1. Fibonacci series.

#include <stdio.h>

int fibonacci(int n);

int main() {

int n\_terms, i;

printf("Enter the number of terms: ");

scanf("%d", &n\_terms);

printf("Fibonacci sequence: ");

for (i = 0; i < n\_terms; i++) {

printf("%d ", fibonacci(i));

}

return 0;

}

int fibonacci(int n) {

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

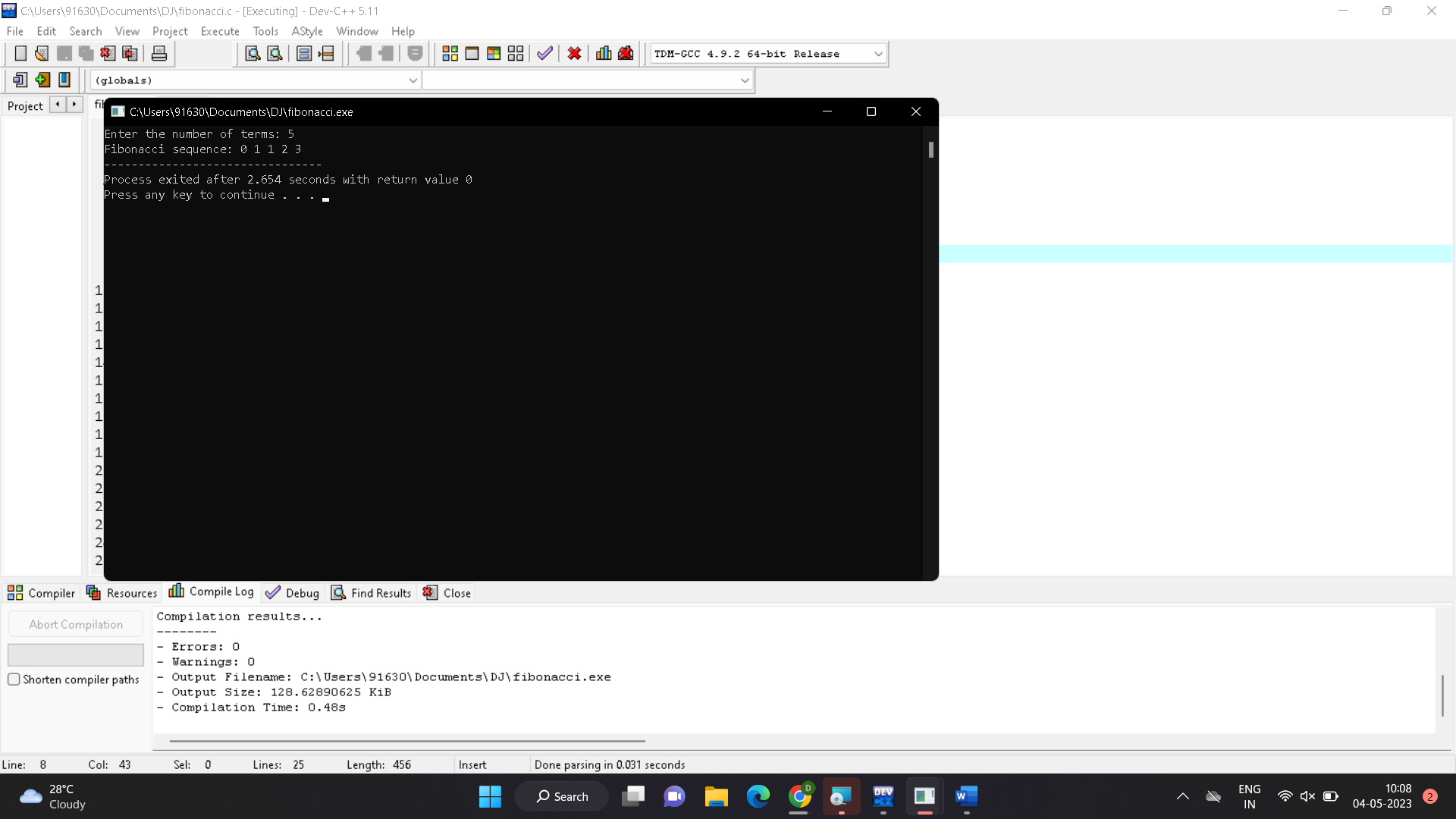
return n;

} else {

return fibonacci(n-1) + fibonacci(n-2);

}

}



1. Armstrong number.

#include <stdio.h>

#include <math.h>

int main()

{

int num, originalNum, remainder, n = 0;

float result = 0.0;

printf("Enter an integer: ");

scanf("%d", &num);

originalNum = num;

while (originalNum != 0) {

originalNum /= 10;

++n;

}

originalNum = num;

while (originalNum != 0) {

remainder = originalNum % 10;

result += pow(remainder, n);

originalNum /= 10;

}

if ((int)result == num) {

printf("%d is an Armstrong number.", num);

}

else {

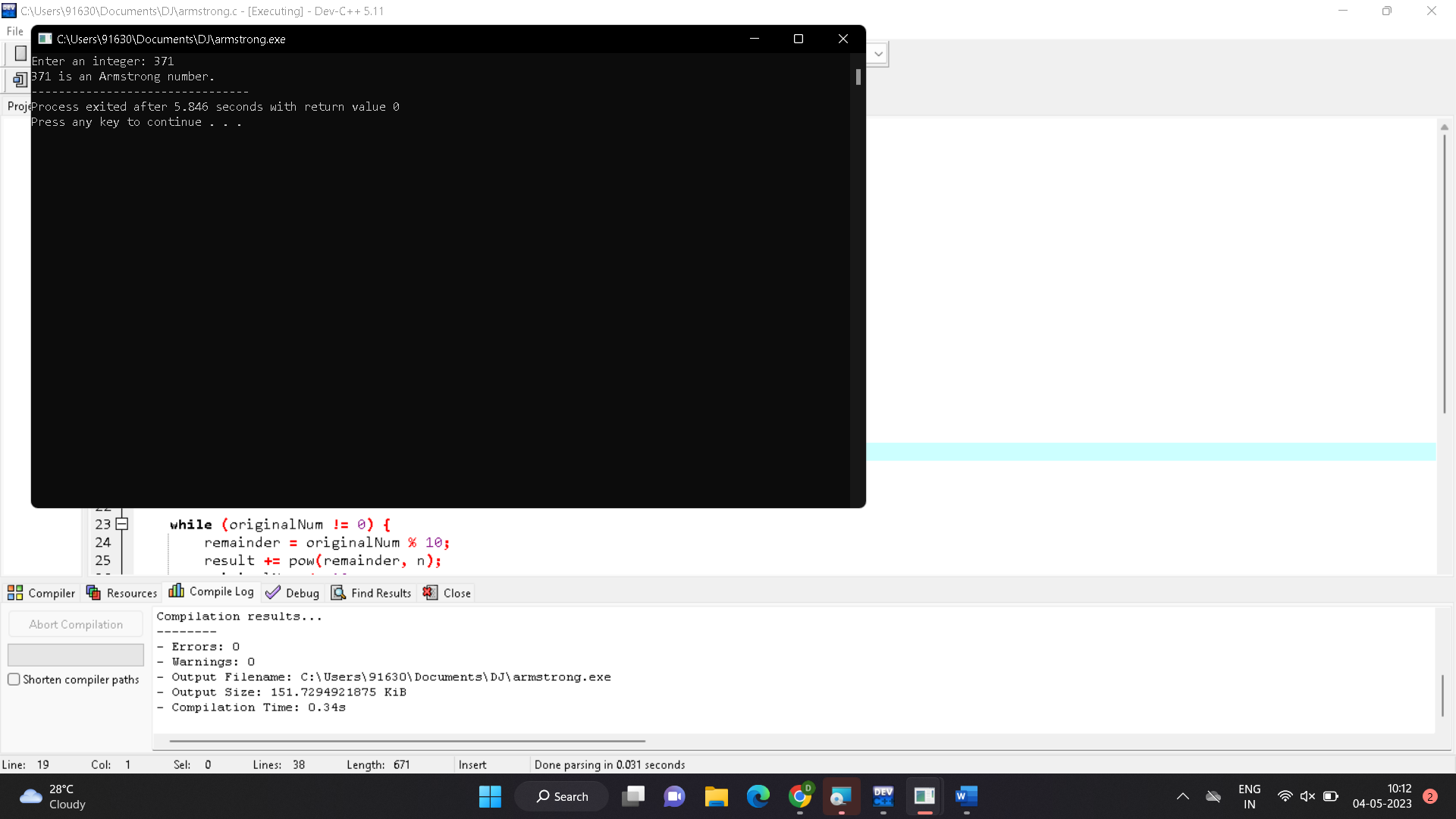
printf("%d is not an Armstrong number.", num);

}

return 0;

}

Out put:



1. Gcd .

#include <stdio.h>

int main() {

int num1, num2, gcd;

printf("Enter two numbers to find their GCD: ");

scanf("%d %d", &num1, &num2);

while (num2 != 0) {

int temp = num2;

num2 = num1 % num2;

num1 = temp;

}

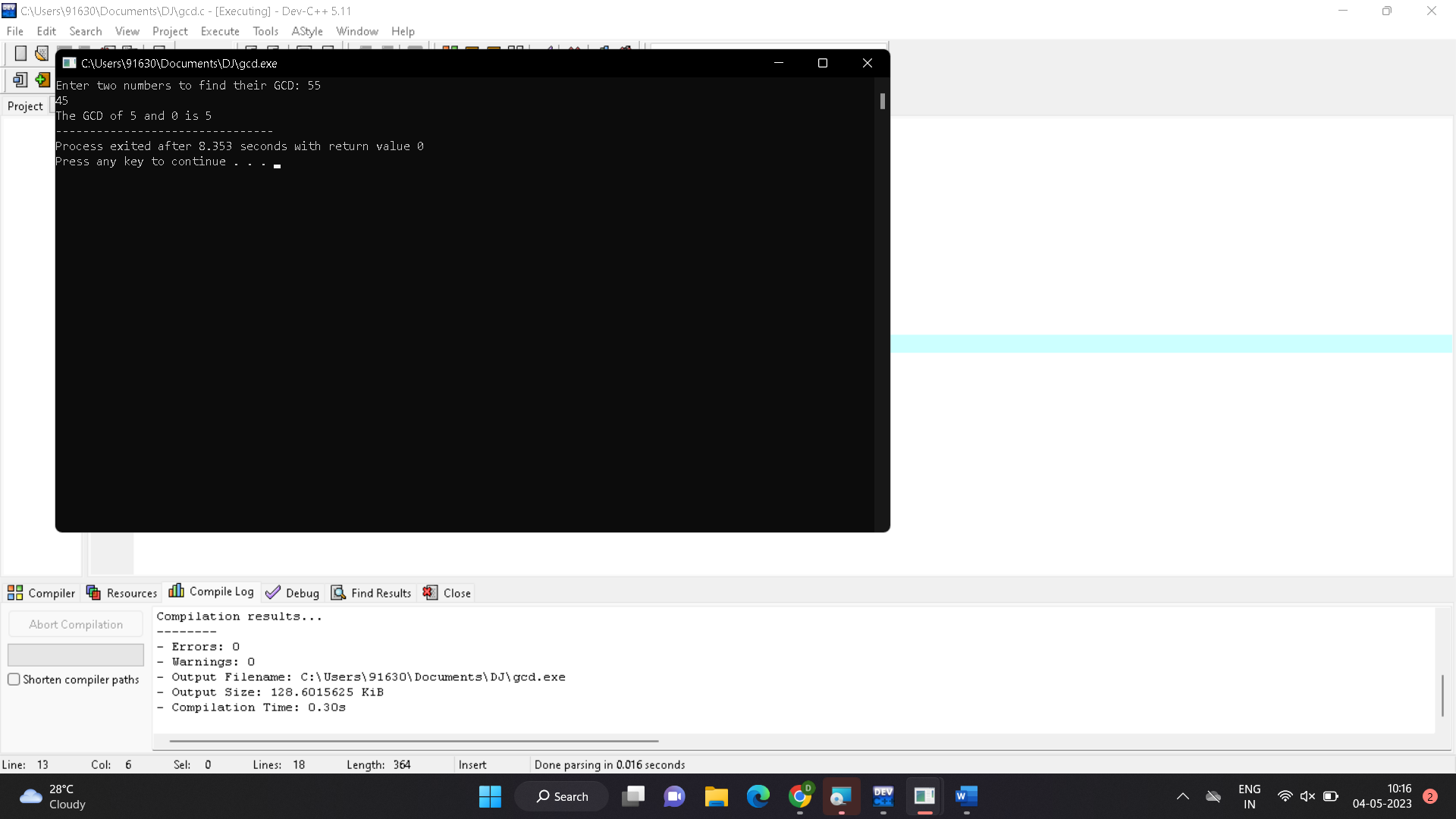
gcd = num1;

printf("The GCD of %d and %d is %d", num1, num2, gcd);

return 0;

}

Output:



1. Largest element.

#include <stdio.h>

int main() {

double n1, n2, n3;

printf("Enter three different numbers: ");

scanf("%lf %lf %lf", &n1, &n2, &n3);

if (n1 >= n2 && n1 >= n3)

printf("%.2f is the largest number.", n1);

if (n2 >= n1 && n2 >= n3)

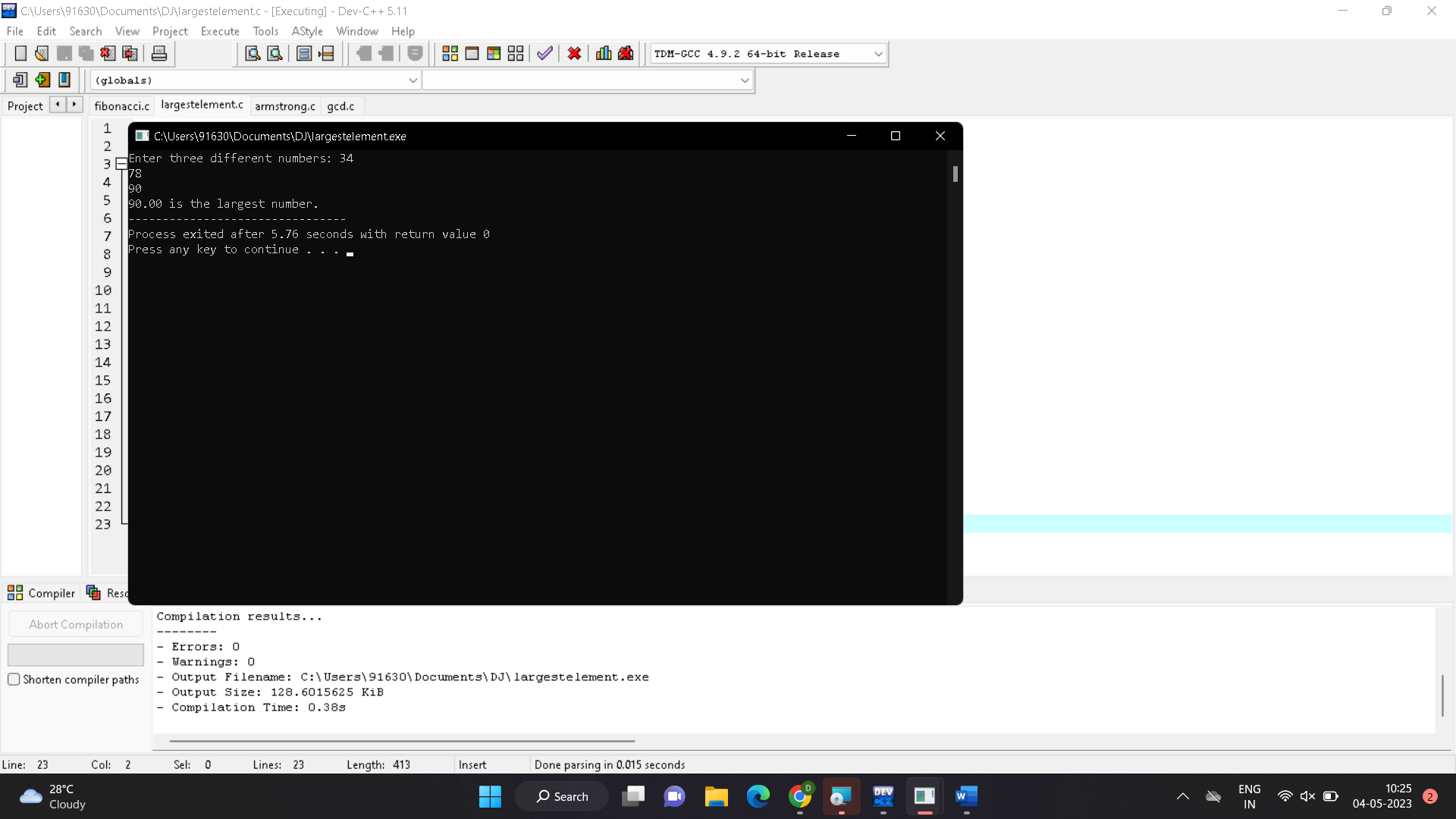
printf("%.2f is the largest number.", n2);

if (n3 >= n1 && n3 >= n2)

printf("%.2f is the largest number.", n3);

return 0;

}

Output: 

1. Factorial.

#include <stdio.h>

int main() {

int num, i;

unsigned long long factorial = 1;

printf("Enter an integer: ");

scanf("%d", &num);

if (num < 0) {

printf("Error: Factorial of a negative number doesn't exist.");

}

else {

for (i = 1; i <= num; ++i) {

factorial \*= i;

}

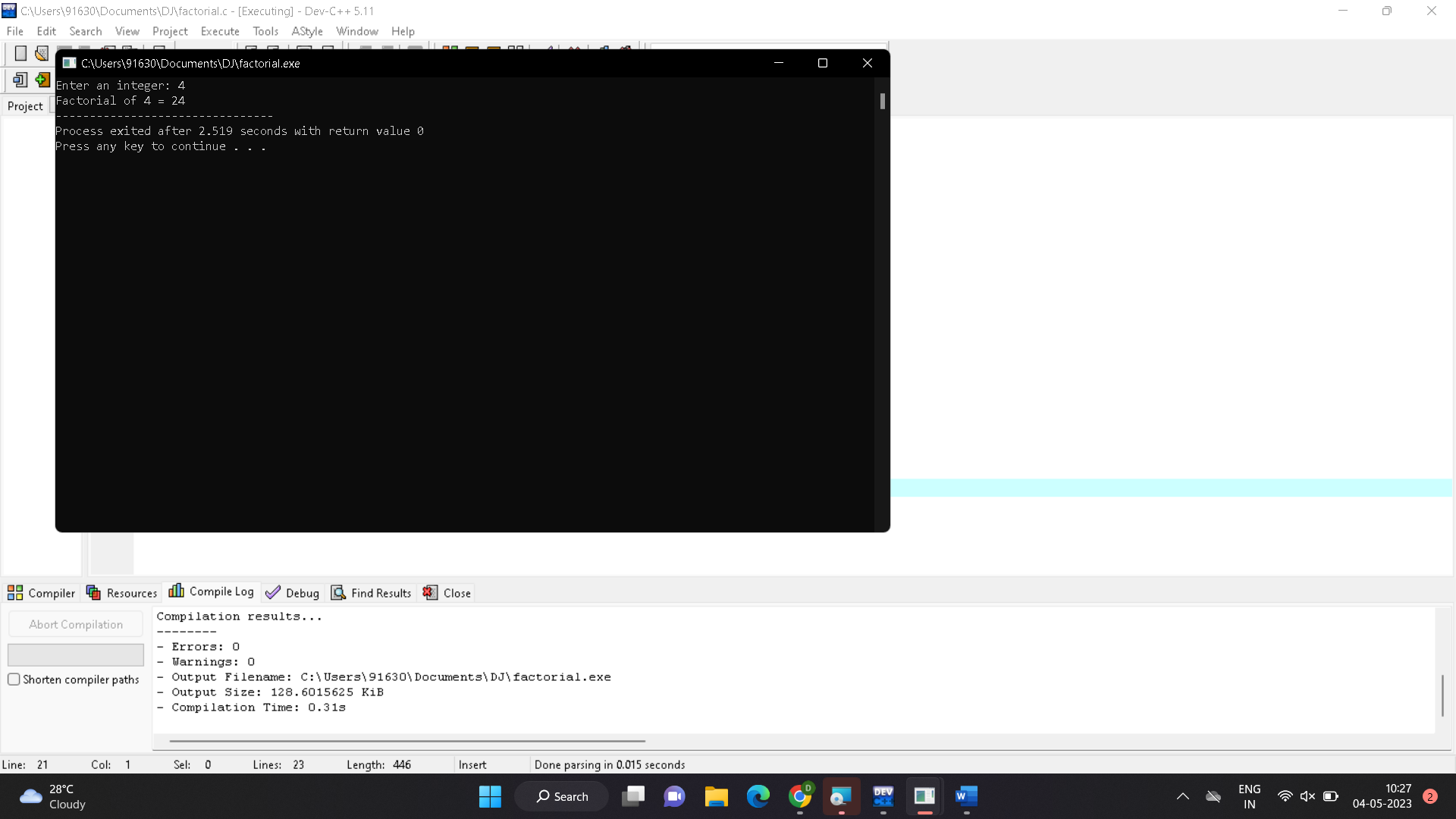
printf("Factorial of %d = %llu", num, factorial);

}

return 0;

}

Output:



1. Prime or not.

#include <stdio.h>

int main() {

int n, i, flag = 0;

printf("Enter a positive integer: ");

scanf("%d", &n);

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

flag = 1;

for (i = 2; i <= n / 2; ++i) {

if (n % i == 0) {

flag = 1;

break;

}

}

if (flag == 0)

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

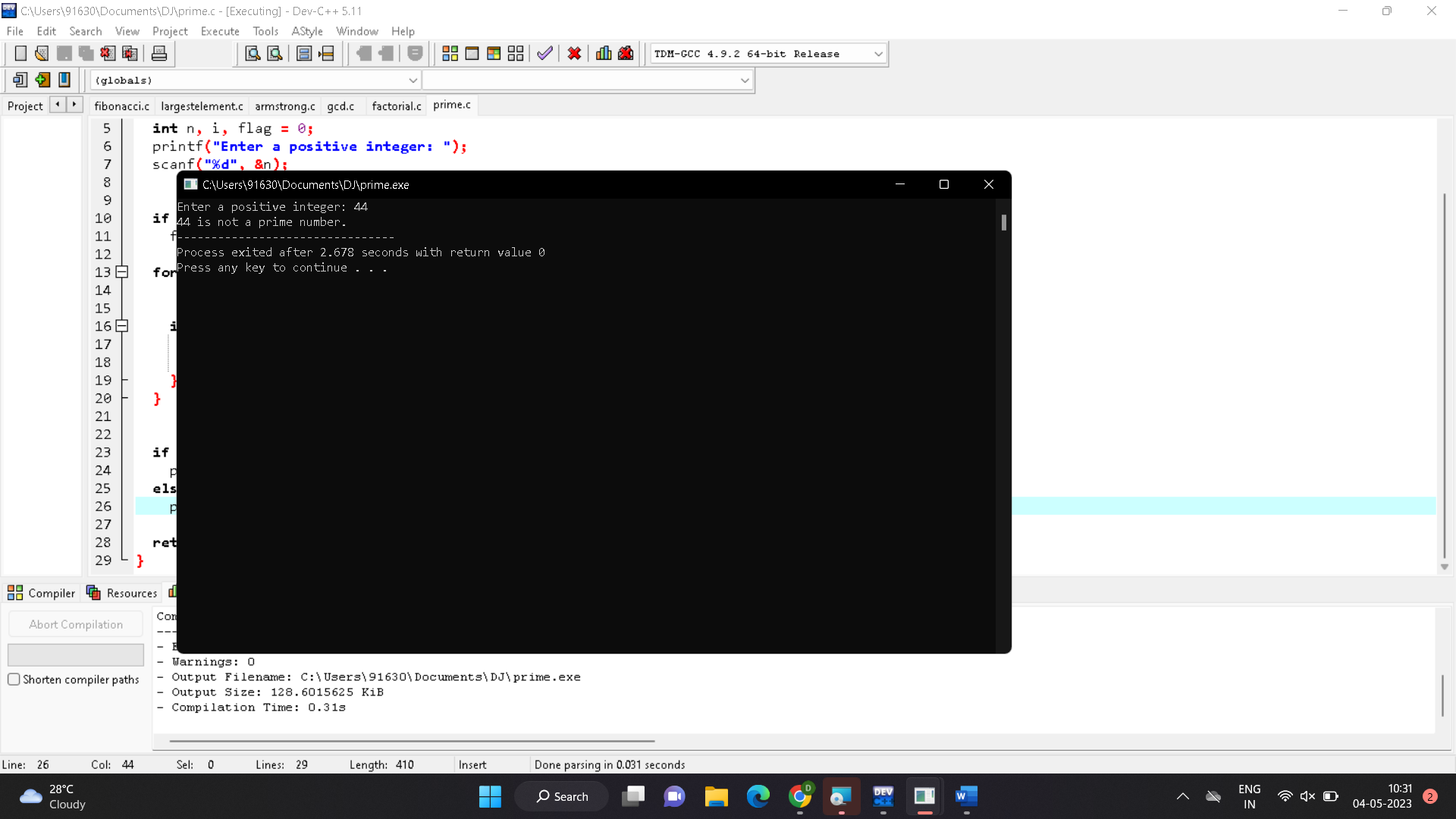
else

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

return 0;

}

Output:



1. Selection sort.

#include <stdio.h>

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

int i, j, minIndex, temp;

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

// Find the index of the smallest element in the unsorted portion of the array

minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

// Swap the smallest element with the first element of the unsorted portion of the array

temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

int main() {

int arr[] = {64, 25, 12, 22, 11};

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

int i;

printf("Original array: ");

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

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

}

selectionSort(arr, n);

printf("\nSorted array: ");

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

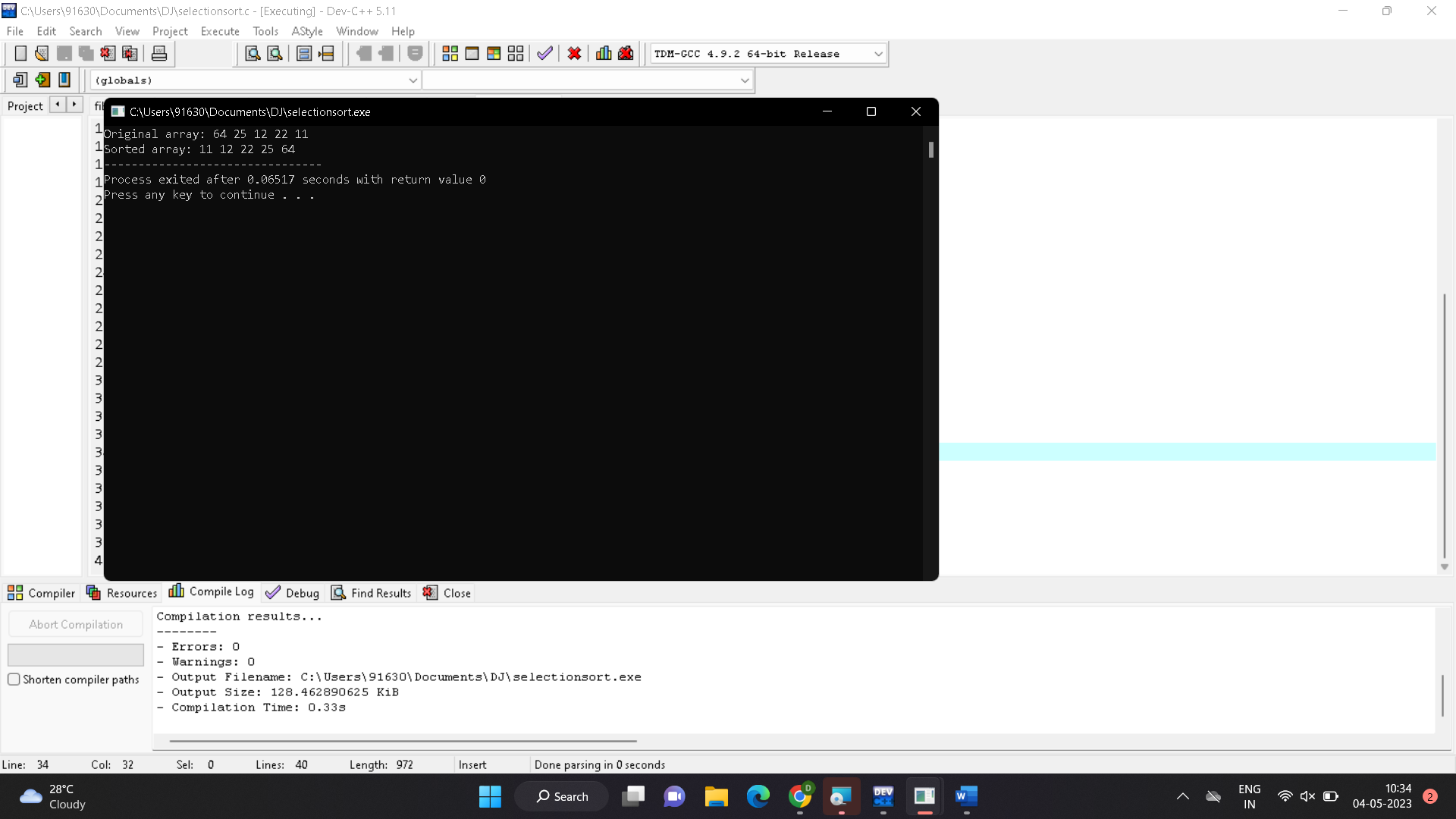
printf("%d ", arr[i]);

}

return 0;

}

Out put:



1. Bubble sort.

#include <stdio.h>

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

int i, j, temp;

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

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

if (arr[j] > arr[j + 1]) {

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

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

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

int i;

printf("Original array: ");

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

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

}

bubbleSort(arr, n);

printf("\nSorted array: ");

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

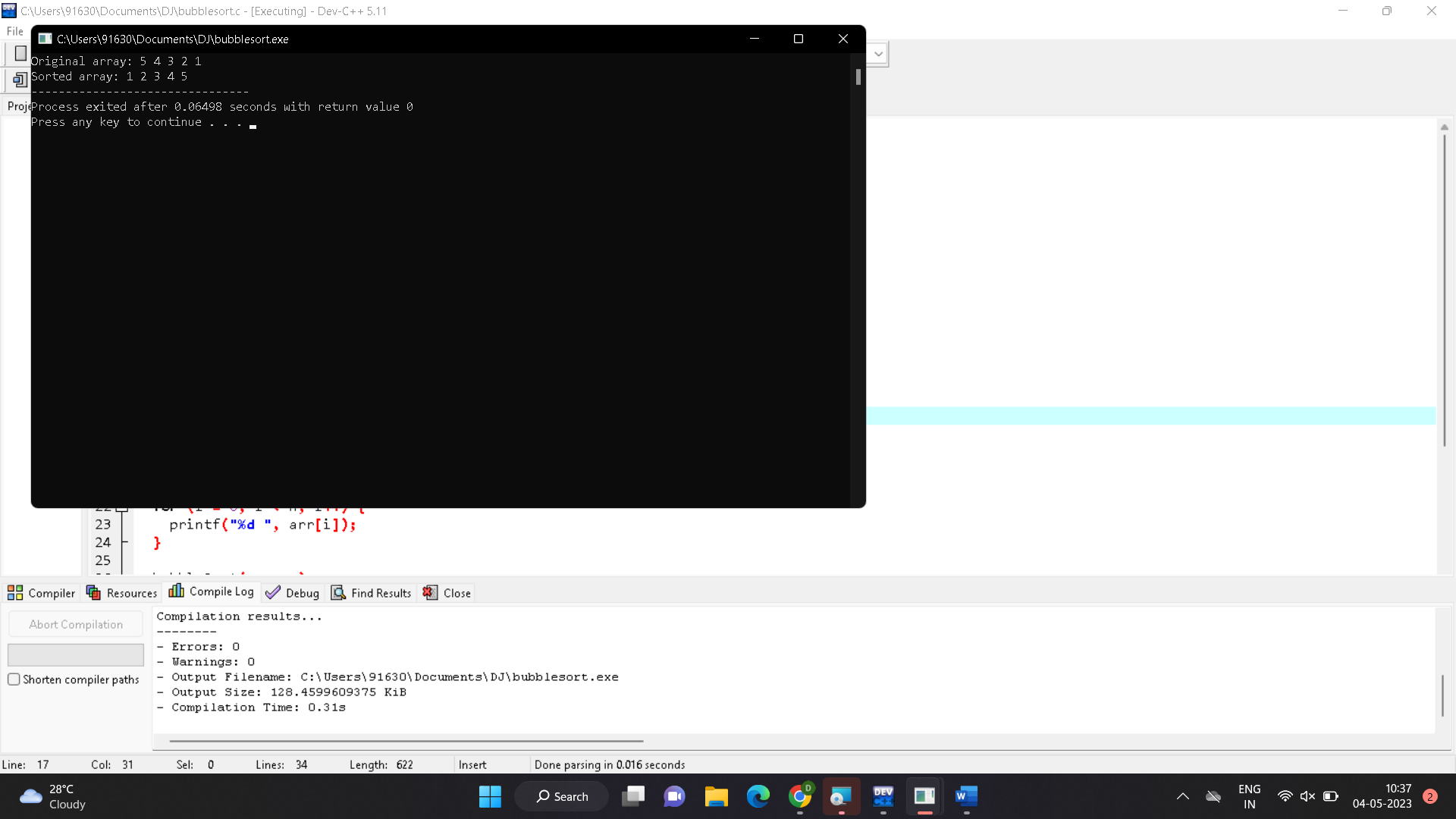
printf("%d ", arr[i]);

}

return 0;

}

Output :



1. Multipy two matrices.

#include <stdio.h>

#define MAX\_ROWS 100

#define MAX\_COLS 100

int main() {

int mat1[MAX\_ROWS][MAX\_COLS], mat2[MAX\_ROWS][MAX\_COLS], result[MAX\_ROWS][MAX\_COLS];

int rows1, cols1, rows2, cols2, i, j, k;

printf("Enter the number of rows and columns of the first matrix: ");

scanf("%d%d", &rows1, &cols1);

printf("Enter the elements of the first matrix:\n");

for (i = 0; i < rows1; i++) {

for (j = 0; j < cols1; j++) {

scanf("%d", &mat1[i][j]);

}

}

printf("Enter the number of rows and columns of the second matrix: ");

scanf("%d%d", &rows2, &cols2);

printf("Enter the elements of the second matrix:\n");

for (i = 0; i < rows2; i++) {

for (j = 0; j < cols2; j++) {

scanf("%d", &mat2[i][j]);

}

}

if (cols1 != rows2) {

printf("Error: The matrices cannot be multiplied!\n");

return 0;

}

for (i = 0; i < rows1; i++) {

for (j = 0; j < cols2; j++) {

result[i][j] = 0;

for (k = 0; k < cols1; k++) {

result[i][j] += mat1[i][k] \* mat2[k][j];

}

}

}

printf("The result matrix is:\n");

for (i = 0; i < rows1; i++) {

for (j = 0; j < cols2; j++) {

printf("%d ", result[i][j]);

}

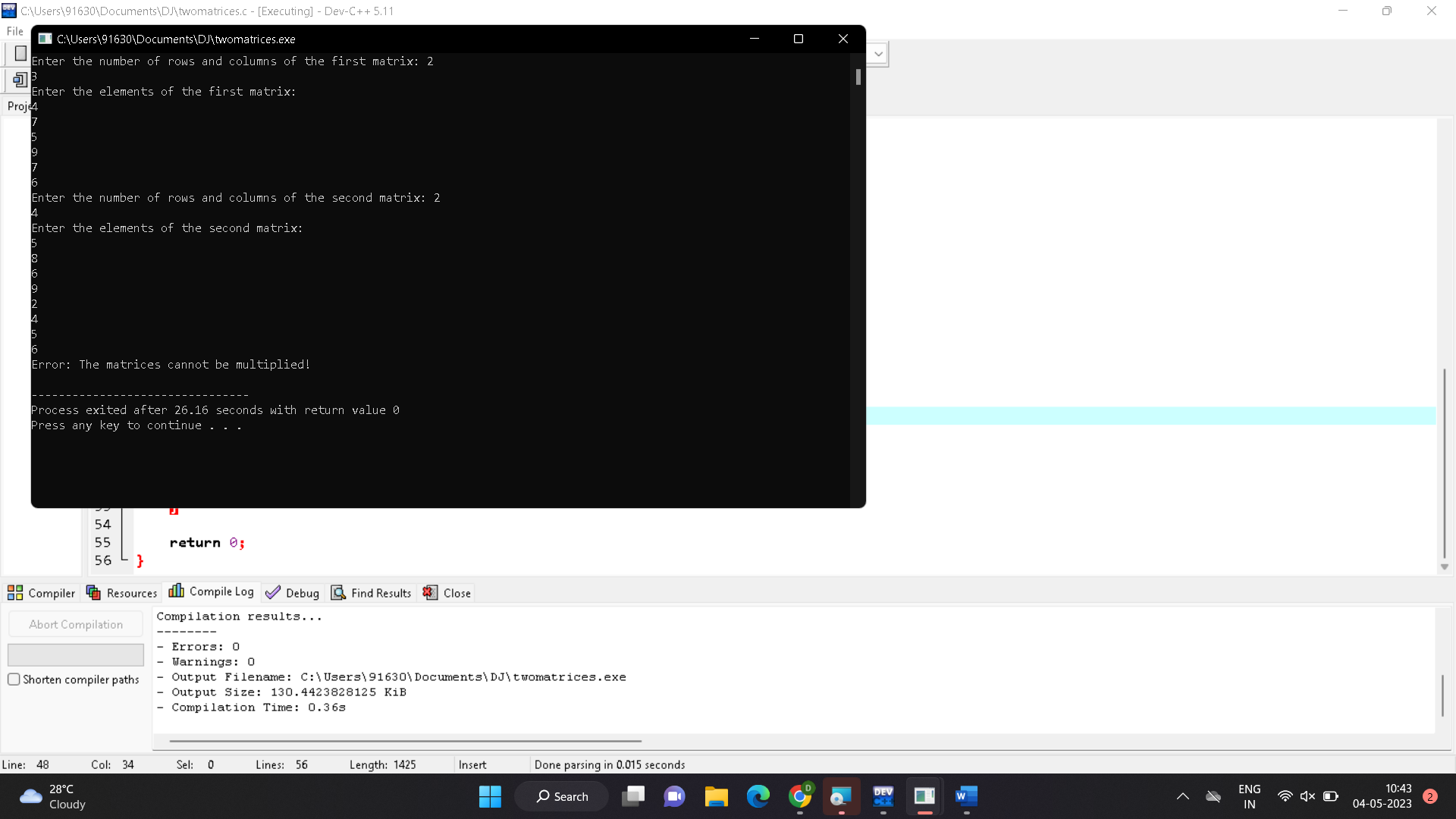
printf("\n");

}

return 0;

}

Output:



1. Palindrome or not.

#include <stdio.h>

#include <string.h>

int main()

{

char str[100];

int i, len, flag = 0;

printf("Enter a string: ");

scanf("%s", str);

len = strlen(str);

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

{

if(str[i] != str[len-i-1])

{

flag = 1;

break;

}

}

if(flag)

{

printf("%s is not a palindrome\n", str);

}

else

{

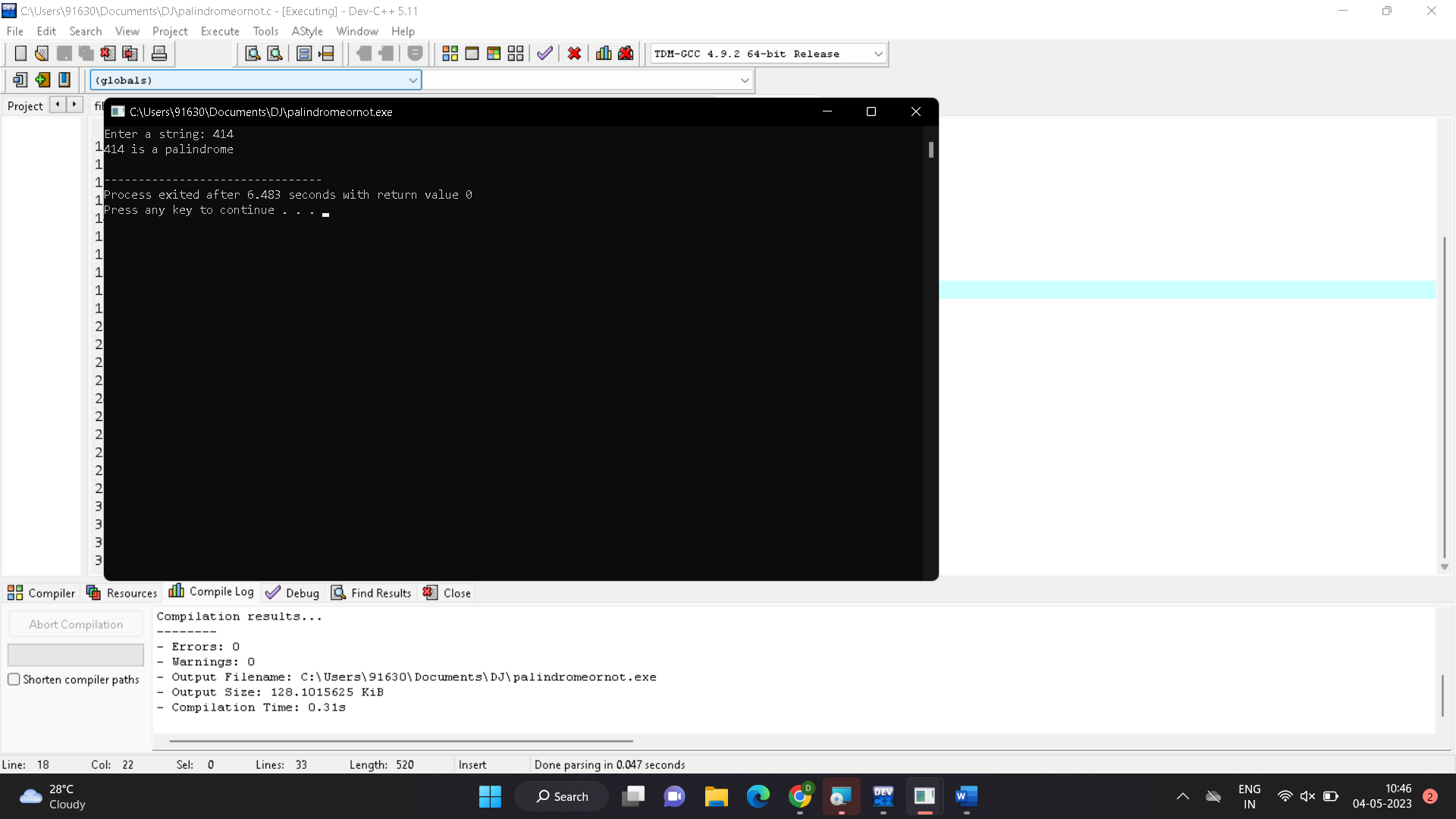
printf("%s is a palindrome\n", str);

}

return 0;

}

Output:



1. Copy one string to another.

#include <stdio.h>

#include <string.h>

int main() {

char source\_string[50], target\_string[50];

printf("Enter a string to copy: ");

fgets(source\_string, sizeof(source\_string), stdin);

strcpy(target\_string, source\_string);

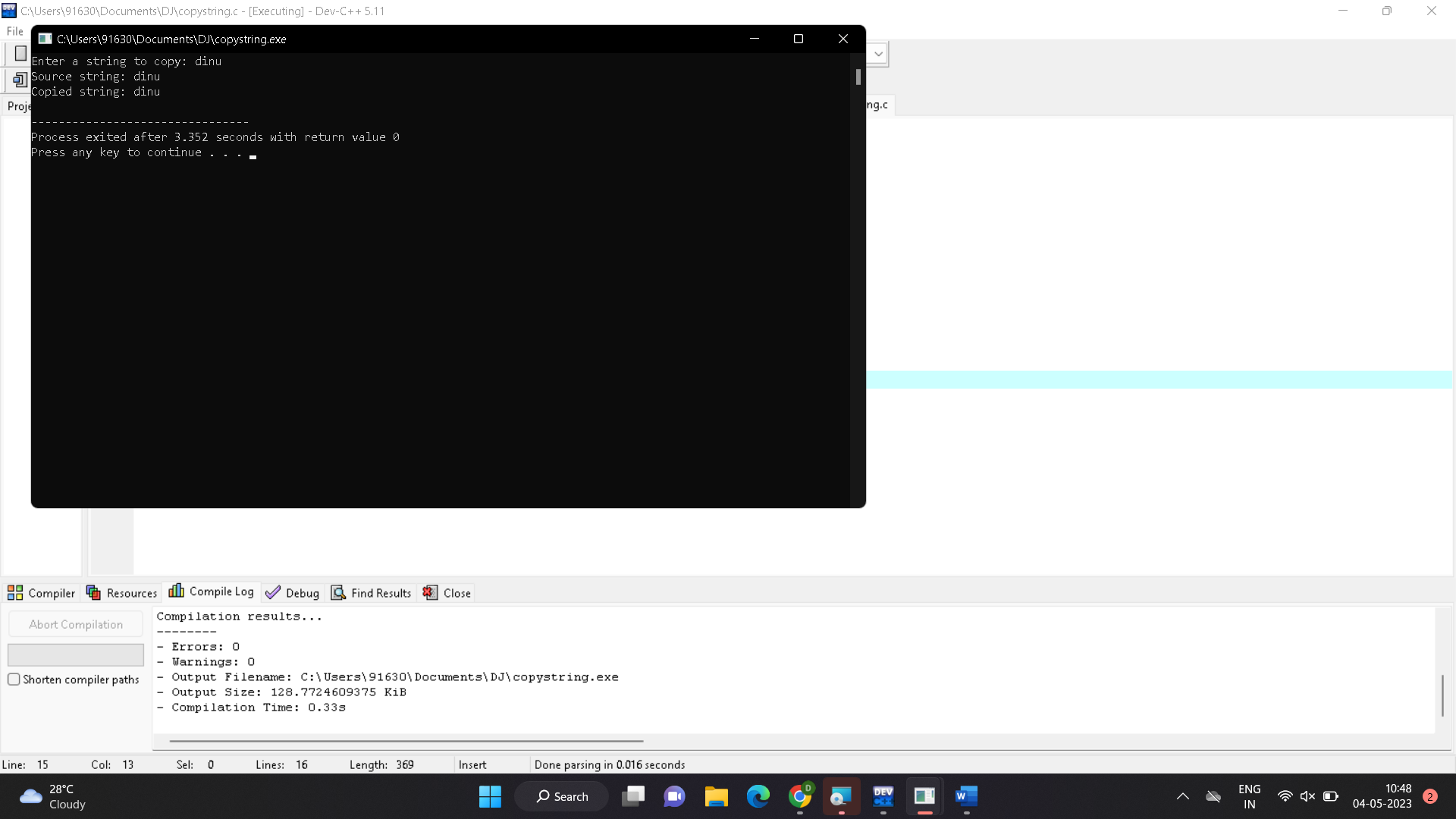
printf("Source string: %s", source\_string);

printf("Copied string: %s", target\_string);

return 0;

}

Output:



1. Binary search.

#include <stdio.h>

int binary\_search(int arr[], int left, int right, int x) {

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == x) {

return mid;

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

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

int main() {

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

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

int x = 5;

int result = binary\_search(arr, 0, n - 1, x);

if (result == -1) {

printf("Element is not present in array");

} else {

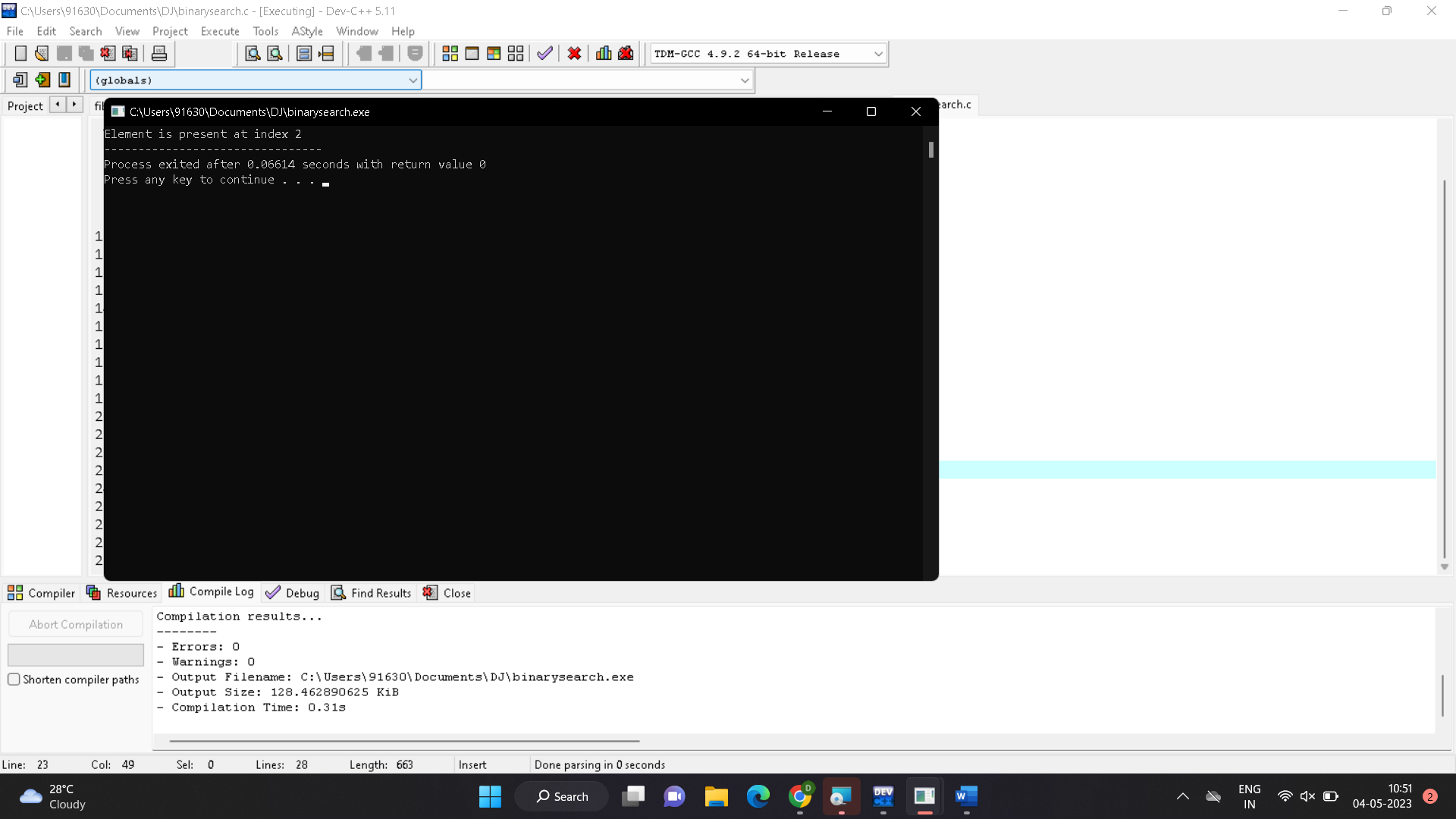
printf("Element is present at index %d", result);

}

return 0;

}

Output :



1. Reverse of a string.

#include <stdio.h>

#include <string.h>

int main() {

char str[100];

int i, len;

printf("Enter a string: ");

scanf("%s", str);

len = strlen(str);

printf("Reverse of the string: ");

for(i = len-1; i >= 0; i--) {

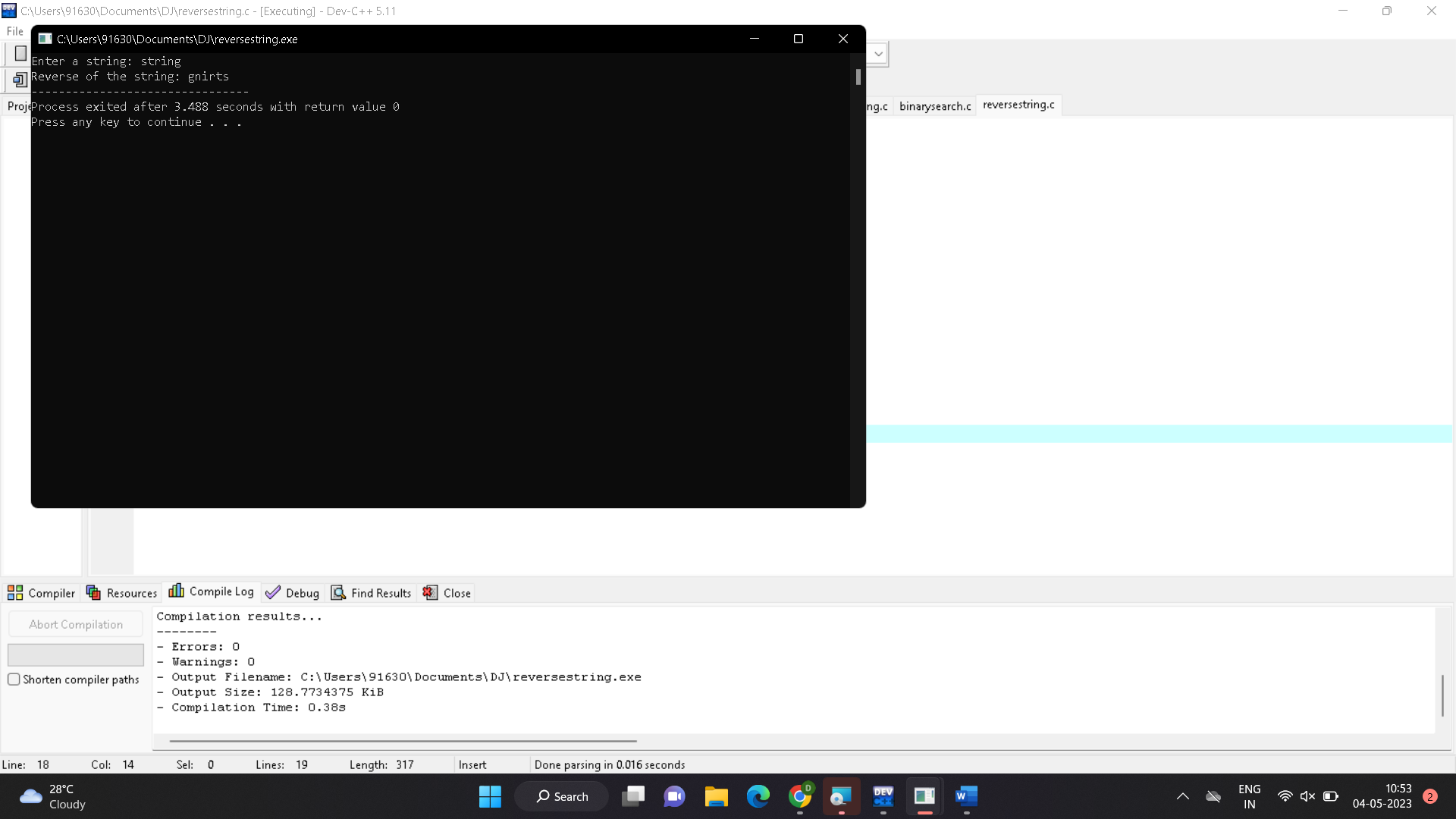
printf("%c", str[i]);

}

return 0;

}

Output:



1. Length of a string.

#include <stdio.h>

#include <string.h>

int main() {

char str[100];

int len;

printf("Enter a string: ");

scanf("%s", str);

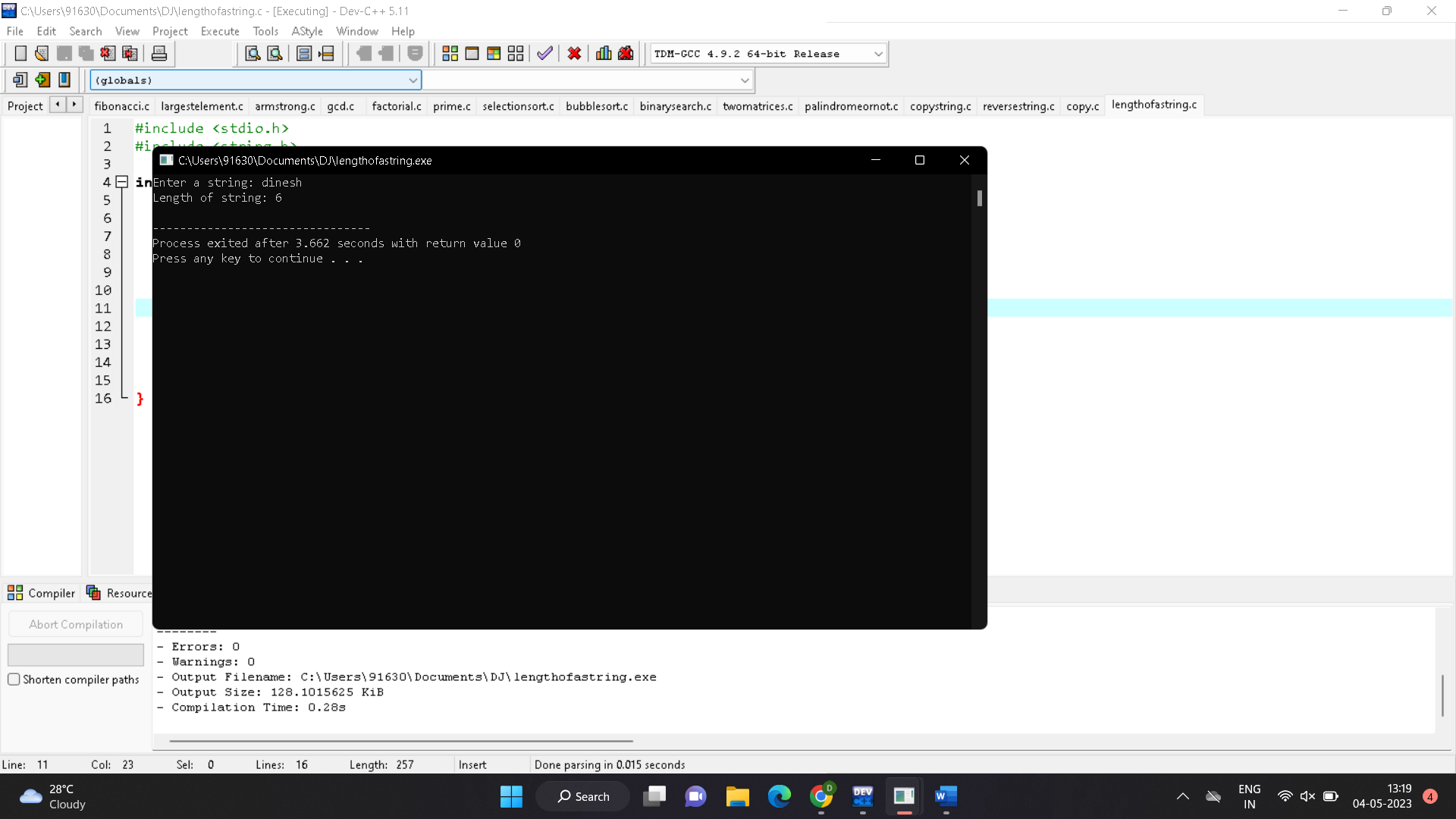
len = strlen(str);

printf("Length of string: %d\n", len);

return 0;

}

Output:

16. Merge sort.

#include <stdio.h>

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int arr[], int size)

{

int i;

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

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

printf("\n");

}

int main()

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

int arr\_size = sizeof(arr) / sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

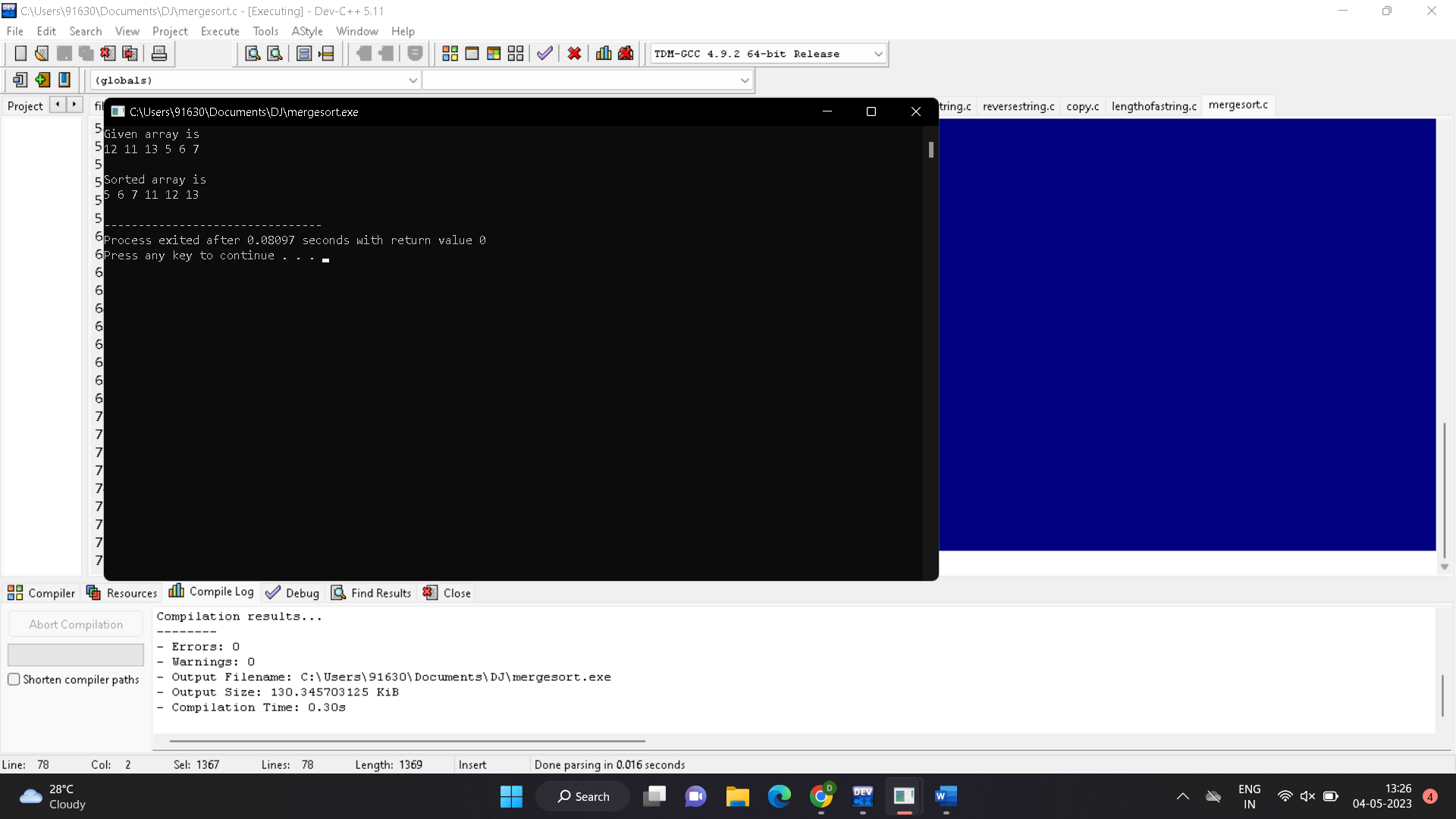
printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}

Output:



17. min&max.

#include <stdio.h>

void findMaxMin(int arr[], int low, int high, int \*max, int \*min) {

int mid, max1, max2, min1, min2;

if (low == high) {

\*max = arr[low];

\*min = arr[low];

return;

}

if (high == low + 1) {

if (arr[low] > arr[high]) {

\*max = arr[low];

\*min = arr[high];

} else {

\*max = arr[high];

\*min = arr[low];

}

return;

}

mid = (low + high) / 2;

findMaxMin(arr, low, mid, &max1, &min1);

findMaxMin(arr, mid + 1, high, &max2, &min2);

if (max1 > max2) {

\*max = max1;

} else {

\*max = max2;

}

if (min1 < min2) {

\*min = min1;

} else {

\*min = min2;

}

}

int main() {

int arr[] = {5, 2, 9, 7, 1, 8, 6};

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

int max, min;

findMaxMin(arr, 0, n - 1, &max, &min);

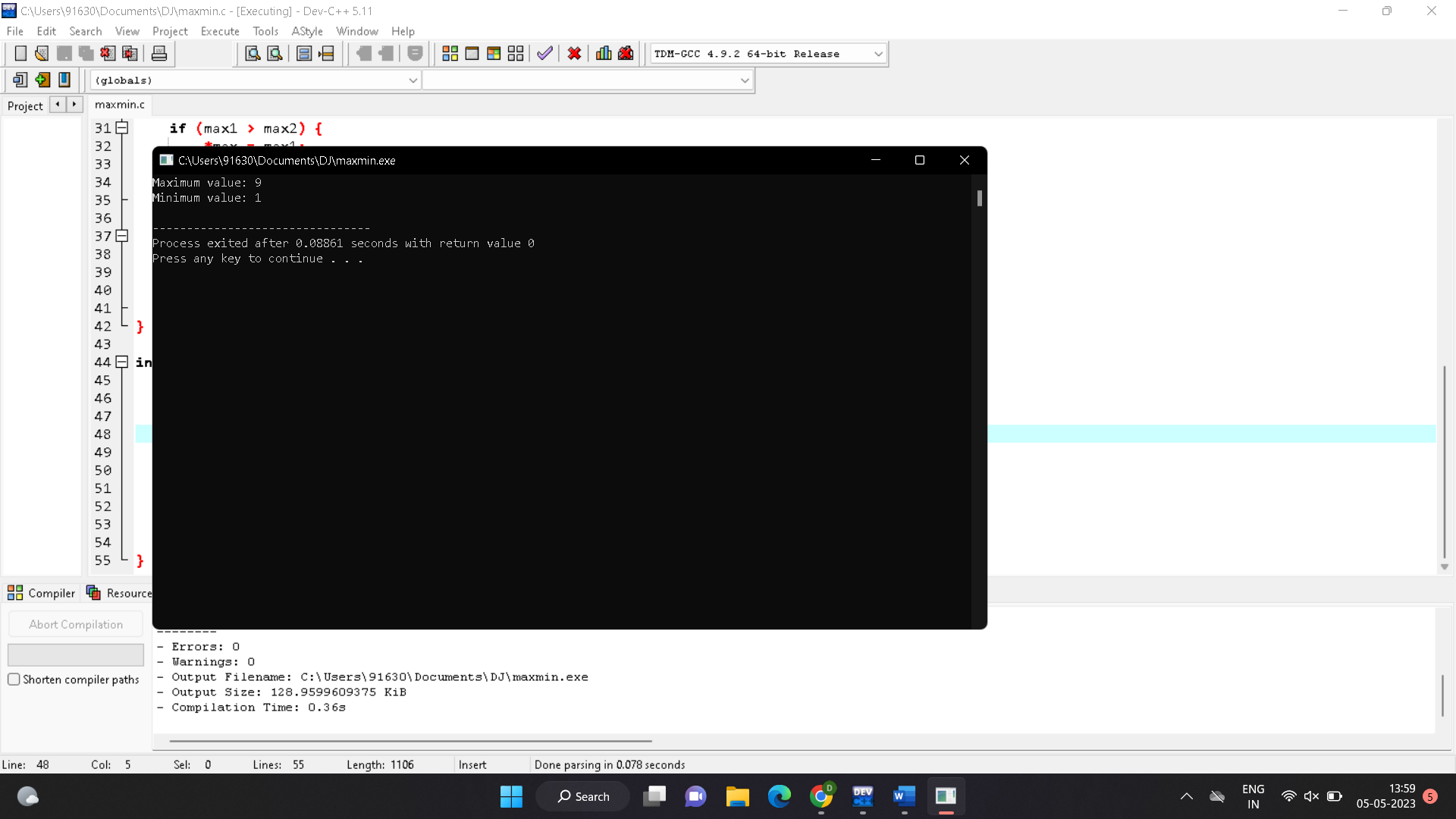
printf("Maximum value: %d\n", max);

printf("Minimum value: %d\n", min);

return 0;

}

Output:



18. to generate all prime numbers.

#include <stdio.h>

#include <stdbool.h>

void sieveOfEratosthenes(int n) {

bool prime[n+1];

int i, p;

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

prime[i] = true;

}

for (p = 2; p \* p <= n; p++) {

if (prime[p] == true) {

for (i = p \* 2; i <= n; i += p) {

prime[i] = false;

}

}

}

for (p = 2; p <= n; p++) {

if (prime[p] == true) {

printf("%d ", p);

}

}

}

int main() {

int n;

printf("Enter the upper bound of the range of numbers: ");

scanf("%d", &n);

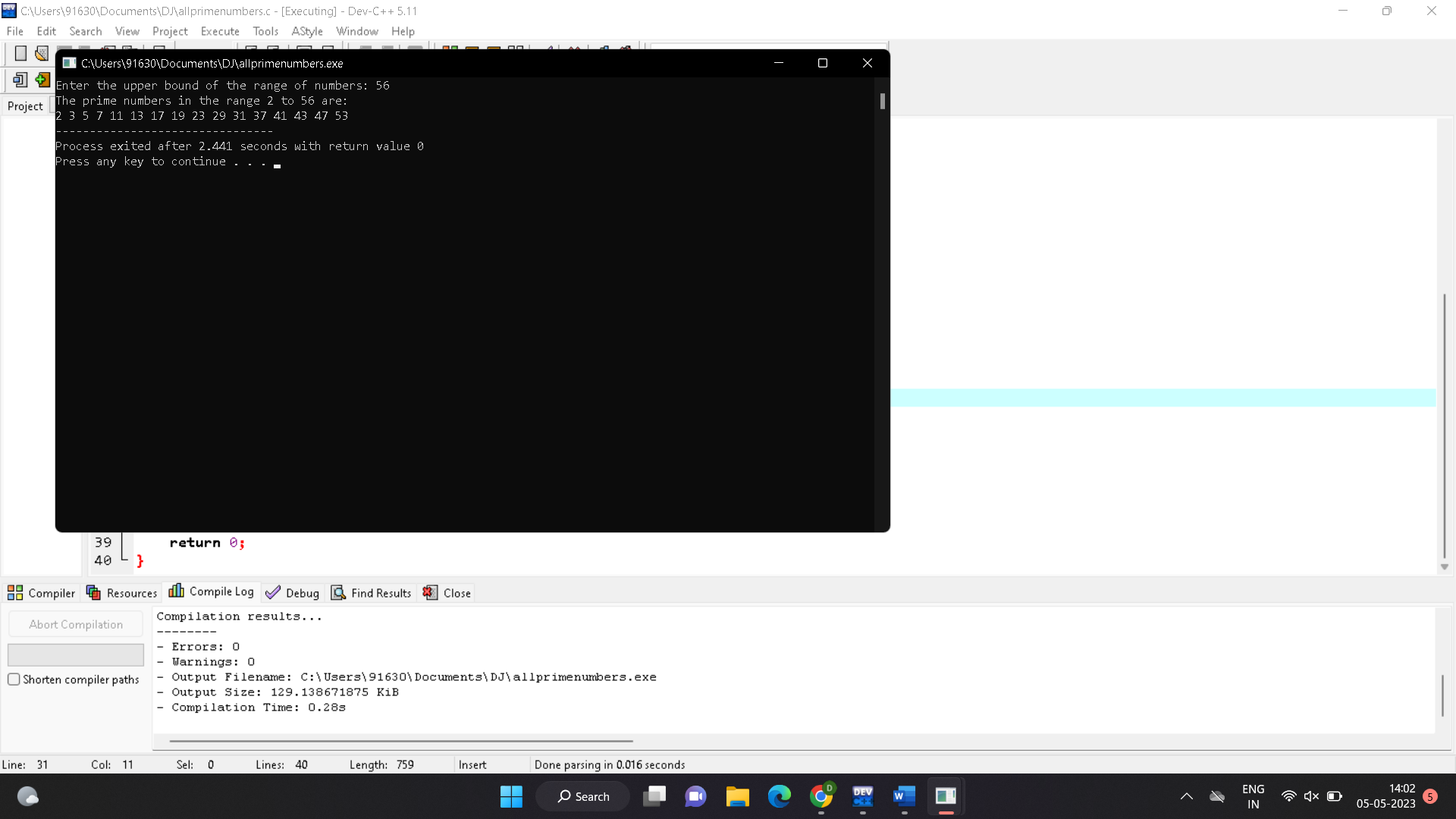
printf("The prime numbers in the range 2 to %d are:\n", n);

sieveOfEratosthenes(n);

return 0;

}

Output:



19. knapsack problem using greedy techniques.

# include<stdio.h>

void knapsack(int n, float weight[], float profit[], float capacity) {

float x[20], tp = 0;

int i, j, u;

u = capacity;

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

x[i] = 0.0;

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

if (weight[i] > u)

break;

else {

x[i] = 1.0;

tp = tp + profit[i];

u = u - weight[i];

}

}

if (i < n)

x[i] = u / weight[i];

tp = tp + (x[i] \* profit[i]);

printf("\nThe result vector is:- ");

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

printf("%f\t", x[i]);

printf("\nMaximum profit is:- %f", tp);

}

int main() {

float weight[20], profit[20], capacity;

int num, i, j;

float ratio[20], temp;

printf("\nEnter the no. of objects:- ");

scanf("%d", &num);

printf("\nEnter the wts and profits of each object:- ");

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

scanf("%f %f", &weight[i], &profit[i]);

}

printf("\nEnter the capacityacity of knapsack:- ");

scanf("%f", &capacity);

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

ratio[i] = profit[i] / weight[i];

}

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

for (j = i + 1; j < num; j++) {

if (ratio[i] < ratio[j]) {

temp = ratio[j];

ratio[j] = ratio[i];

ratio[i] = temp;

temp = weight[j];

weight[j] = weight[i];

weight[i] = temp;

temp = profit[j];

profit[j] = profit[i];

profit[i] = temp;

}

}

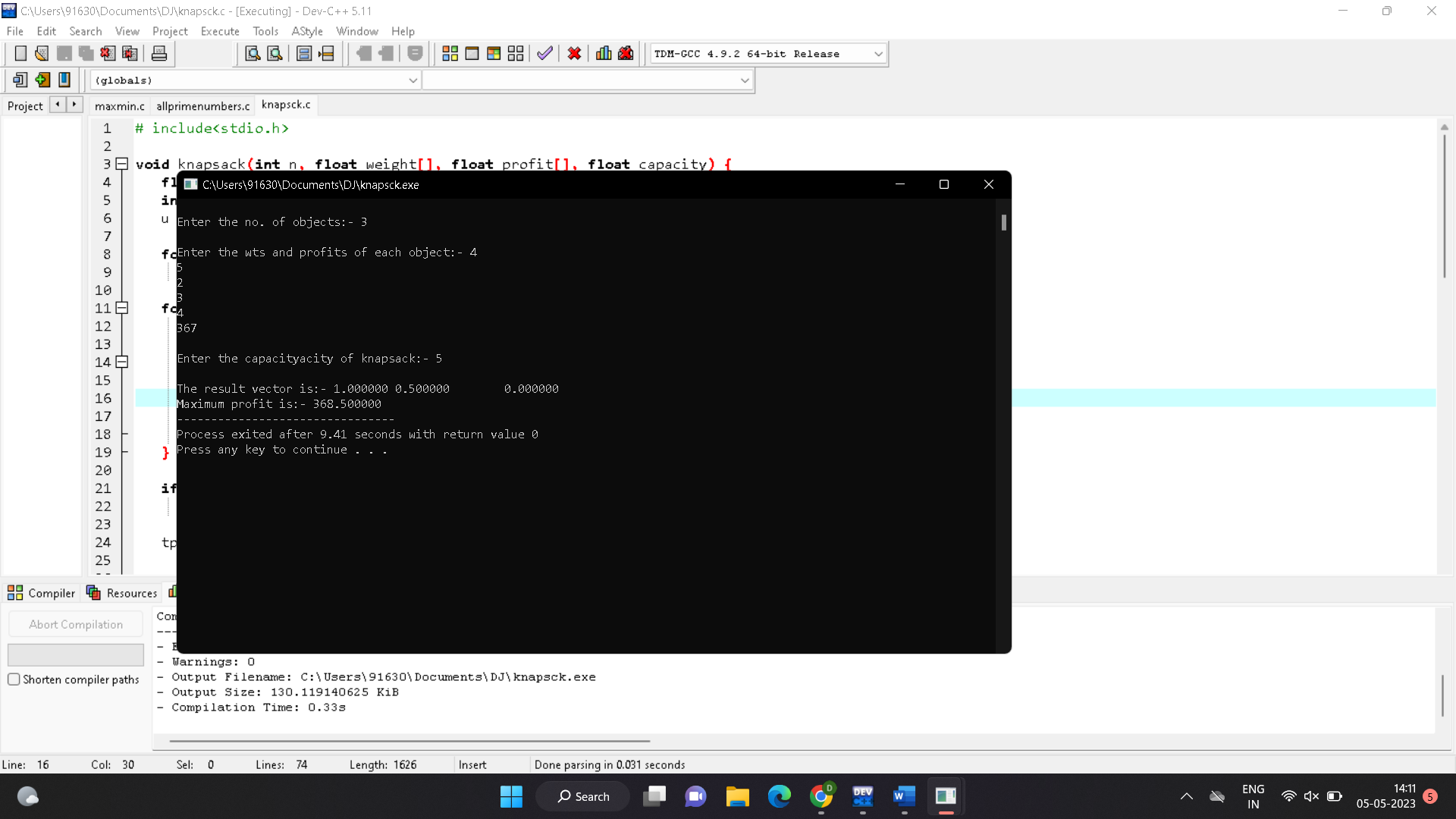
}

knapsack(num, weight, profit, capacity);

return(0);

}

Output:



20. MST using greed techniques.

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define V 5

int findMinKey(int key[], int mstSet[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++) {

if (mstSet[v] == 0 && key[v] < min) {

min = key[v];

min\_index = v;

}

}

return min\_index;

}

void printMST(int parent[], int graph[V][V]) {

printf("Edge \tWeight\n");

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

printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);

}

}

void primMST(int graph[V][V]) {

int parent[V];

int key[V];

int mstSet[V];

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

key[i] = INT\_MAX;

mstSet[i] = 0;

}

key[0] = 0;

parent[0] = -1;

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

int u = findMinKey(key, mstSet);

mstSet[u] = 1;

for (int v = 0; v < V; v++) {

if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

}

printMST(parent, graph);

}

int main() {

int graph[V][V] = {{0, 2, 0, 6, 0},

{2, 0, 3, 8, 5},

{0, 3, 0, 0, 7},

{6, 8, 0, 0, 9},

{0, 5, 7, 9, 0}};

primMST(graph);

return 0;

}

Out put:

