1. **Maximum Subarray Sum – Kadane’s Algorithm**

Given an array arr[ ], the task is to find the subarray that has the maximum sum and return its sum.

**APPROACH** : Kadane algorithm

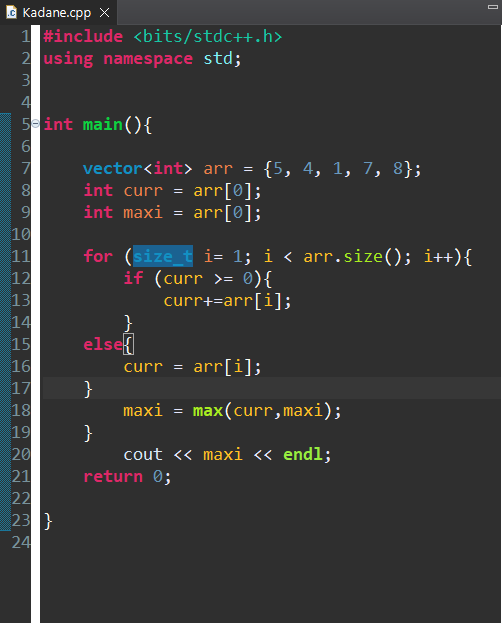
Option 1: Choose to include the current element in the sum

Option 2: Start a new sum from the current element

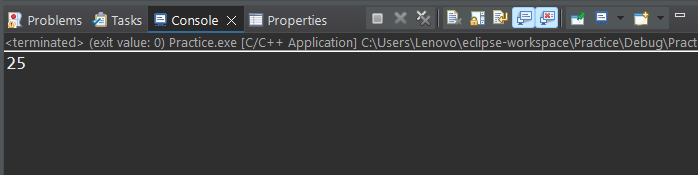
Time complexity = O(n)

Space complexity = O(1)

**CODE :**



**OUTPUT :**



**Various test cases :**

[-1, -2, -3, -4]

[1, 2, 3, 4, 5, 6 ]

[-5]

[1, -2, 3, -4, 5, -6]

**My code’s constraint handling capacity :**

1 <= n <= 10^6

1. **Maximum Product Subarray**

Given an integer array, the task is to find the maximum product of any subarray

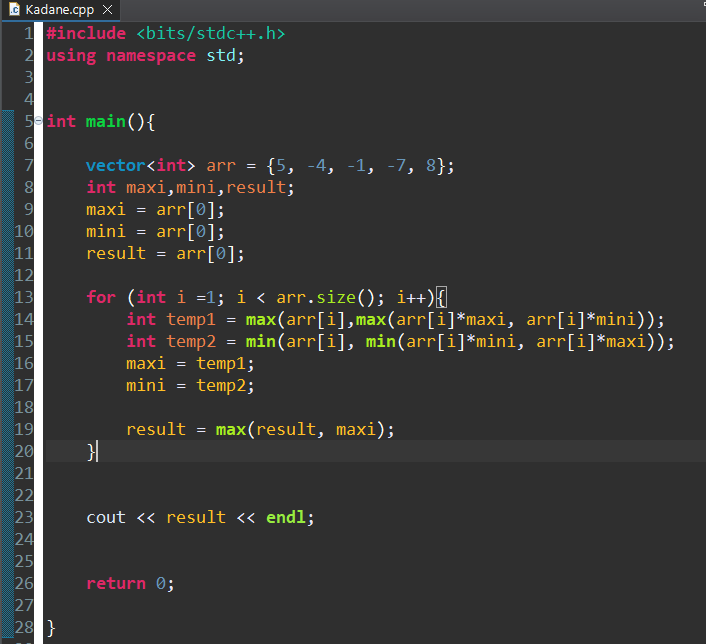
**APPROACH:**

Using two variables to keep track of the maximum and minimum value at each element. Returning the global maximum.

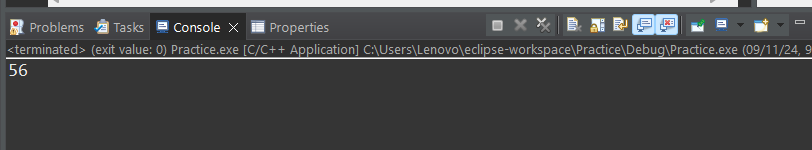
Time complexity : O(n)

Space complexity: O(1)

**CODE :**



**OUTPUT :**

****

**Various test cases :**

[-1, -2, -3, -4]

[1, 2, 3, 4, 5, 6 ]

[-5]

[1, -2, 3, -4, 5, -6]

**My code’s constraint handling capacity :**

1 <= n <= 10^6

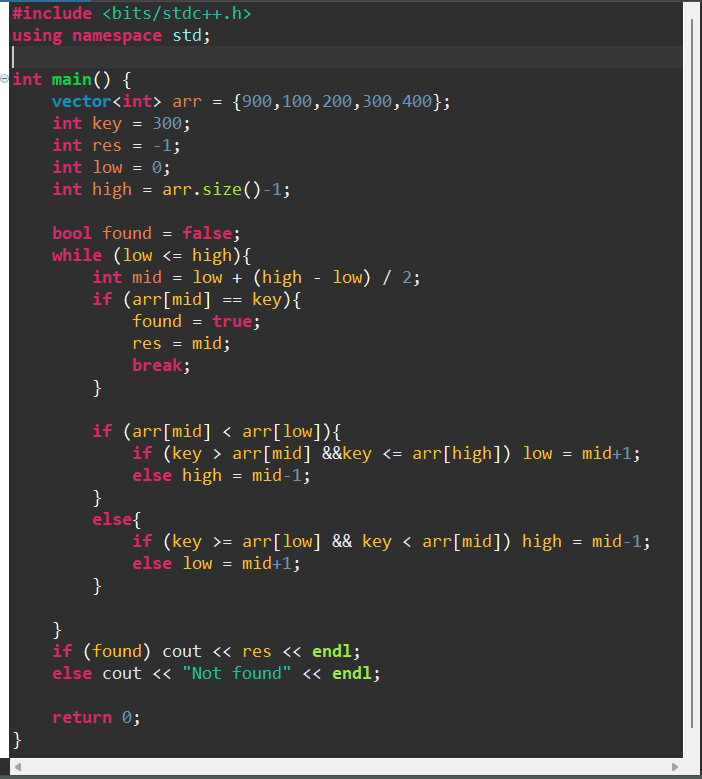
1. **Search in a sorted and rotated Array**

Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1.

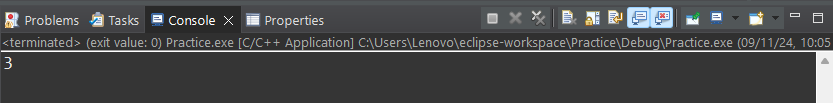
**APPROACH** : Using binary search to verify the sorted half, comparing the key element and proceeding with the correct half. Binary search approach

Time complexity: O(log N)

Space complexity : O(1)



**OUTPUT**

****

**Various test cases :**

[6, 7, 9, 15, 19, 2, 3] Key is in the first part

[15, 19, 2, 3, 6, 7, 9] Key is in the second part

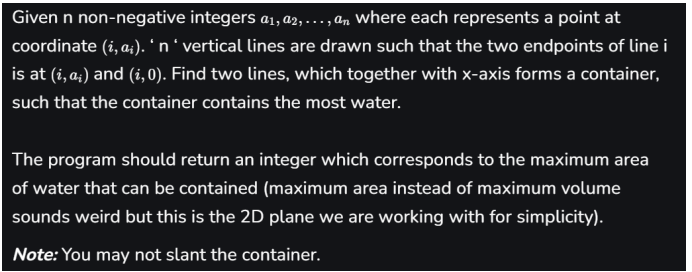
[4, 5, 6, 7, 0, 1, 2] Key is in the middle

[15, 19, 2, 3, 6, 7, 9, 10] even length

**My code’s constraint handling capacity :**

1 <= n <= 10^6

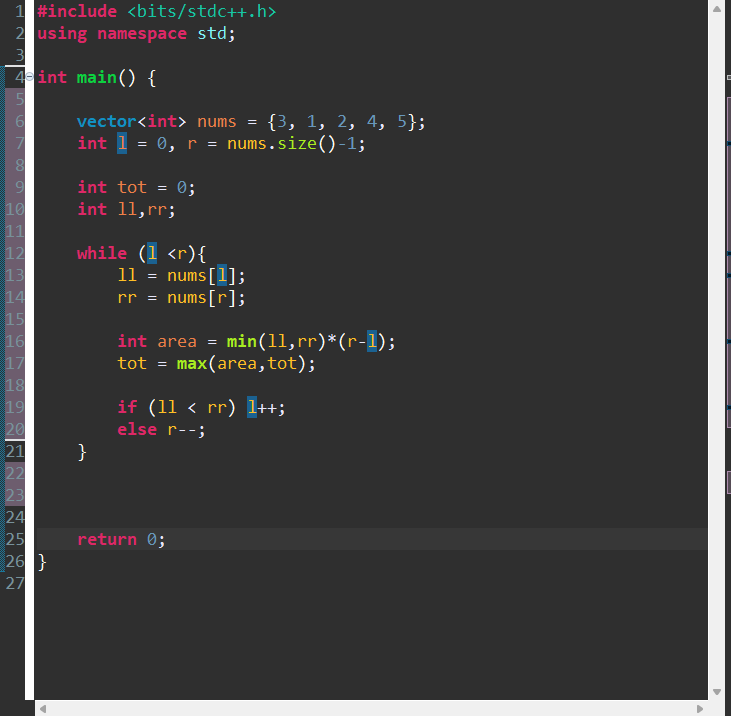
**4. Container with Most Water**

****

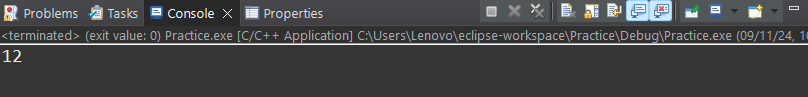
**APPROACH :** Two pointers approach, place left at the start and right at the end, calculate the max area attainable and then move l if l <r or move r to left, if r < l

Time complexity : O(N)

Space complexity : O(1)

****

**OUTPUT :**

****

**Various test cases :**

[1, 1]

[1]

[1, 3, 2, 4, 1, 3, 2, 4, 1]

[1, 2, 3, ..., 100000]

[100000, 99999, ..., 1]

**My code’s constraint handling capacity :**

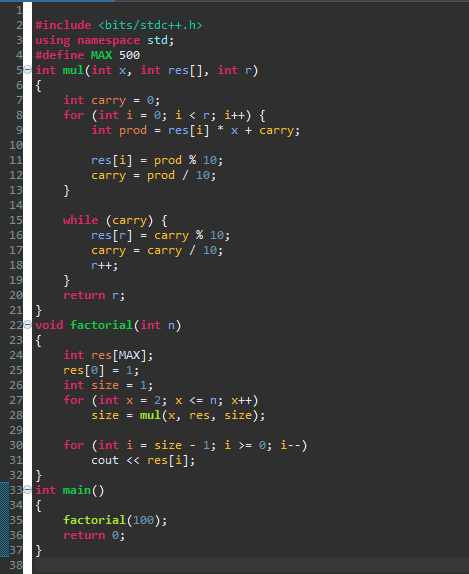
1 <= n <= 10^6

**5) Find the Factorial of a large number**

**APPROACH** : Since large numbers defy the logic of the integers, we have to use carry and then append all the digits of the result to the array of MAX = 500 size. The digits are stored in reverse order… hence print them in reverse from end to the start.

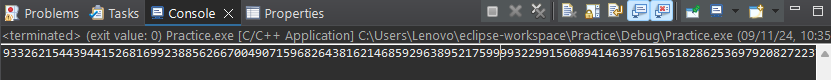
Time complexity : O(N)

Space complexity : O(1)

****

**OUTPUT :**

93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

****

**Various test cases :**

0

1

1000

**My code’s constraint handling capacity :**

1 <= n <= 1000

**6) Trapping Rainwater Problem**

Given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.

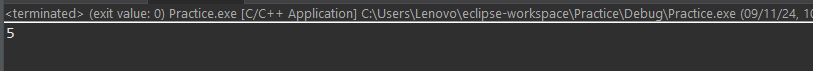
**APPROACH :** Two pointer approach. Check whether left is greater or right is greater, if left, then compare the left with the maximum left encountered until now, if greater, update maximum else add the diff to the result. If right > left, then compare the maximum with the right, and update if not greater than the right maximum, add their diff to res.

Time complexity : O(N)

Space complexity : O(1)



**OUTPUT :**

****

**Various test cases:**

[1]

[5, 5, 5, 5, 5]

[1, 2, 3, ..., 100000]

[100000, 99999, ..., 1]

[0,0,0]

**My code’s constraint handling capacity :**

1 <= n <= 10^6

**7) Chocolate Distribution Problem**

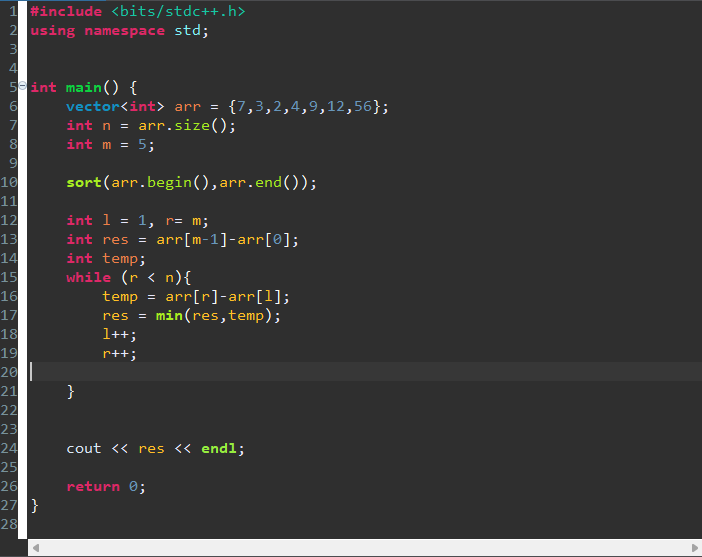
Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates.

There are m students, the task is to distribute chocolate packets such that: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimised

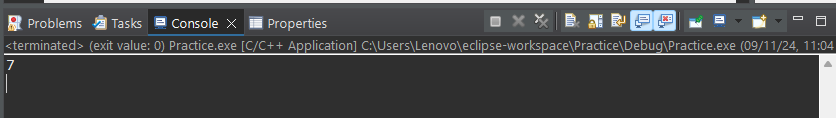
**APPROACH** : Sort the array and move the left and right pointer with ‘m’ gap between them from the start until the right reaches the end… calculate the diff between the r and l pointer and update the global minimum value. Return the global minimum.

Time complexity : O(N log N)

Space complexity : O(1)

.

**OUTPUT :**

****

**Various test cases :**

[1, 2, 3, 4, 5], m = 5

[1000, 2000, 3000, 4000, 5000], m=3

[1, 5, 3, 7, 9], m=2

[1, 5, 8, 12, 15, 20, 25], m=4

**My code’s constraint handling capacity :**

1 <= n <= 10^4

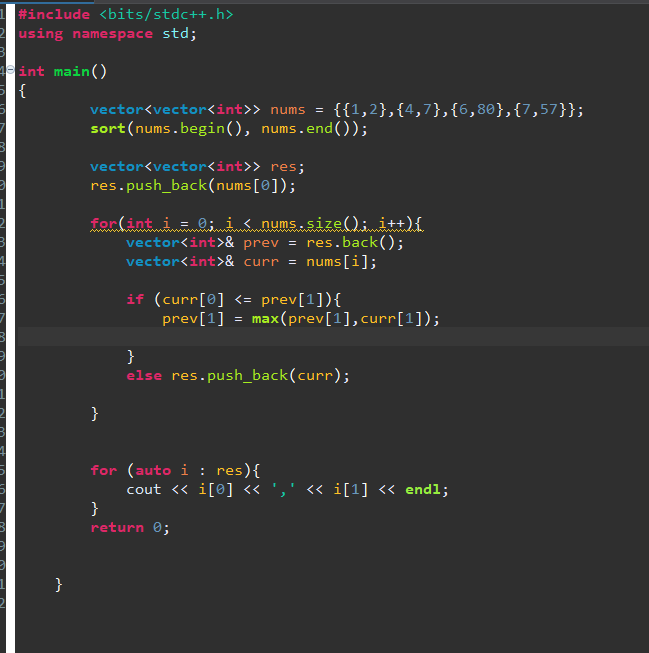
**8) Merge Overlapping Intervals.**

Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

**APPROACH** : Sorting the intervals in the ascending order and traversing through the sorted array and keeping track of the previous elements ending time and current elements start time. If the start time of the current element is lesser or equal to the previous element’s ending time, we update the current ending time as the prev and continue traversing. Else add the prev to the result array.

Time complexity: O(N log N)

Space complexity : O(N)



**OUTPUT :**

**Input ->** [[1,2],[4,7],[6,80],[17,57]]

****

**Various test cases :**

[[1, 2], [3, 4], [5, 6]] No overlapping

[[6, 8], [2, 4], [1, 3], [5, 7]] reverse order

[[1, 2], [10, 20], [30, 40]] many groups between single large group

[[3, 5], [1, 4], [2, 6], [7, 10]] one starts before the other ends (normal overlapping)

**My code’s constraint handling capacity :**

1 <= n <= 10^4

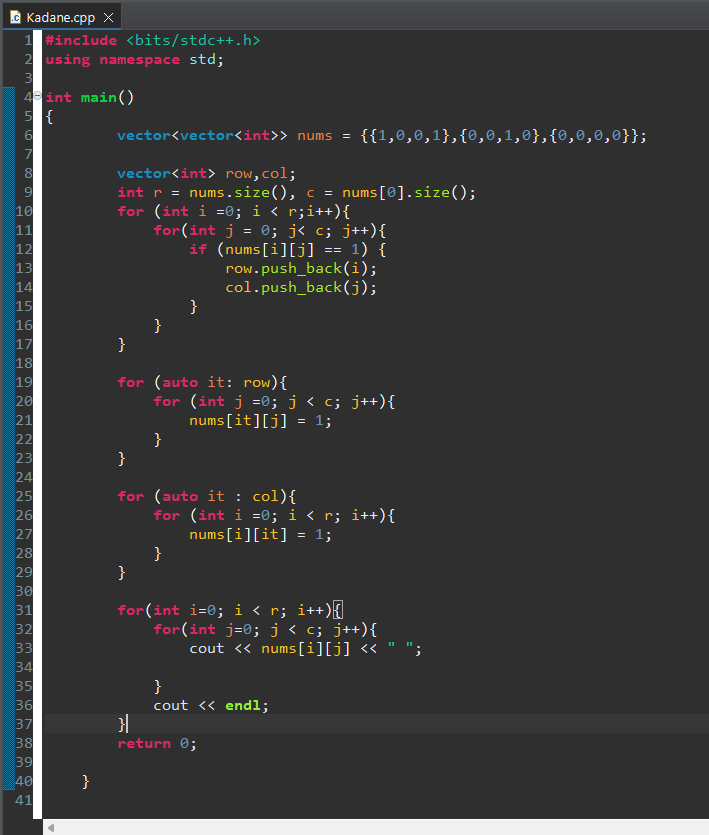
**9. A Boolean Matrix**

Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

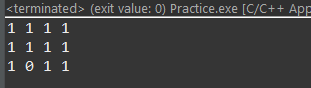
**APPROACH** : Finding the rows and columns of all the values that are 1 and then traversing through the rows and column list and changing it in the matrix.

Time complexity : O(m\*n)

Space complexity : O(m+n)

****

**OUTPUT :**

****

**Various test cases :**

[[1]]

[[0]]

[[0, 0, 0], [0, 0, 0], [0, 0, 0]]

[[1, 1, 1], [1, 1, 1], [1, 1, 1]]

[[0], [1], [0]]

**My code’s constraint handling capacity :**

1 <= n,m <= 10^3

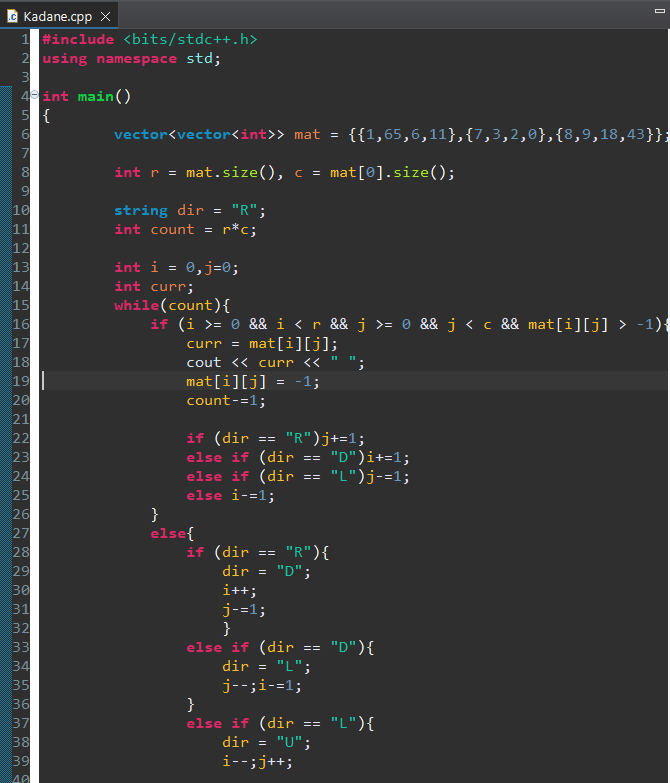
**10) Print a given matrix in spiral form**

Given an m x n matrix, the task is to print all elements of the matrix in spiral form.

**APPROACH** : Check whether the current row or column whether overflows the matrix, or if is already visited, change the direction according to the given conditions.

Time complexity : O(N\*M)

Space complexity : O(1)



**OUTPUTS :**

1. [1, 2, 3, 4, 5]



[1, 2, 3, 4],

[5, 6, 7, 8],

[9, 10, 11, 12],

[13, 14, 15, 16]



1. [42]



1. [ [1]

[2]

[3]

[4]

[5] ]



**My code’s constraint handling capacity :**

1 <= n,m <= 10^3

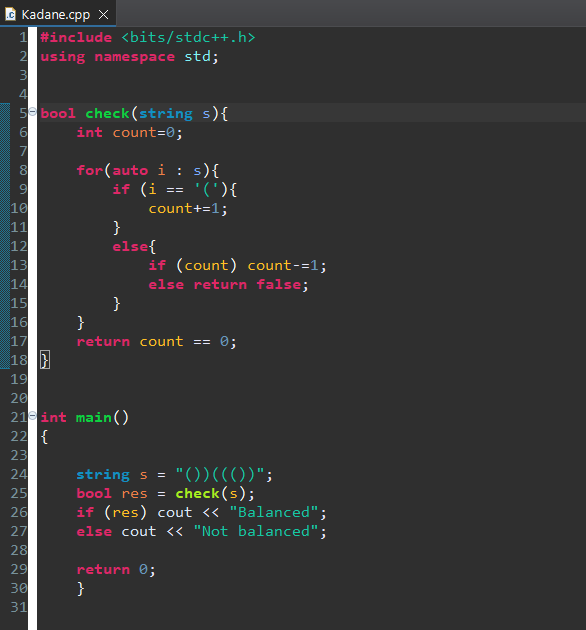
**13) Check if given Parentheses expression is balanced or not**

Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not.

**APPROACH** : Stack based approach. Push until it’s open and if close is encountered pop if the top is open else return unbalanced.

Time complexity : O(N)

Space complexity : O(1)



**OUTPUTS :**

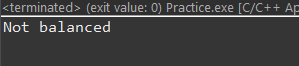
str = "((()()(()(()))(())))((()))()()"



str = "((()(()((()()(()()))(()(())))())))"



str = "("



str = "("\*99999 + ")"



**My code’s constraint handling capacity :**

1 <= n <= 10^6

**14) Check if two Strings are Anagrams of each other**

Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

**APPROACH** : Using hashmap and keeping track of the frequency of both strings in 2 diff hashmaps. If map1 == map2, return true, else false.

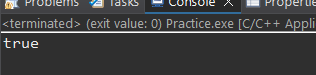
Time complexity : O(m+n)

Space complexity : O(1)

****

**OUTPUT :**

s1 = "" s2 = ""

****

S1 = “shgdyhak” s2 = “abdgteyh”

****

S1 = “” s2 = “u”

****

S1 = “silent” s2 = “listen”

****

**My code’s constraint handling capacity :**

1 <= n <= 10^6

1 <= m <= 10^6

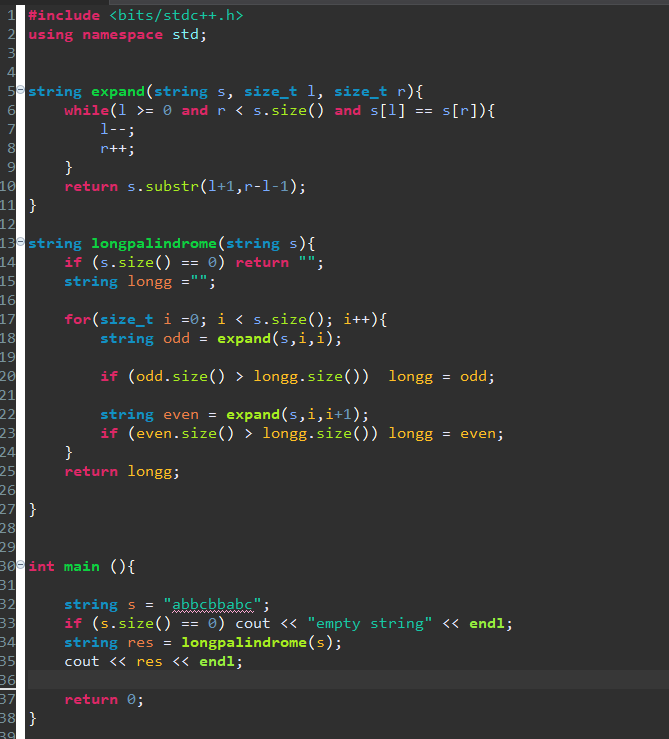
**15) Longest Palindromic Substring**

Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

**APPROACH** : Two pointers to expand from the in to out as long the end characters match or either left or right pointer runs out of their bounds. Updating long substring if needed and returning it at last.

Time complexity : O(n^2)

Space complexity : O(1)



**OUTPUT :**

**“a”**

****

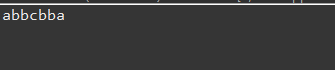
**“”**

****

**xyzzzzxy**

****

**abbcbbabc**

****

**My code’s constraint handling capacity :**

1 <= n <= 10^4

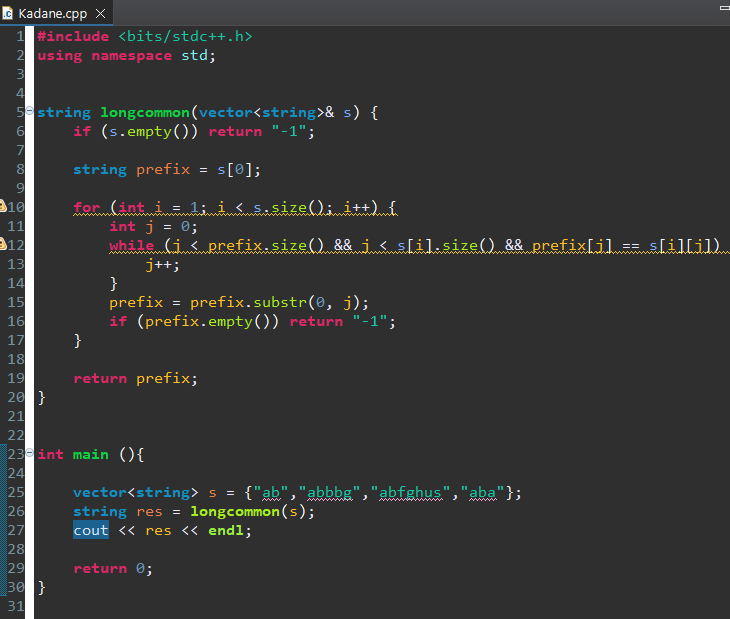
**16) Longest Common Prefix using Sorting**

Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there‟s no prefix common in all the strings, return “-1”

**APPROACH** : Start with the first string as the common prefix and compare it with all string. For each prefix, update the current matching prefix. If at any point, there is no common prefix, return "-1".

Time complexity : O(n\*m)

Space complexity : O(m)

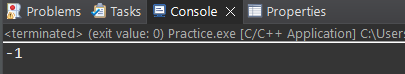


**OUTPUT :**

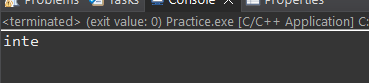
“single”

****

["", ""]

****

["intelligence", "integrate", "integrity"]

****

["in", "iiiin", "iiiiiin"]



**My code’s constraint handling capacity :**

1 <= n <= 10^4

1 <= m <= 10^3

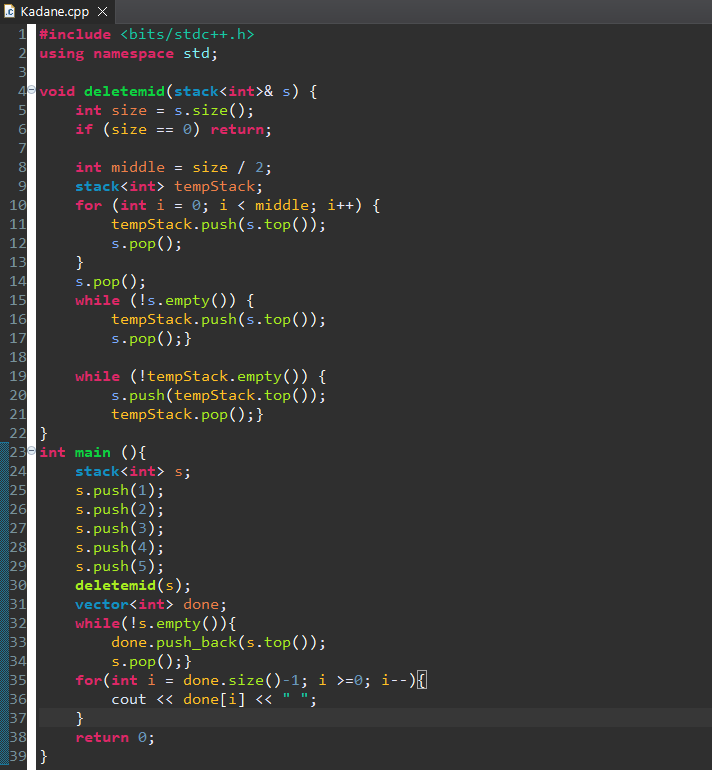
**17) Delete middle element of a stack**

Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

**APPROACH** : Find the size of the array and divide it by two and find the middle index. Pop from the stack until it reaches the mid and then push all the remaining elements into the array.

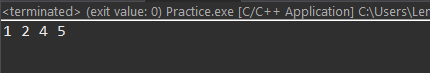
Time complexity : O(N)

Space complexity : O(N)

;

**OUTPUT :**

[1,2,3,4,5]

****

[1,2,3,4,5,6]

****

**My code’s constraint handling capacity :** 1 <= n <= 10^6

**18) Next Greater Element (NGE) for every element in given Array**

Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1

**APPROACH** : Traverse the array from right to left. Pop elements from stack that are smaller than or equal to arr[i] If stack != empty, NGE(arr[i]) = top and then push the current element onto the stack

Time complexity : O(N)

Space complexity : O(N)

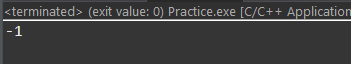


**OUTPUT :**

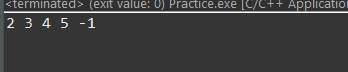
{9, 8, 7, 6, 5, 4, 3, 2, 1}



{42}



{1, 2, 3, 4, 5}



{1, 2, 3, 1, 1, 1, 4}



**My code’s constraint handling capacity :**  1 <= n <= 10^6

**19) . Print Right View of a Binary Tree**

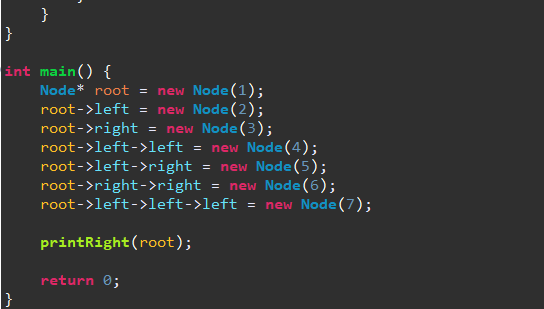
Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level

**APPROACH** : level-order traversal (BFS) queue is used for traversal, and for each level, check if current element is the last element. If yes, print the element. Continue until all levels are processed.

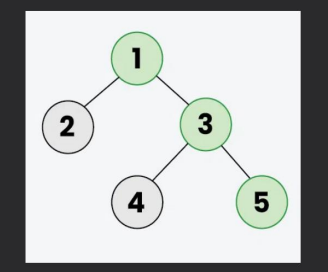
Time complexity : O(N)

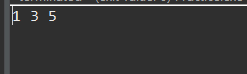
Space complexity : O(N)

****

****

**OUTPUT :**

****

****

**My code’s constraint handling capacity :**

1 <= n <= 10^6

**20. Maximum Depth or Height of Binary Tree**

Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node

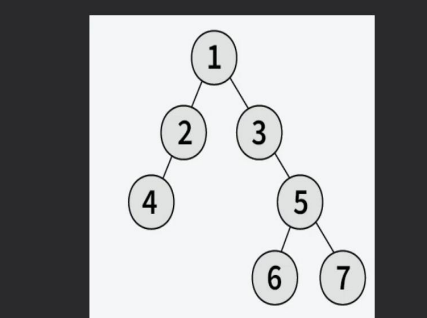
**APPROACH** : DFS

Time complexity : O(N) number of nodes

Space complexity : O(H) height of tree



**OUTPUT :**

****

****

**My code’s constraint handling capacity :**

1 <= n <= 10^6 // For balanced trees.

1 <= n <= 1000 // For highly unbalanced