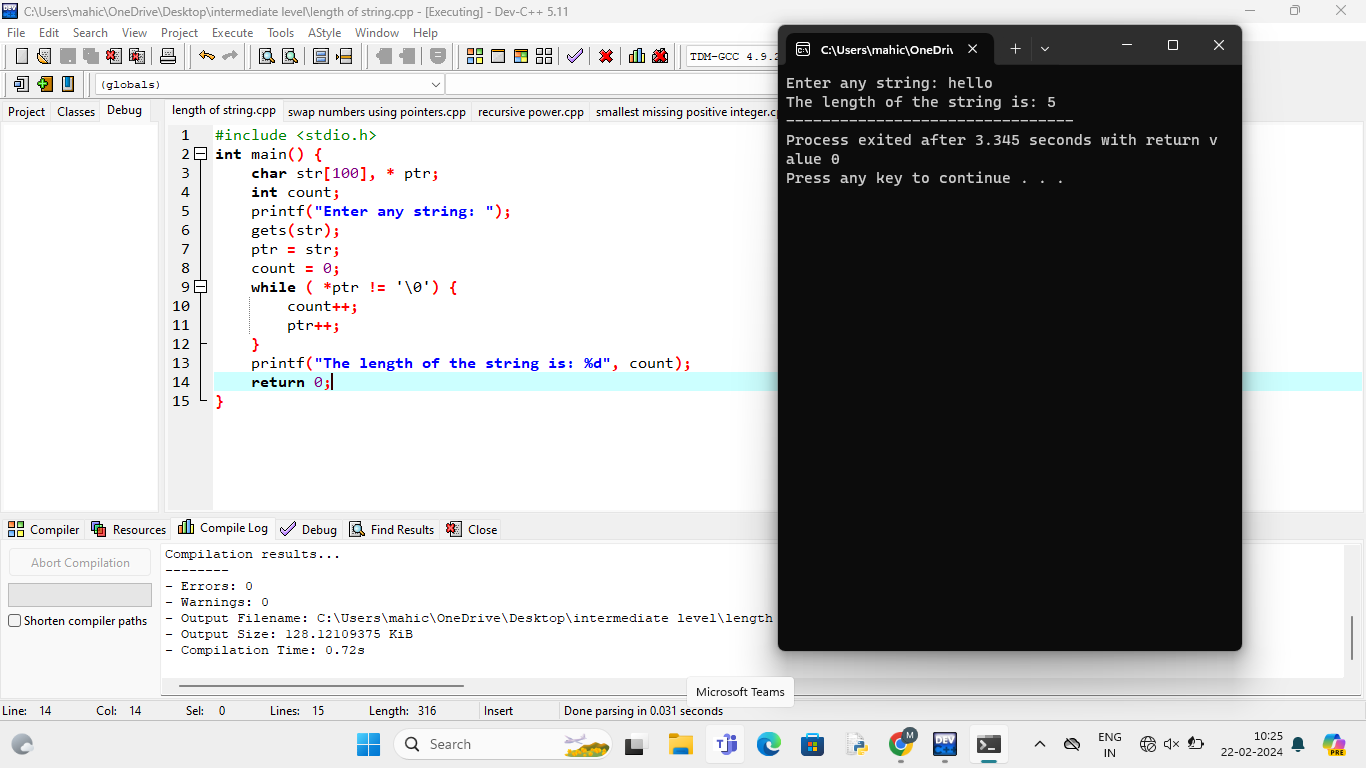
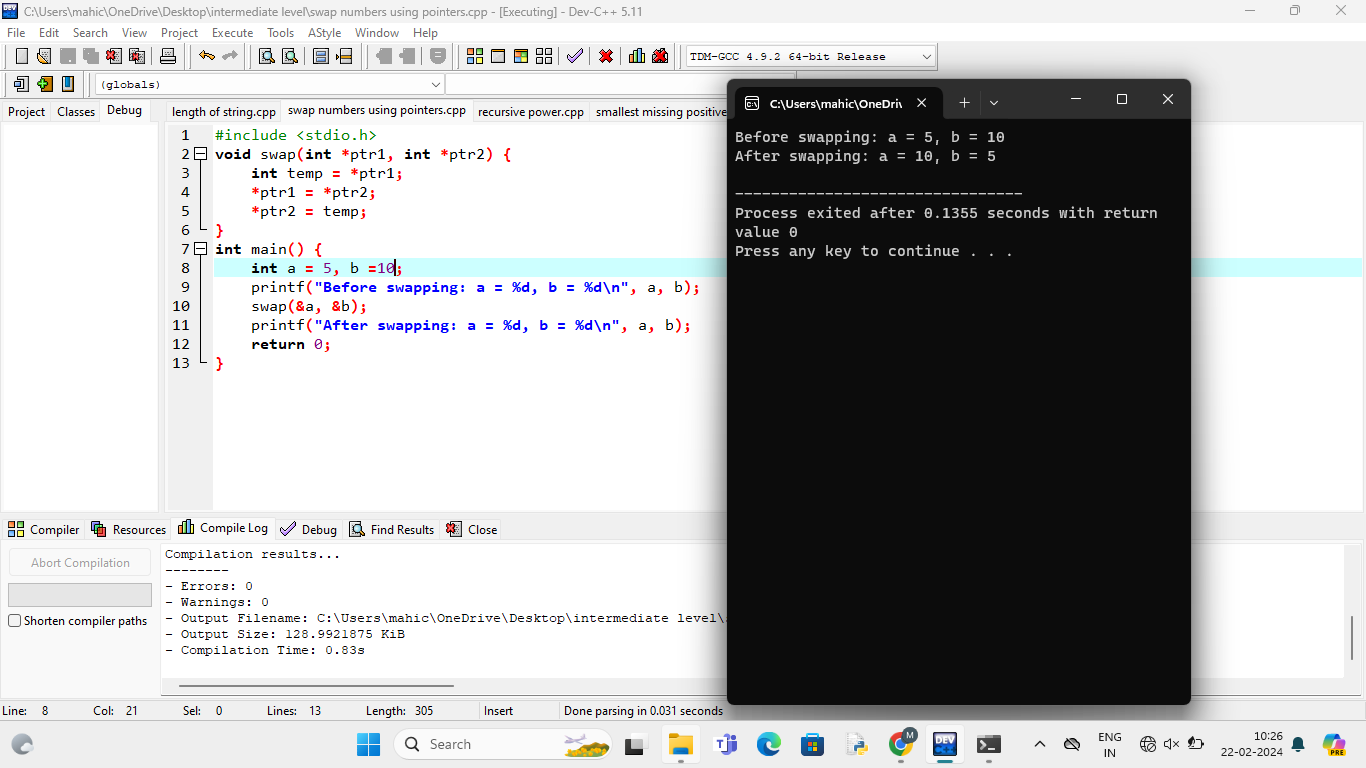
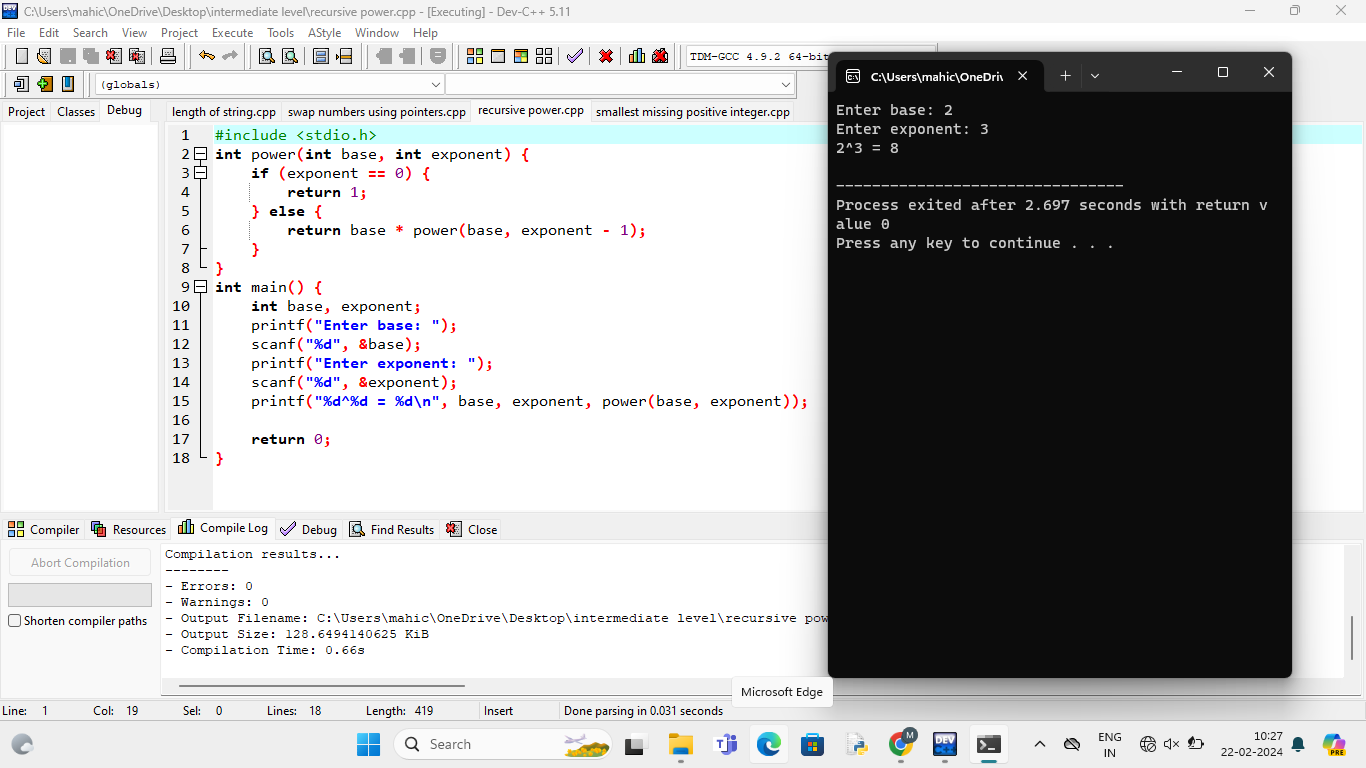
**1.Length of string using pointers**



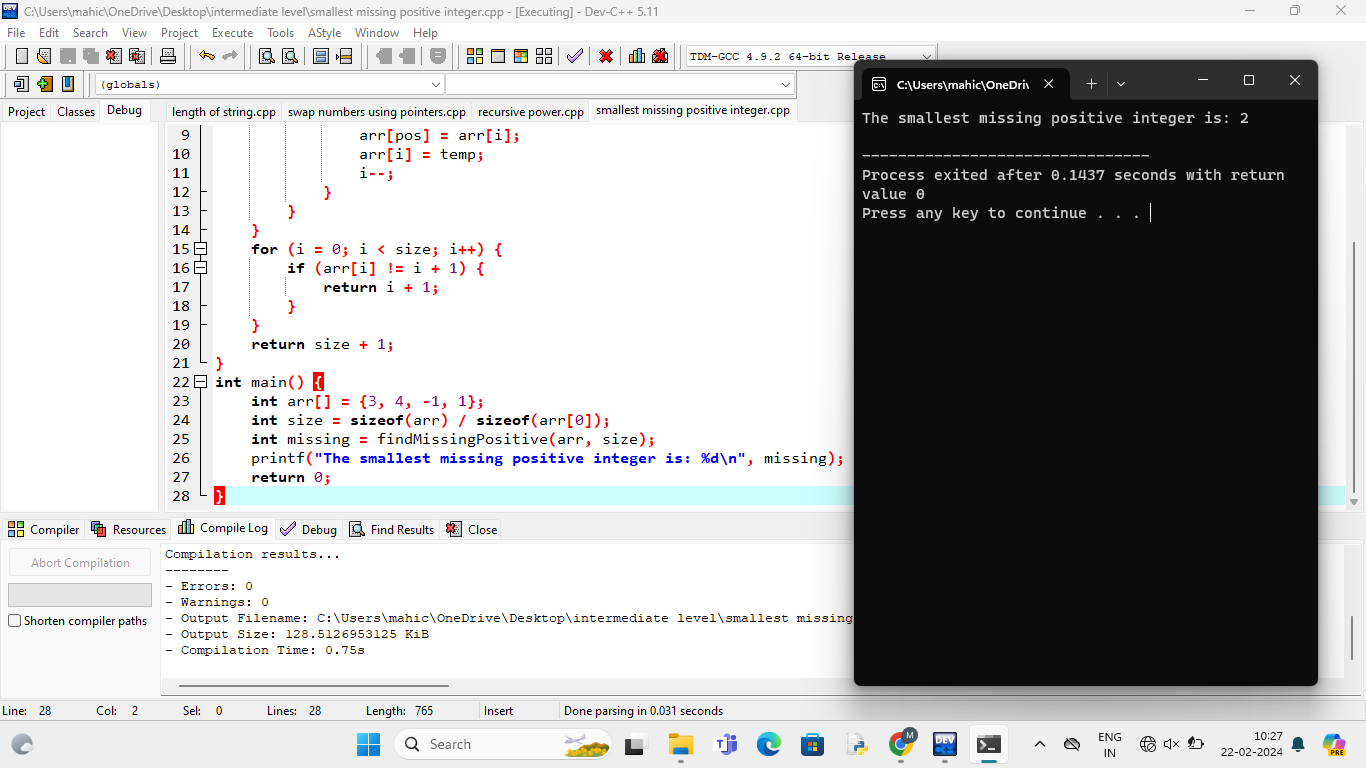
**2.Swap integers using pointers**



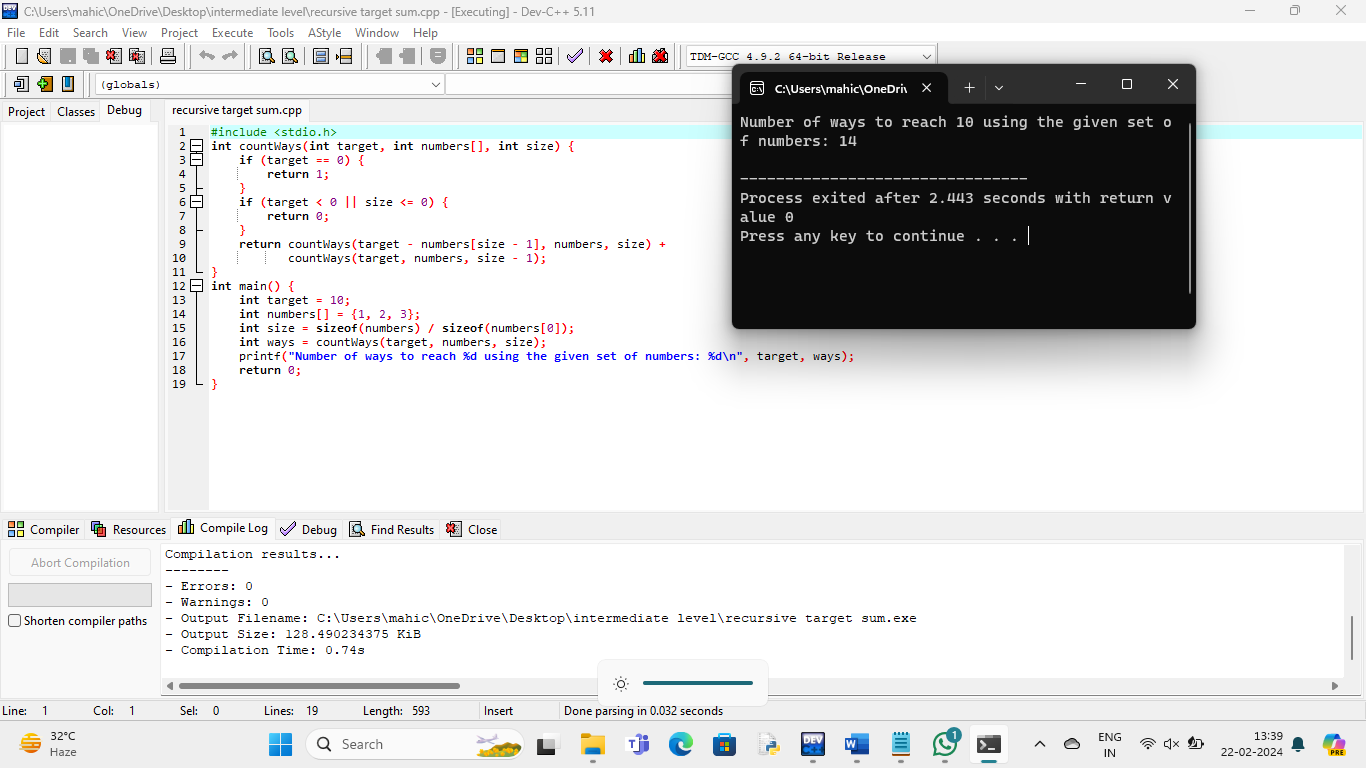
**3.Recursive power**



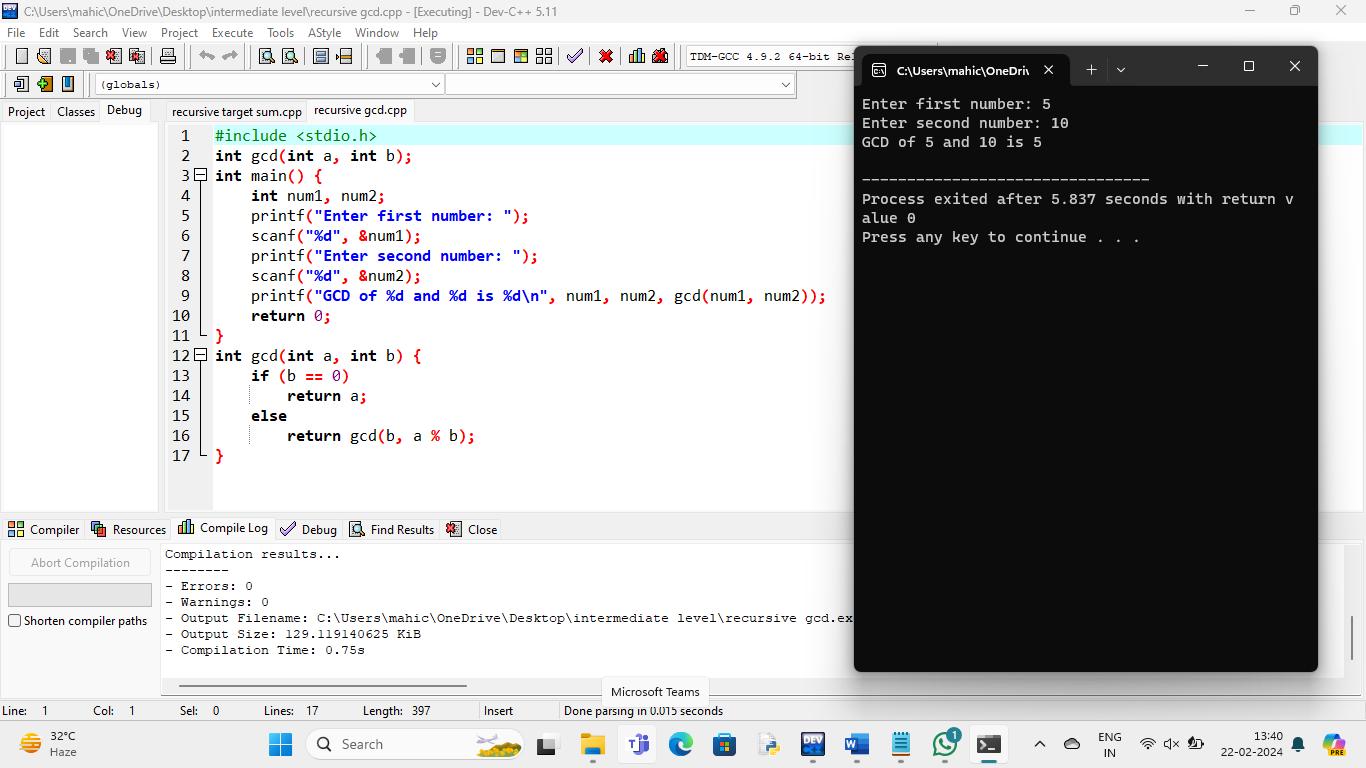
**4.Smallest missing positive integer**



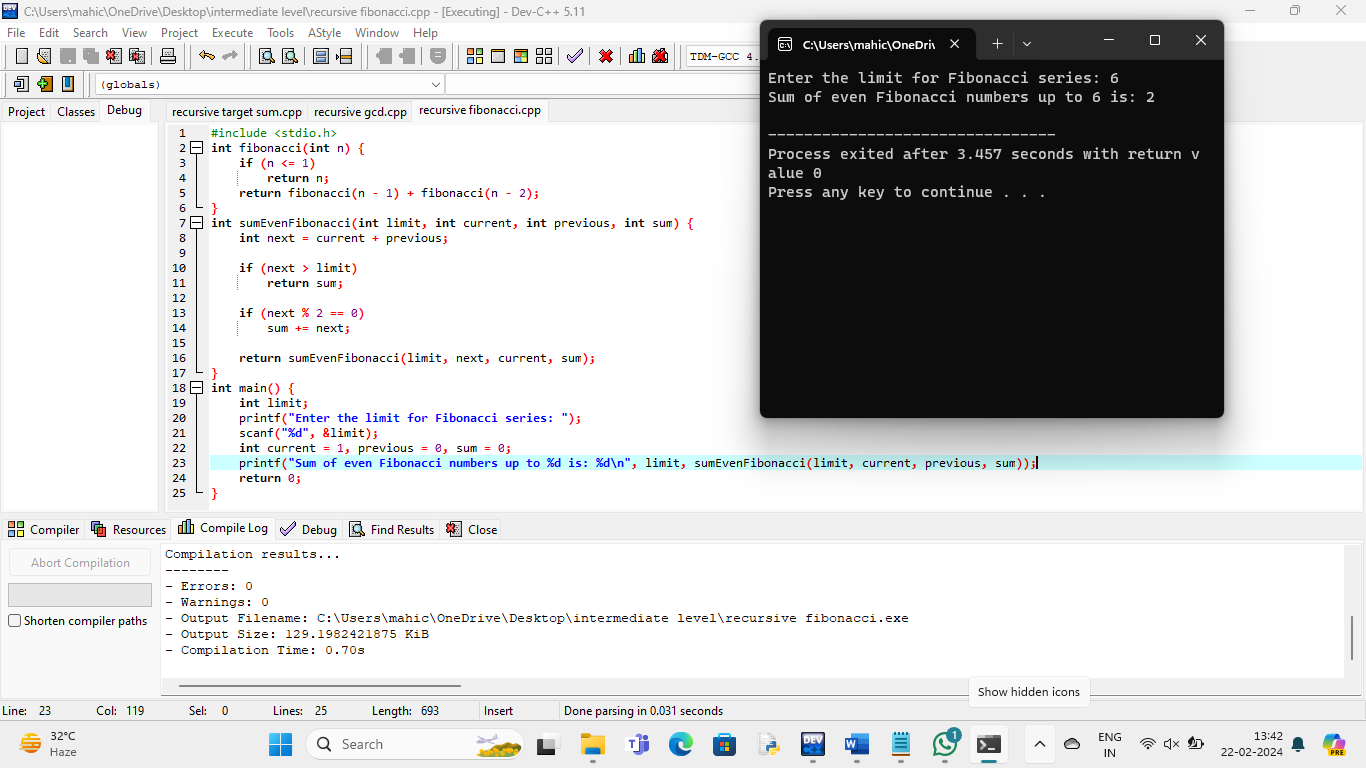
**5.Recursive target sum**



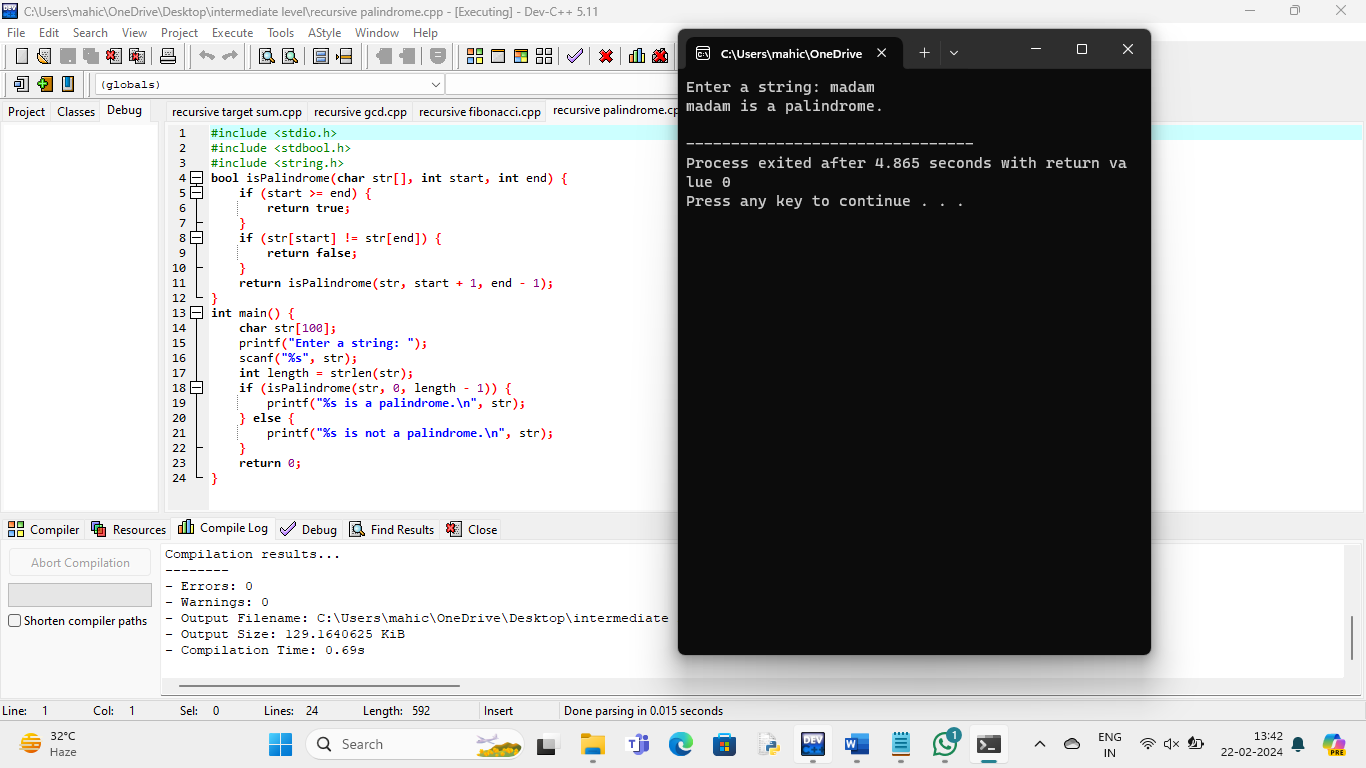
**6.Recursive gcd**



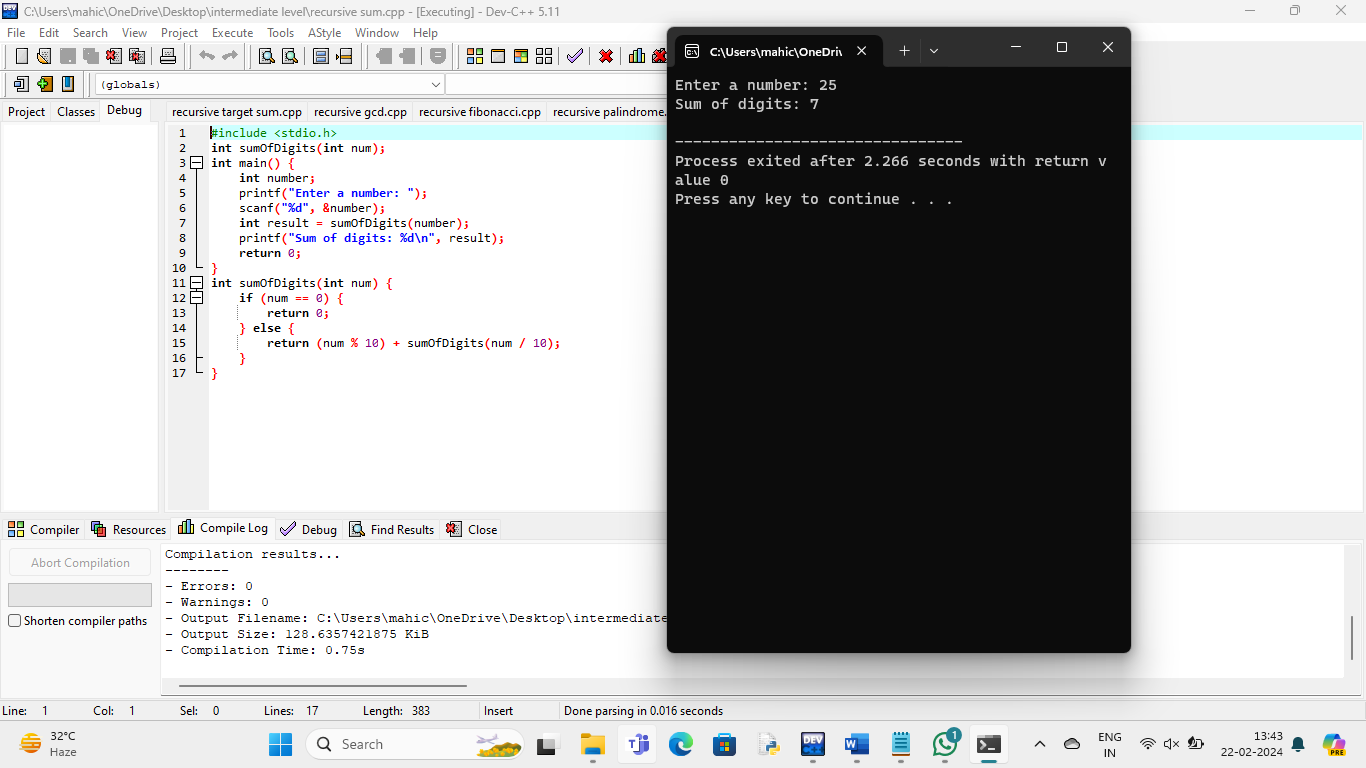
**7.Recursive Fibonacci**



**8.Recursive palindrome**



**9.Recursive sum**



**10.Longest sub string using characters:**

#include <stdio.h>

int findFirstOccurrence(int arr[], int size, int target) {

int low = 0;

int high = size - 1;

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == target) {

result = mid;

high = mid - 1;

} else if (arr[mid] < target) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return result;}

int main() {

int arr[] = {1, 2, 3, 4, 4, 4, 5, 6};

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

int target = 3;

int index = findFirstOccurrence(arr, size, target);

if (index != -1) {

printf("First occurrence of %d is at index %d\n", target, index);

} else {

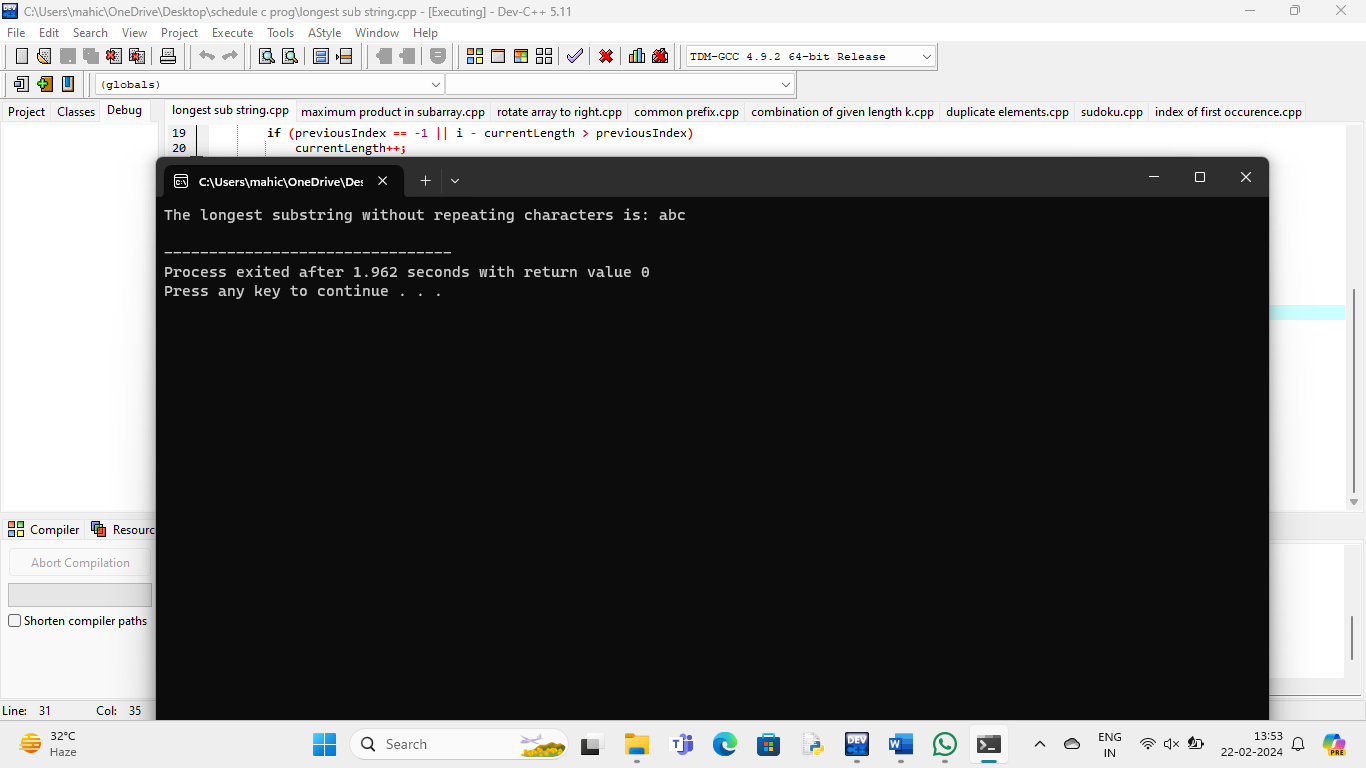
printf("%d not found in the array\n", target);

}

return 0;

}

**Output:**



**11.Maximum product in 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 maxProductSubarray(int arr[], int n) {

int maxEndingHere = 1;

int minEndingHere = 1;

int maxProduct = 1;

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

if (arr[i] > 0) {

maxEndingHere = maxEndingHere \* arr[i];

minEndingHere = min(minEndingHere \* arr[i], 1);

}

else if (arr[i] == 0) {

maxEndingHere = 1;

minEndingHere = 1;

}

else {

int temp = maxEndingHere;

maxEndingHere = max(minEndingHere \* arr[i], 1);

minEndingHere = temp \* arr[i];

}

if (maxProduct < maxEndingHere) {

maxProduct = maxEndingHere;

}

}

return maxProduct;

}

int main() {

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

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

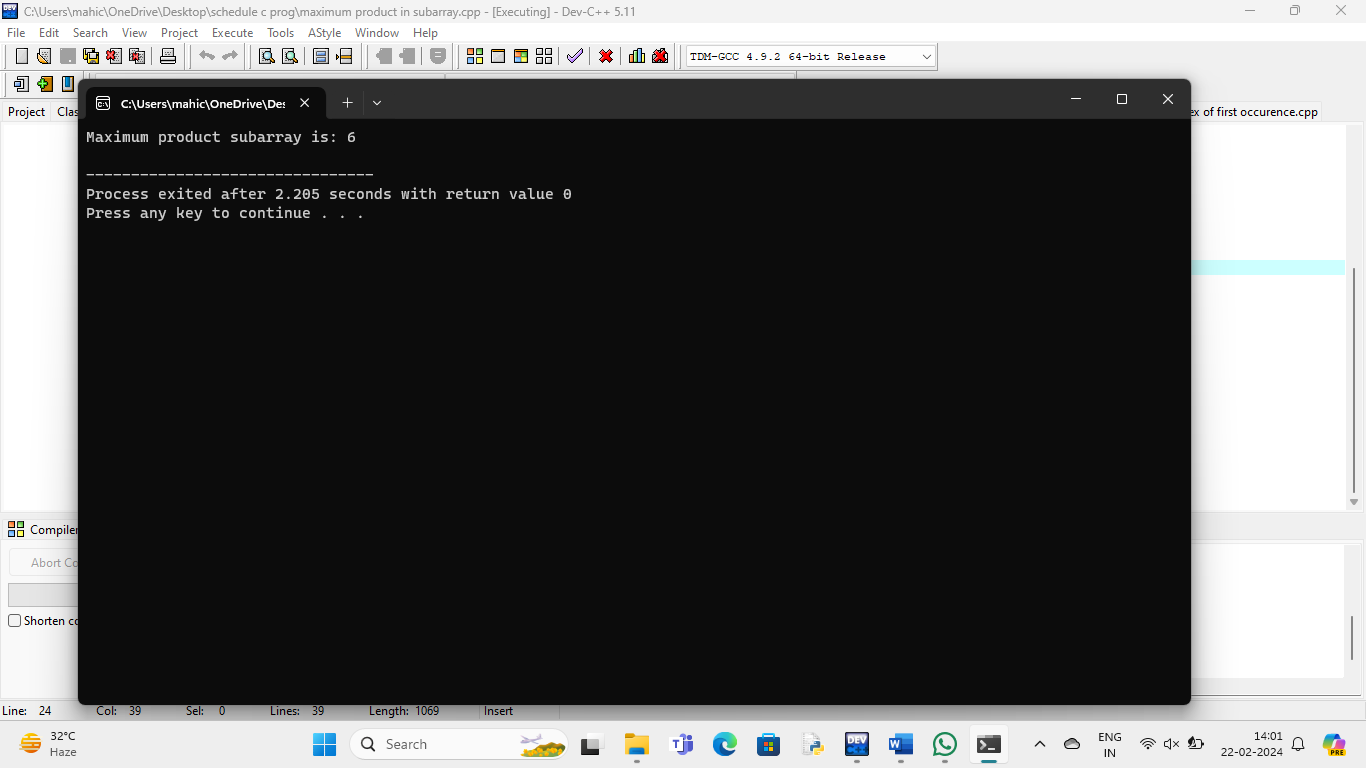
int maxProd = maxProductSubarray(arr, n);

printf("Maximum product subarray is: %d\n", maxProd);

return 0;

}

**Output:**



**12.Rotate array to right:**

#include <stdio.h>

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

k = k % n;

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 = 3;

printf("Original array: ");

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

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

}

printf("\n");

rotateArray(arr, n, k);

printf("Rotated array: ");

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

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

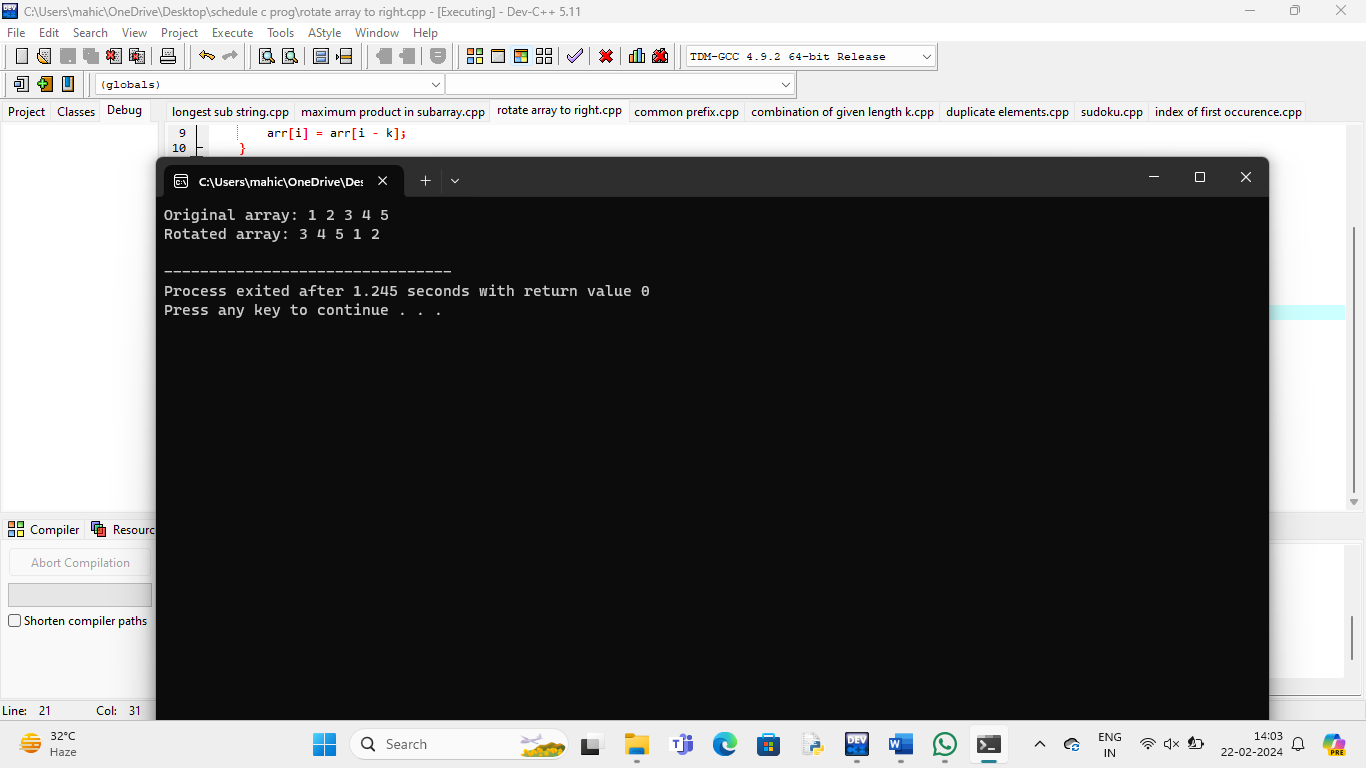
}

printf("\n");

return 0;

}

**Output:**



**13.Common prefix:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int min(int a, int b) {

return (a < b) ? a : b;

}

char\* longestCommonPrefix(char\*\* strs, int strsSize) {

if (strsSize == 0)

return "";

int minLen = strlen(strs[0]);

for (int i = 1; i < strsSize; ++i)

minLen = min(minLen, strlen(strs[i]));

int i, j;

for (i = 0; i < minLen; ++i) {

for (j = 1; j < strsSize; ++j) {

if (strs[j][i] != strs[j - 1][i])

break;

}

if (j != strsSize)

break;

}

char\* commonPrefix = (char\*)malloc(sizeof(char) \* (i + 1));

strncpy(commonPrefix, strs[0], i);

commonPrefix[i] = '\0';

return commonPrefix;

}

int main() {

char \*strings[] = {"flower", "flow", "flight"};

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

char \*prefix = longestCommonPrefix(strings, size);

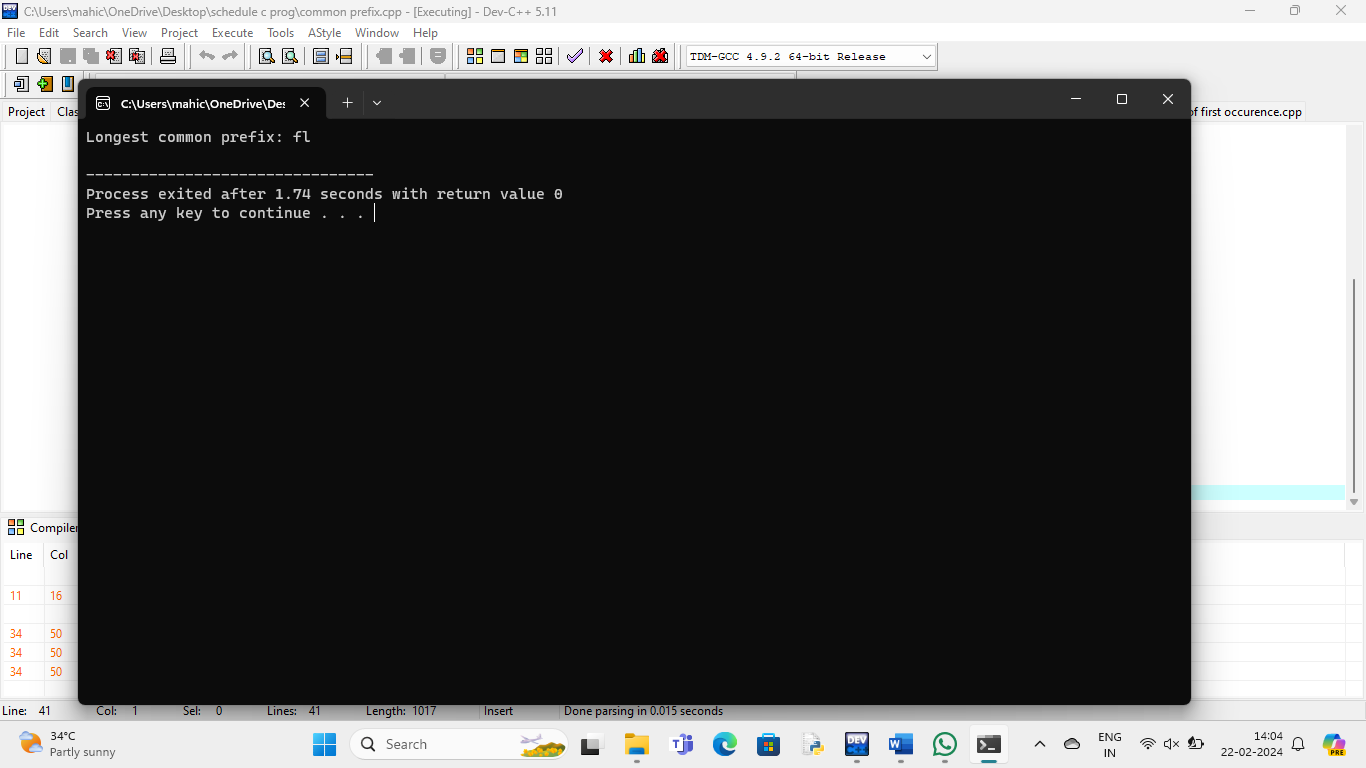
printf("Longest common prefix: %s\n", prefix);

free(prefix);

return 0;

}

**Output:**



**14.Combination of given length k:**

#include <stdio.h>

void findCombinations(int arr[], int n, int k, int index, int data[], int i) {

if (index == k) {

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

printf("%d ", data[j]);

printf("\n");

return;

}

if (i >= n)

return;

data[index] = arr[i];

findCombinations(arr, n, k, index + 1, data, i + 1);

while (arr[i] == arr[i+1])

i++;

findCombinations(arr, n, k, index, data, i + 1);

}

int main() {

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

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

int k = 2;

int data[k];

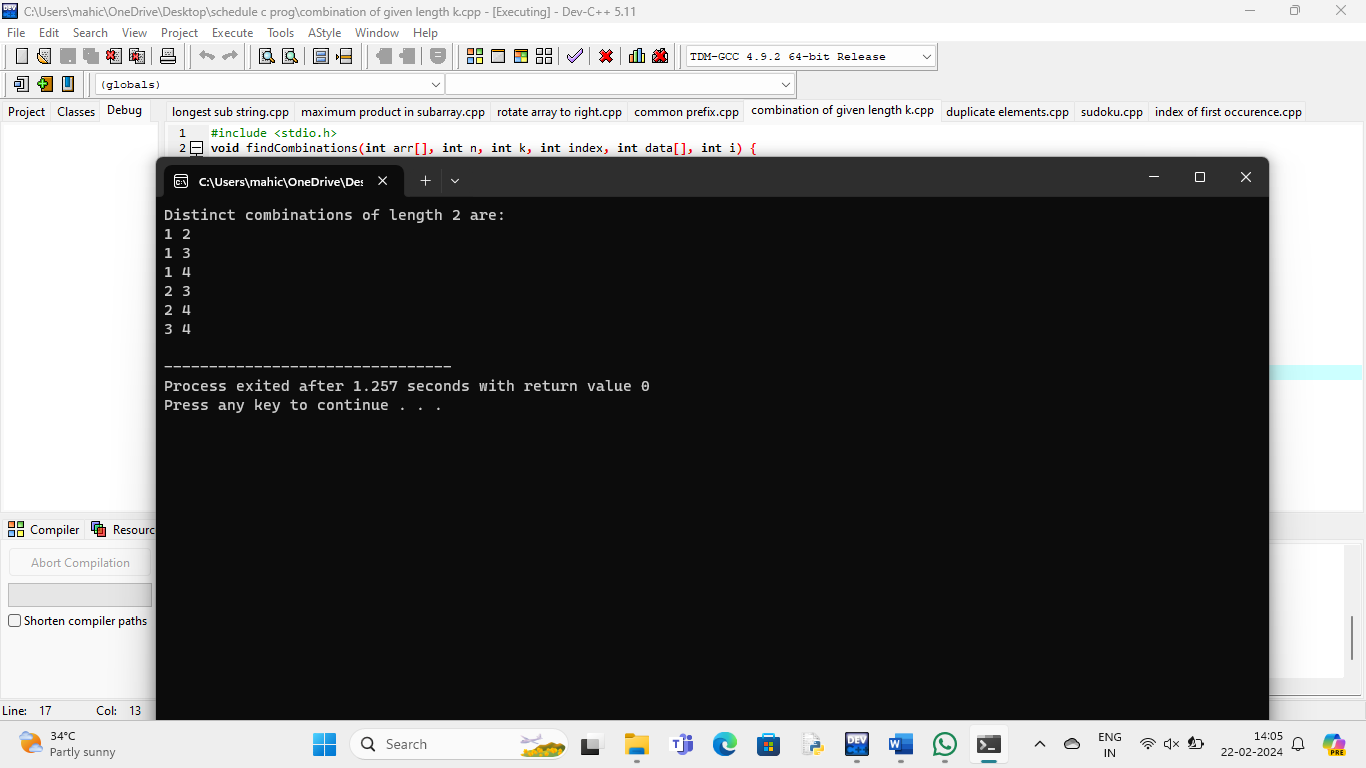
printf("Distinct combinations of length %d are:\n", k);

findCombinations(arr, n, k, 0, data, 0);

return 0;

}

**Output:**



**15.Duplicate elements:**

#include <stdio.h>

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

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

return n;

int index = 0;

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

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

arr[index++] = arr[i];

}

}

arr[index++] = arr[n - 1];

return index;

}

int main() {

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

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

printf("Original Array: ");

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

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

}

printf("\n");

int newLength = removeDuplicates(arr, n);

printf("Array after removing duplicates: ");

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

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

}

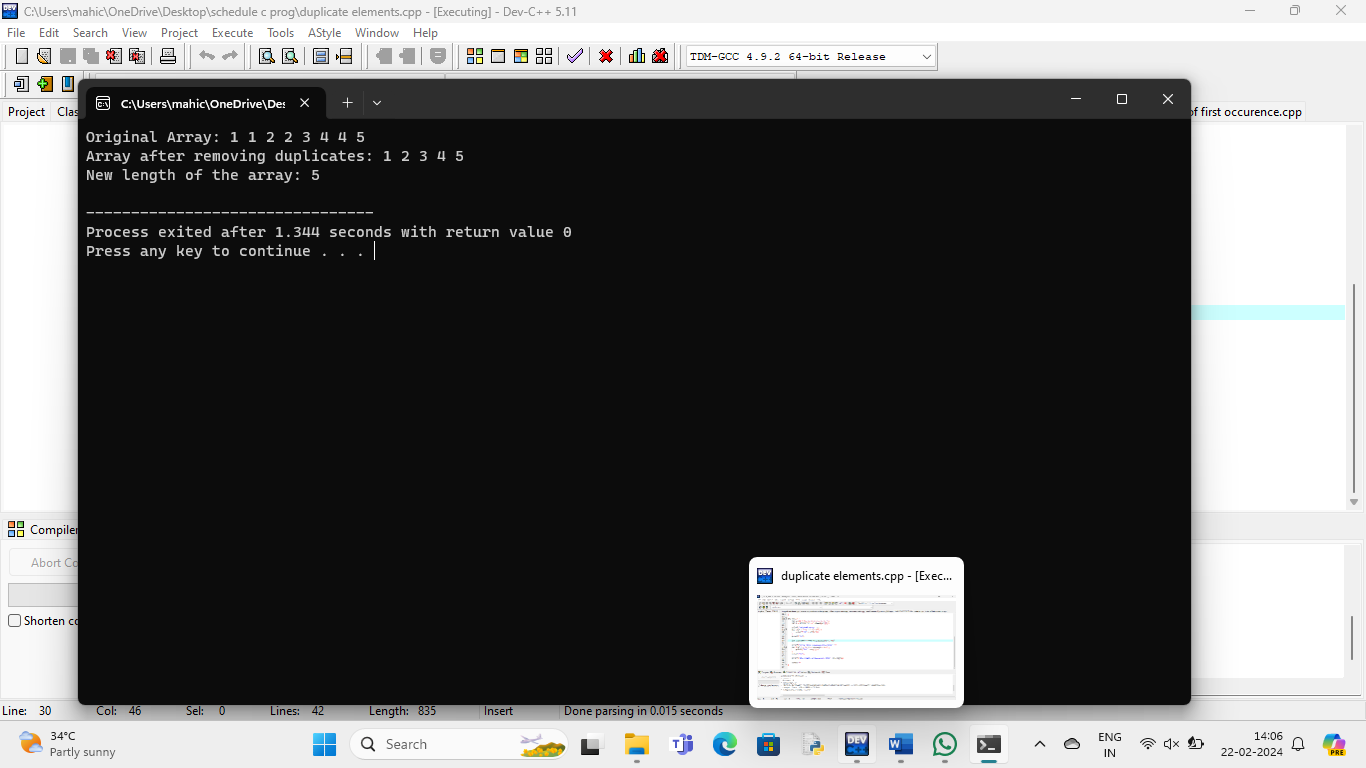
printf("\n");

printf("New length of the array: %d\n", newLength);

return 0;

}

**Output:**



**16.Sudoku:**

#include <stdio.h>

#include <stdbool.h>

#define SIZE 9

bool isValidRow(char board[SIZE][SIZE], int row) {

bool visited[SIZE + 1] = {false};

for (int col = 0; col < SIZE; col++) {

if (board[row][col] != '.') {

int num = board[row][col] - '0';

if (visited[num])

return false;

visited[num] = true;

}

}

return true;

}

bool isValidCol(char board[SIZE][SIZE], int col) {

bool visited[SIZE + 1] = {false};

for (int row = 0; row < SIZE; row++) {

if (board[row][col] != '.') {

int num = board[row][col] - '0';

if (visited[num])

return false;

visited[num] = true;

}

}

return true;

}

bool isValidSubgrid(char board[SIZE][SIZE], int startRow, int startCol) {

bool visited[SIZE + 1] = {false};

for (int row = startRow; row < startRow + 3; row++) {

for (int col = startCol; col < startCol + 3; col++) {

if (board[row][col] != '.') {

int num = board[row][col] - '0';

if (visited[num])

return false;

visited[num] = true;

}

}

}

return true;

}

bool isValidSudoku(char board[SIZE][SIZE]) {

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

if (!isValidRow(board, i))

return false;

}

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

if (!isValidCol(board, i))

return false;

}

for (int startRow = 0; startRow < SIZE; startRow += 3) {

for (int startCol = 0; startCol < SIZE; startCol += 3) {

if (!isValidSubgrid(board, startRow, startCol))

return false;

}

}

return true;

}

int main() {

char board[SIZE][SIZE] = {

{'5', '3', '.', '.', '7', '.', '.', '.', '.'},

{'6', '.', '.', '1', '9', '5', '.', '.', '.'},

{'.', '9', '8', '.', '.', '.', '.', '6', '.'},

{'8', '.', '.', '.', '6', '.', '.', '.', '3'},

{'4', '.', '.', '8', '.', '3', '.', '.', '1'},

{'7', '.', '.', '.', '2', '.', '.', '.', '6'},

{'.', '6', '.', '.', '.', '.', '2', '8', '.'},

{'.', '.', '.', '4', '1', '9', '.', '.', '5'},

{'.', '.', '.', '.', '8', '.', '.', '7', '9'}

};

if (isValidSudoku(board))

printf("The Sudoku board is valid.\n");

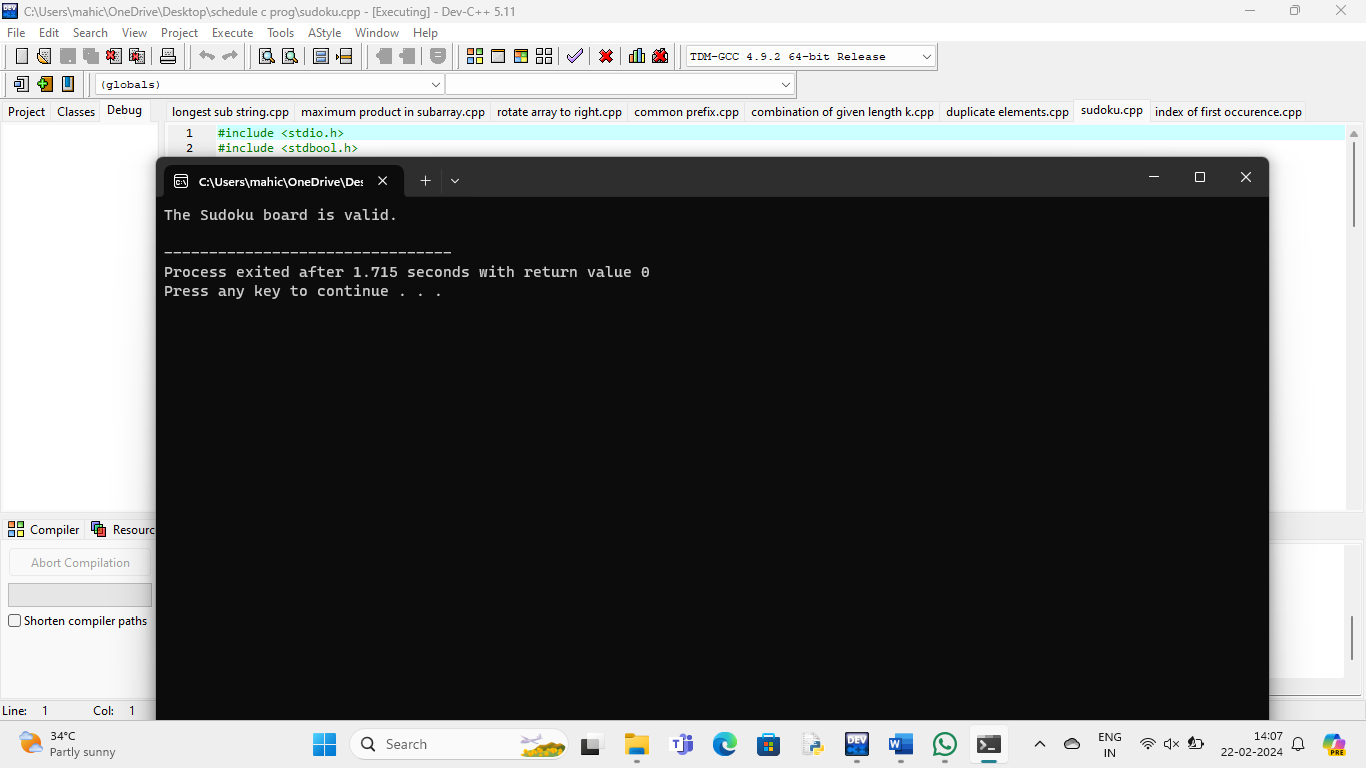
else

printf("The Sudoku board is not valid.\n");

return 0;

}

**Output:**



**17.Index of the first occurrence:**

#include <stdio.h>

int findFirstOccurrence(int arr[], int size, int target) {

int low = 0;

int high = size - 1;

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == target) {

result = mid;

high = mid - 1;

} else if (arr[mid] < target) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return result;

}

int main() {

int arr[] = {1, 2, 3, 4, 4, 4, 5, 6};

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

int target = 3;

int index = findFirstOccurrence(arr, size, target);

if (index != -1) {

printf("First occurrence of %d is at index %d\n", target, index);

} else {

printf("%d not found in the array\n", target);

}

return 0;

}

**Output:**

