Assignment 5

A1. Given two integer arrays nums1 and nums2, return *the maximum length of a subarray that appears in* ***both*** *arrays*.

**Example 1:**

**Input:** nums1 = [1,2,3,2,1], nums2 = [3,2,1,4,7]   
Output: 3   
Explanation: The repeated subarray with maximum length is [3,2,1].

**Example 2:**

**Input:** nums1 = [0,0,0,0,0], nums2 = [0,0,0,0,0]   
Output: 5 

Code:

**#include <iostream>**

**using namespace std;**

**int maxSubArray(int A[], int B[], int e, int f)**

**{**

**int c[e + 1][f + 1];**

**for (int i = 0; i <= e; i++)**

**for (int j = 0; j <= f; j++)**

**c[i][j] = 0;**

**for (int i = e - 1; i >= 0; i--) {**

**for (int j = f - 1; j >= 0; j--) {**

**if (A[i] == B[j])**

**c[i][j] = c[i + 1][j + 1] + 1;**

**}**

**}**

**int maxl = 0;**

**for (int i = 0; i < e; i++) {**

**for (int j = 0; j < f; j++) {**

**maxl = max(maxl, c[i][j]);**

**}**

**}**

**return maxl;**

**}**

**int main()**

**{**

**int A[100],B[100],e,f;**

**cout<<"Enter the size of the first array : " <<endl;**

**cin>>e;**

**cout<<"Enter the elements of the first array : "<<endl;**

**for(int i=0;i<e;i++){**

**cin>>A[i];**

**}**

**cout<<"Enter the size of the second array : "<<endl;**

**cin>>f;**

**cout<<"Enter the elements of the subarray : "<<endl;**

**for(int i=0;i<f;i++){**

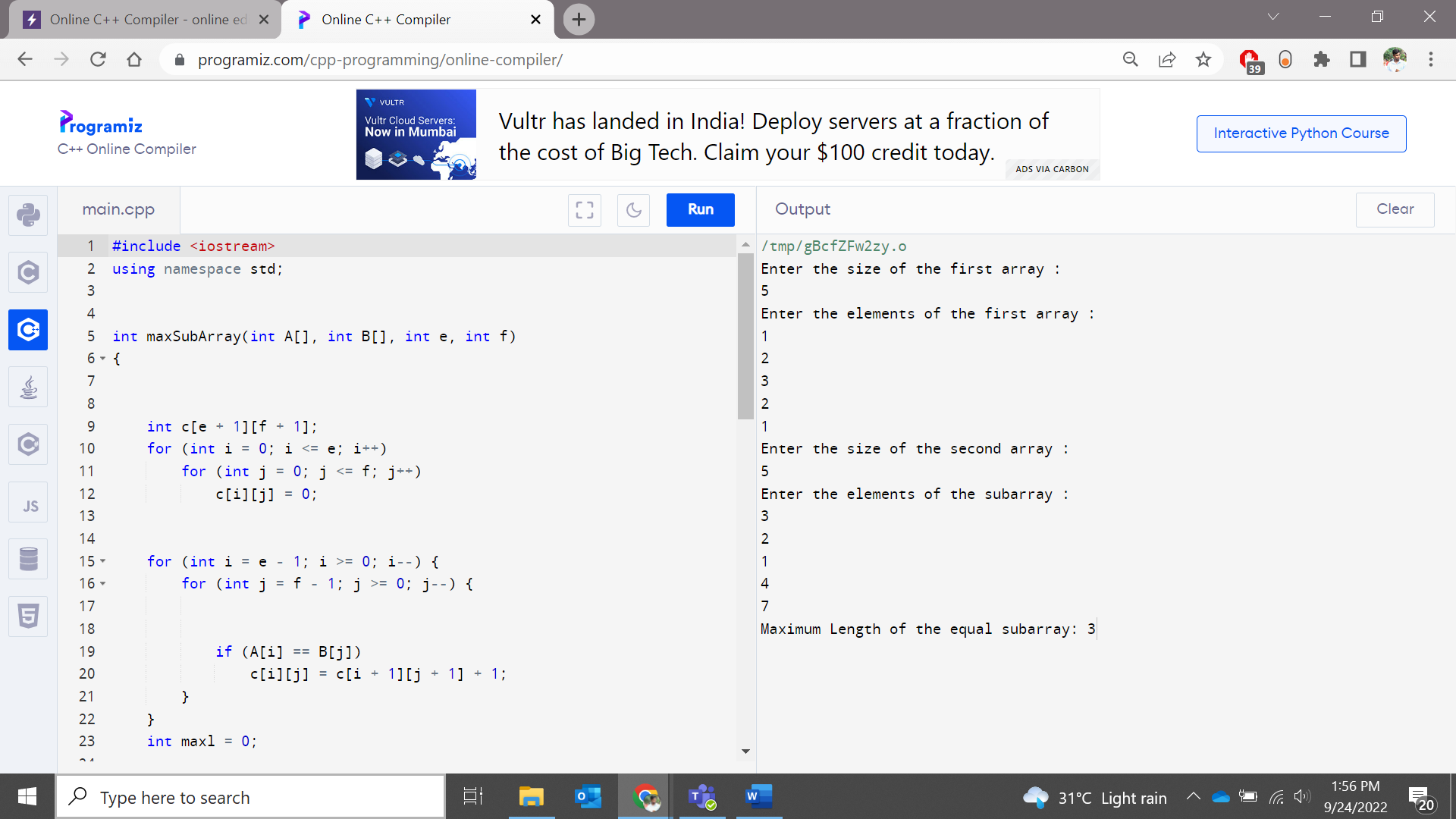
**cin>>B[i];**

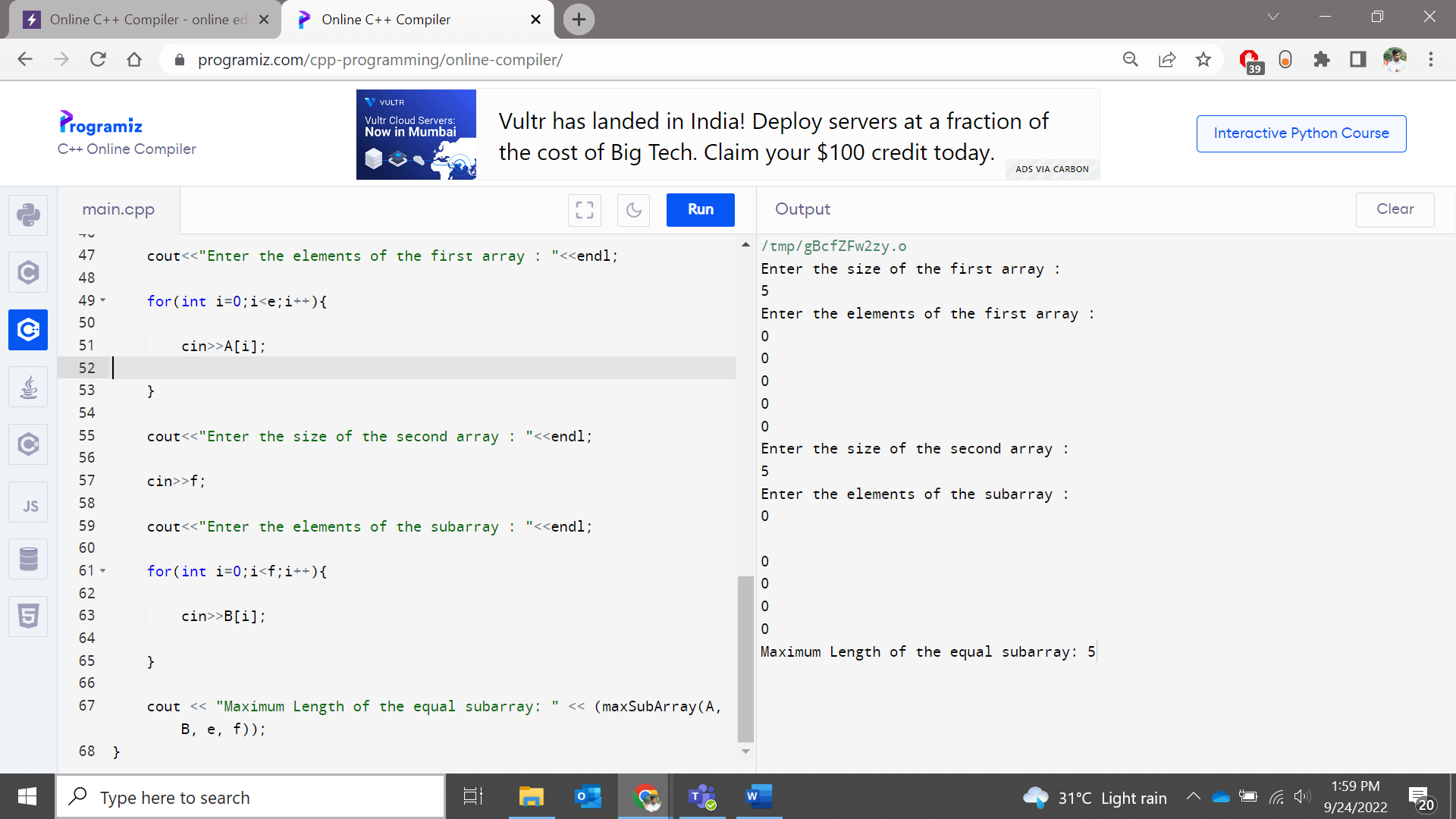
**}**

**cout << "Maximum Length of the equal subarray: " << (maxSubArray(A, B, e, f));**

**}**

Output:





A2. Given an integer array nums sorted in **non-decreasing order**, remove the duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array nums. More formally, if there are k elements after removing the duplicates, then the first k elements of nums should hold the final result. It does not matter what you leave beyond the first k elements.

Return k *after placing the final result in the first* k *slots of* nums.

Do **not** allocate extra space for another array. You must do this by **modifying the input array** [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

**Custom Judge:**

The judge will test your solution with the following code:

int[] nums = [...]; // Input array   
int[] expectedNums = [...]; // The expected answer with correct length   
   
int k = removeDuplicates(nums); // Calls your implementation   
   
assert k == expectedNums.length;   
for (int i = 0; i < k; i++) {   
    assert nums[i] == expectedNums[i];   
} 

If all assertions pass, then your solution will be **accepted**.

**Example 1:**

**Input:** nums = [1,1,2]   
Output: 2, nums = [1,2,\_]   
Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.   
It does not matter what you leave beyond the returned k (hence they are underscores). 

**Example 2:**

**Input:** nums = [0,0,1,1,1,2,2,3,3,4]   
Output: 5, num = [0,1,2,3,4,\_,\_,\_,\_,\_]   
Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.   
It does not matter what you leave beyond the returned k (hence they are underscores). 

**Constraints:**

* 1 <= nums.length <= 3 \* 104
* -100 <= nums[i] <= 100
* nums is sorted in **non-decreasing** order.

**Code :**

**#include<iostream>**

**using namespace std;**

**int removeDup(int arr[], int n)**

**{**

**if (n==0 || n==1)**

**return n;**

**int j = 0;**

**for (int i=0; i < n-1; i++)**

**if (arr[i] != arr[i+1])**

**arr[j++] = arr[i];**

**arr[j++] = arr[n-1];**

**return j;**

**}**

**int main()**

**{**

**int n;**

**cout << "Enter the size of the array: " << endl;**

**cin >> n;**

**int arr[n];**

**cout << "Enter the elements of the array: " << endl;**

**for (int i=0;i<n;i++){**

**cin >> arr[i];**

**}**

**n = removeDup(arr, n);**

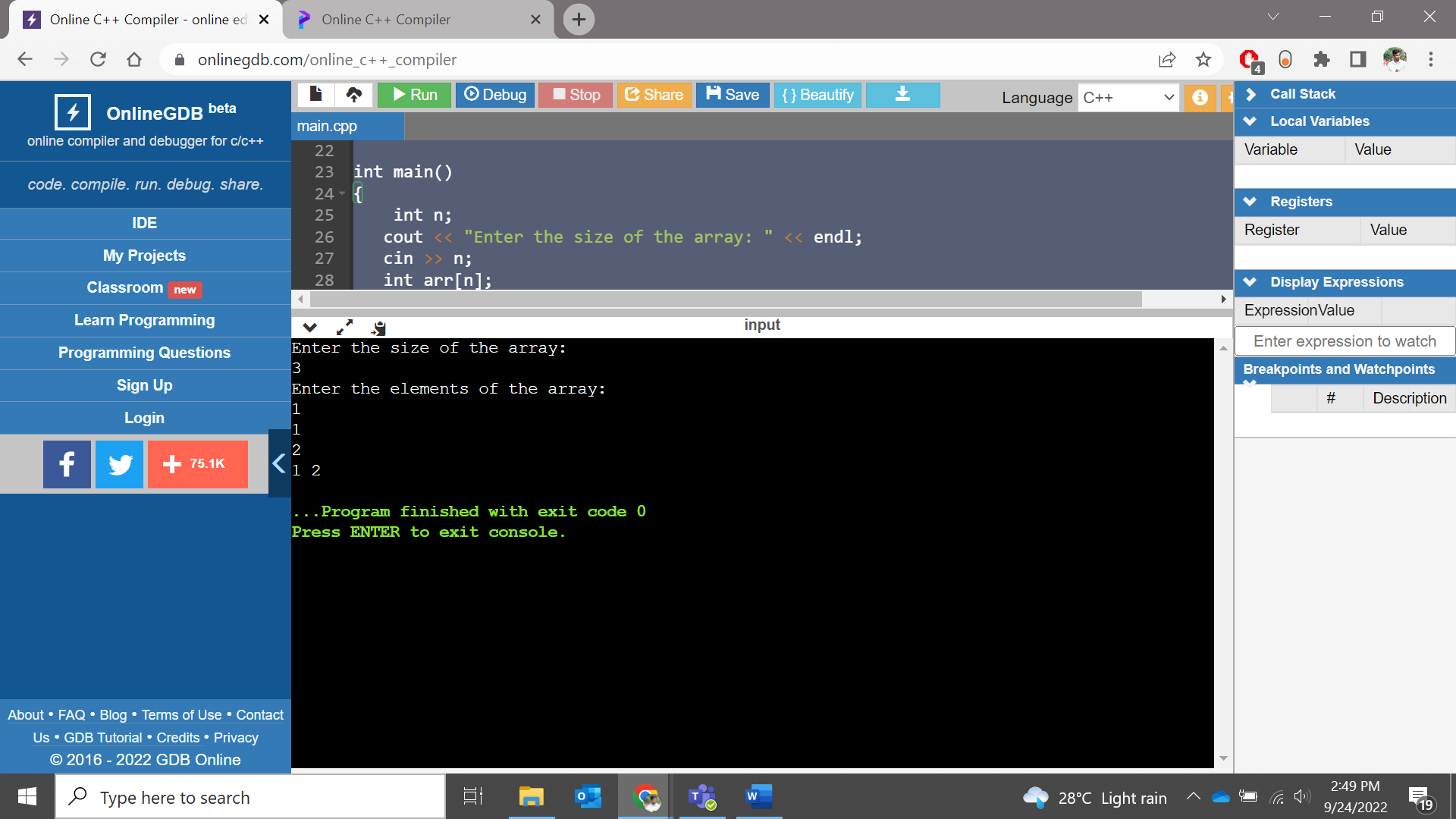
**for (int i=0; i<n; i++)**

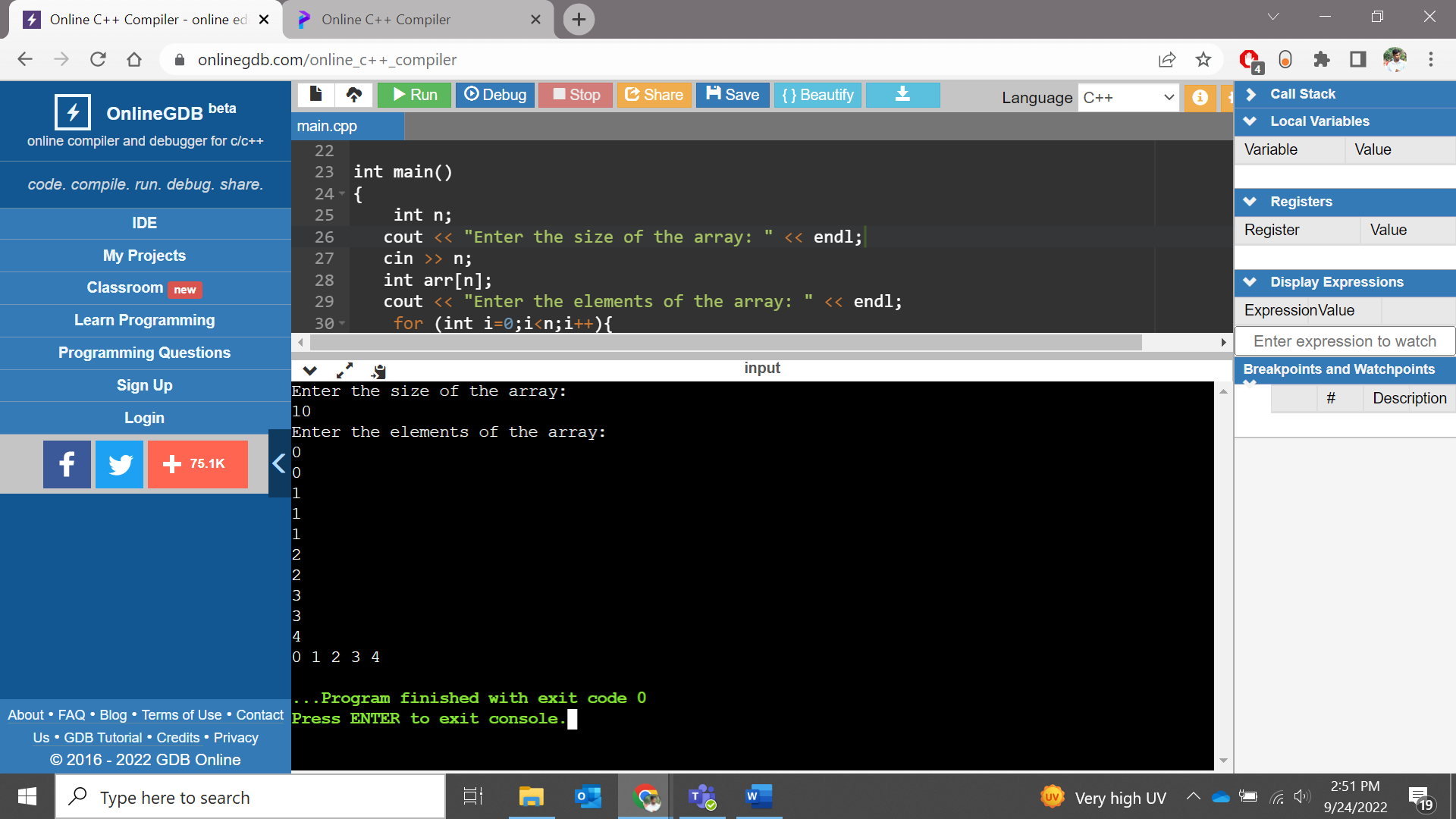
**cout << arr[i] << " ";**

**return 0;**

**}**

**Output:**





**A3**. Given a string s and a character c that occurs in s, return *an array of integers* answer *where* answer.length == s.length *and* answer[i] *is the* ***distance*** *from index* i *to the* ***closest*** *occurrence of character* c *in* s.

The **distance** between two indices i and j is abs(i - j), where abs is the absolute value function.

**Example 1:**

**Input:** s = "loveleetcode", c = "e"   
Output: [3,2,1,0,1,0,0,1,2,2,1,0]   
Explanation: The character 'e' appears at indices 3, 5, 6, and 11 (0-indexed).   
The closest occurrence of 'e' for index 0 is at index 3, so the distance is abs(0 - 3) = 3.   
The closest occurrence of 'e' for index 1 is at index 3, so the distance is abs(1 - 3) = 2.   
For index 4, there is a tie between the 'e' at index 3 and the 'e' at index 5, but the distance is still the same: abs(4 - 3) == abs(4 - 5) = 1.   
The closest occurrence of 'e' for index 8 is at index 6, so the distance is abs(8 - 6) = 2. 

**Example 2:**

**Input:** s = "aaab", c = "b"   
Output: [3,2,1,0] 

**Constraints:**

* 1 <= s.length <= 104
* s[i] and c are lowercase English letters.
* It is guaranteed that c occurs at least once in s.

**Code:**

**#include <string>**

**#include <iostream>**

**#include <bits/stdc++.h>**

**using namespace std;**

**void shortestD(string S, char C)**

**{**

**int prev = INT\_MAX;**

**vector<int> ans;**

**for (int i = 0; i < S.length(); i++)**

**{**

**if (S[i] == C)**

**prev = i;**

**if (prev == INT\_MAX)**

**ans.push\_back(INT\_MAX);**

**else**

**ans.push\_back(i - prev);**

**}**

**prev = INT\_MAX;**

**for (int i = S.length() - 1; i >= 0; i--)**

**{**

**if (S[i] == C)**

**prev = i;**

**if (prev != INT\_MAX)**

**ans[i] = min(ans[i], prev - i);**

**}**

**for (auto val: ans)**

**cout << val << ' ';**

**}**

**int main()**

**{**

**string S = "loveleetcode";**

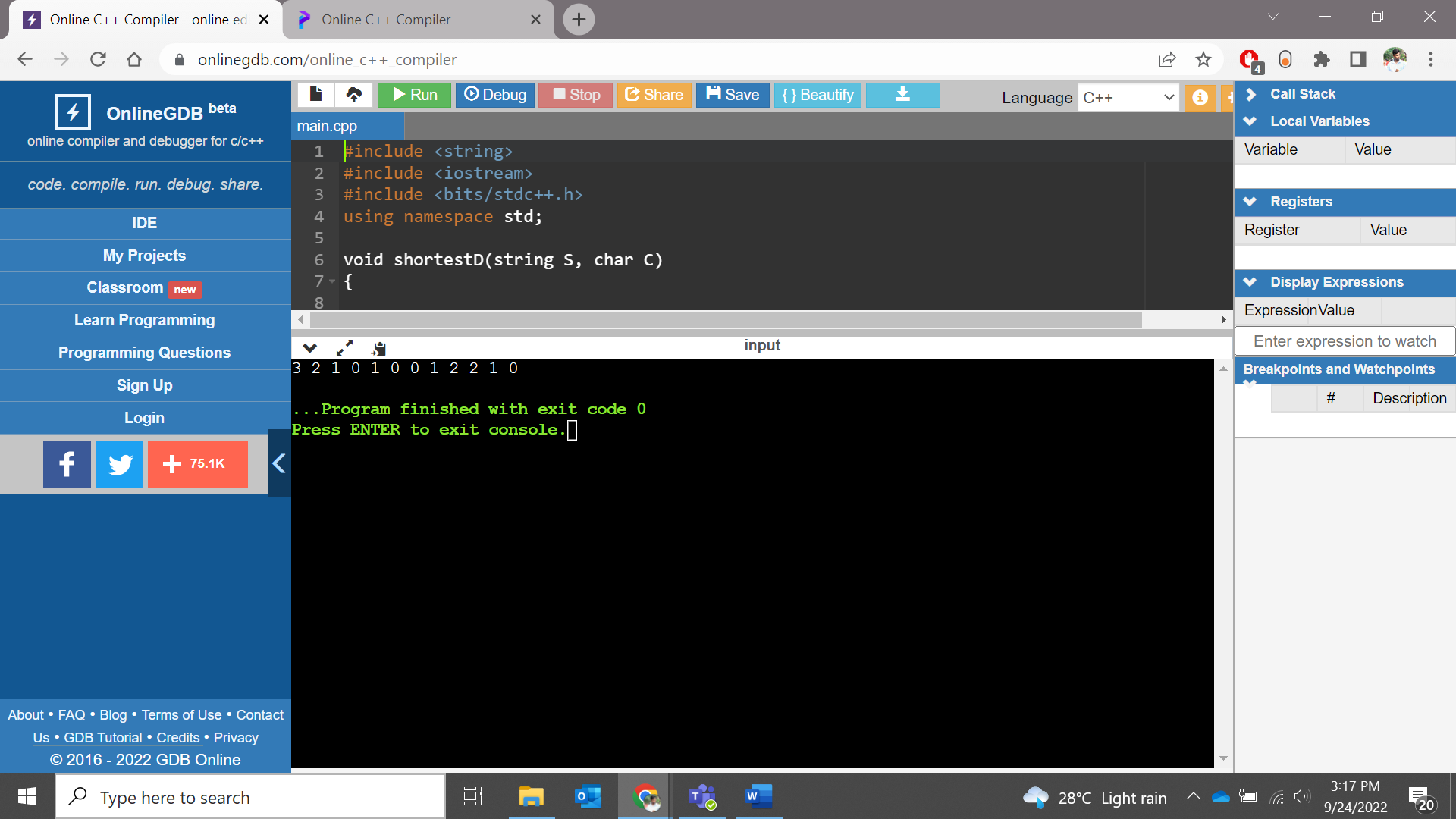
**char C = 'e';**

**shortestD(S, C);**

**return 0;**

**}**

**Output:**



Graphical user interface

Description automatically generated

A4. Calculate total payment of employees using structure and function (10 employees, employee details like id,name,and salary in the structure and function for adding the salary)

**Code:**

**#include <iostream>**

**using namespace std;**

**struct Employee {**

**char name[50];**

**int salary;**

**int employeeCode;**

**char dept[5];**

**}e[10];**

**int add(struct Employee \*e,int n){**

**int total\_salary=0;**

**for(int i=0; i<n;i++)**

**{**

**total\_salary+=e[i].salary;**

**}**

**return total\_salary;**

**}**

**int main() {**

**int n;**

**cout << "Enter the number of employees: " << endl;**

**cin >> n;**

**for(int i=0;i<n;i++){**

**cout << "Enter name of employee : ";**

**cin >> e[i].name;**

**cout << "Enter department : ";**

**cin >> e[i].dept;**

**cout << "Enter salary of employee : ";**

**cin >> e[i].salary;**

**cout << "Enter employee code : ";**

**cin >> e[i].employeeCode;**

**cout <<" " <<endl;**

**}**

**cout << "\n\*\*\* Employee Details \*\*\*" << endl;**

**for(int i=0;i<n;i++){**

**cout << "Name : " << e[i].name << endl << "Salary : " << e[i].salary << endl;**

**cout << "Employee Code : " << e[i].employeeCode << endl << "Department : " << e[i].dept << endl;**

**}**

**int sum=add(e,n);**

**cout << "Total Salary: " << sum <<endl;**

**return 0;**

**}**

**Output:**

