C Operator Precedence Table

Listed in order of precedence.

		Associativity
( )	Parentheses: grouping or function call	left-to-right
[ ]	Brackets: array subscript	
.	Member selection via object name	
→	Member selection via pointer	
++ --	Postfix increment/decrement	
++ --	Prefix increment/decrement	right-to-left
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(type)	Cast (convert value to temporary value of type)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes	

* / %	Multiplication/ division/ modulus	left-to-right
+ -	Addition/ subtraction	
<< >>	Bitwise shift left, bitwise shift right	
< <=	Relational less than/ less than or equal to	
> >=	Relational greater than / greater than or equal to	
== !=	Relational is equal than/ is not equal to	
&	Bitwise AND	
^	Bitwise exclusive OR	
|	Bitwise inclusive OR	
&&	Logical AND	
||	Logical OR	
?:	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/ substraction assignment	
*= /=	multiplication/ division assignment	
%= &=	modulus / bitwise AND assignment	
^= |=	Bitwise exclusive/ inclusive OR assignment	
<<= >>=	Bitwise shift left / right assignment	
,	comma (Separate expressions)	left-to-right

Vector of Vectors in C++

↳ vectos: dynamic arrays

↳ two-dimensional vector

↳ accessed using iterators

↳ Syntax: vector<vector<data_type>>vec;

Example: vector<vector<int>> vec {{9, 8, 7},
{6, 5, 4},
{3, 2, 1}};

Insertion in vectors:

↳ vector_name.push_back(value)
↳ function used for insertion

Example: v2 = {1, 2, 3}
v1.push_back(v2);

This function pushes vector v2 into vector of vectors v1. Therefore v1 becomes {{1,2,3}}.

Removal or deletion in a vector of vectors :

↳ Syntax : vector_name[row-position].pop_back()

Example : Let v={{1,2,3}, {4,5,6}, {7,8,9}}

v[2].pop_back()

Output : {{1,2,3}, {4,5,6}, {7,8}}

The function removes element 9 from the last row vector.

Example : v[1].pop_back()

This function removes element 6 from the last row vector.

∴ v becomes {{1,2,3}, {4,5}, {7,8}}

Traversal of a Vector of Vectors :

↳ Syntax :

```
for i in [0,n)
{
    for (iterator it = v[i].begin(); it != v[i].end(); it++)
    {
        // operations to be done
        print(*it)
    }
}
```

Example :

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    vector<vector<int>> vec{{1,2,3},
                            {4,5,6},
                            {7,8,9}};

    for (int i=0; i<3; i++){
        for (
```

```
        auto it = vec[i].begin();
        it != vec[i].end(); it++)
        cout << *it << " ";
    cout << endl;
  }
  return 0;
}
```

Output:

```
1  2  3
4  5  6
7  8  9
```

Approach to find missing number in an array

① **Sorting:** Sort the array and compare each index with the value of that index if it is not equal then return the index. TC = O(nlogn), SC = O(1)

② **Hashing:** Create a hashmap and store all the values inside it. Iterate over the range 0 to the size of array. Return the first element not present inside the set or hashmap TC = O(n), SC = O(n)

Sum of N whole numbers: Store the sum of array. Store the sum of N whole numbers. Return their difference.

TC = O(n), SC = O(1)

Solutions: ①

```
int MissingNumber(vector<int>& array, int n) {
    int sum = accumulate(array.begin(), array.end(), 0);
    return n*(n+1)/2 - sum;
}
```

This will return an integer value.

Its return float, use 0.0.

(2)
```
int MissingNumber(vector<int>& array, int n){
    int res = array[0];
    for(int i=1; i<n-1; i++)
    {
        res^= array[i];
    }
    for(int i=1; i<=n; i++)
    {
        res^= i;
    }
    return res;
}
};
```

Use of C++ Bitwise XOR Operator

↳ Returns 1 if and only if one of the operands is 1.

↳ If both are 0/1, result is 0.

a	b	a^b
0	0	0
0	1	1
1	0	1
1	1	0

For example:

12 = 00001100
25 = 00011001
Bitwise XOR
00010101 = 21 (in decimal)

↳ use the same logic in above.

Example: array = {1, 2, 3, 5}

res = 1

Step 1: res = 1^2 (01^10 = 11) in binary

2: res = (11)^(11) (00)

3: res = (00)^(101) (101)

res = 1 correct ans.

2nd loop 1: res = (001)^(101) [100]

2: res = (100)^(010) [110]

3: res = (110)^(011) [101]

4: res = (101)^(100) [001]

Minimum XOR value pair:

Example: N = 3

arr[] = {9, 5, 3}

Output = 6

Explanation: There are 3 pairs -

9 ^ 5 = 12

5 ^ 3 = 6

9 ^ 3 = 10

So minimum is 5 ^ 3 = 6.

①
```
int minxorpair (int N, int arr[]){
    sort(arr, arr+N);
    int min=9999;
    for(int i=0; i<N; i++)
    {
        if ((arr[i] ^ arr[i+1]) < min)
        min = arr[i] ^ arr[i+1];
    }
    return min;
}
```

TC = O(nlogn) SC = O(n)

②
```
int minxorpair(int N; int arr[]){
    int minXor = INT_MAX;
    int val = 0;
    sort(arr, arr+N);
    for(int i=0; i<N; i++)
    {
        val = arr[i] ^ arr[i+1];
        minXor = min(minXor, val);
    }
    return minXor;
}
```

Trie (Insert & Search)

- It reduces search complexities to optimal limit (key length).
- Every node of Trie consists of multiple branches.
  - Each branch represents a possible character of keys.
  - The last node of every key is marked as end of word node
  - isEndOfWord is used to distinguish the node as end of word node.

Two Pointers Technique

↳ typically used for searching pairs in a sorted array.

Naive solution

```
for(int i=0 ; i<N; i++)
    {
      for(j=0; j< N; j++)
        {
          ||
        }
    }
```

Time Complexity = $O(n^2)$

↳ which can be reduced to $O(n)$

Police and Thieves Problem (gfg)

↳ using greedy algorithm

↳ if |index(PI) − index(TI)| <= k, the thief will be caught.

```
#include <bits/stdc++.h>
using namespace std;
int policeThief(char arr[], int n, int k){
    int caught=0;
    vector<int> thieves;
    vector<int> policeman;
```

```
for(int i=0; i<n;i++){
   if (arr[i]=='P')
      policeman.push_back(i);
   else if(arr[i]=='T')
      thieves.push_back(i);
 }
int thief=0, police=0;
while( thief<thieves.size() && police<policeman.size()){
   if(abs(thieves[thief]- policeman[police])<=k){
      caught++;
      thief++;
      police++;
    }
    else if( thieves[thief]< policeman[police])
     thief++;
    else
     police++;
   }
return caught;
}
```

• Next greater element (gfg)

Example: Input = [2, 7, 3, 5, 4, 6, 8]

Output = [7, 8, 5, 8, 6, 8, -1] ×

[7, 8, 5, 6, 6, 8, -1]

Steps:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
2	7	3	5	4	6	8

I/P

a1[0]	a2[1]	a2[2]	a2[3]	a2[4]	a2[5]	a[6]
7	8	5	6	6	8	-1

1st element: a[0] < a[1], a1[0] = a[1] = 7

2nd ele: a[2] < a[1] → a[3] < a[1]

a[4] < a[1], a[5] < a[1]

a[6] > a[1]

∴ a2[1] = a[6] = 8

3rd ele: a[2] < a[3]

∴ a2[2] = a[3] = 5

4th ele: a[3] > a[4]

a[3] < a[5]

∴ a2[3] = a[5] = 6

5th ele: a[4] < a[5]

∴ a2[4] = a[5] = 6

6th ele:

a[5] < a[6]

∴ a2[5] = a[6] = 8

7th ele:

a[6] = 8

Since there is no element after that and greater than 8.

∴ a2[6] = -1

(given value)

Brute force Approach

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

void nextgreaterElement( int arr[], int n)
{
    for (int i=0; i<n; i++)
    {
```

```
            int next = -1;
            for(int j= i+1; j< n;j++)
            {
              if(arr[j] > arr[i])
               {
                 next = arr[j];
                 break;
               }
            }
          cout << next << "  ";
      }
   }
int main()
{
    int arr[]= {2,7,3,5,4,6,8};
    int n = sizeof(arr)/sizeof(arr[0]);
    nextGreaterElement(arr,n);
    return 0;
}
```

Using Stack

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

passing a vector in the func"

```
vector<int> findNextGreaterElement(vector<int> input)
{
   int n= input.size();
   vector<int>result(n,-1);
```

→ initializing the vector with -1 value

```
   stack<int>s;
```

→ declaring a stack

```
   for(int i=0; i<n; i++)
    {
       while(!s.empty() && input[s.top()] < input[i])
```

checking if stack is empty

comparing the stack values

```
        {
```

```
            result[s.top()] = input[i];   if found a greater value then insert
            s.pop();                      into result vector.
        }                    pop that
        s.push(i);           value from the stack s
    }                                  push the value in the stack
    return result;
}

int main()
{
    vector<int> input = { 2,7,3,5,4,6,8 };
    vector<int> result = findNextGreaterElement(input);
    for(int i:result)
    {
        cout<<i<<" ";
    }
    return 0;
}
```

Steps:

1. Create an empty stack.
2. Loop till we have a greater element on top or stack becomes empty.
3. Keep popping elements from the stack smaller than the current element, and set their next greater element to the current element.
4. Push current index into the stack