans ⇒ 0000 0010

Logic 10 ⇒ 0000 1010

STEP1 Clear ith bit

N ⇒ 0000 0010

STEP2 Create mask (target << i)

target ⇒ 0000 0000

mask ⇒ 000 $\boxed{0000\ 0000}$

× × ×

MASK

STEP3 N | mask

N ⇒ 0000 0010

mask ⇒ 0000 0000

Ans 0000 0010

Ex2

Input N=10 , i=2 , target =1

Output 14

☺ Target will be 0 or 1 only

Explaination

2nd bit

10 ⇒ 0000 1010

Ans ⇒ 0000 1110

Logic 10 ⇒ 0000 1010

STEP1 Clear ith bit

N ⇒ 0000 1010

STEP2 Create mask (target << i)

target ⇒ 0000 0001

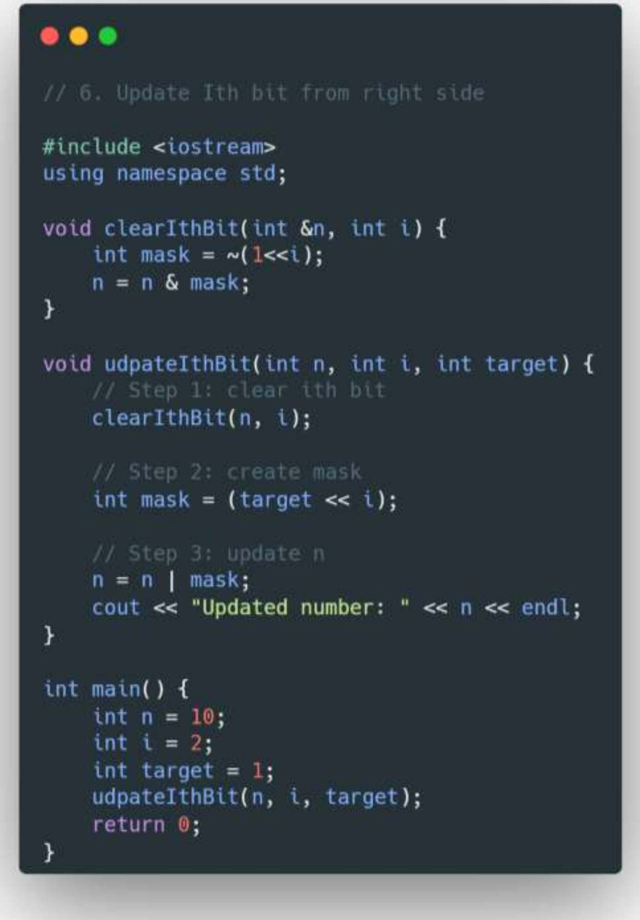
mask ⇒ $\underset{\times\times}{00}\,\underset{\text{MASK}}{\boxed{0000\ 0100}}$

STEP3 N | mask

N ⇒ 0000 1010

mask ⇒ 0000 0100

Ans 0000 1110



```cpp
// 6. Update Ith bit from right side

#include <iostream>
using namespace std;

void clearIthBit(int &n, int i) {
    int mask = ~(1<<i);
    n = n & mask;
}

void udpateIthBit(int n, int i, int target) {
    // Step 1: clear ith bit
    clearIthBit(n, i);

    // Step 2: create mask
    int mask = (target << i);

    // Step 3: update n
    n = n | mask;
    cout << "Updated number: " << n << endl;
}

int main() {
    int n = 10;
    int i = 2;
    int target = 1;
    udpateIthBit(n, i, target);
    return 0;
}
```

Another way to solve this Problem

↳ if(target ==0)

↳ clear bit

↳ if(target ==1)

↳ set bit

7. Single number (Leetcode-136)

Problem Statement:
Given a non-empty array of integers nums, every element appears twice except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:
Input: nums = [2,2,1]
Output: 1

Example 2:
Input: nums = [4,1,2,1,2]
Output: 4

Example 3:
Input: nums = [1]
Output: 1

Logic   4 1 2 1 2

$\Rightarrow 0 \wedge 4 \wedge \cancel{1} \wedge \cancel{2} \wedge \cancel{1} \wedge \cancel{2}$

$\Rightarrow 4$

property $\boxed{0 \wedge X = X}$

$\boxed{0000\ 0100}$

DRY RUN

0 ⇒ 0000 0000
4 ⇒ ^ 0000 0100

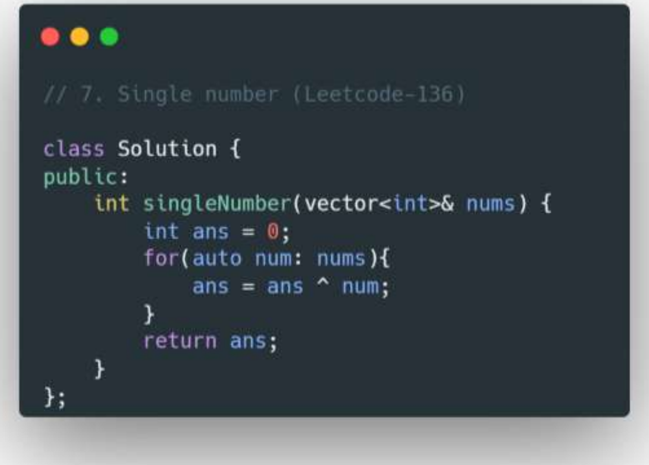
X ⇒ 0000 0100
1 ⇒ 0000 0001

Y ⇒ 0000 0101
2 ⇒ 0000 0010

Z ⇒ 0000 0111
1 ⇒ 0000 0001

M ⇒ 0000 0110
2 ⇒ 0000 0010

output



```
// 7. Single number (Leetcode-136)

class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int ans = 0;
        for(auto num: nums){
            ans = ans ^ num;
        }
        return ans;
    }
};
```

T.C. ⇒ O(N)

S.C. ⇒ O(1)

Related Questions

① single Number II

② single Number III

8. Clear n bits from last

Input N=15 and i=3

Output 8

Explanation 15 ⇒ 0000 1$\boxed{111}$ → Last 3 bits

Ans ⇒ 0000 1000

Logic

15 ⇒ 0000 1111

Mask ⇒ 1111 1000

N & mask ⇒ 0000 1000 → output

How to create this mask ?

Take 1 ⇒ 0000 0001

1 << 3 ⇒ 0000 1000

Take 2's complmnt ⇒ 1111 1000

mask

How to take 2's complmnt ?

How to Represent the -1 in binary

$-1 \Rightarrow \boxed{1}000 \quad 0001$

-ve

$\boxed{\begin{array}{l} \text{2's compliment of 1} \\ 1111 \quad 1111 \end{array}}$

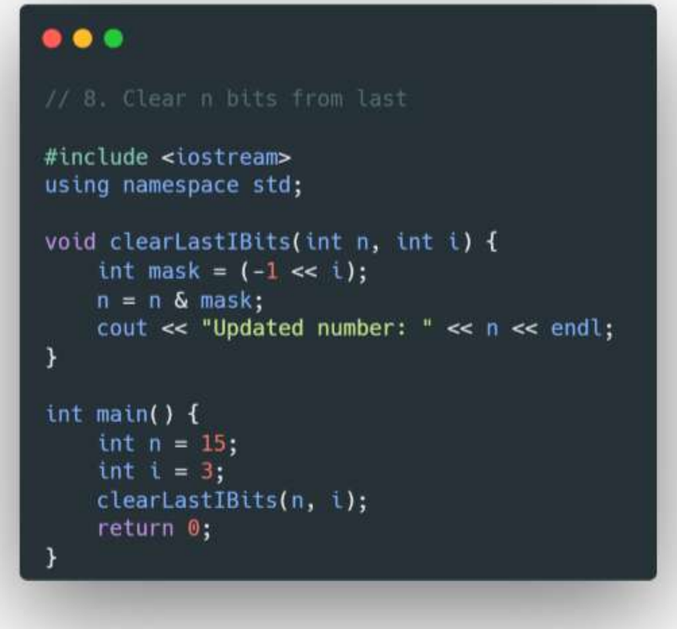
This is the -1

$\boxed{mask = -1 << i}$

$-1 \Rightarrow 1111 \quad 1111$

$-1 << 3 \Rightarrow \underset{\times\ \times\ \times}{111} \ \underset{mask}{\boxed{1111 \quad 1000}}$

$\boxed{Ans = N \ \& \ mask}$



```
// 8. Clear n bits from last

#include <iostream>
using namespace std;

void clearLastIBits(int n, int i) {
    int mask = (-1 << i);
    n = n & mask;
    cout << "Updated number: " << n << endl;
}

int main() {
    int n = 15;
    int i = 3;
    clearLastIBits(n, i);
    return 0;
}
```

[Read about 2's complement to better understanding]

9. Check power of two

input N = 16

output True

$\boxed{2^4 = 16}$

input N = 12

output False

$$\boxed{\begin{array}{l} 2^0 = 1 \\ 2^1 = 2 \\ 2^3 = 8 \\ 2^4 = 16 \end{array}}$$

power of 2

2 ⇒ 0000 0010

4 ⇒ 0000 0100

8 ⇒ 0000 1000

16 ⇒ 0001 0000

set bit count is 1

→ TRUE

Not power of 2

1 ⇒ 0000 0001

3 ⇒ 0000 0011

10 ⇒ 0000 1010

12 ⇒ 0000 1100

set bit count is not 1

→ False

Method I

Count 1 jab tak N != 0 Hai

4 => 0000 0100

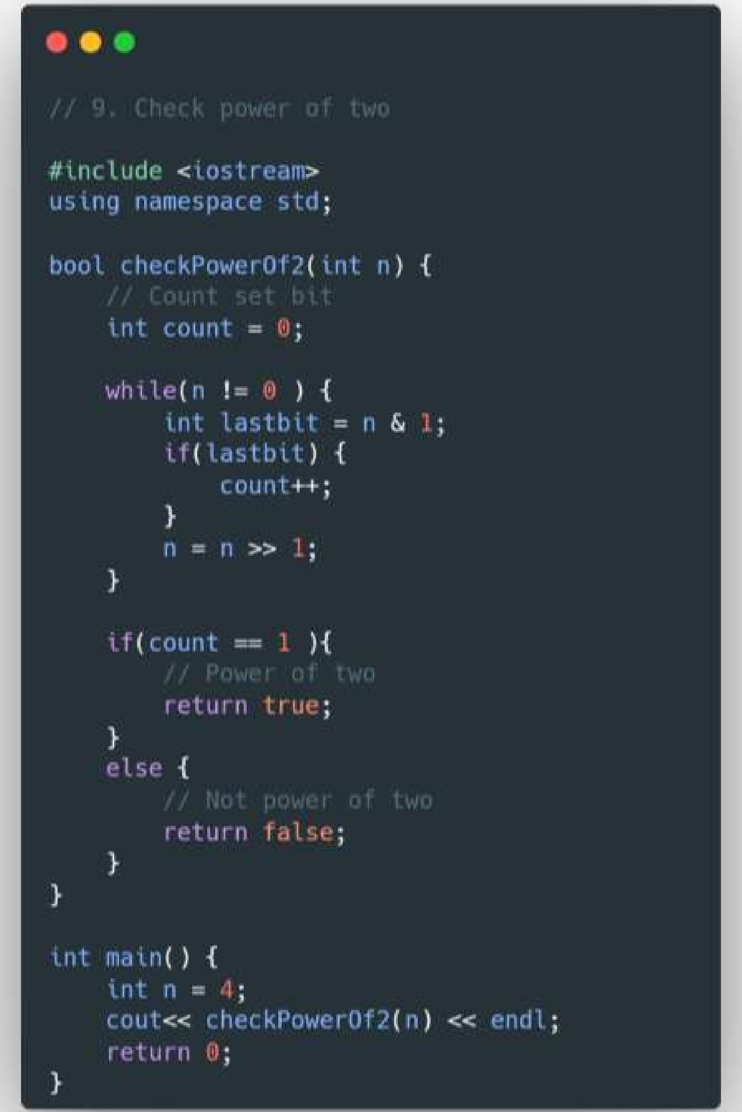
iteration1 N=4 count = 0
Lastbit = 0
N = N >> 1
$\boxed{N = 0000\ 0010}$

itern2 N=2 count = 0
Lastbit = 0
N= N >> 1
$\boxed{N = 0000\ 0001}$

itern3 N=1 count=0
Lastbit = 1
Count = 1
N = N >> 1
$\boxed{N = 0000\ 0000}$

itern 4 N=0 count = 1
↳ stop $\boxed{N != 0}$

$\boxed{\text{Count} == 1 \ ↳ \text{TRUE}}$



```cpp
// 9. Check power of two

#include <iostream>
using namespace std;

bool checkPowerOf2(int n) {
    // Count set bit
    int count = 0;

    while(n != 0 ) {
        int lastbit = n & 1;
        if(lastbit) {
            count++;
        }
        n = n >> 1;
    }

    if(count == 1 ){
        // Power of two
        return true;
    }
    else {
        // Not power of two
        return false;
    }
}

int main() {
    int n = 4;
    cout<< checkPowerOf2(n) << endl;
    return 0;
}
```

T.C. = O(N)

S.C. = O(1)

METHOD II

(Remove last set bit)

$$\underbrace{\text{Formula} = N \,\&\, (N-1)}_{0 \Rightarrow \text{power of Two}}$$

N=4

Ex 4 ⇒ 0000 0100

3 ⇒ 0000 0011

4 & 3 ⇒ 0000 0000

power of 2 = 4

$\boxed{2^2}$

N=6

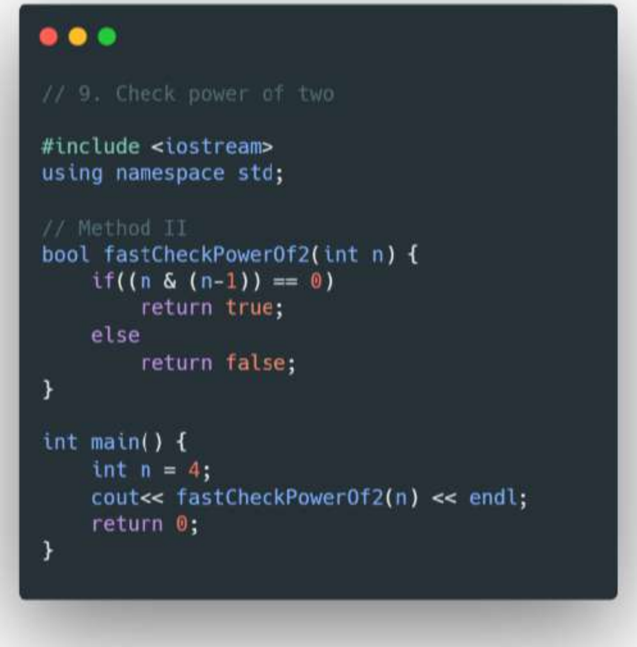
Ex

6 ⇒ 0000 0110

5 ⇒ 0000 0101

6 & 5 ⇒ 0000 0100

Not power of 2 = 6



```
// 9. Check power of two

#include <iostream>
using namespace std;

// Method II
bool fastCheckPowerOf2(int n) {
    if((n & (n-1)) == 0)
        return true;
    else
        return false;
}

int main() {
    int n = 4;
    cout<< fastCheckPowerOf2(n) << endl;
    return 0;
}
```

T.C. & S.C. = O(1)

10. Count set bits

Input N = 10

Output 2

Explanation 10 ⇒ 0000 1010

set bits = 2

Method I

using Loop

T.C. = O(N)

Slow S.C. = O(1)

Entire traversal in case of $\boxed{10}$

Method II

using Formula

T.C. = O(N)

Fast S.C. = O(1)

Not Entire traversal in case of $\boxed{10}$

DRY RUN   N = 10

N ⇒ 0000 1010

Itern 1   N = 10   count = 0

N != 0

↳ count = 1

N = (N) & (N-1)

$\boxed{N = 0000\ 1000}$

Itern 2   N = 8   count = 1

N != 0

↳ count = 2

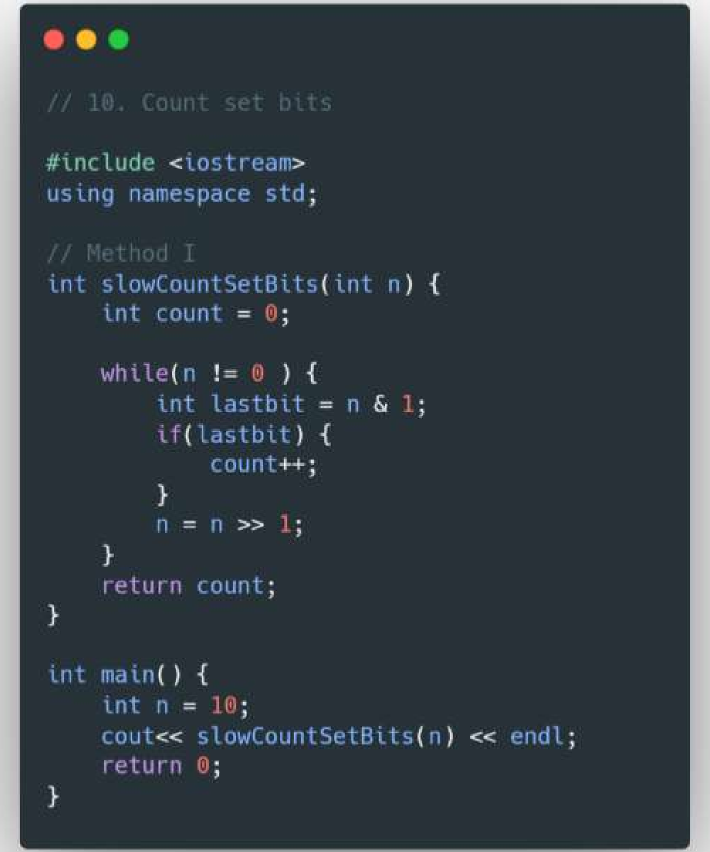
N = (N) & (N-1)

$\boxed{N = 0000\ 0000}$

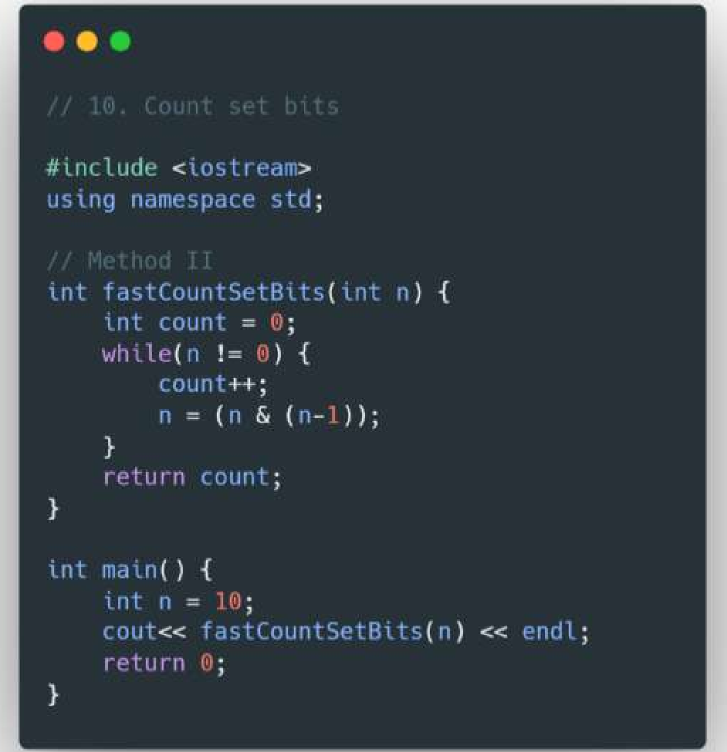
Itern 3   N = 0   count = 2

↳ stop (N != 0)
X

$\boxed{Ans = 2}$



```cpp
// 10. Count set bits

#include <iostream>
using namespace std;

// Method I
int slowCountSetBits(int n) {
    int count = 0;

    while(n != 0 ) {
        int lastbit = n & 1;
        if(lastbit) {
            count++;
        }
        n = n >> 1;
    }
    return count;
}

int main() {
    int n = 10;
    cout<< slowCountSetBits(n) << endl;
    return 0;
}
```



```cpp
// 10. Count set bits

#include <iostream>
using namespace std;

// Method II
int fastCountSetBits(int n) {
    int count = 0;
    while(n != 0) {
        count++;
        n = (n & (n-1));
    }
    return count;
}

int main() {
    int n = 10;
    cout<< fastCountSetBits(n) << endl;
    return 0;
}
```

11. Clear bits in range

[0-Index based]

Input N=255 , i=4 , j=1

Output 225

Explanation

i=4 j=1

N => 1111 1111

Ans => 1110 0001

Right to left

Logic

255 => 1111 1111

mask => 1110 0001

N & mask => 11100001 ← output

How to create this mask?

Create mask

using $i = 4$

$$\boxed{a = (-1 << (i+1))}$$

$$-1 \Rightarrow 1111\ 1111$$

$$-1 << 5 \Rightarrow \underset{\times\times\times\times}{1111}\ \underset{\times\times}{11}\underbrace{\boxed{1100\ 0000}}_{a}$$

using $j = 1$

$$\boxed{b = \sim(-1 << j)}$$

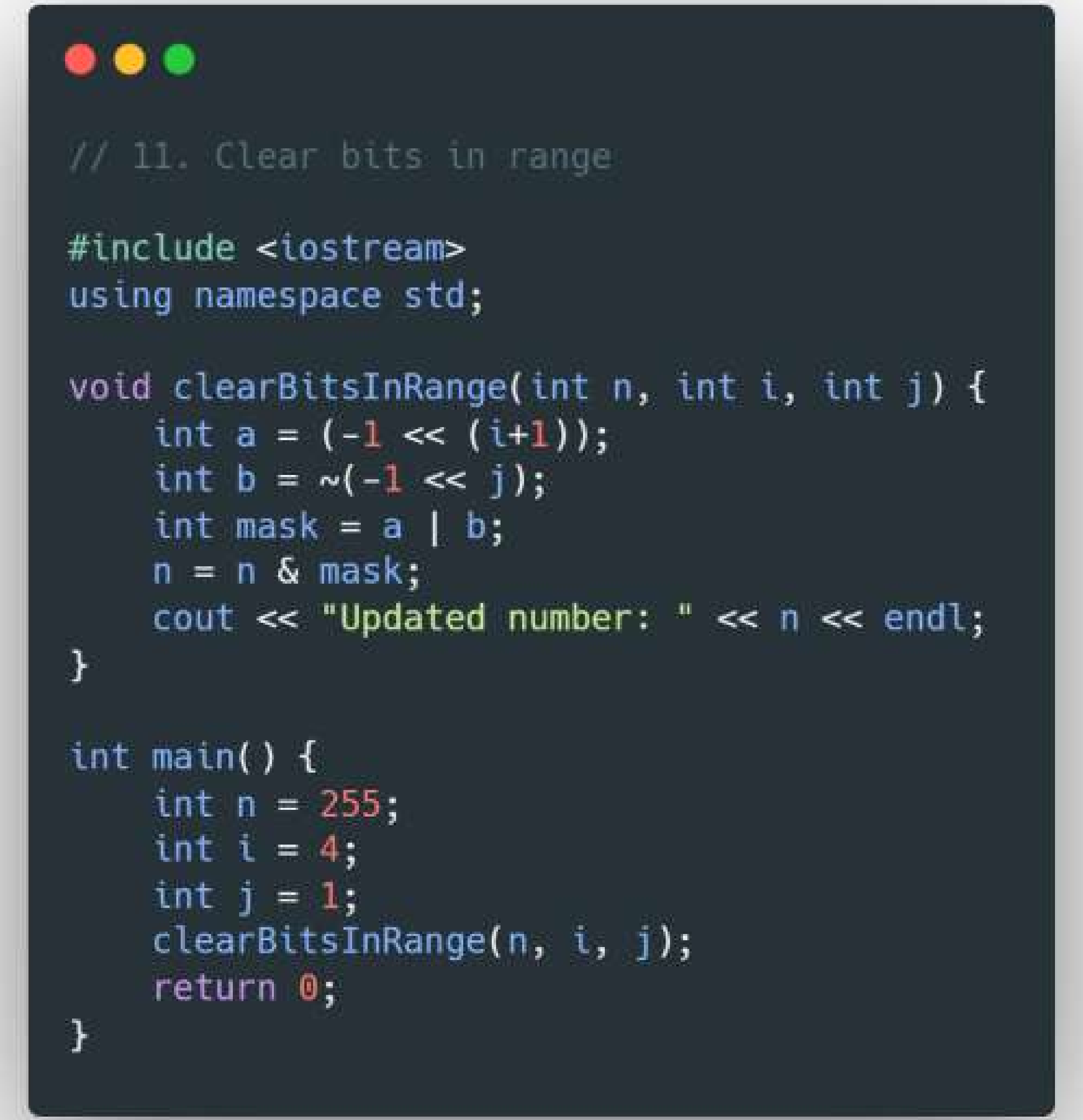
$$-1 \Rightarrow 1111\ 1111$$

$$-1 << 1 \Rightarrow \underset{\times}{1}\boxed{1111\ 1110}$$

$$\sim(-1 << 1) \Rightarrow \underbrace{\boxed{0000\ 0001}}_{b}$$

Final mask :

$$a \text{ OR } b \Rightarrow \begin{aligned} a &\Rightarrow \quad 1100\ 0000 \\ b &\Rightarrow \text{OR } 0000\ 0001 \\ \hline &\qquad 1100\ 0001 \longleftarrow \underline{\text{mask}} \\ \hline \end{aligned}$$



```cpp
// 11. Clear bits in range

#include <iostream>
using namespace std;

void clearBitsInRange(int n, int i, int j) {
    int a = (-1 << (i+1));
    int b = ~(-1 << j);
    int mask = a | b;
    n = n & mask;
    cout << "Updated number: " << n << endl;
}

int main() {
    int n = 255;
    int i = 4;
    int j = 1;
    clearBitsInRange(n, i, j);
    return 0;
}
```