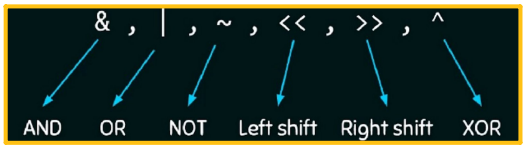
# BITWISE OPERATORS AND LOOPS

03 September 2023 18:48

class : 5th — 03/09/23

https://www.linkedin.com/in/manojoffcialmj/

**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 :**

SAME = 0
Diff. = 1

| A | B | AND | OR | XOR |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

| A | NOT |
|---|---|
| 0 | 1 |
| 1 | 0 |

**Program of Bitwise Operators :**

```cpp
#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 :**

```cpp
#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 → 2s 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:

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

int main(){
    bool num=1;

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

    return 0;
}

@manojofficialmj
```

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

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

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

    return 0;
}

@manojofficialmj
```

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

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

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

    return 0;
}

@manojofficialmj
```

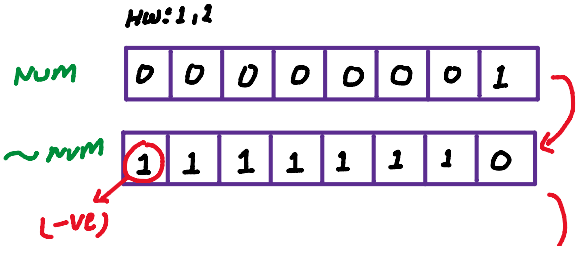
**Note**

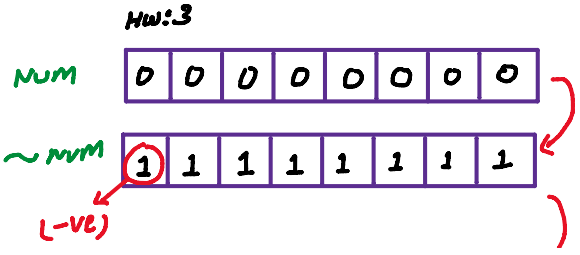
num = 1 → True (1)

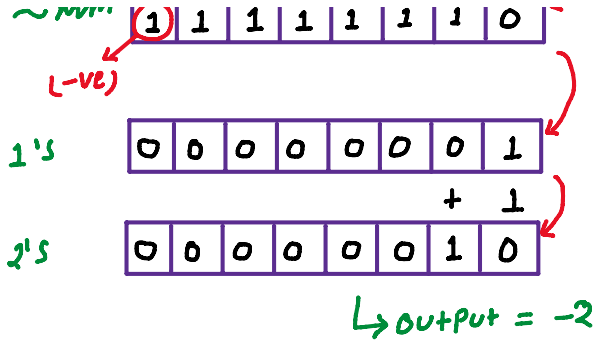
num = 0 → False (0)

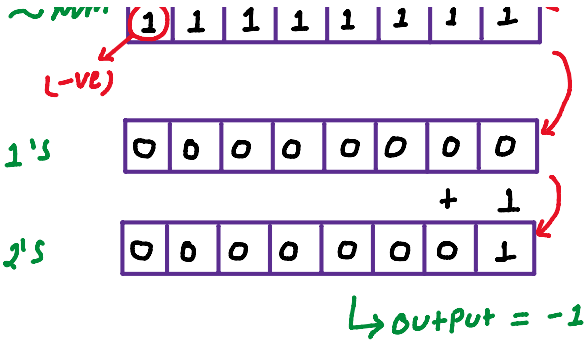
num = 2 → True (1)

- 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.






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

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

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

    return 0;
}
```

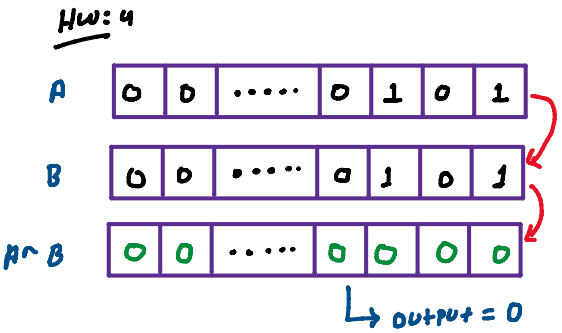
@manojofficialmj

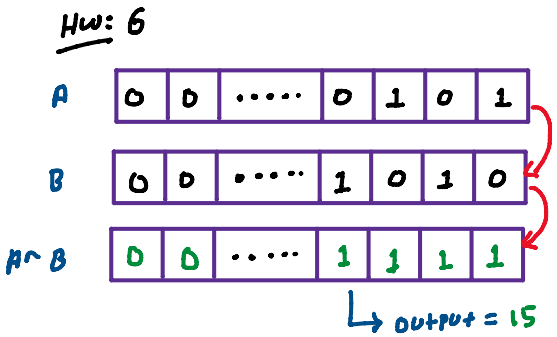
```cpp
// Homework 05
#include<iostream>
using namespace std;

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

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

    return 0;
}
```

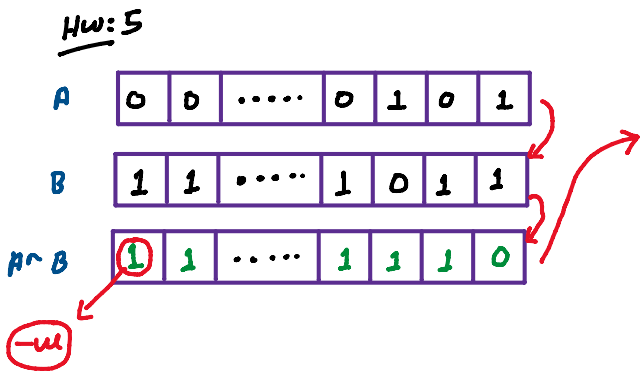
@manojofficialmj

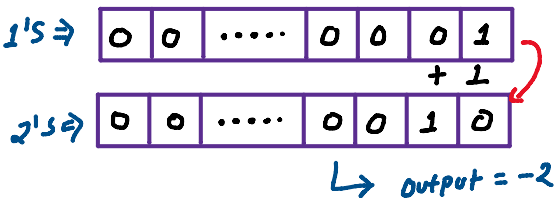
```cpp
// Homework 06
#include<iostream>
using namespace std;

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

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

    return 0;
}
```

@manojofficialmj



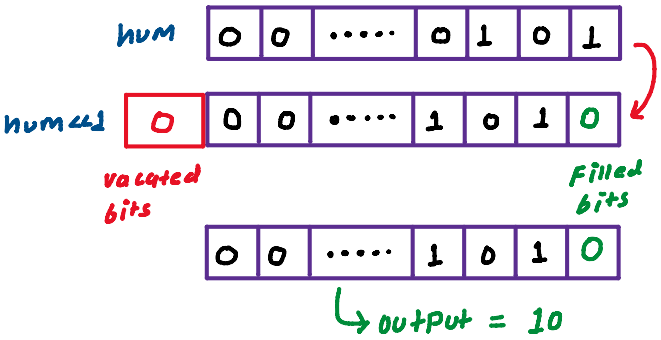




Bitwise left and right shift operators :

1

```cpp
#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;
}
```

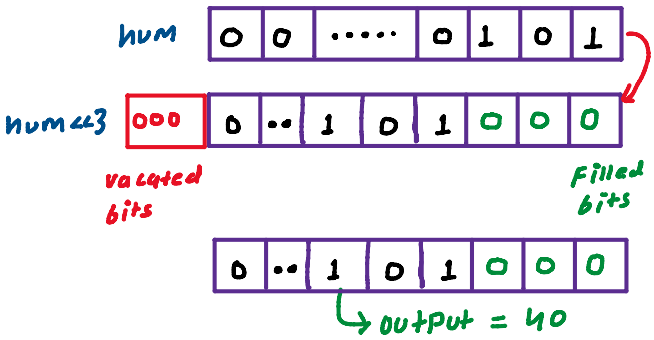
@manojofficialmj




2

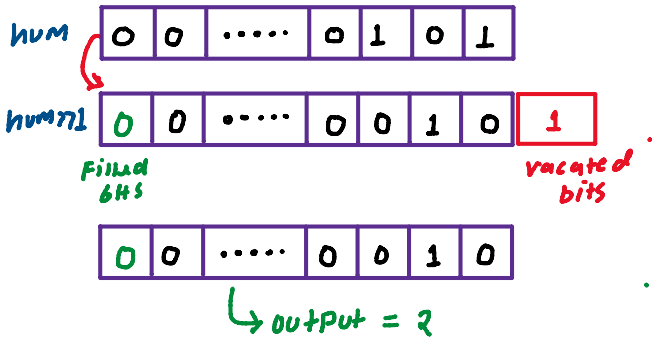
```cpp
#include<iostream>
using namespace std;

int main(){
    int num=5;

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

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

    return 0;
}
```

@manojofficialmj



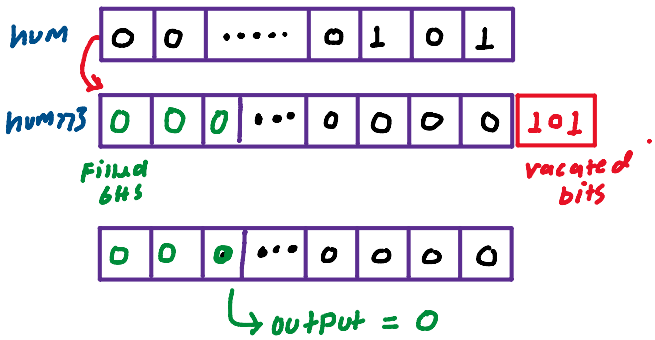

```cpp
#include<iostream>
using namespace std;

int main(){
    int num=5;

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

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

    return 0;
}
```

@manojofficialmj




```cpp
#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;
}
```

@manojofficialmj



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.

Unsigned

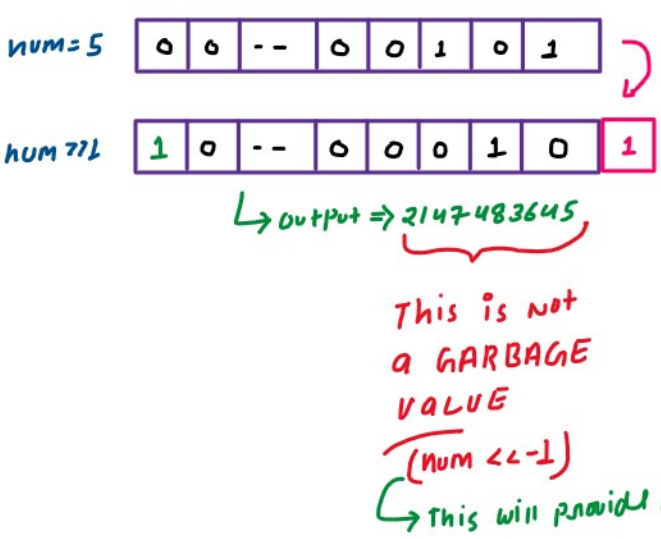
```cpp
#include<iostream>
using namespace std;

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

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

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

    return 0;
}
```

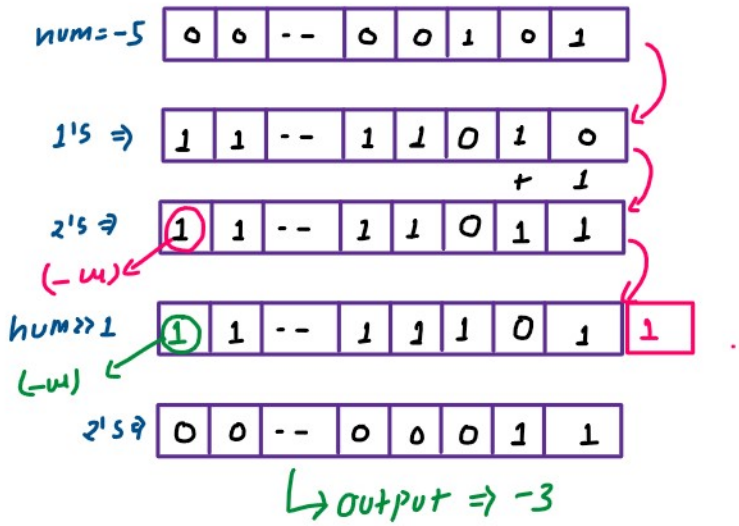
@manojofficialmj

signed

```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;
}
```

@manojofficialmj




## Pre/Post Increment and Decrement operators :

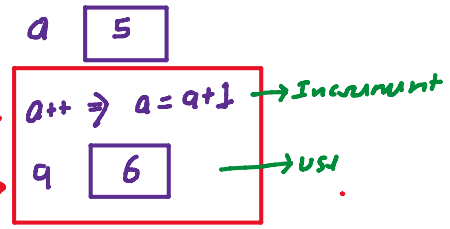
| | |
|---|---|
| Pre Increment ++a : first incremented by one, and then use<br>Post Increment a++ : first use , and then incremented by one | Pre Decrement --a : first decremented by one, and then use<br>Post Decrement a-- : first use , and then decremented by one |

1

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

int main(){
    int a=5;

    (++a); // Incremented by one, and then use
    cout<<a; // 6

    return 0;
}
```

@manojofficialmj

O/P = 6

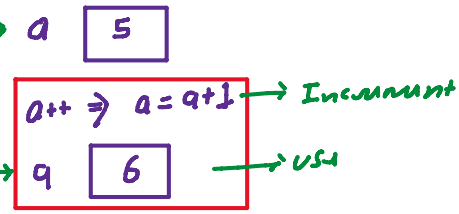



2

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

int main(){
    int a=5;

    cout<<(++a); // Incremented by one, and then use = 6

    return 0;
}
```

@manojofficialmj

O/P = 6

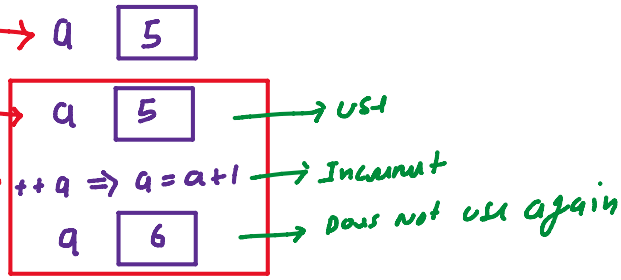



3

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

int main(){
    int a=5;

    cout<<(a++); // first use = 5, then incremented by one = 6

    return 0;
}
```

@manojofficialmj

O/P = 5

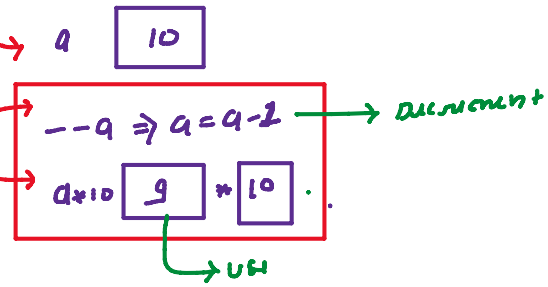



4

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

int main(){
    int a=10;

    cout<<(--a)*10; // first decremented by one = 9, and then use = 90

    return 0;
}
```

@manojofficialmj

O/P = 90

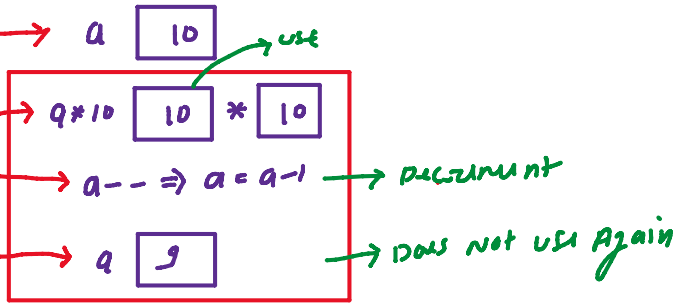



5

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

int main(){
    int a=10;

    cout<<(a--)*10; // first use = 100 , and then decremented by one = 9

    return 0;
}
```

@manojofficialmj

O/P = 100



6

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

int main(){
    int a=21;

    cout<<(++a)<<endl; // 22
    cout<<(a++)<<endl; // 22
    cout<<(a)<<endl;   // 23

    return 0;
}
```

@manojofficialmj

7

```cpp
// Program 07
#include<iostream>
using namespace std;

int main(){
    int a=10;

    cout<<(++a)*10<<endl; // 11*10=110
    cout<<(a++)*10<<endl; // 11*10=110
    cout<<(a)<<endl;      // 12

    return 0;
}
```

@manojofficialmj

Online C++ Compiler

C++ Online Compiler | Run Code | Output

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

int main(){
    int a=10;

    cout<<((++a)*(a++)); // 132
    return 0;
}
```

```
132
```

## Break and continue keyword :

**Using break to exit a Loop:**
*It is used to get out of the loop when a particular condition occurs.*

**Using continue to continue a Loop:**
*It is used to skip the iteration of the loop when a particular condition occurs.*

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

int main(){
    for(int i=0; i<5; i++){
        if(i==2){
            break;
        }
        cout<<i<<endl;
    }
    return 0;
}

/*
OUTPUT:
0
1
*/
```

Scope of Loop

@manojofficialmj

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

int main(){
    for(int i=0; i<5; i++){
        if(i==2){
            continue;
        }
        cout<<i<<endl;
    }
    return 0;
}

/*
OUTPUT:
0
1
3
4
*/
```

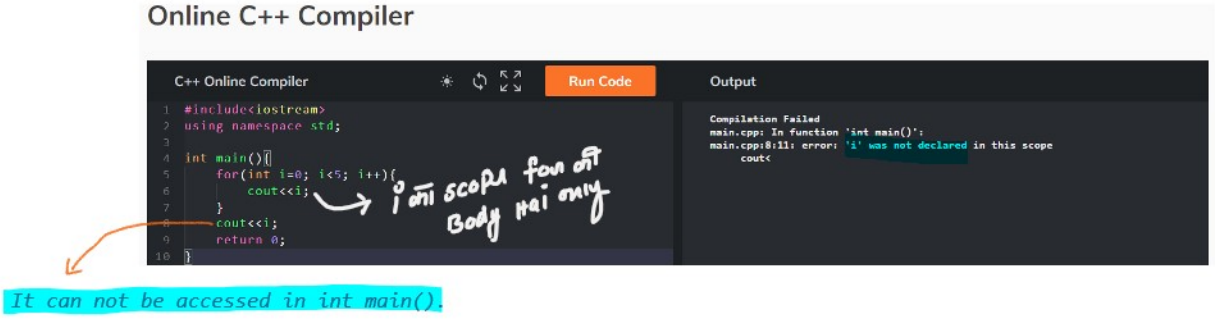
Scope of Loop

@manojofficialmj

## Variable scoping :

**Local variable**
*It can not be redefined in the same scope.*
*It is limited to curly braces {---}.*

**Global variable**
*It can be used anywhere in the same file.*
*It is bad practice to use in a program.*




*It can not be accessed in int main().*

## Operator precedence table :

1. Does not need to learn this table.
2. Use BRACKETS() to avoid learning of this table.
3. Cant be compared to the BODMAS rule

Ex ((2*3) + (5/10)) - 2

↓ ↓ ↓

((6) + (0.5)) - 2

(6 + 0.5) - 2

(6.5) - 2

6.5 → Final output

| OPERATOR | TYPE | ASSOCIAVITY |
|---|---|---|
| () [] . -> | | 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 |
| ^ | Bitwise EX-OR Operator | left-to-right |
| \| | Bitwise OR Operator | left-to-right |
| && | Logical AND Operator | left-to-right |
| \|\| | Logical OR Operator | left-to-right |
| ? : | Ternary Conditional Operator | right-to-left |
| = += -= *= /= %= &= ^= \|= <<= >>= | Assignment Operator | right-to-left |
| , | Comma | left-to-right |