**Majority element:**

#include <stdio.h>

int findMajorityElement(int arr[], int size) {

int candidate = arr[0];

int count = 1;

for (int i = 1; i < size; i++) {

if (arr[i] == candidate) {

count++;

} else {

count--;

if (count == 0) {

candidate = arr[i];

count = 1;

}

}

}

count = 0;

for (int i = 0; i < size; i++) {

if (arr[i] == candidate) {

count++;

}

}

if (count > size / 2) {

return candidate;

} else {

return -1;

}

}

int main() {

int arr[] = {3, 3, 4, 2, 4, 4, 2, 4, 4};

int size = sizeof(arr) / sizeof(arr[0]);

int majority = findMajorityElement(arr, size);

if (majority != -1) {

printf("Majority element is: %d\n", majority);

} else {

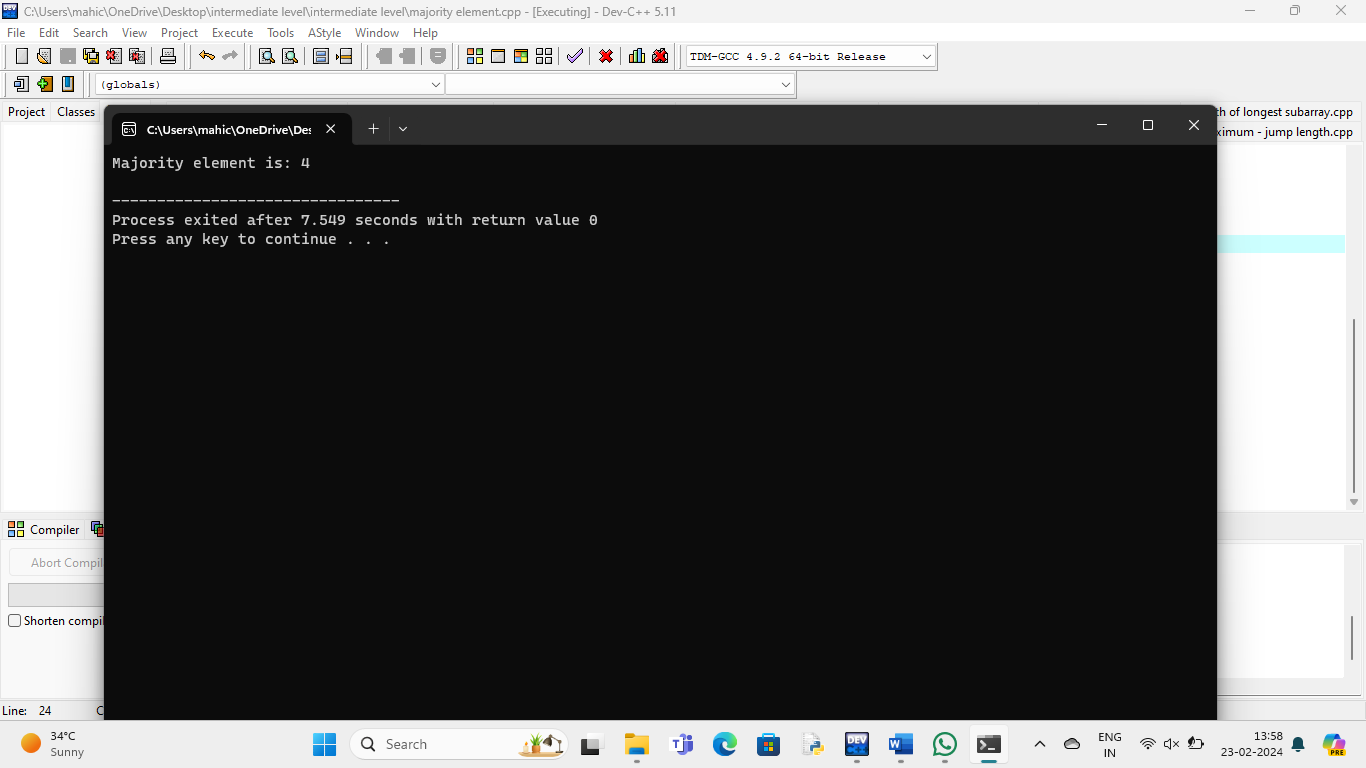
printf("No majority element found\n");

}

return 0;

}

**Output:**



**Rearrange array:**

#include <stdio.h>

void rearrange(int arr[], int n) {

int left = 0, right = n - 1;

while (left < right) {

while (arr[left] % 2 == 0 && left < right)

left++;

while (arr[right] % 2 != 0 && left < right)

right--;

if (left < right) {

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

}

int main() {

int arr[] = {12, 34, 45, 9, 8, 90, 3};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Original array: ");

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

printf("%d ", arr[i]);

}

printf("\n");

rearrange(arr, n);

printf("Array after rearrangement: ");

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

printf("%d ", arr[i]);

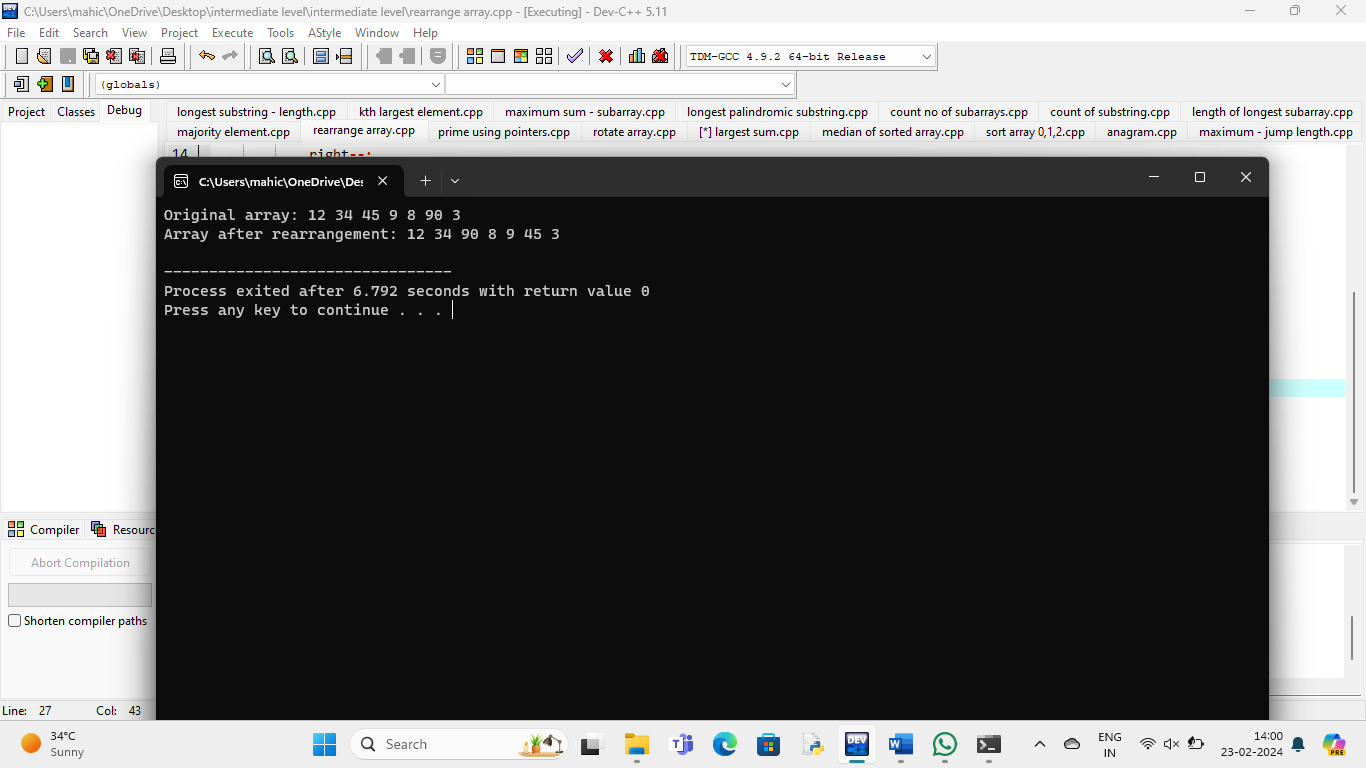
}

printf("\n");

return 0;

}

**Output:**



**Check prime using pointers:**

#include <stdio.h>

int isPrime(int \*num) {

if (\*num <= 1) {

return 0;

}

for (int i = 2; i <= \*num / 2; ++i) {

if (\*num % i == 0) {

return 0;

}

}

return 1;

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if (isPrime(&num)) {

printf("%d is a prime number.\n", num);

} else {

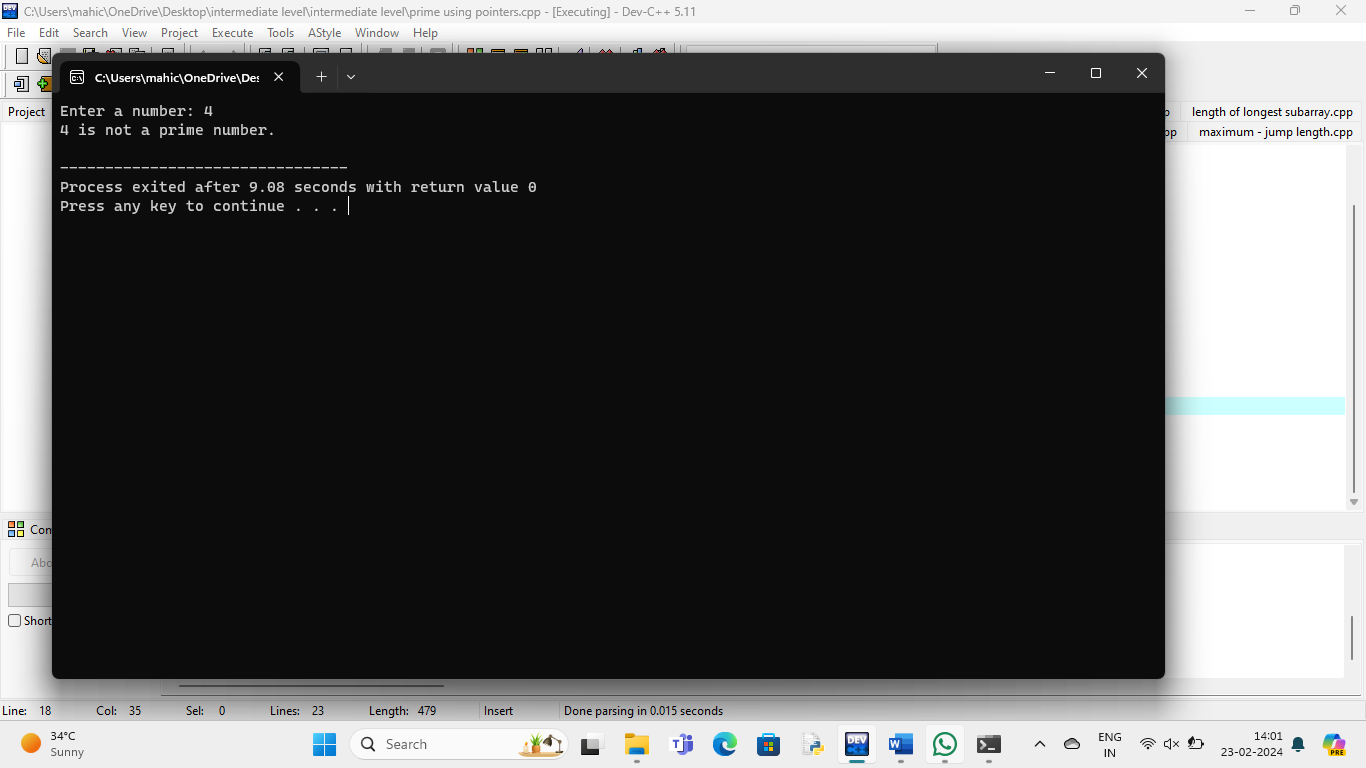
printf("%d is not a prime number.\n", num);

}

return 0;

}

**Output:**



**Rotate array to right :**

#include <stdio.h>

void rotateArray(int arr[], int n, int k) {

int temp[k];

for (int i = 0; i < k; i++)

temp[i] = arr[n - k + i];

}

for (int i = n - 1; i >= k; i--) {

arr[i] = arr[i - k];

}

for (int i = 0; i < k; i++) {

arr[i] = temp[i];

}

}

int main() {

int arr[] = {1, 2, 3, 4, 5};

int n = sizeof(arr) / sizeof(arr[0]);

int k = 2;

printf("Original array: ");

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

printf("%d ", arr[i]);

}

rotateArray(arr, n, k);

printf("\nArray after rotating by %d positions: ", k);

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

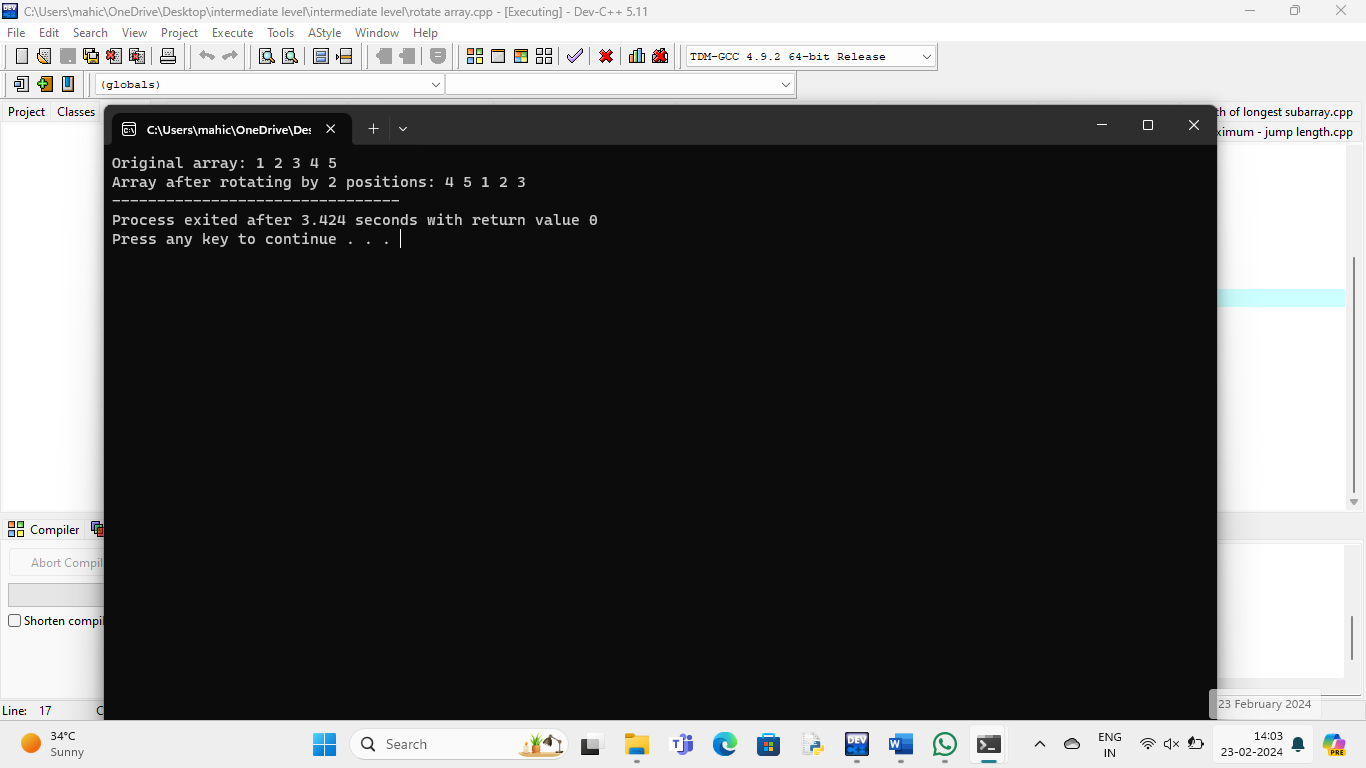
printf("%d ", arr[i]);

}

return 0;

}

**Output:**



**Largest sum in subarray:**

#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int maxSubArraySum(int arr[], int n) {

int maxEndingHere = arr[0];

int maxSoFar = arr[0];

for (int i = 1; i < n; i++) {

maxEndingHere = max(arr[i], maxEndingHere + arr[i]);

maxSoFar = max(maxSoFar, maxEndingHere);

}

return maxSoFar;

}

int main() {

int arr[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4};

int n = sizeof(arr) / sizeof(arr[0]);

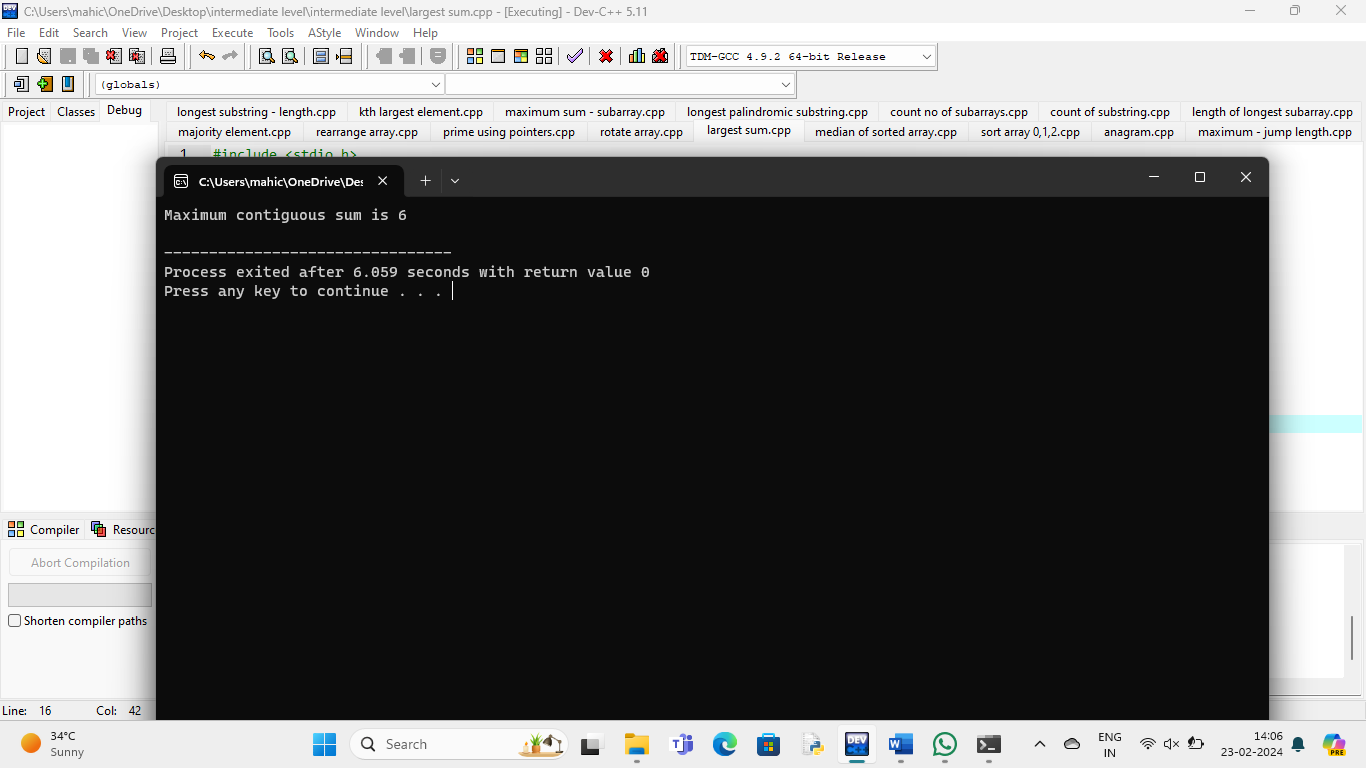
int maxSum = maxSubArraySum(arr, n);

printf("Maximum contiguous sum is %d\n", maxSum);

return 0;

}

**Output:**



**Median of sorted array:**

#include <stdio.h>

double findMedianSortedArrays(int arr1[], int arr2[], int n) {

int mergedArray[2 \* n];

int i = 0, j = 0, k = 0;

while (i < n && j < n) {

if (arr1[i] < arr2[j])

mergedArray[k++] = arr1[i++];

else

mergedArray[k++] = arr2[j++];

}

while (i < n)

mergedArray[k++] = arr1[i++];

while (j < n)

mergedArray[k++] = arr2[j++];

return (mergedArray[n - 1] + mergedArray[n]) / 2.0;

}

int main() {

int arr1[] = {1, 3, 5};

int arr2[] = {2, 4, 6};

int n = sizeof(arr1) / sizeof(arr1[0]);

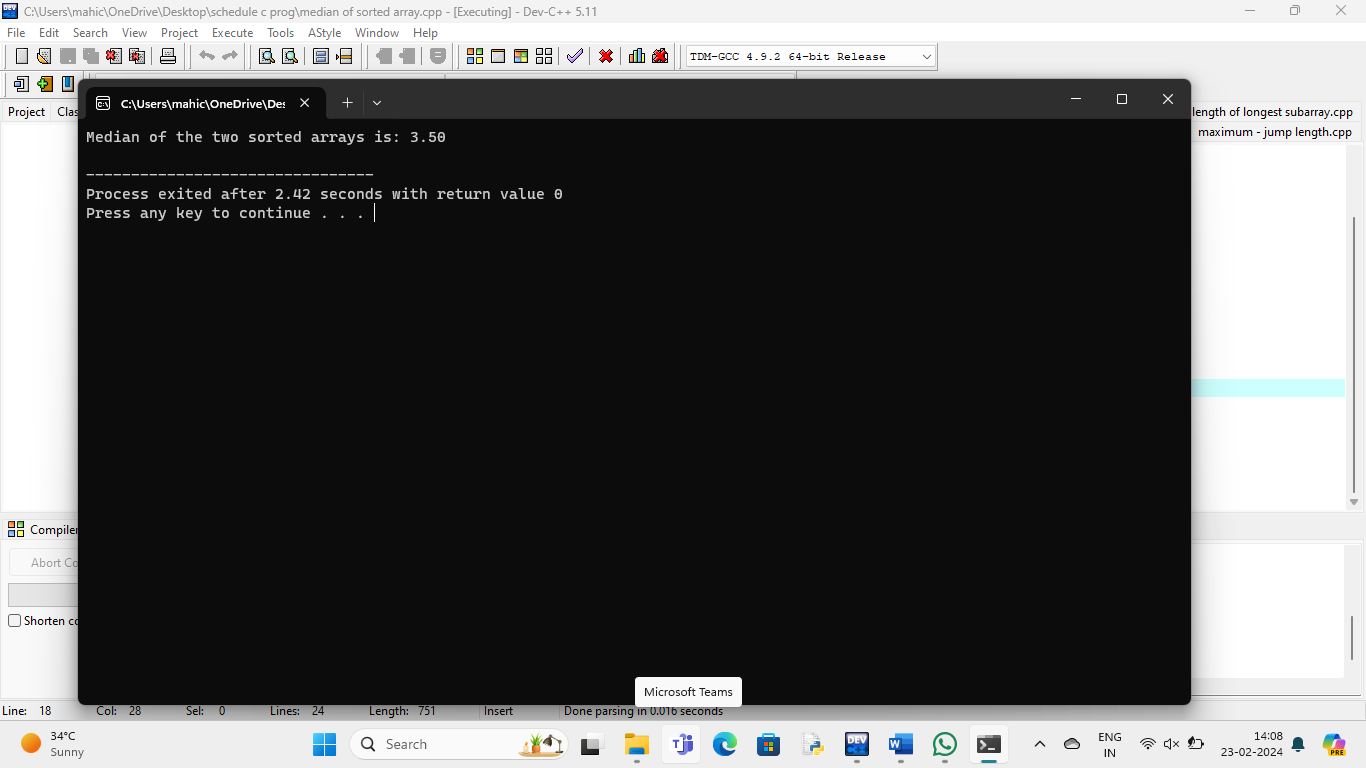
double median = findMedianSortedArrays(arr1, arr2, n);

printf("Median of the two sorted arrays is: %.2lf\n", median);

return 0;

}

**Output:**



**Sort an array of 0s, 1s and 2s:**

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void sortColors(int\* nums, int numsSize) {

int low = 0, mid = 0, high = numsSize - 1;

while (mid <= high) {

switch (nums[mid]) {

case 0:

swap(&nums[low], &nums[mid]);

low++;

mid++;

break;

case 1:

mid++;

break;

case 2:

swap(&nums[mid], &nums[high]);

high--;

break;

}

}

}

int main() {

int nums[] = {2, 0, 1, 2, 1, 0};

int numsSize = sizeof(nums) / sizeof(nums[0]);

printf("Original Array: ");

for (int i = 0; i < numsSize; i++) {

printf("%d ", nums[i]);

}

printf("\n");

sortColors(nums, numsSize);

printf("Sorted Array: ");

for (int i = 0; i < numsSize; i++) {

printf("%d ", nums[i]);

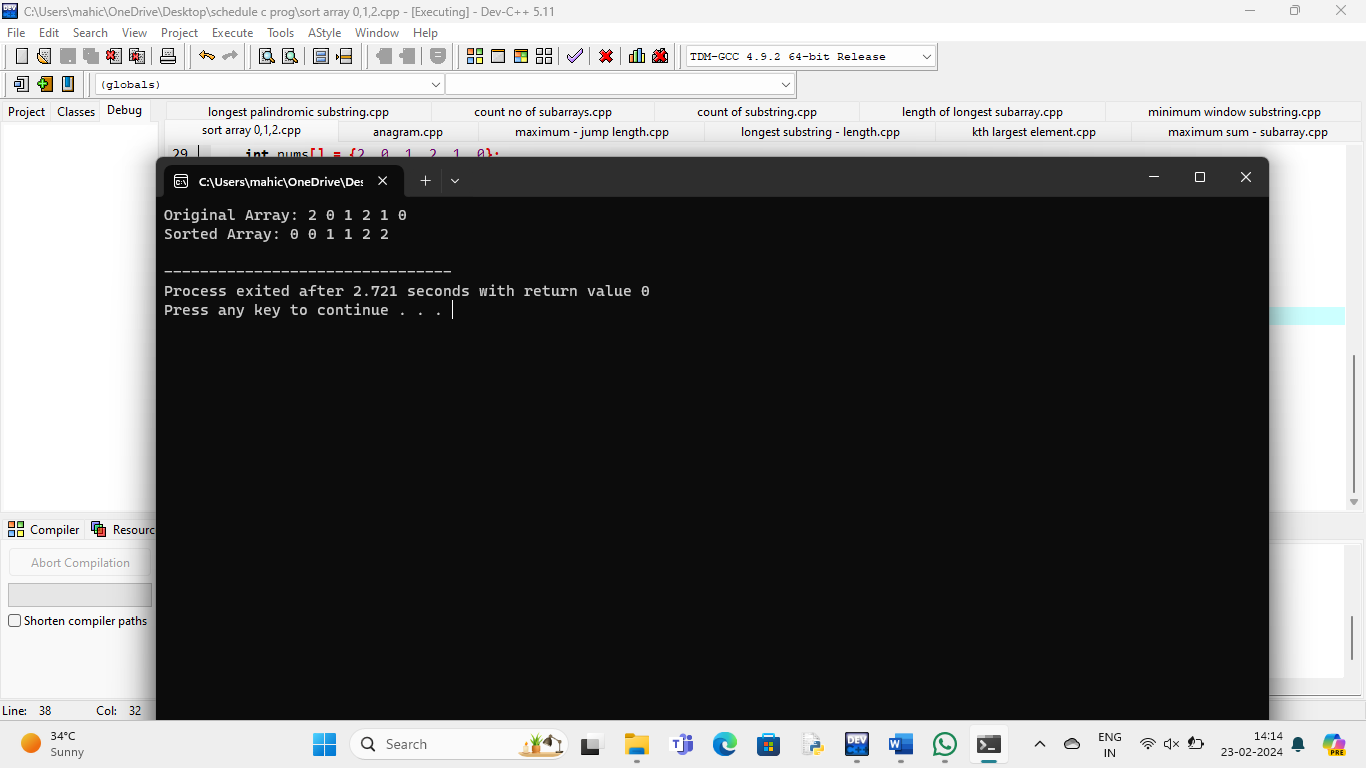
}

printf("\n");

return 0;

}

**Output:**



**Anagram:**

#include <stdio.h>

#include <string.h>

#define MAX\_CHARS 256

int areAnagrams(char \*str1, char \*str2) {

int count1[MAX\_CHARS] = {0};

int count2[MAX\_CHARS] = {0};

int i;

for (i = 0; str1[i] != '\0'; i++)

count1[str1[i]]++;

for (i = 0; str2[i] != '\0'; i++)

count2[str2[i]]++;

for (i = 0; i < MAX\_CHARS; i++) {

if (count1[i] != count2[i])

return 0;

}

return 1;

}

int main() {

char str1[] = "listen";

char str2[] = "silent";

if (areAnagrams(str1, str2))

printf("%s and %s are anagrams.\n", str1, str2);

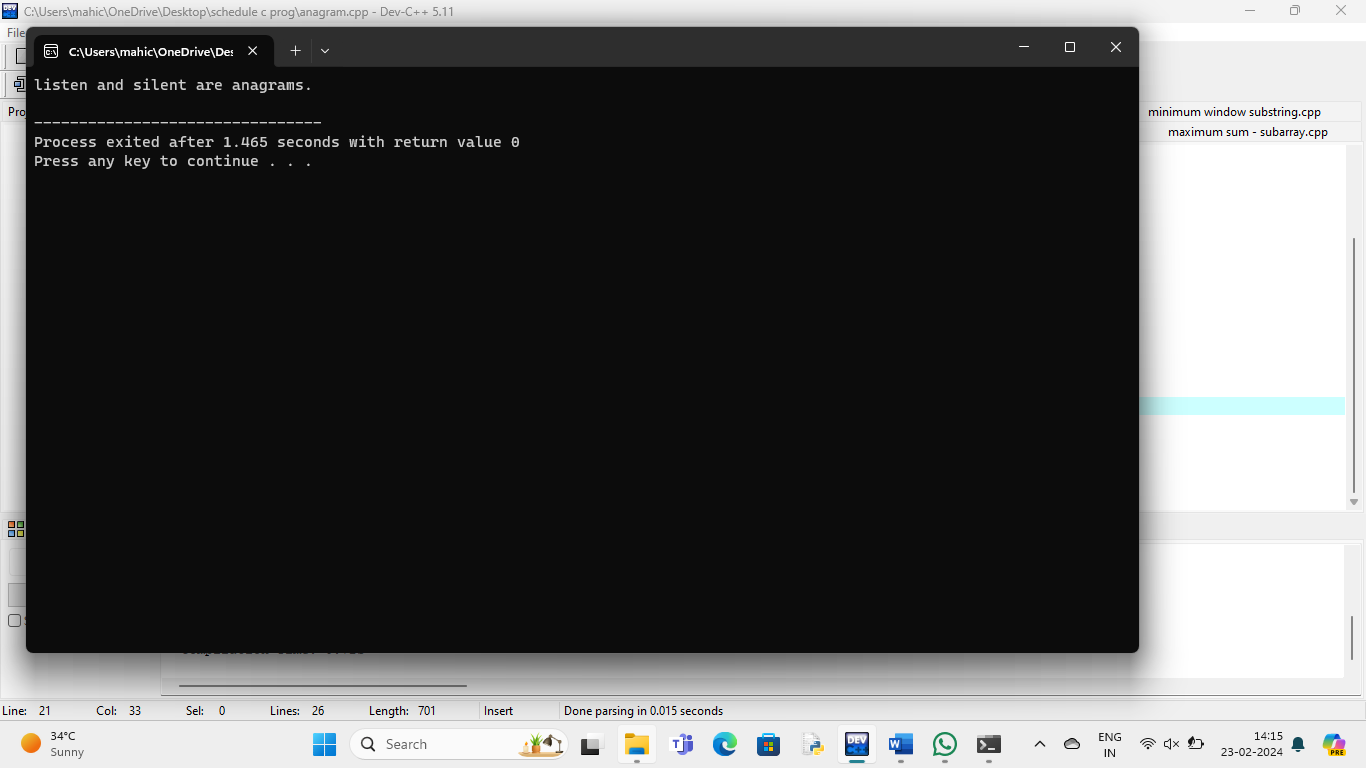
else

printf("%s and %s are not anagrams.\n", str1, str2);

return 0;

}

**Output:**



**Maximum array – jump length:**

#include <stdio.h>

int minJumps(int arr[], int n) {

if (n <= 1)

return 0;

if (arr[0] == 0)

return -1;

int maxReach = arr[0];

int steps = arr[0];

int jumps = 1;

for (int i = 1; i < n; i++) {

if (i == n - 1)

return jumps;

maxReach = (maxReach > i + arr[i]) ? maxReach : i + arr[i];

steps--;

if (steps == 0) {

jumps++;

if (i >= maxReach)

return -1;

steps = maxReach - i;

}

}

return -1;

}

int main() {

int arr[] = {2, 3, 1, 1, 4};

int n = sizeof(arr) / sizeof(arr[0]);

int minJumpsNeeded = minJumps(arr, n);

if (minJumpsNeeded != -1)

printf("Minimum number of jumps needed: %d\n", minJumpsNeeded);

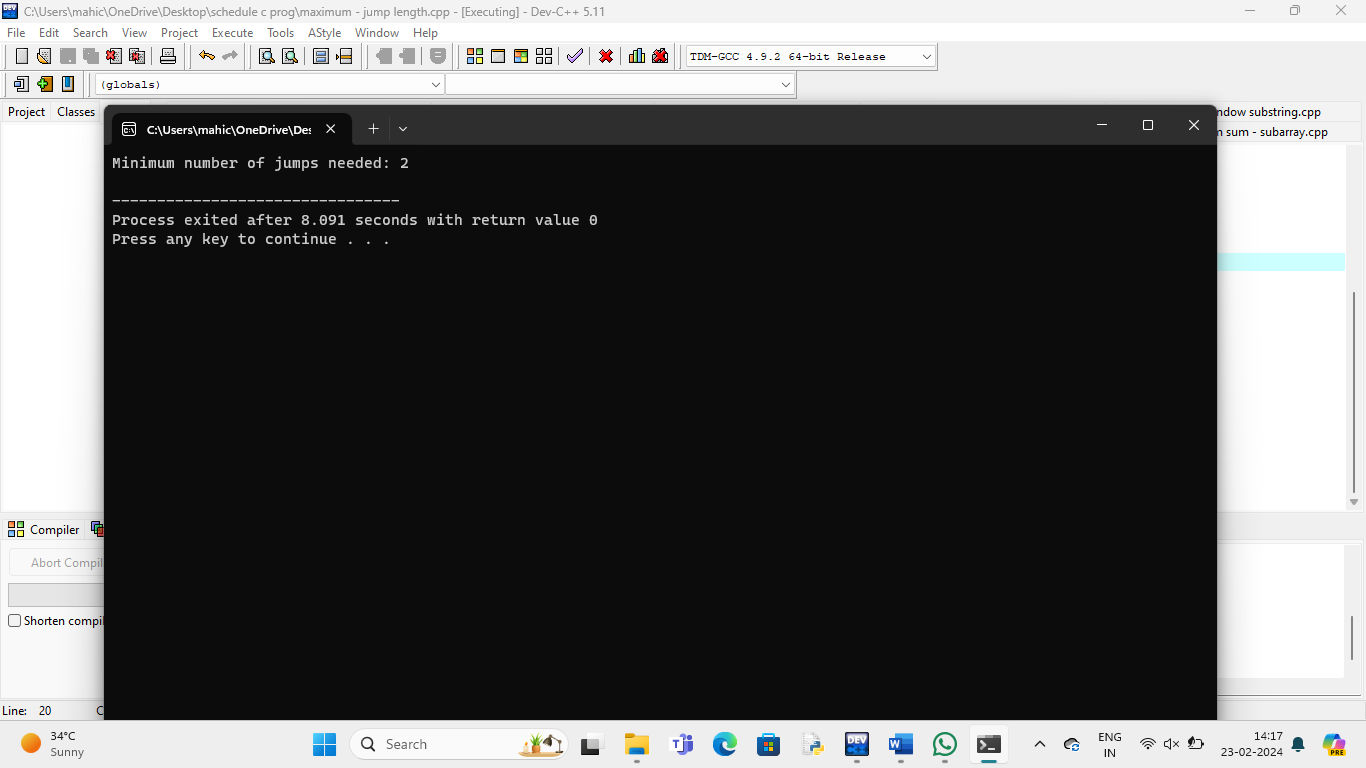
else

printf("It's not possible to reach the end of the array.\n");

return 0;

}

**Output:**



**Longest substring – length:**

#include <stdio.h>

#include <string.h>

#define MAX\_CHARS 256

int max(int a, int b) {

return (a > b) ? a : b;

}

int longestUniqueSubsttr(char \*str) {

int n = strlen(str);

int visited[MAX\_CHARS];

int currentLength = 1;

int maxLength = 1;

int previousIndex;

for (int i = 0; i < MAX\_CHARS; i++)

visited[i] = -1;

visited[str[0]] = 0;

for (int i = 1; i < n; i++) {

previousIndex = visited[str[i]];

if (previousIndex == -1 || i - currentLength > previousIndex)

currentLength++;

else {

maxLength = max(maxLength, currentLength);

currentLength = i - previousIndex;

}

visited[str[i]] = i;

}

maxLength = max(maxLength, currentLength);

return maxLength;

}

int main() {

char str[] = "abcabcbb";

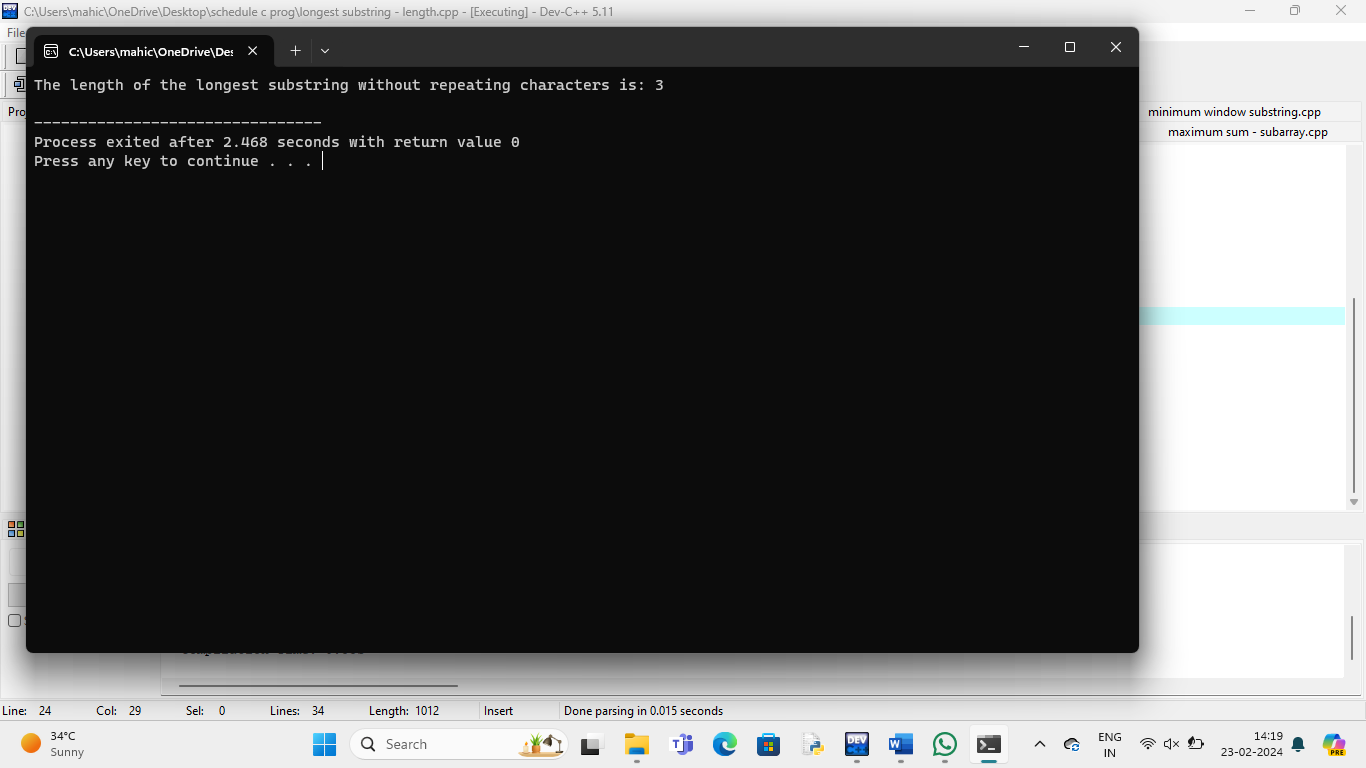
int length = longestUniqueSubsttr(str);

printf("The length of the longest substring without repeating characters is: %d\n", length);

return 0;

}

**Output:**



**Kth largest element:**

#include <stdio.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = low - 1;

int j;

for (j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int findKthLargest(int arr[], int size, int k) {

quickSort(arr, 0, size - 1);

return arr[size - k];

}

int main() {

int arr[] = {3, 1, 4, 2, 5};

int size = sizeof(arr) / sizeof(arr[0]);

int k = 3;

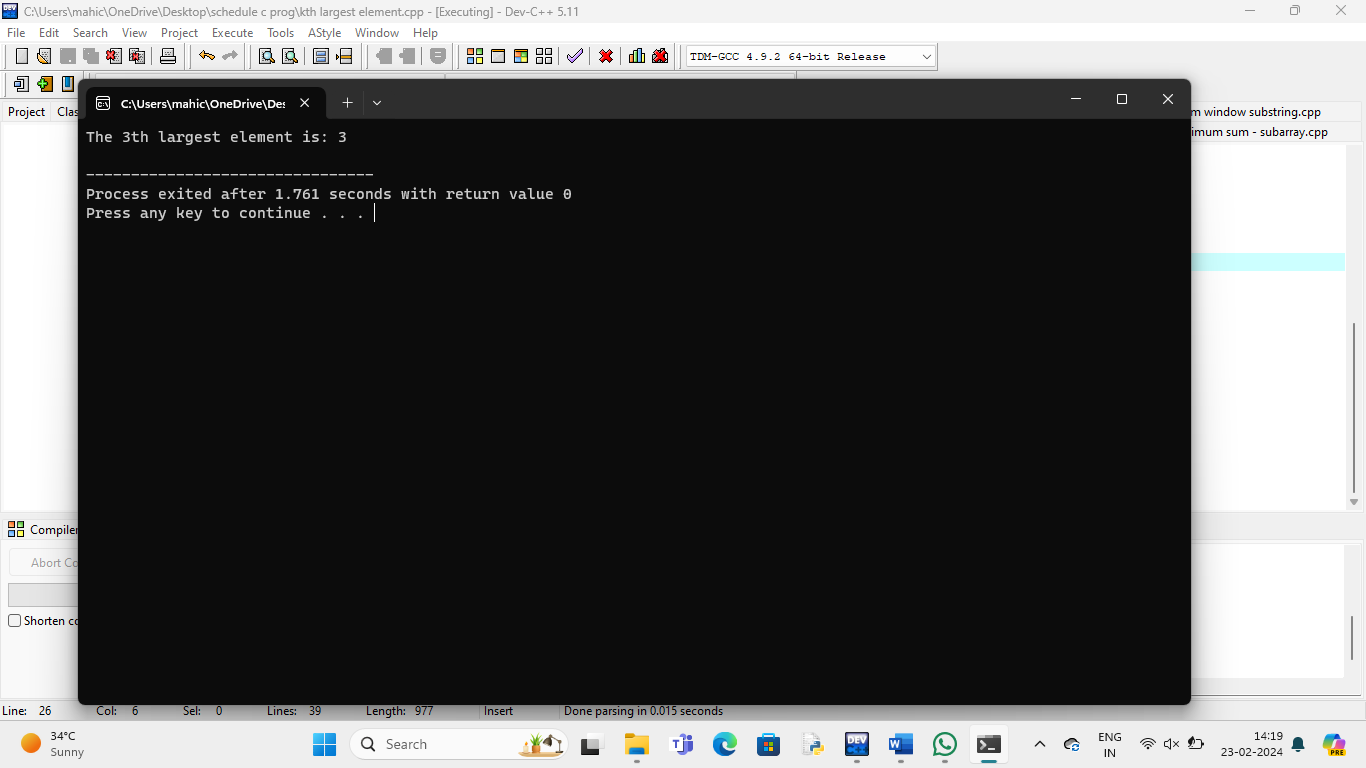
int kthLargest = findKthLargest(arr, size, k);

printf("The %dth largest element is: %d\n", k, kthLargest);

return 0;

}

**Output:**



**Maximum sum – subarray:**

#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int min(int a, int b) {

return (a < b) ? a : b;

}

int kadane(int arr[], int n) {

int maxEndingHere = arr[0];

int maxSoFar = arr[0];

for (int i = 1; i < n; i++) {

maxEndingHere = max(arr[i], maxEndingHere + arr[i]);

maxSoFar = max(maxSoFar, maxEndingHere);

}

return maxSoFar;

}

int maxCircularSum(int arr[], int n) {

int maxKadane = kadane(arr, n);

if (maxKadane < 0) {

return maxKadane;

}

int totalSum = 0;

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

totalSum += arr[i];

arr[i] = -arr[i];

}

int maxCircularSum = totalSum + kadane(arr, n);

return max(maxKadane, maxCircularSum);

}

int main() {

int n;

printf("Enter the size of the circular array: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements of the circular array: ");

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

scanf("%d", &arr[i]);

}

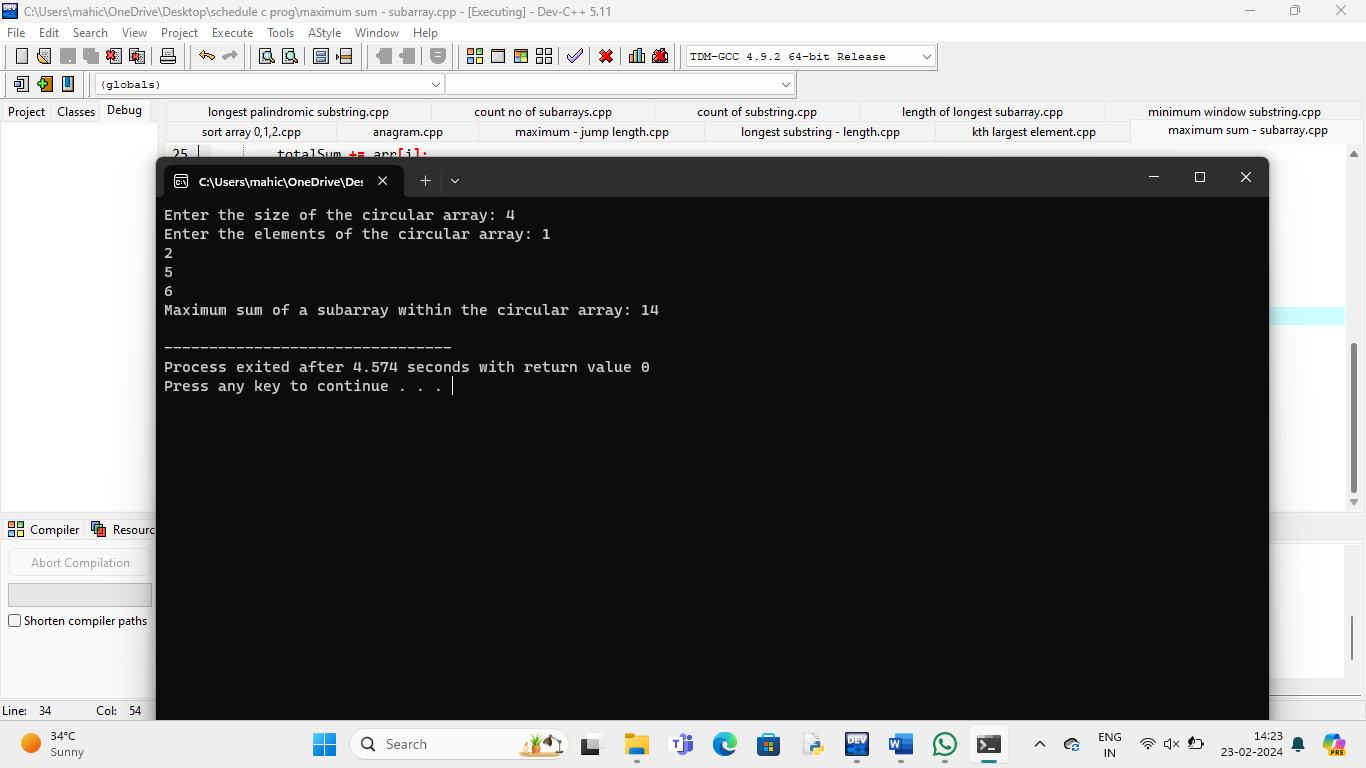
int result = maxCircularSum(arr, n);

printf("Maximum sum of a subarray within the circular array: %d\n", result);

return 0;

}

**Output:**



**Longest palindromic substring:**

#include <stdio.h>

#include <string.h>

int isPalindrome(char \*str, int start, int end) {

while (start < end) {

if (str[start] != str[end])

return 0;

start++;

end--;

}

return 1;

}

void longestPalindromicSubstring(char \*str) {

int n = strlen(str);

int maxLength = 1;

int start = 0;

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

for (int j = 0; i - j >= 0 && i + j < n; j++) {

if (str[i - j] != str[i + j])

break;

int length = 2 \* j + 1;

if (length > maxLength) {

maxLength = length;

start = i - j;

}

}

for (int j = 0; i - j >= 0 && i + 1 + j < n; j++) {

if (str[i - j] != str[i + 1 + j])

break;

int length = 2 \* j + 2;

if (length > maxLength) {

maxLength = length;

start = i - j;

}

}

}

printf("Longest palindromic substring is: ");

for (int i = start; i < start + maxLength; i++)

printf("%c", str[i]);

printf("\n");

}

int main() {

char str[100];

printf("Enter a string: ");

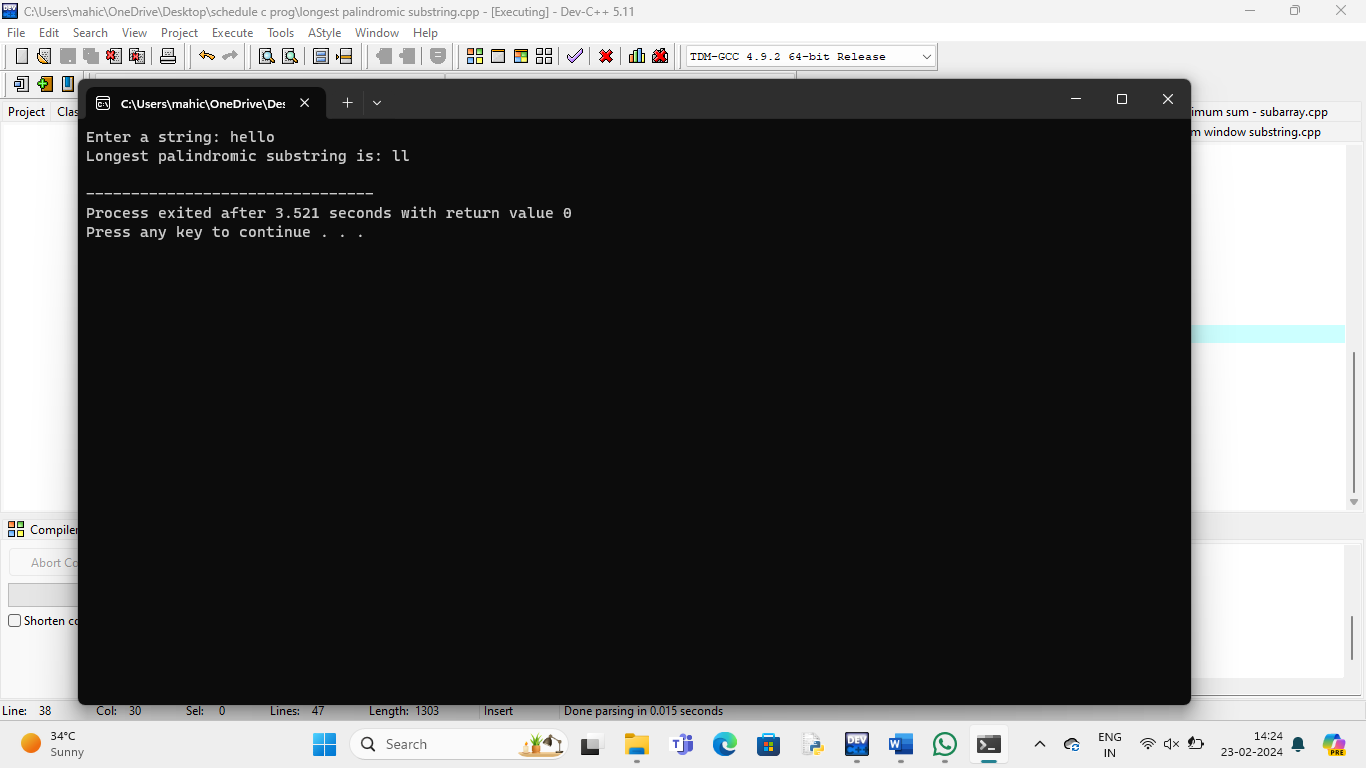
scanf("%s", str);

longestPalindromicSubstring(str);

return 0;

}

**Output:**



**Count no of subarrays:**

#include <stdio.h>

int countSubarraysWithSumLessThan(int arr[], int n, int target) {

int count = 0;

int sum = 0;

int start = 0;

for (int end = 0; end < n; end++) {

sum += arr[end];

while (sum >= target && start <= end) {

sum -= arr[start];

start++;

}

count += end - start + 1;

}

count -= n;

return count;

}

int main() {

int arr[] = {1, 4, 2, 3};

int n = sizeof(arr) / sizeof(arr[0]);

int target = 6;

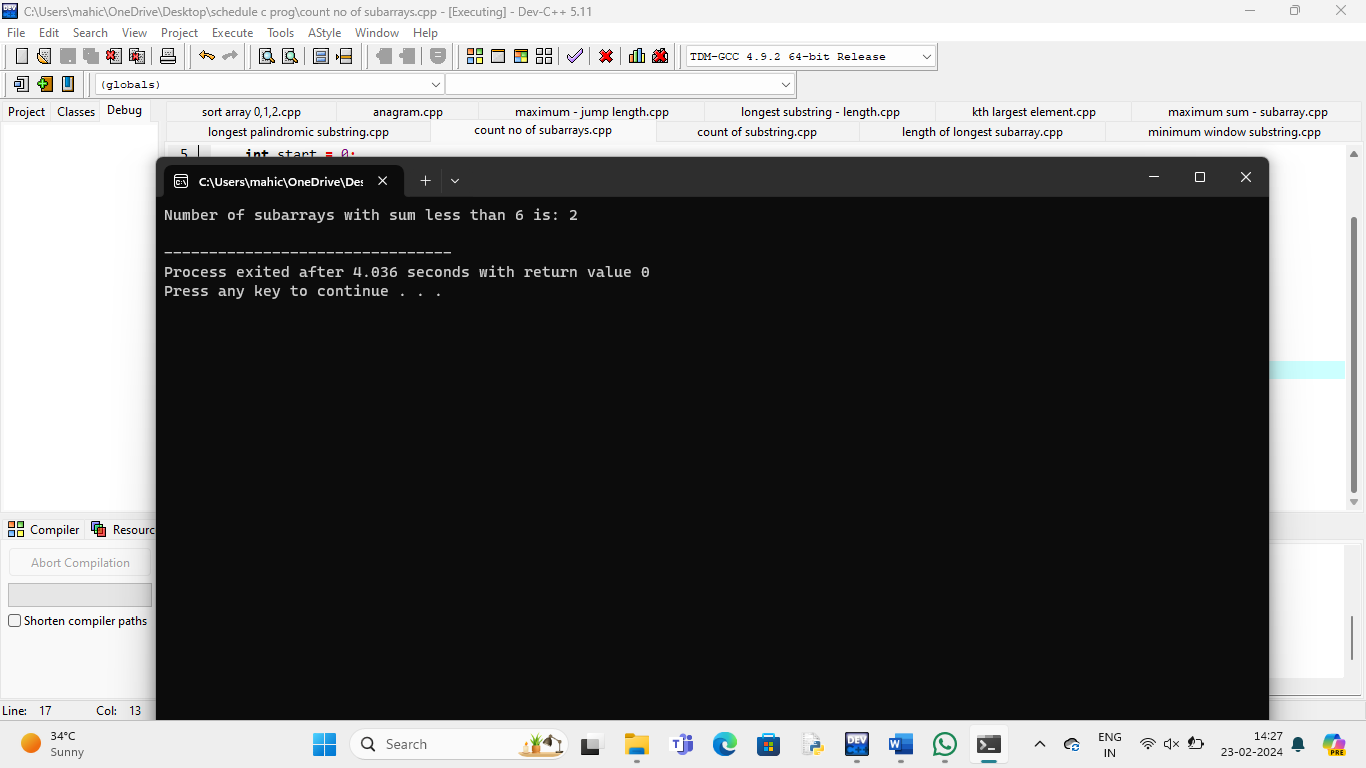
int numSubarrays = countSubarraysWithSumLessThan(arr, n, target);

printf("Number of subarrays with sum less than %d is: %d\n", target, numSubarrays);

return 0;

}

**Output:**



**Count of substring:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LEN 100

struct TrieNode {

struct TrieNode\* children[26];

int count;

};

struct TrieNode\* createNode() {

struct TrieNode\* newNode = (struct TrieNode\*)malloc(sizeof(struct TrieNode));

newNode->count = 0;

for (int i = 0; i < 26; i++) {

newNode->children[i] = NULL;

}

return newNode;

}

void insert(struct TrieNode\* root, char\* str) {

struct TrieNode\* current = root;

int len = strlen(str);

for (int i = 0; i < len; i++) {

int index = str[i] - 'a';

if (current->children[index] == NULL) {

current->children[index] = createNode();

}

current = current->children[index];

}

current->count++;

}

int countDistinctSubstrings(char\* str) {

int n = strlen(str);

struct TrieNode\* root = createNode();

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

for (int len = 1; len <= n - i; len++) {

char\* substr = (char\*)malloc((len + 1) \* sizeof(char));

strncpy(substr, &str[i], len);

substr[len] = '\0';

insert(root, substr);

free(substr);

}

}

int distinctCount = 0;

for (int i = 0; i < 26; i++) {

if (root->children[i] != NULL) {

distinctCount += root->children[i]->count;

}

}

return distinctCount;

}

int main() {

char str[MAX\_LEN];

printf("Enter a string: ");

fgets(str, sizeof(str), stdin);

str[strcspn(str, "\n")] = '\0';

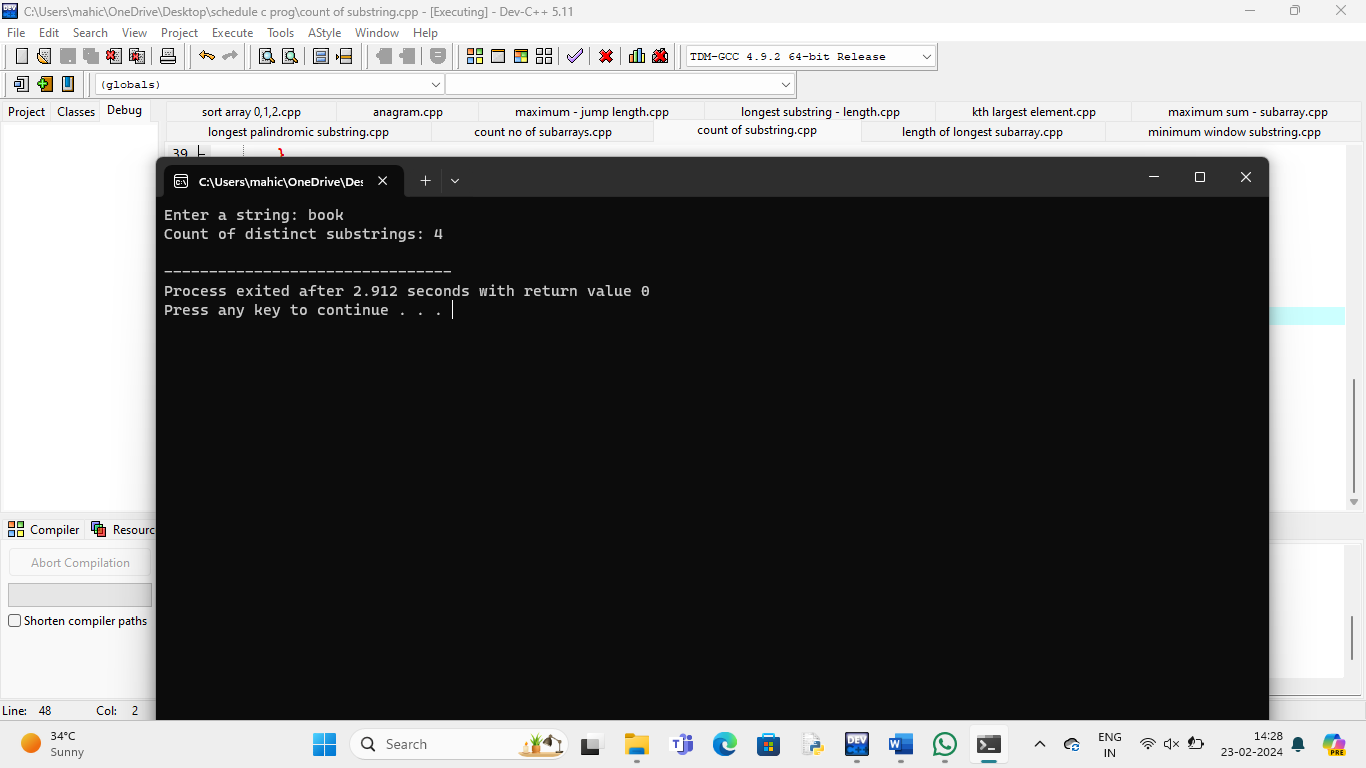
int distinctSubstrings = countDistinctSubstrings(str);

printf("Count of distinct substrings: %d\n", distinctSubstrings);

return 0;

}

**Output:**



**Length of longest subarray:**

#include <stdio.h>

int longestIncreasingSubarray(int arr[], int size) {

if (size == 0) {

return 0;

}

int maxLength = 1;

int currentLength = 1;

for (int i = 1; i < size; i++) {

if (arr[i] > arr[i - 1]) {

currentLength++;

} else {

if (currentLength > maxLength) {

maxLength = currentLength;

}

currentLength = 1;

}

}

if (currentLength > maxLength) {

maxLength = currentLength;

}

return maxLength;

}

int main() {

int arr[] = {1, 2, 3, 2, 5, 7, 8, 9, 10};

int size = sizeof(arr) / sizeof(arr[0]);

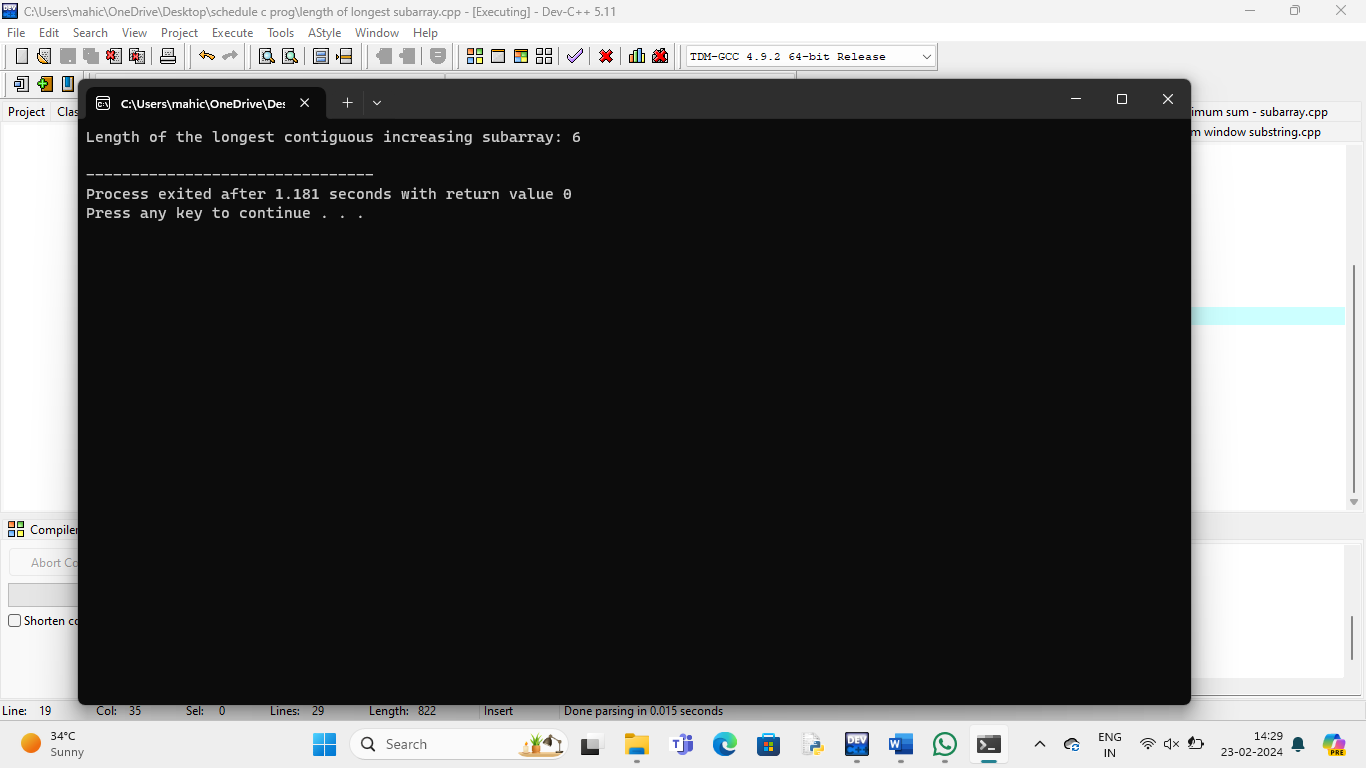
int result = longestIncreasingSubarray(arr, size);

printf("Length of the longest contiguous increasing subarray: %d\n", result);

return 0;

}

**Output:**



**Minimum window substring:**

#include <stdio.h>

#include <string.h>

#include <limits.h>

#define MAX\_CHARS 256

int allCharactersFound(int arr[]) {

for (int i = 0; i < MAX\_CHARS; i++) {

if (arr[i] > 0) {

return 0;

}

}

return 1;

}

char\* minWindowSubstring(char\* str, char\* pattern) {

int patternCount[MAX\_CHARS] = {0};

int strCount[MAX\_CHARS] = {0};

int patternLength = strlen(pattern);

int strLength = strlen(str);

for (int i = 0; i < patternLength; i++) {

patternCount[pattern[i]]++;

}

int minWindowSize = INT\_MAX;

char\* minWindowStart = NULL;

int left = 0, right = 0;

int count = 0;

int startPos = -1;

while (right < strLength) {

if (patternCount[str[right]] > 0) {

strCount[str[right]]++;

if (strCount[str[right]] <= patternCount[str[right]]) {

count++;

}

}

if (count == patternLength) {

while (patternCount[str[left]] == 0 || strCount[str[left]] > patternCount[str[left]]) {

if (strCount[str[left]] > patternCount[str[left]]) {

strCount[str[left]]--;

}

left++;

}

if (right - left + 1 < minWindowSize) {

minWindowSize = right - left + 1;

minWindowStart = &str[left];

startPos = left;

}

}

right++;

}

if (startPos != -1) {

minWindowStart[minWindowSize] = '\0';

}

return minWindowStart;

}

int main() {

char str[] = "ADOBECODEBANC";

char pattern[] = "ABC";

char\* minWindow = minWindowSubstring(str, pattern);

if (minWindow != NULL) {

printf("Minimum window substring containing all characters of pattern: %s\n", minWindow);

} else {

printf("No window found\n");

}

return 0;

}

**Output:**

