**P.RUPASH**

**192221105**

**CSA0313**

**DATA STRUCTURE**

**1.Write a C program to perform Matrix Multiplication**

#include <stdio.h>

int main() {

int first[10][10], second[10][10], result[10][10];

int r1, c1, r2, c2, i, j, k;

printf("Enter rows and columns for first matrix: ");

scanf("%d %d", &r1, &c1);

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

for(i = 0; i < r1; i++)

for(j = 0; j < c1; j++)

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

printf("Enter rows and columns for second matrix: ");

scanf("%d %d", &r2, &c2);

if(c1 != r2) {

printf("Multiplication not possible!\n");

return 1;

}

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

for(i = 0; i < r2; i++)

for(j = 0; j < c2; j++)

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

for(i = 0; i < r1; i++) {

for(j = 0; j < c2; j++) {

result[i][j] = 0;

for(k = 0; k < c1; k++)

result[i][j] += first[i][k] \* second[k][j];

}

}

printf("Result of multiplication:\n");

for(i = 0; i < r1; i++) {

for(j = 0; j < c2; j++)

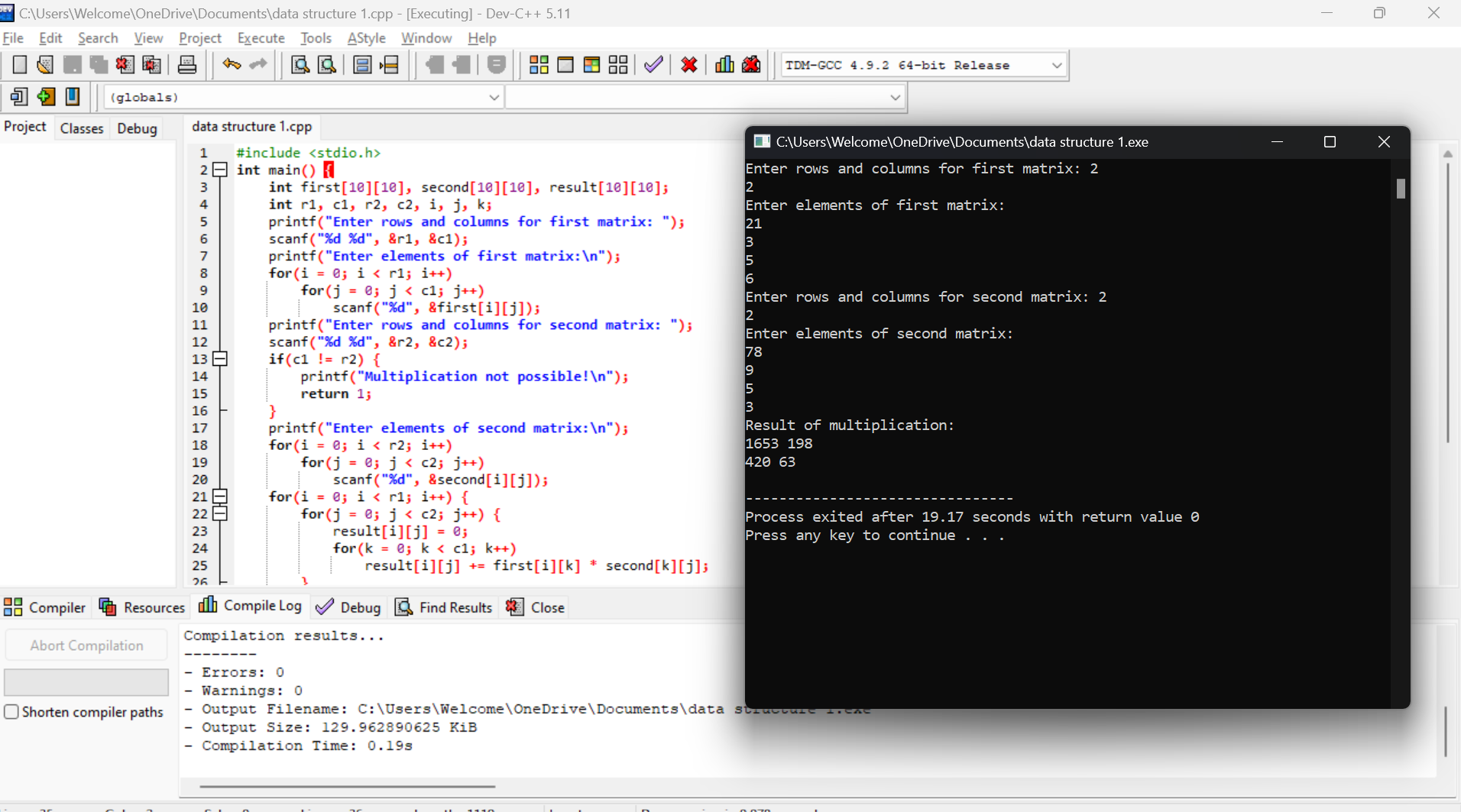
printf("%d ", result[i][j]);

printf("\n");

}

return 0;

}



**2. Write a C program to find Odd or Even number from a given set of numbers**

#include<stdio.h>

int main(){

int num;

printf("enter the number:");

scanf("%d",&num);

if (num % 2==0)

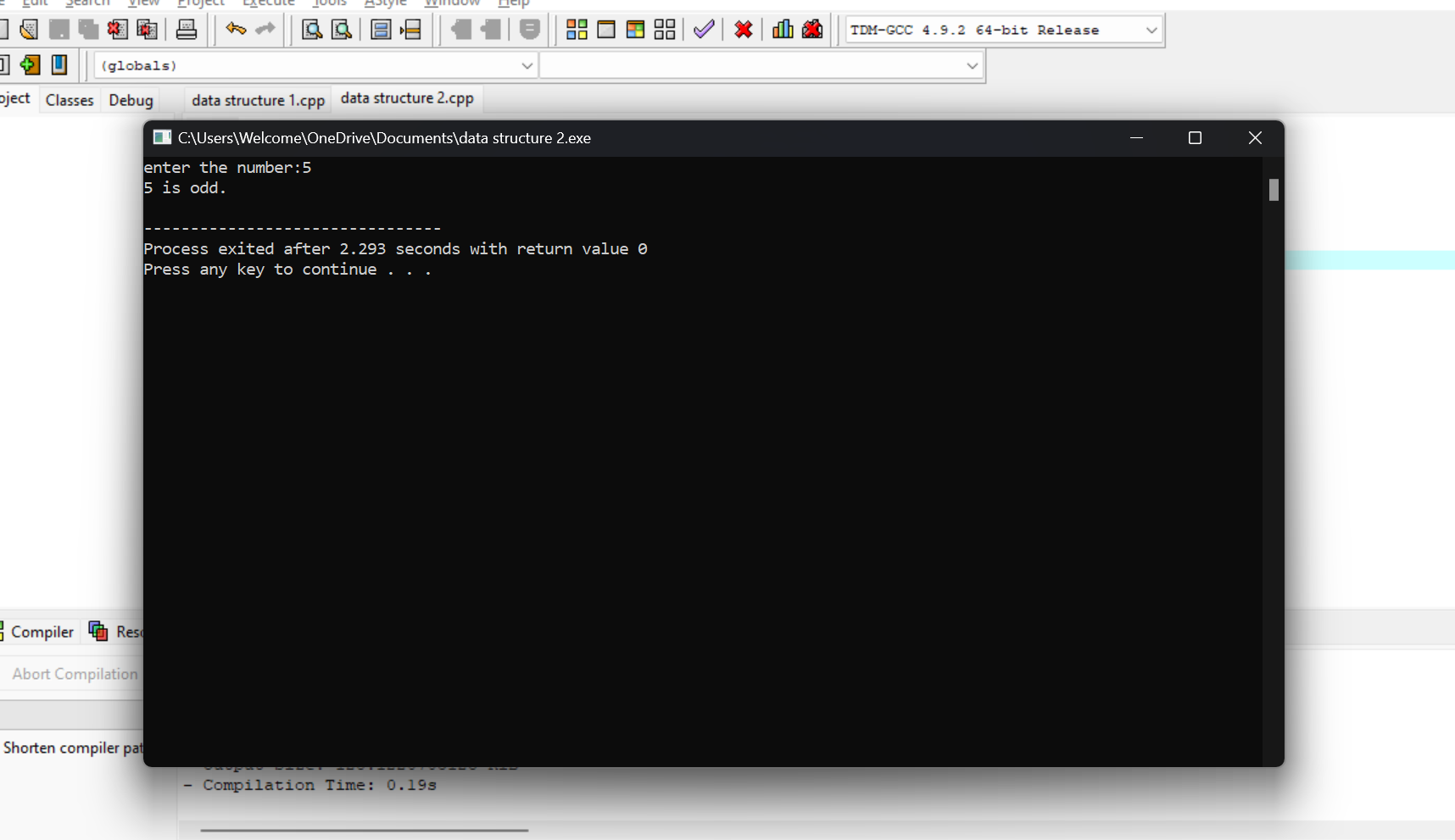
printf("%d is even.\n",num);

else

printf("%d is odd.\n",num);

return 0;

}



**3. Write a C program to find Factorial of a given number without using Recursion**

#include <stdio.h>

int main() {

int num, fact = 1;

printf("Enter a number: ");

scanf("%d", &num);

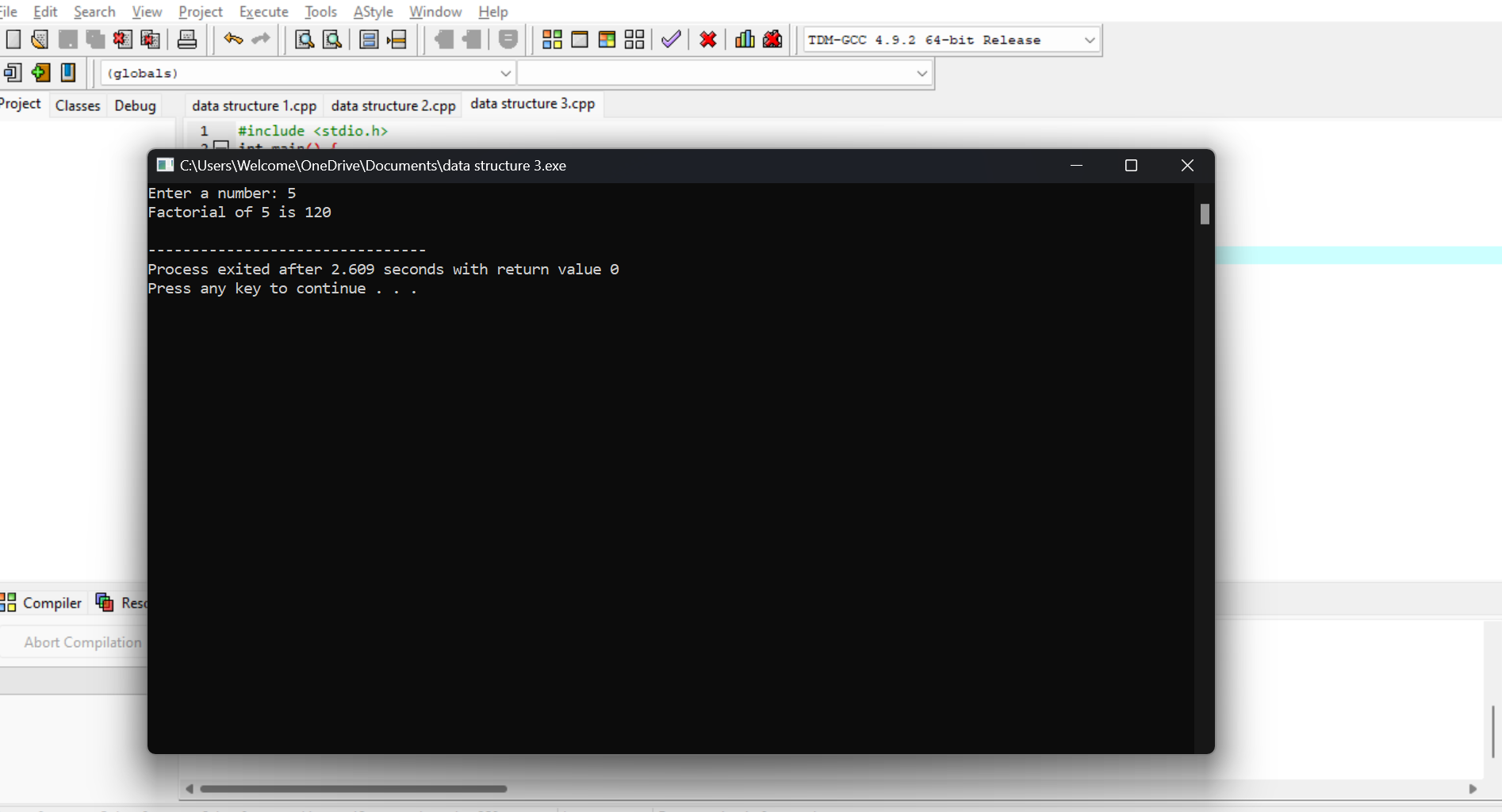
for(int i = 1; i <= num; i++) {

fact \*= i;

}

printf("Factorial of %d is %d\n", num, fact);

return 0;

}

**4. Write a C program to find Fibonacci series without using Recursion**

**#include <stdio.h>**

**int main() {**

**int n, t1 = 0, t2 = 1, nextTerm;**

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

**scanf("%d", &n);**

**printf("Fibonacci Series: ");**

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

**printf("%d, ", t1);**

**nextTerm = t1 + t2;**

**t1 = t2;**

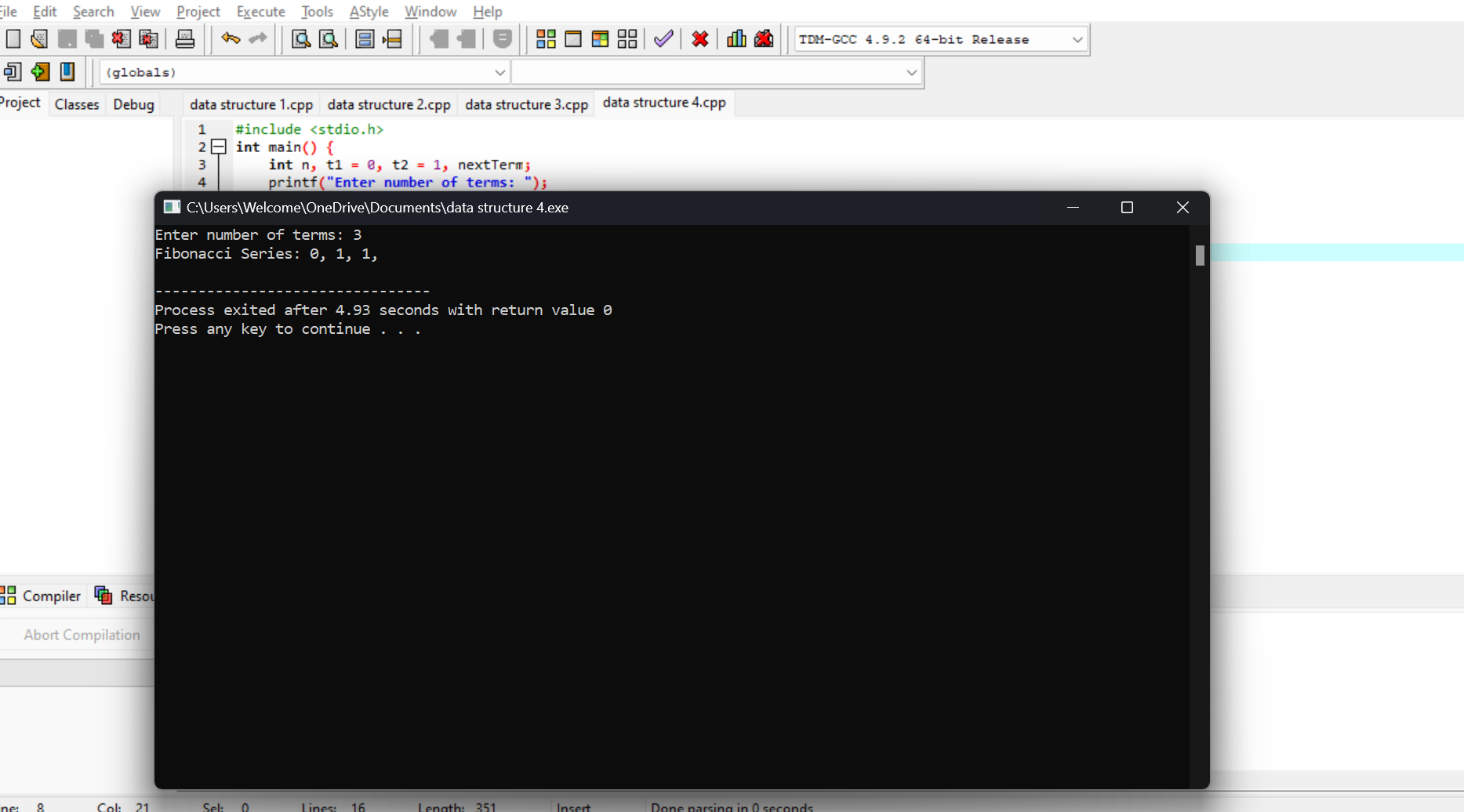
**t2 = nextTerm;**

**}**

**printf("\n");**

**return 0;**

**}**

****

**5. Write a C program to find Factorial of a given number using Recursion**

**#include <stdio.h>**

**int factorial(int n) {**

**if (n == 0)**

**return 1;**

**else**

**return n \* factorial(n - 1);**

**}**

**int main() {**

**int num;**

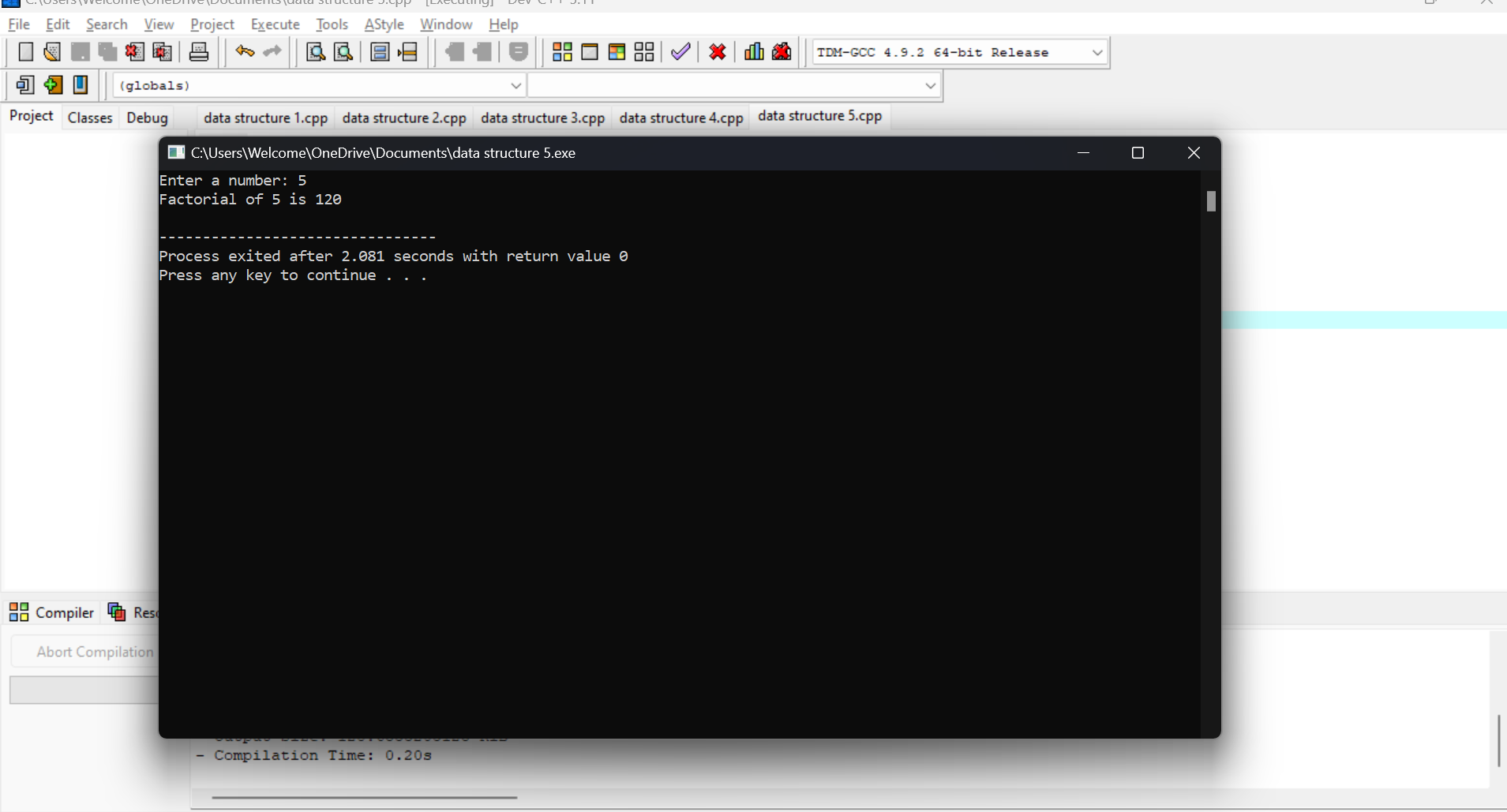
**printf("Enter a number: ");**

**scanf("%d", &num);**

**printf("Factorial of %d is %d\n", num, factorial(num));**

**return 0;**

**}**

****

**6. Write a C program to find Fibonacci series using Recursion**

**#include <stdio.h>**

**int fibonacci(int n) {**

**if (n <= 1)**

**return n;**

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

**}**

**int main() {**

**int n;**

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

**scanf("%d", &n);**

**printf("Fibonacci Series: ");**

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

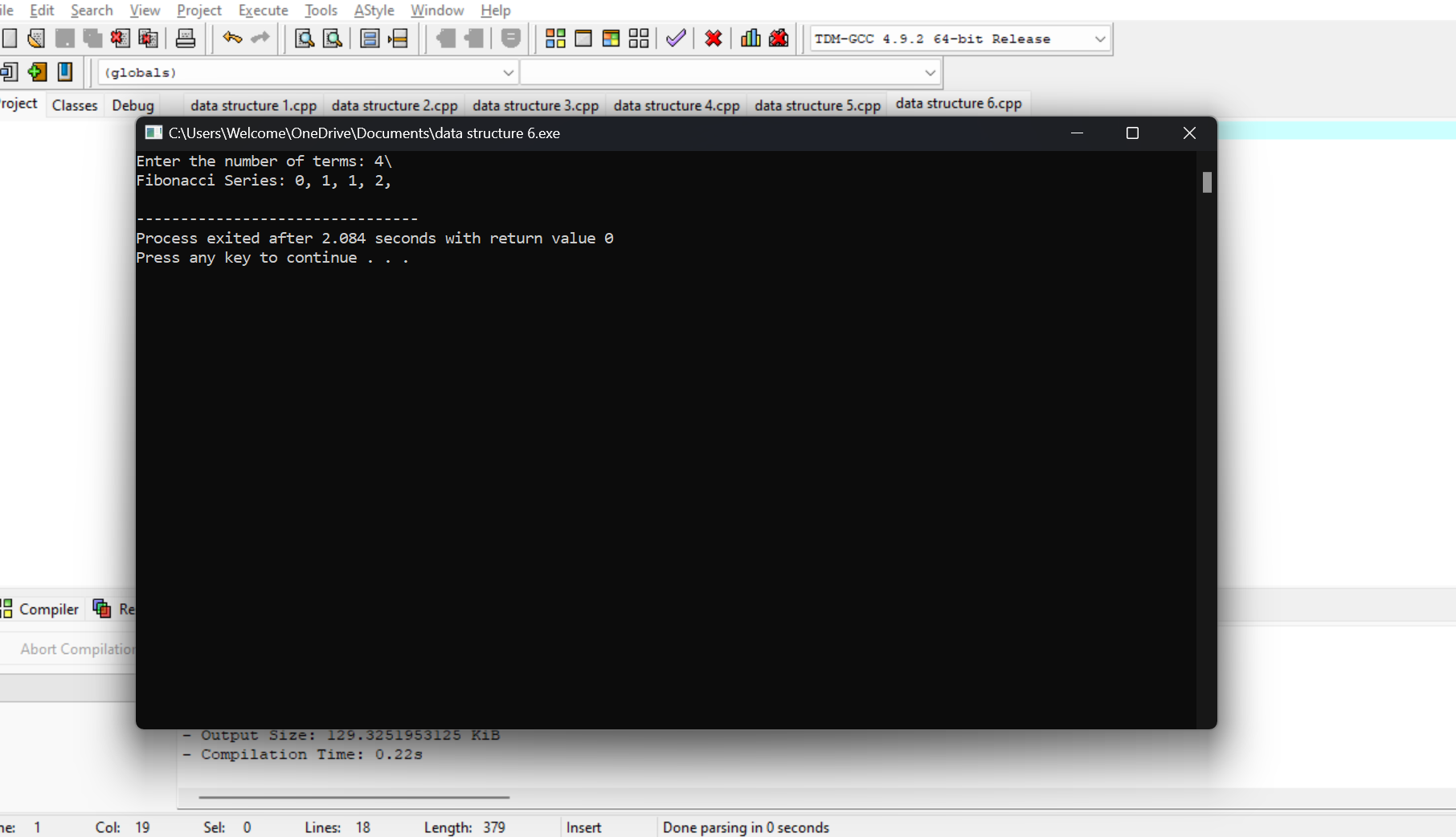
**printf("%d, ", fibonacci(i));**

**}**

**printf("\n");**

**return 0;**

**}**

****

**7. Write a C program to implement Array operations such as Insert, Delete and Display.**

**#include <stdio.h>**

**void insert(int arr[], int \*size, int value, int pos) {**

**for (int i = \*size; i > pos; i--) {**

**arr[i] = arr[i - 1];**

**}**

**arr[pos] = value;**

**(\*size)++;**

**}**

**void delete(int arr[], int \*size, int pos) {**

**for (int i = pos; i < \*size - 1; i++) {**

**arr[i] = arr[i + 1];**

**}**

**(\*size)--;**

**}**

**void display(int arr[], int size) {**

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

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

**}**

**printf("\n");**

**}**

**int main() {**

**int arr[100], size = 0, value, pos, choice;**

**while (1) {**

**printf("1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**printf("Enter value and position: ");**

**scanf("%d %d", &value, &pos);**

**insert(arr, &size, value, pos);**

**break;**

**case 2:**

**printf("Enter position to delete: ");**

**scanf("%d", &pos);**

**delete(arr, &size, pos);**

**break;**

**case 3:**

**display(arr, size);**

**break;**

**case 4:**

**return 0;**

**}**

**}**

**}**

**8,Write a C program to search a number using Linear Search method**

**#include <stdio.h>**

**int linearSearch(int arr[], int size, int key) {**

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

**if (arr[i] == key) {**

**return i;**

**}**

**}**

**return -1;**

**}**

**int main() {**

**int arr[] = {10, 20, 30, 40, 50};**

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

**int key = 40;**

**int result = linearSearch(arr, size, key);**

**if (result != -1) {**

**printf("Element found at index %d\n", result);**

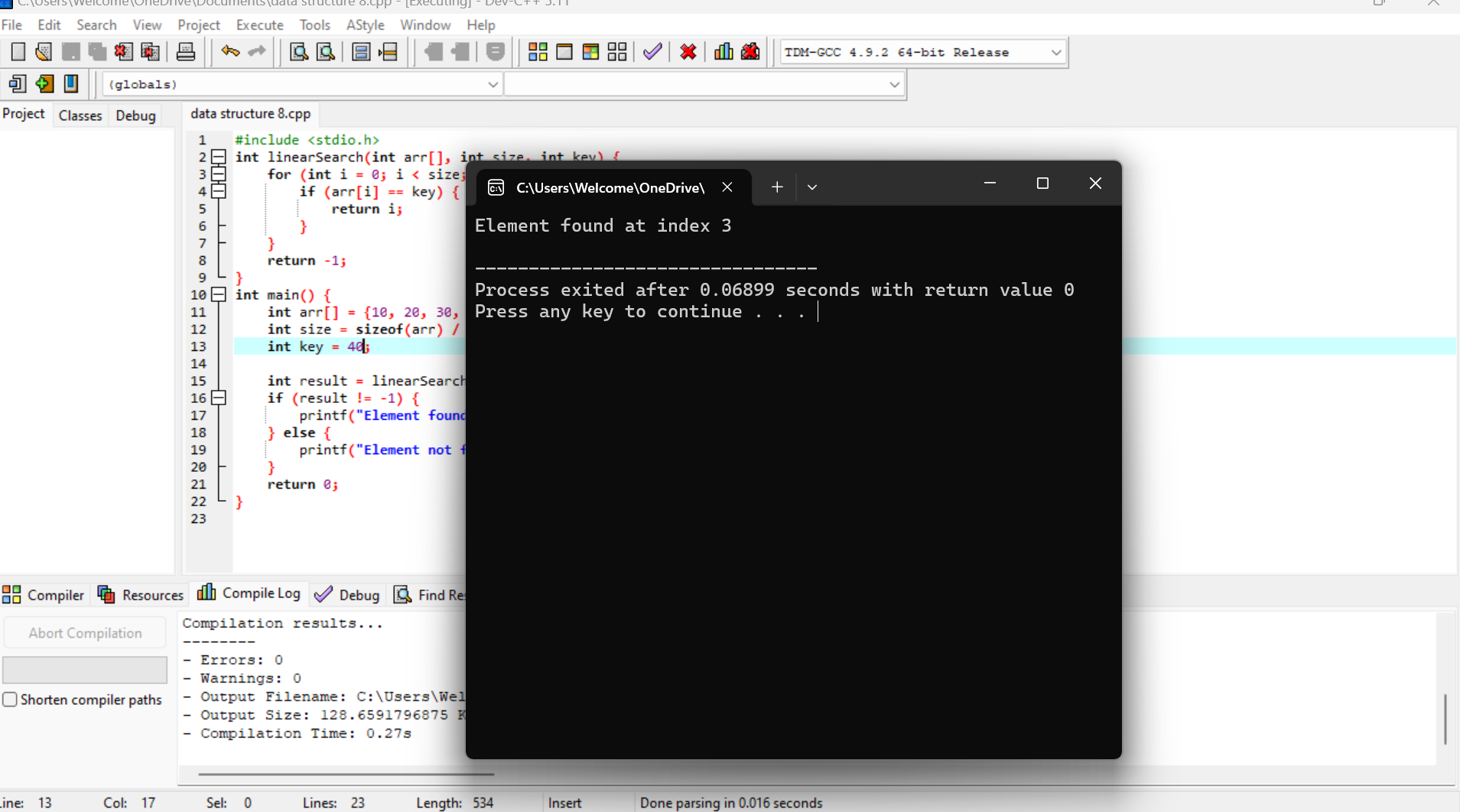
**} else {**

**printf("Element not found\n");**

**}**

**return 0;**

**}**

****

**9. Write a C program to search a number using Binary Search method**

**#include <stdio.h>**

**int binarySearch(int arr[], int size, int key) {**

**int left = 0, right = size - 1;**

**while (left <= right) {**

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

**if (arr[mid] == key)**

**return mid;**

**if (arr[mid] < key)**

**left = mid + 1;**

**else**

**right = mid - 1;**

**}**

**return -1;**

**}**

**int main() {**

**int arr[] = {10, 20, 30, 40, 50};**

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

**int key = 30;**

**int result = binarySearch(arr, size, key);**

**if (result != -1) {**

**printf("Element found at index %d\n", result);**

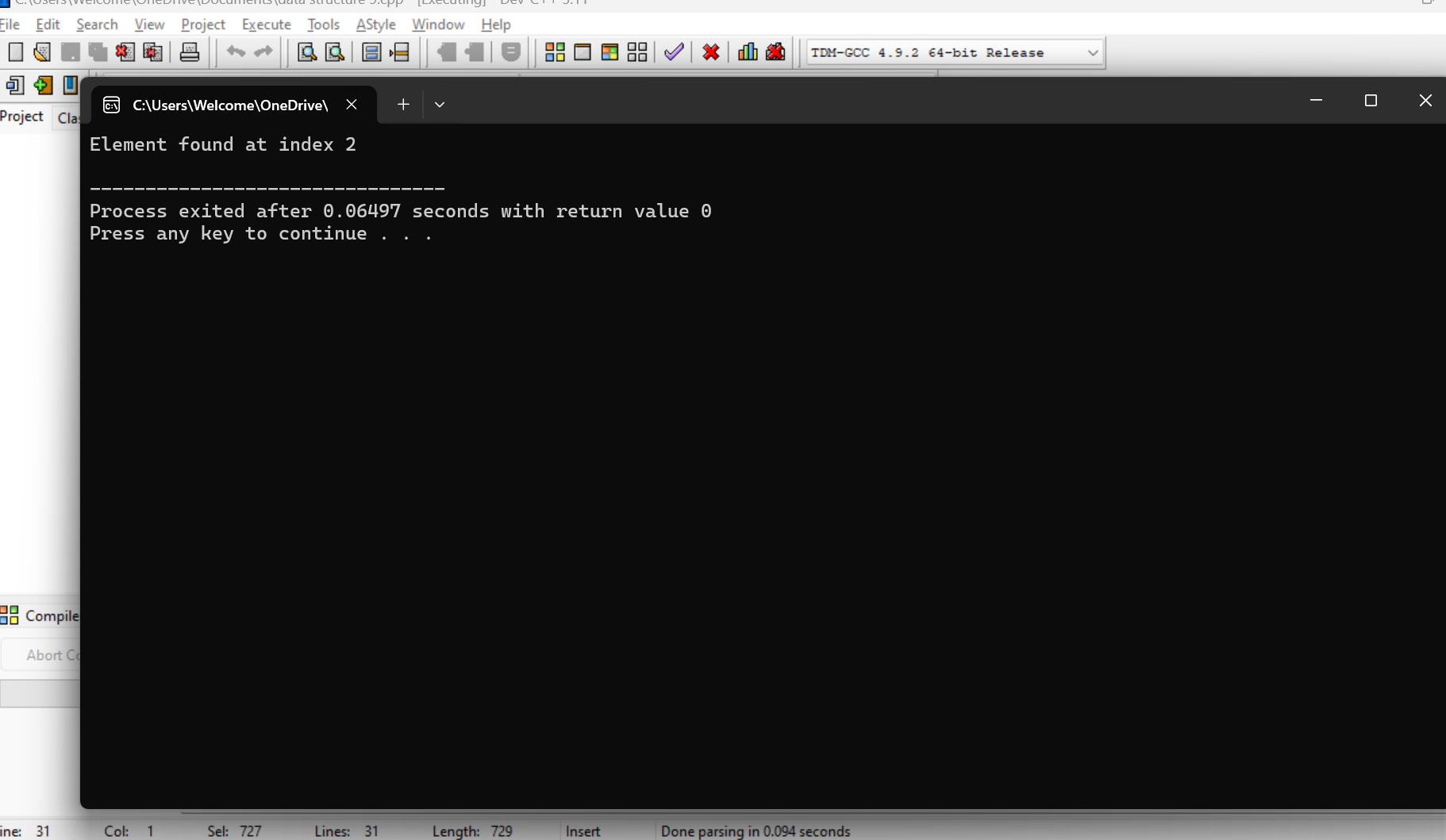
**} else {**

**printf("Element not found\n");**

**}**

**return 0;**

**}**

****

**10. Write a C program to implement Linked list operations**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node\* next;**

**};**

**struct Node\* createNode(int data) {**

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

**newNode->data = data;**

**newNode->next = NULL;**

**return newNode;**

**}**

**void insertAtEnd(struct Node\*\* head, int data) {**

**struct Node\* newNode = createNode(data);**

**if (\*head == NULL) {**

**\*head = newNode;**

**return;**

**}**

**struct Node\* temp = \*head;**

**while (temp->next != NULL) {**

**temp = temp->next;**

**}**

**temp->next = newNode;**

**}**

**void insertAtBeginning(struct Node\*\* head, int data) {**

**struct Node\* newNode = createNode(data);**

**newNode->next = \*head;**

**\*head = newNode;**

**}**

**void deleteNode(struct Node\*\* head, int key) {**

**struct Node\* temp = \*head;**

**struct Node\* prev = NULL;**

**if (temp != NULL && temp->data == key) {**

**\*head = temp->next;**

**free(temp);**

**return;**

**}**

**while (temp != NULL && temp->data != key) {**

**prev = temp;**

**temp = temp->next;**

**}**

**if (temp == NULL) {**

**printf("Node with value %d not found.\n", key);**

**return;**

**}**

**prev->next = temp->next;**

**free(temp);**

**}**

**void displayList(struct Node\* head) {**

**struct Node\* temp = head;**

**while (temp != NULL) {**

**printf("%d -> ", temp->data);**

**temp = temp->next;**

**}**

**printf("NULL\n");**

**}**

**void createInitialList(struct Node\*\* head) {**

**int n, value;**

**printf("Enter the number of elements in the linked list: ");**

**scanf("%d", &n);**

**int i;**

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

**printf("Enter element %d: ", i + 1);**

**scanf("%d", &value);**

**insertAtEnd(head, value);**

**}**

**printf("Initial linked list created.\n");**

**}**

**int main() {**

**struct Node\* head = NULL;**

**int choice, value;**

**createInitialList(&head);**

**while (1) {**

**printf("\n1. Insert at Beginning\n2. Insert at End\n3. Delete Node\n4. Display List\n5. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**printf("Enter value to insert at beginning: ");**

**scanf("%d", &value);**

**insertAtBeginning(&head, value);**

**break;**

**case 2:**

**printf("Enter value to insert at end: ");**

**scanf("%d", &value);**

**insertAtEnd(&head, value);**

**break;**

**case 3:**

**printf("Enter value to delete: ");**

**scanf("%d", &value);**

**deleteNode(&head, value);**

**break;**

**case 4:**

**printf("Linked List: ");**

**displayList(head);**

**break;**

**case 5:**

**exit(0);**

**default:**

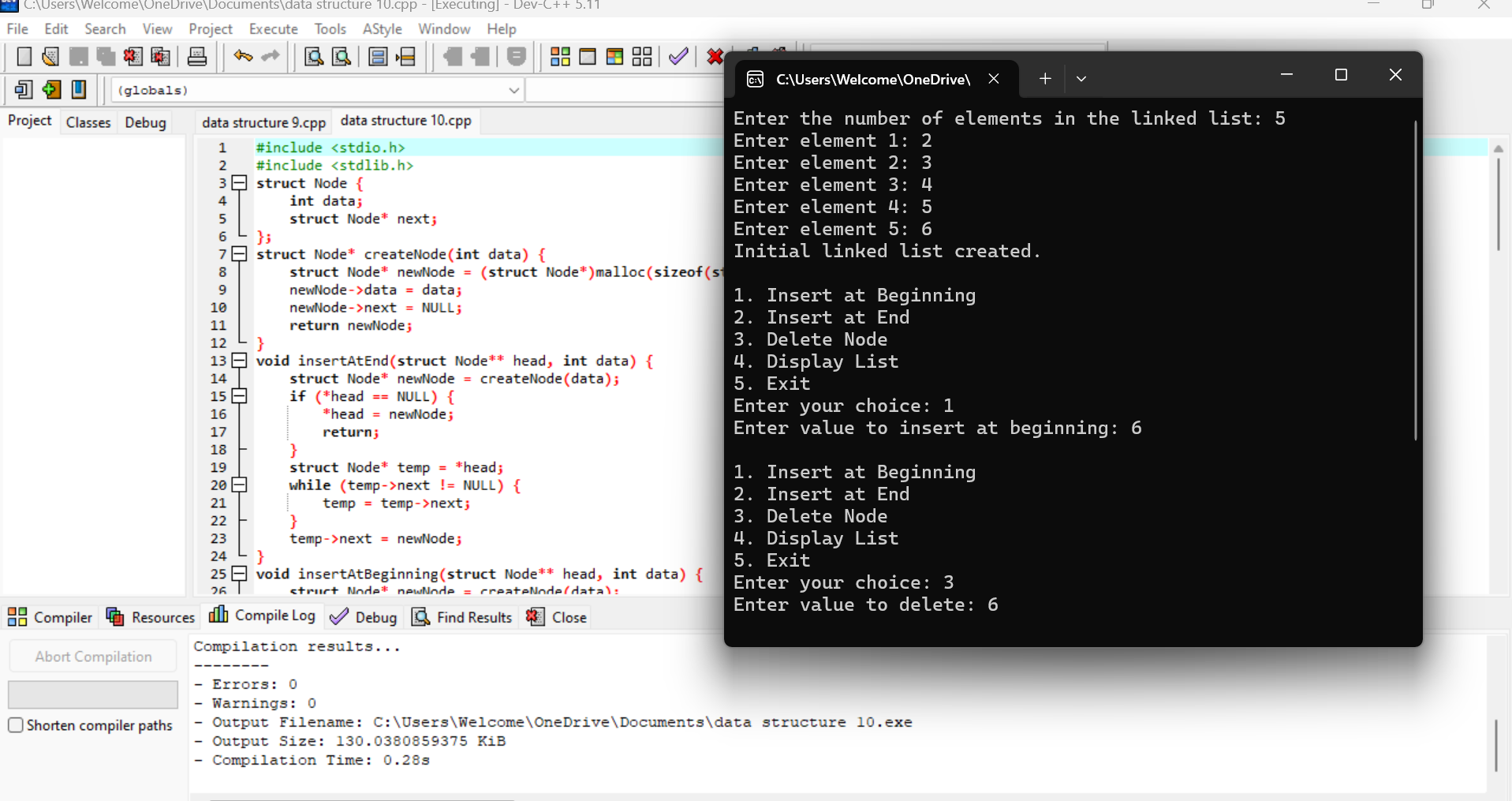
**printf("Invalid choice!\n");**

**}**

**}**

**return 0;**

**}**

****

**11. C Program to Implement Stack Operations (PUSH, POP, PEEK)**

**#include <stdio.h>**

**#define MAX 5**

**int stack[MAX];**

**int top = -1;**

**void push(int value) {**

**if (top < MAX - 1) {**

**stack[++top] = value;**

**printf("%d pushed to stack\n", value);**

**} else {**

**printf("Stack Overflow\n");**

**}**

**}**

**int pop() {**

**if (top >= 0) {**

**return stack[top--];**

**} else {**

**printf("Stack Underflow\n");**

**return -1;**

**}**

**}**

**int peek() {**

**if (top >= 0) {**

**return stack[top];**

**} else {**

**printf("Stack is Empty\n");**

**return -1;**

**}**

**}**

**int main() {**

**push(10);**

**push(20);**

**push(30);**

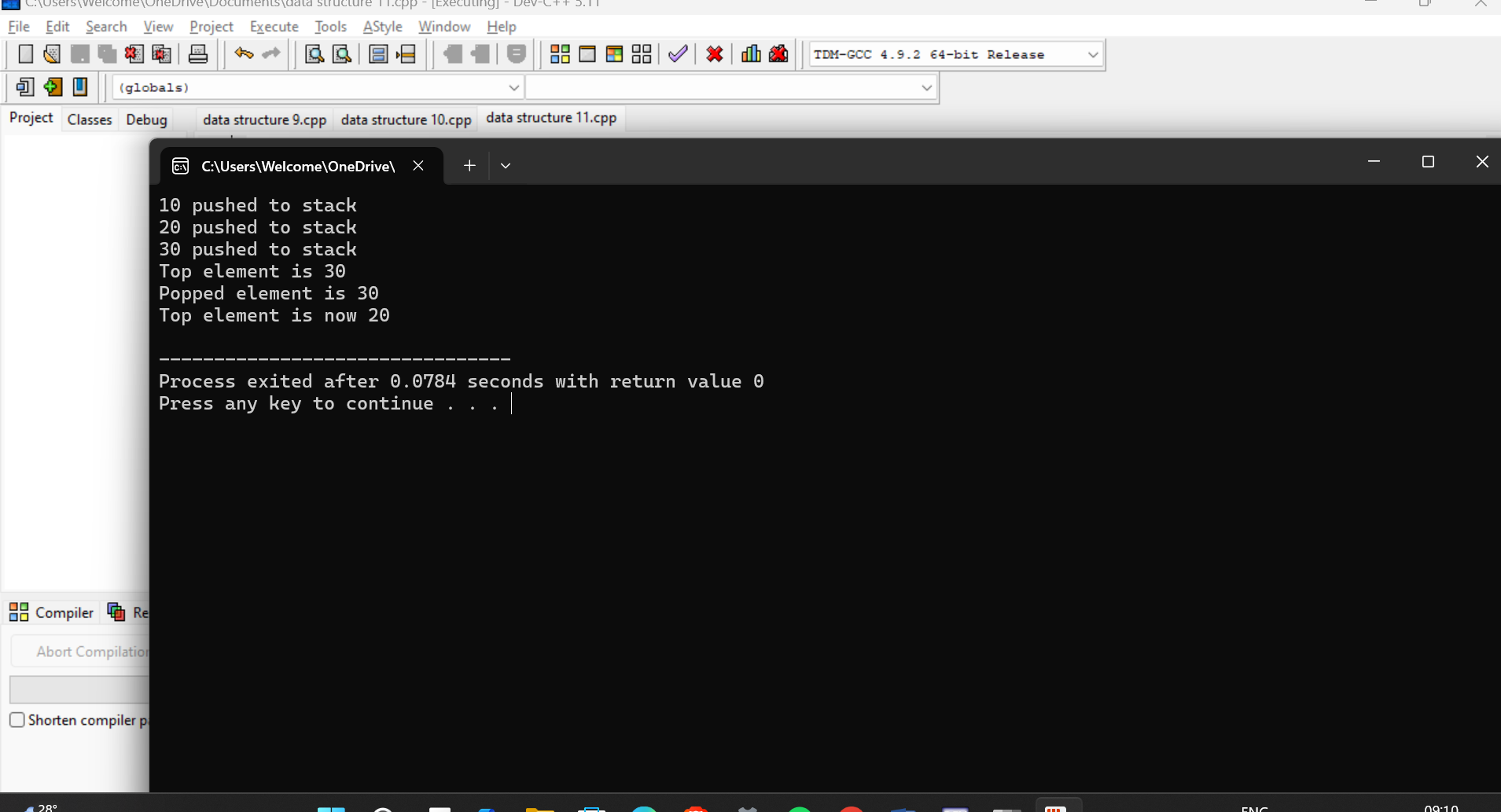
**printf("Top element is %d\n", peek());**

**printf("Popped element is %d\n", pop());**

**printf("Top element is now %d\n", peek());**

**return 0;**

**}**

****

**12. C Program to Implement Stack Application (Infix to Postfix)**

**#include <stdio.h>**

**#include <ctype.h>**

**#define MAX 50**

**char stack[MAX];**

**int top = -1;**

**void push(char c) {**

**stack[++top] = c;**

**}**

**char pop() {**

**return stack[top--];**

**}**

**int precedence(char c) {**

**if (c == '+' || c == '-') return 1;**

**if (c == '\*' || c == '/') return 2;**

**return 0;**

**}**

**void infixToPostfix(char \*infix, char \*postfix) {**

**int i = 0, j = 0;**

**while (infix[i] != '\0') {**

**char c = infix[i];**

**if (isalnum(c)) {**

**postfix[j++] = c;**

**} else if (c == '(') {**

**push(c);**

**} else if (c == ')') {**

**while (top >= 0 && stack[top] != '(') {**

**postfix[j++] = pop();**

**}**

**top--; // pop '('**

**} else {**

**while (top >= 0 && precedence(stack[top]) >= precedence(c)) {**

**postfix[j++] = pop();**

**}**

**push(c);**

**}**

**i++;**

**}**

**while (top >= 0) {**

**postfix[j++] = pop();**

**}**

**postfix[j] = '\0';**

**}**

**int main() {**

**char infix[MAX], postfix[MAX];**

**printf("Enter infix expression: ");**

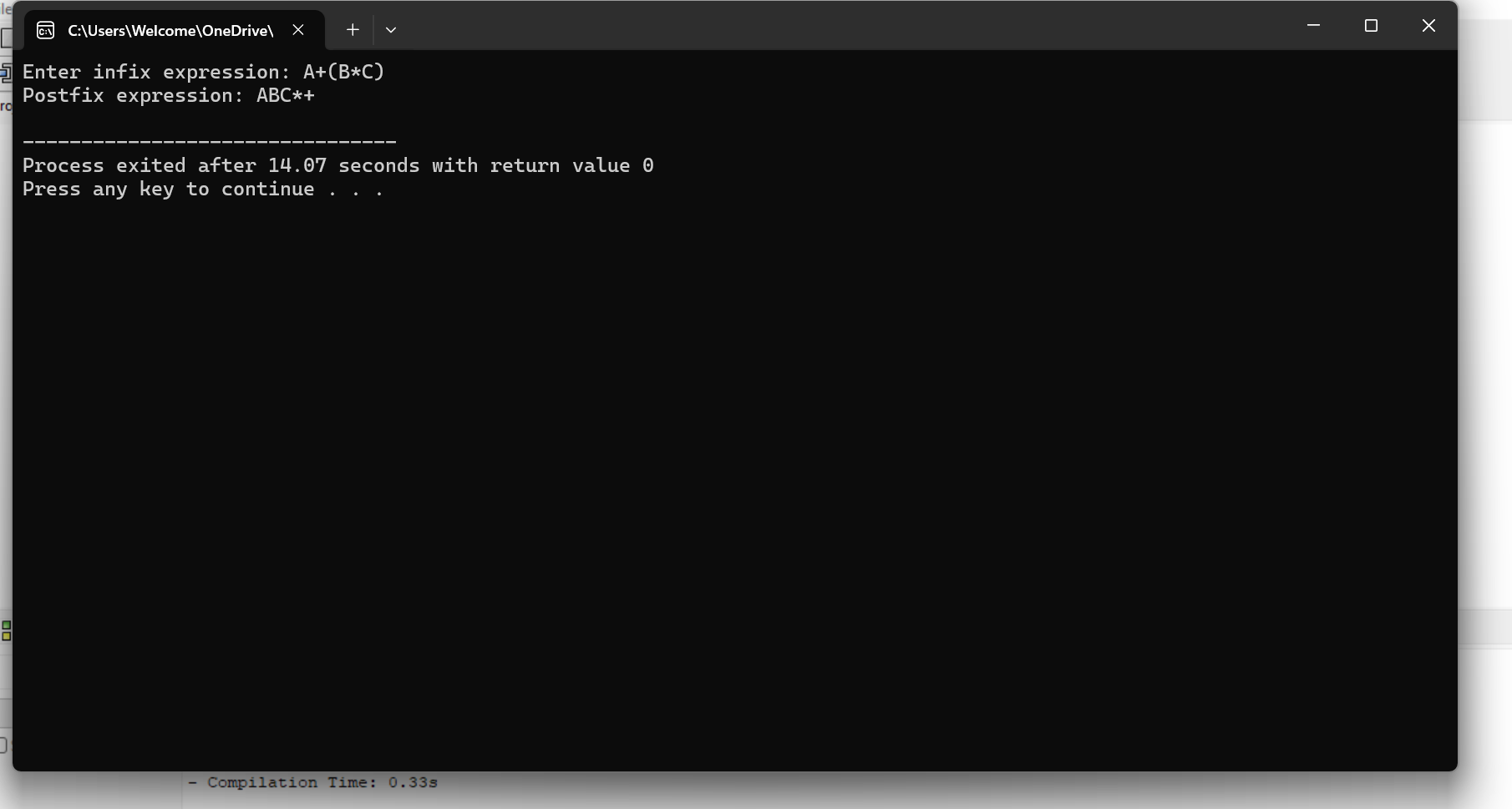
**scanf("%s", infix);**

**infixToPostfix(infix, postfix);**

**printf("Postfix expression: %s\n", postfix);**

**return 0;**

**}**

****

**13. C Program to Implement Queue Operations (ENQUEUE, DEQUEUE, DISPLAY)**

**#include <stdio.h>**

**#define MAX 5**

**int queue[MAX];**

**int front = -1, rear = -1;**

**void enqueue(int value) {**

**if (rear < MAX - 1) {**

**if (front == -1) front = 0;**

**queue[++rear] = value;**

**printf("%d enqueued to queue\n", value);**

**} else {**

**printf("Queue Overflow\n");**

**}**

**}**

**int dequeue() {**

**if (front == -1 || front > rear) {**

**printf("Queue Underflow\n");**

**return -1;**

**} else {**

**return queue[front++];**

**}**

**}**

**void display() {**

**if (front == -1 || front > rear) {**

**printf("Queue is empty\n");**

**} else {**

**for (int i = front; i <= rear; i++) {**

**printf("%d ", queue[i]);**

**}**

**printf("\n");**

**}**

**}**

**int main() {**

**enqueue(10);**

**enqueue(20);**

**enqueue(30);**

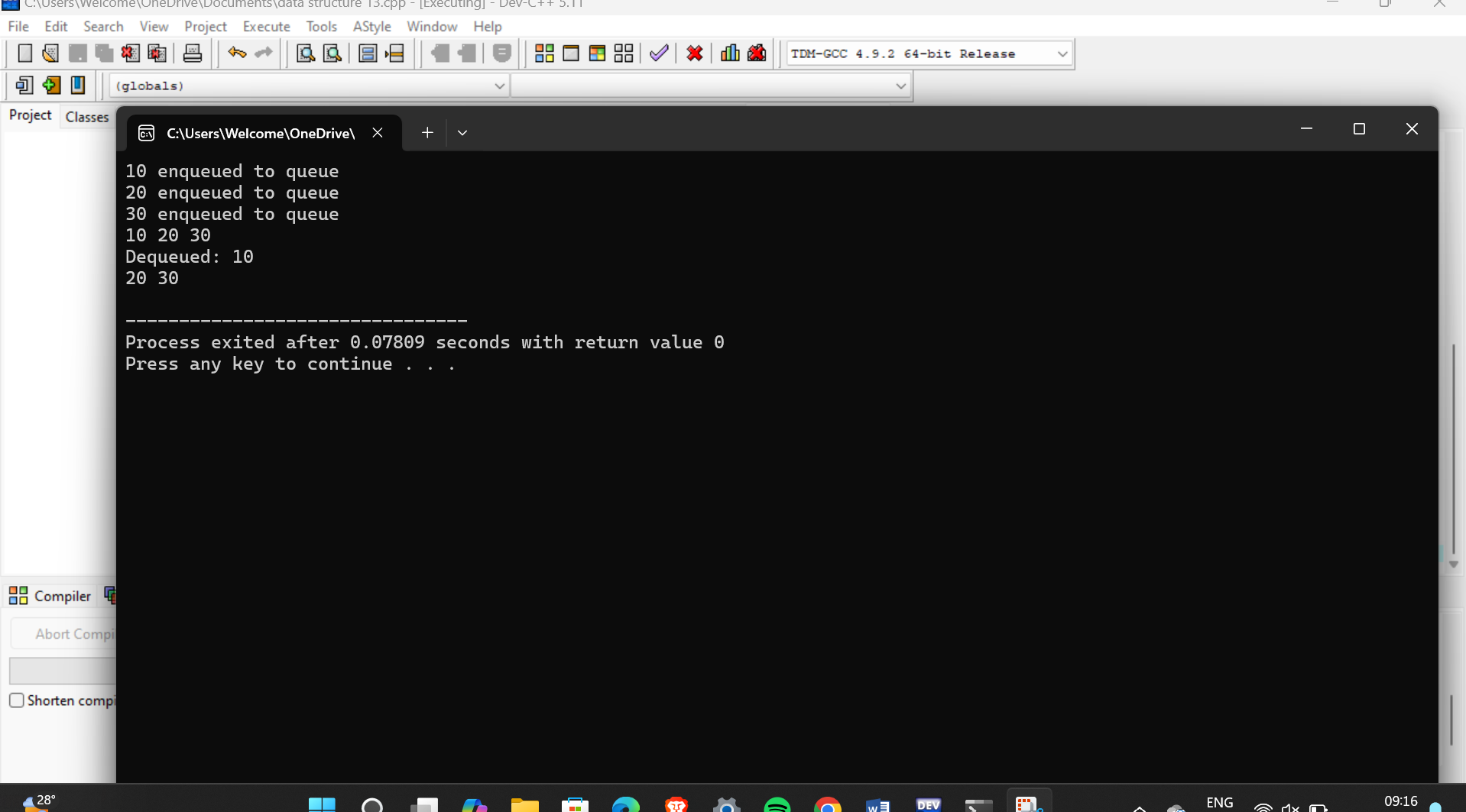
**display();**

**printf("Dequeued: %d\n", dequeue());**

**display();**

**return 0;**

**}**

****

**14. C Program to Implement Tree Traversals (Inorder, Preorder, Postorder)**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Node {**

**int data;**

**struct Node \*left, \*right;**

**};**

**struct Node\* createNode(int data) {**

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

**newNode->data = data;**

**newNode->left = newNode->right = NULL;**

**return newNode;**

**}**

**void inorder(struct Node\* root) {**

**if (root != NULL) {**

**inorder(root->left);**

**printf("%d ", root->data);**

**inorder(root->right);**

**}**

**}**

**void preorder(struct Node\* root) {**

**if (root != NULL) {**

**printf("%d ", root->data);**

**preorder(root->left);**

**preorder(root->right);**

**}**

**}**

**void postorder(struct Node\* root) {**

**if (root != NULL) {**

**postorder(root->left);**

**postorder(root->right);**

**printf("%d ", root->data);**

**}**

**}**

**int main() {**

**struct Node\* root = createNode(1);**

**root->left = createNode(2);**

**root->right = createNode(3);**

**root->left->left = createNode(4);**

**root->left->right = createNode(5);**

**printf("Inorder: ");**

**inorder(root);**

**printf("\n");**

**printf("Preorder: ");**

**preorder(root);**

**printf("\n");**

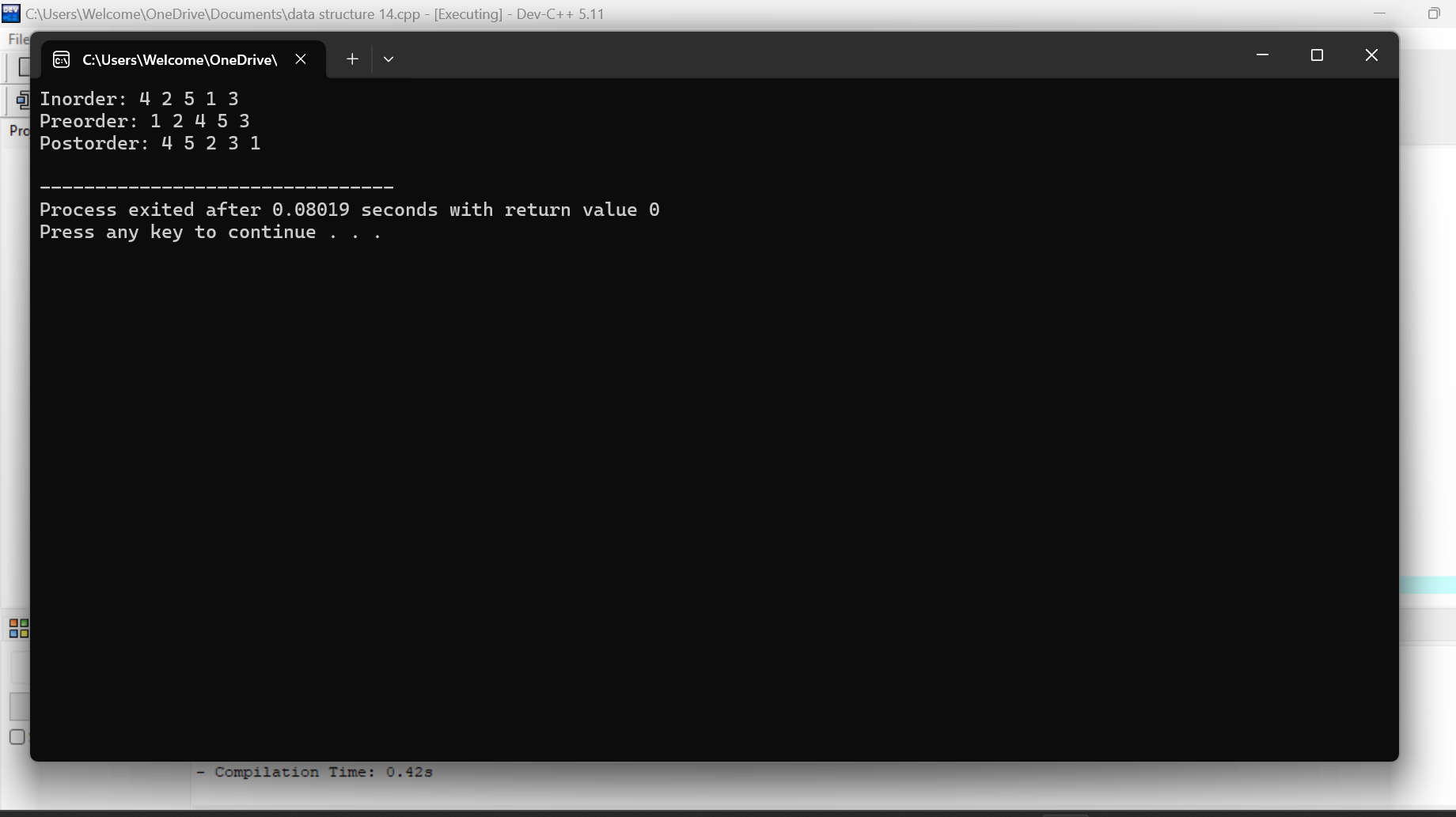
**printf("Postorder: ");**

**postorder(root);**

**printf("\n");**

**return 0;**

**}**

****

### 15. C Program to Implement Hashing Using Linear Probing Method

#include <stdio.h>

#include <stdlib.h>

#define TABLE\_SIZE 10

int hashTable[TABLE\_SIZE];

void initializeHashTable() {

for (int i = 0; i < TABLE\_SIZE; i++) {

hashTable[i] = -1;

}

}

int hashFunction(int key) {

return key % TABLE\_SIZE;

}

void insert(int key) {

int index = hashFunction(key);

while (hashTable[index] != -1) {

index = (index + 1) % TABLE\_SIZE;

}

hashTable[index] = key;

}

void displayHashTable() {

for (int i = 0; i < TABLE\_SIZE; i++) {

if (hashTable[i] != -1) {

printf("Index %d: %d\n", i, hashTable[i]);

} else {

printf("Index %d: Empty\n", i);

}

}

}

int main() {

initializeHashTable() ;

insert(10);

insert(20);

insert(30);

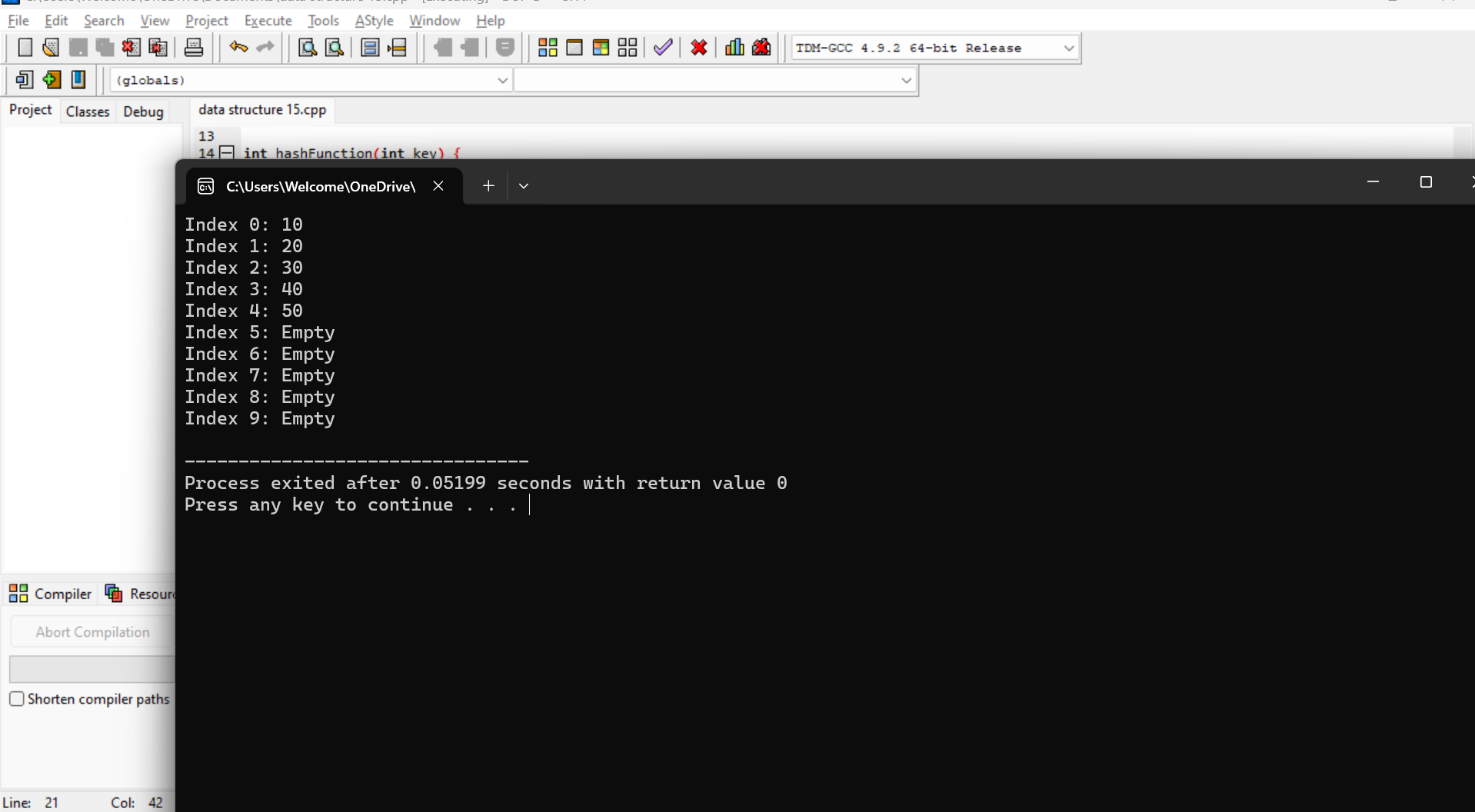
insert(40);

insert(50);

displayHashTable();

return 0;

}

****

### 16. C Program to Arrange a Series of Numbers Using Insertion Sort

#include <stdio.h>

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

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

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

j--;

}

arr[j + 1] = key;

}

}

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

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

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

}

printf("\n");

}

int main() {

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

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

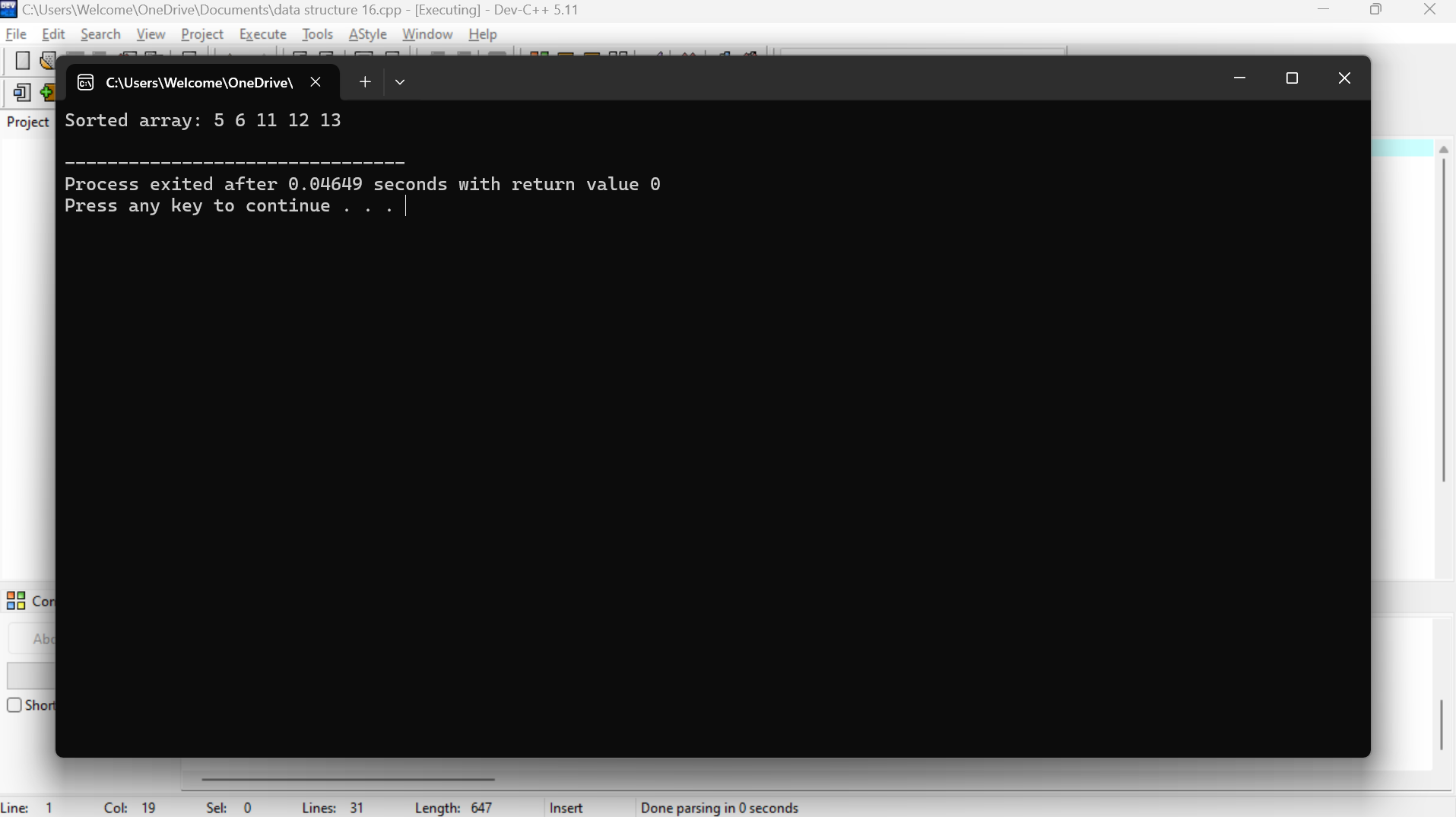
insertionSort(arr, n);

printf("Sorted array: ");

displayArray(arr, n);

return 0;

}



**17. C Program to Arrange a Series of Numbers Using Merge Sort**

**#include <stdio.h>**

**void merge(int arr[], int left, int mid, int right) {**

**int i, j, k;**

**int n1 = mid - left + 1;**

**int n2 = right - mid;**

**int L[n1], R[n2];**

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

**L[i] = arr[left + i];**

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

**R[j] = arr[mid + 1 + j];**

**i = 0;**

**j = 0;**

**k = left;**

**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 left, int right) {**

**if (left < right) {**

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

**mergeSort(arr, left, mid);**

**mergeSort(arr, mid + 1, right);**

**merge(arr, left, mid, right);**

**}**

**}**

**void displayArray(int arr[], int n) {**

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

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

**}**

**printf("\n");**

**}**

**int main() {**

**int arr[] = {12, 11, 13, 4, 8, 7};**

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

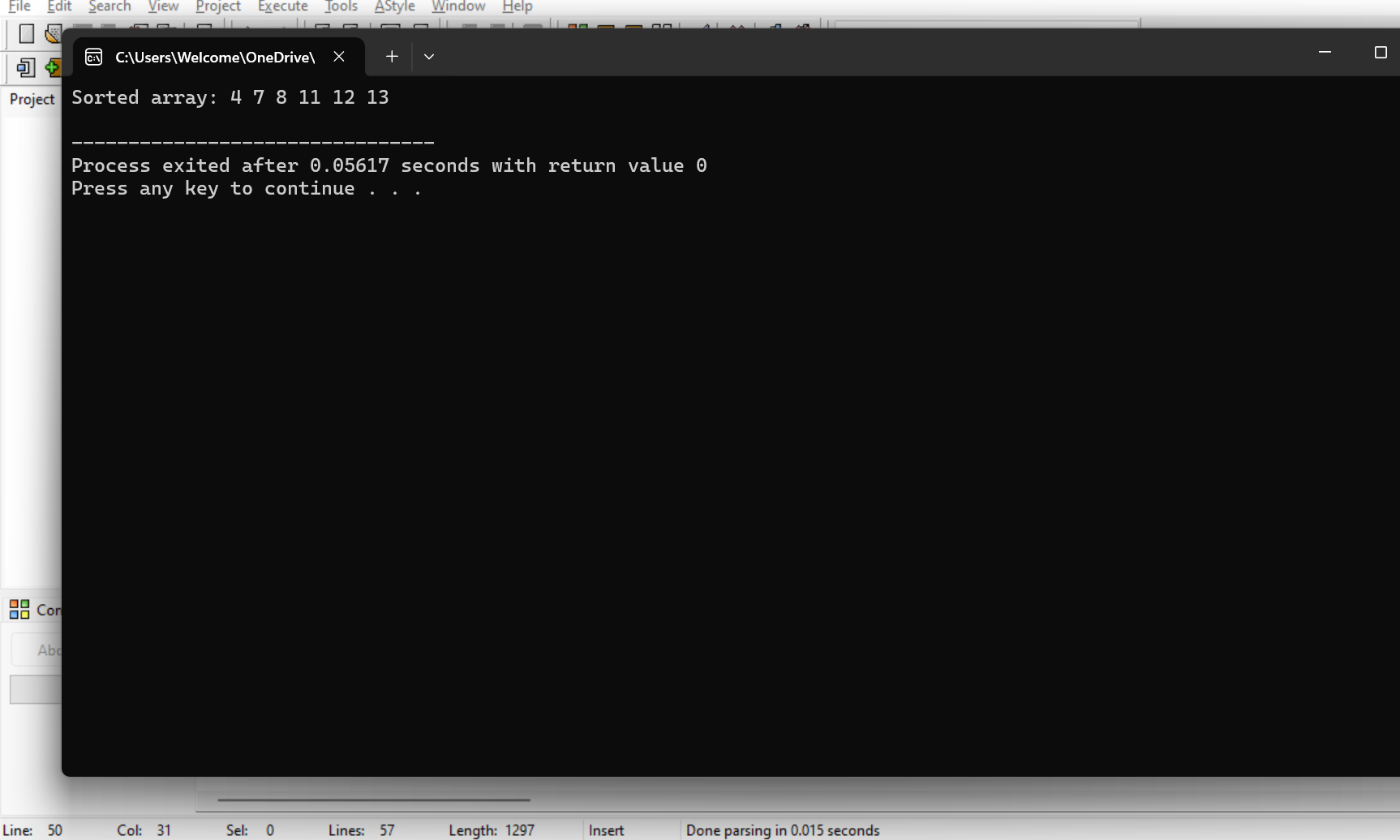
**mergeSort(arr, 0, n - 1);**

**printf("Sorted array: ");**

**displayArray(arr, n);**

**return 0;**

**}**

****

**18. C Program to Arrange a Series of Numbers Using Quick Sort**

**#include <stdio.h>**

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

**int pivot = arr[high];**

**int i = (low - 1);**

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

**if (arr[j] <= pivot) {**

**i++;**

**int temp = arr[i];**

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

**arr[j] = temp;**

**}**

**}**

**int temp = arr[i + 1];**

**arr[i + 1] = arr[high];**

**arr[high] = temp;**

**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);**

**}**

**}**

**void display(int arr[], int n) {**

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

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

**}**

**printf("\n");**

**}**

**int main() {**

**int arr[] = {38, 27, 43, 3, 9, 82, 10};**

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

**printf("Original array: ");**

**display(arr, n);**

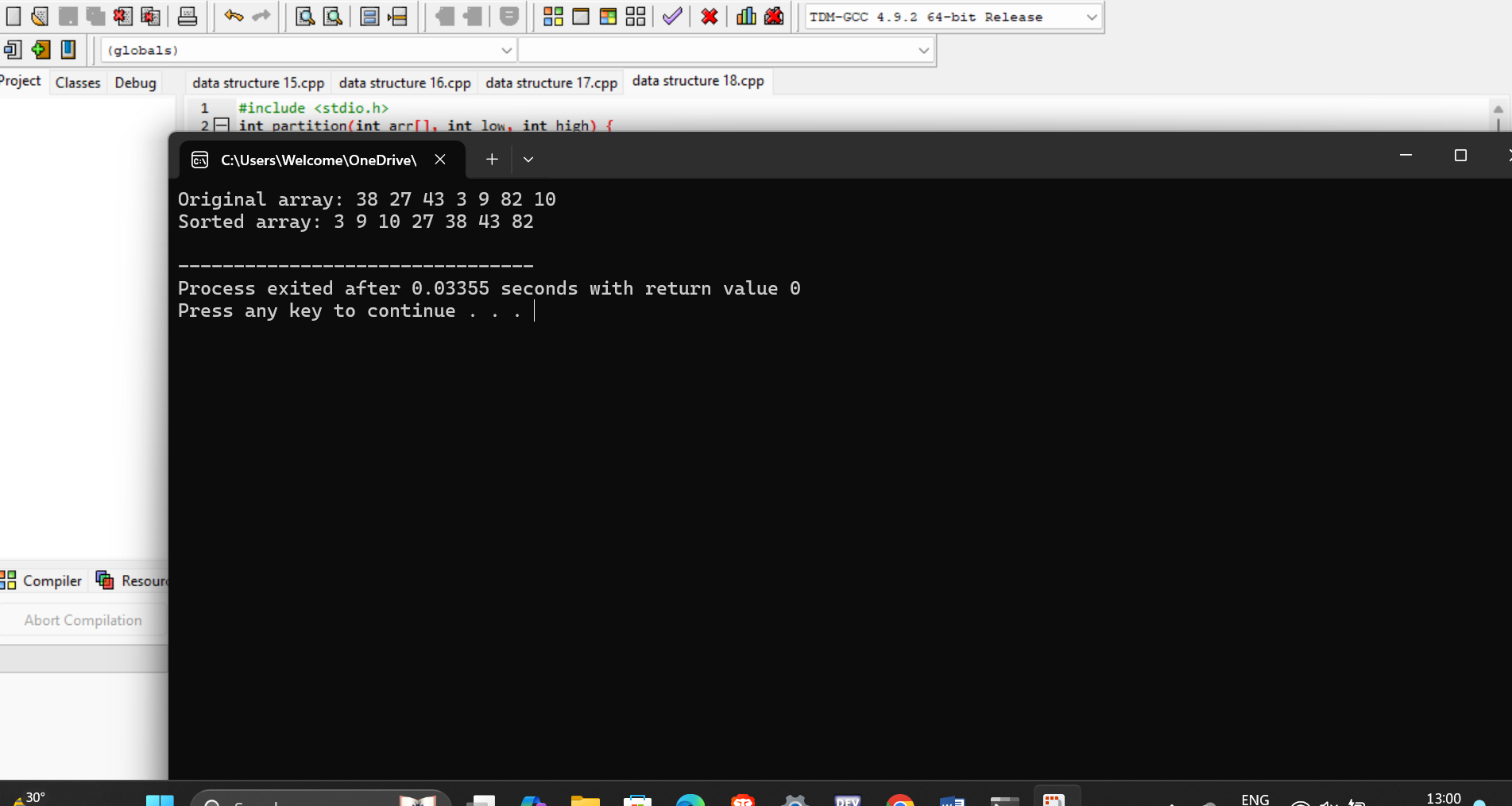
**quickSort(arr, 0, n - 1);**

**printf("Sorted array: ");**

**display(arr, n);**

**return 0;**

**}**

****

### 19. C Program to Implement Heap Sort

**#include <stdio.h>**

**void swap(int \*x, int \*y) {**

**int temp = \*x;**

**\*x = \*y;**

**\*y = temp;**

**}**

**void heapify(int arr[], int n, int i) {**

**int largest = i;**

**int left = 2 \* i + 1;**

**int right = 2 \* i + 2;**

**if (left < n && arr[left] > arr[largest])**

**largest = left;**

**if (right < n && arr[right] > arr[largest])**

**largest = right;**

**if (largest != i) {**

**swap(&arr[i], &arr[largest]);**

**heapify(arr, n, largest);**

**}**

**}**

**void heapSort(int arr[], int n) {**

**for (int i = n / 2 - 1; i >= 0; i--)**

**heapify(arr, n, i);**

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

**swap(&arr[0], &arr[i]);**

**heapify(arr, i, 0);**

**}**

**}**

**void display(int arr[], int n) {**

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

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

**printf("\n");**

**}**

**int main() {**

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

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

**printf("Original array: ");**

**display(arr, n);**

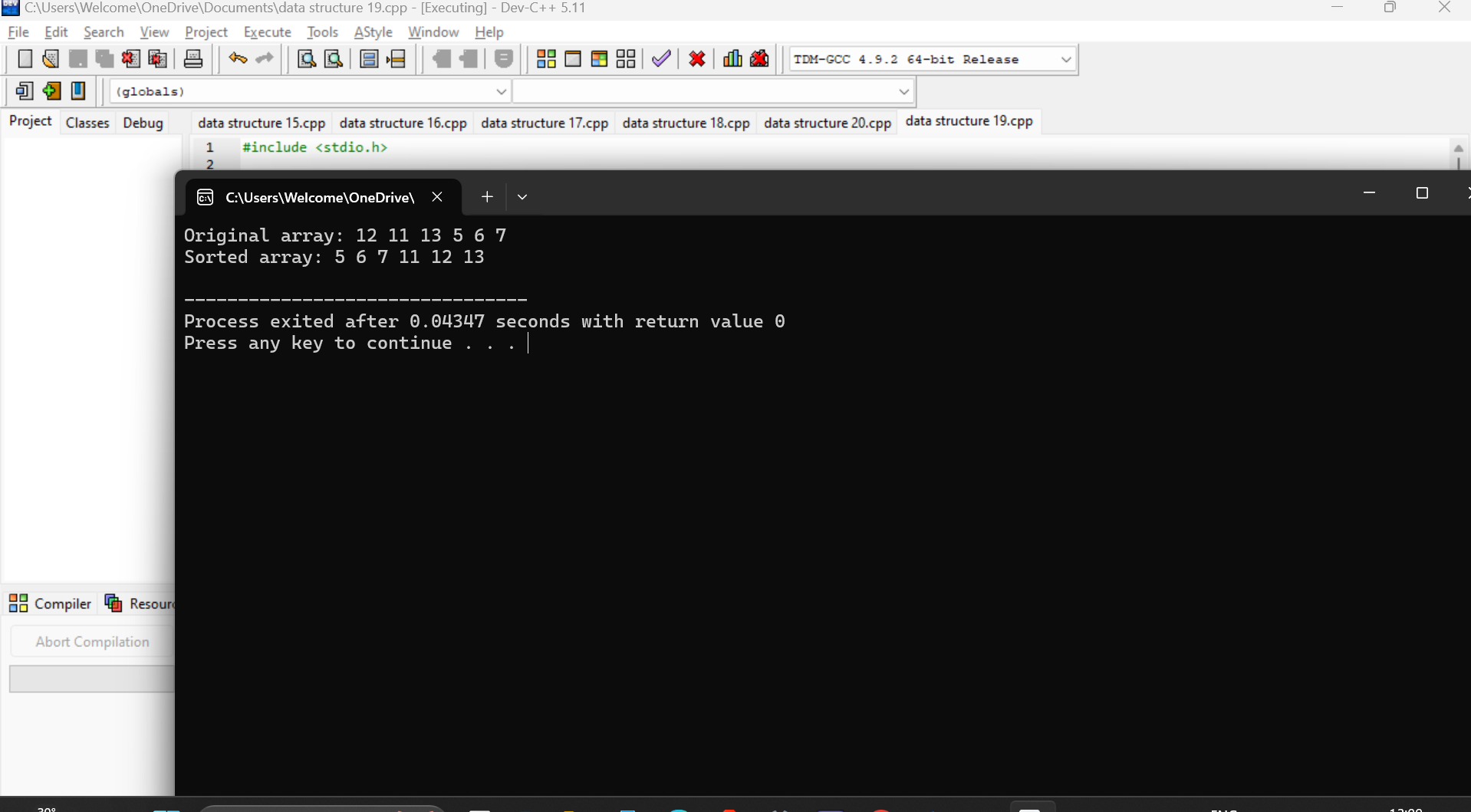
**heapSort(arr, n);**

**printf("Sorted array: ");**

**display(arr, n);**

**return 0;**

**}**

****

**20.Write a program to perform the following operations: a) Insert an element into a AVL tree. b) Delete an element from a AVL tree. c) Search for a key element in a AVL tree.**

**#include <stdio.h>**

**#include <stdlib.h>**

**typedef struct TreeNode {**

**int key;**

**struct TreeNode \*left;**

**struct TreeNode \*right;**

**int height;**

**} TreeNode;**

**int max(int a, int b) {**

**return (a > b) ? a : b;**

**}**

**int height(TreeNode \*node) {**

**if (node == NULL)**

**return 0;**

**return node->height;**

**}**

**TreeNode \*createNode(int key) {**

**TreeNode \*node = (TreeNode \*)malloc(sizeof(TreeNode));**

**node->key = key;**

**node->left = node->right = NULL;**

**node->height = 1;**

**return node;**

**}**

**TreeNode \*rightRotate(TreeNode \*y) {**

**TreeNode \*x = y->left;**

**TreeNode \*T2 = x->right;**

**x->right = y;**

**y->left = T2;**

**y->height = max(height(y->left), height(y->right)) + 1;**

**x->height = max(height(x->left), height(x->right)) + 1;**

**return x;**

**}**

**TreeNode \*leftRotate(TreeNode \*x) {**

**TreeNode \*y = x->right;**

**TreeNode \*T2 = y->left;**

**y->left = x;**

**x->right = T2;**

**x->height = max(height(x->left), height(x->right)) + 1;**

**y->height = max(height(y->left), height(y->right)) + 1;**

**return y;**

**}**

**int getBalance(TreeNode \*node) {**

**if (node == NULL)**

**return 0;**

**return height(node->left) - height(node->right);**

**}**

**TreeNode \*insert(TreeNode \*node, int key) {**

**if (node == NULL)**

**return createNode(key);**

**if (key < node->key)**

**node->left = insert(node->left, key);**

**else if (key > node->key)**

**node->right = insert(node->right, key);**

**else**

**return node;**

**node->height = 1 + max(height(node->left), height(node->right));**

**int balance = getBalance(node);**

**if (balance > 1 && key < node->left->key)**

**return rightRotate(node);**

**if (balance < -1 && key > node->right->key)**

**return leftRotate(node);**

**if (balance > 1 && key > node->left->key) {**

**node->left = leftRotate(node->left);**

**return rightRotate(node);**

**}**

**if (balance < -1 && key < node->right->key) {**

**node->right = rightRotate(node->right);**

**return leftRotate(node);**

**}**

**return node;**

**}**

**TreeNode \*minValueNode(TreeNode \*node) {**

**TreeNode \*current = node;**

**while (current->left != NULL)**

**current = current->left;**

**return current;**

**}**

**TreeNode \*deleteNode(TreeNode \*root, int key) {**

**if (root == NULL)**

**return root;**

**if (key < root->key)**

**root->left = deleteNode(root->left, key);**

**else if (key > root->key)**

**root->right = deleteNode(root->right, key);**

**else {**

**if ((root->left == NULL) || (root->right == NULL)) {**

**TreeNode \*temp = root->left ? root->left : root->right;**

**if (temp == NULL) {**

**temp = root;**

**root = NULL;**

**} else**

**\*root = \*temp;**

**free(temp);**

**} else {**

**TreeNode \*temp = minValueNode(root->right);**

**root->key = temp->key;**

**root->right = deleteNode(root->right, temp->key);**

**}**

**}**

**if (root == NULL)**

**return root;**

**root->height = 1 + max(height(root->left), height(root->right));**

**int balance = getBalance(root);**

**if (balance > 1 && getBalance(root->left) >= 0)**

**return rightRotate(root);**

**if (balance > 1 && getBalance(root->left) < 0) {**

**root->left = leftRotate(root->left);**

**return rightRotate(root);**

**}**

**if (balance < -1 && getBalance(root->right) <= 0)**

**return leftRotate(root);**

**if (balance < -1 && getBalance(root->right) > 0) {**

**root->right = rightRotate(root->right);**

**return leftRotate(root);**

**}**

**return root;**

**}**

**TreeNode \*search(TreeNode \*root, int key) {**

**if (root == NULL || root->key == key)**

**return root;**

**if (key < root->key)**

**return search(root->left, key);**

**return search(root->right, key);**

**}**

**void preOrder(TreeNode \*root) {**

**if (root != NULL) {**

**printf("%d ", root->key);**

**preOrder(root->left);**

**preOrder(root->right);**

**}**

**}**

**int main() {**

**TreeNode \*root = NULL;**

**root = insert(root, 10);**

**root = insert(root, 20);**

**root = insert(root, 30);**

**root = insert(root, 40);**

**root = insert(root, 50);**

**root = insert(root, 25);**

**printf("Preorder traversal of the AVL tree is: ");**

**preOrder(root);**

**printf("\n");**

**root = deleteNode(root, 10);**

**printf("Preorder traversal after deletion of 10: ");**

**preOrder(root);**

**printf("\n");**

**int key = 30;**

**TreeNode \*found = search(root, key);**

**if (found)**

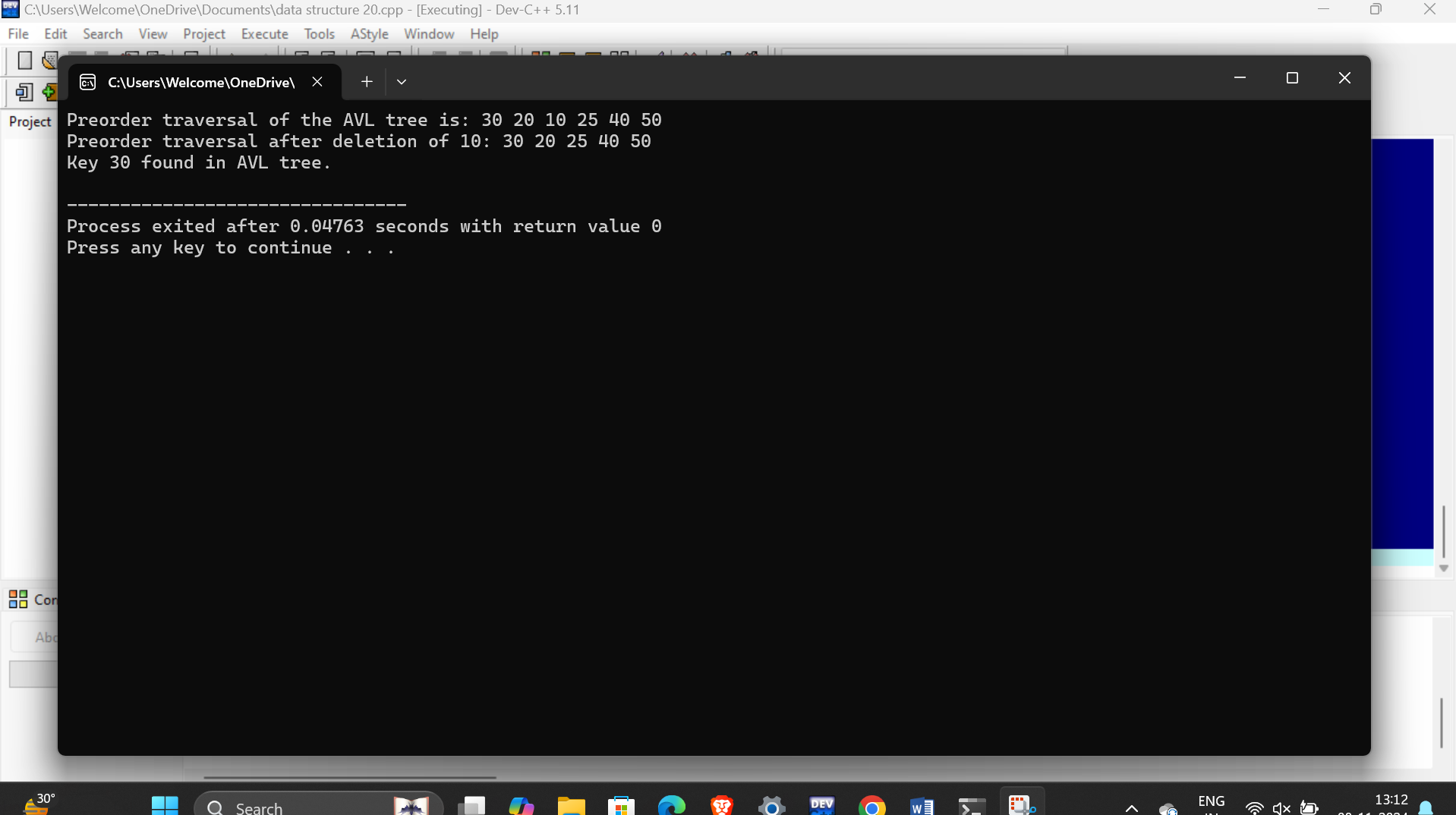
**printf("Key %d found in AVL tree.\n", key);**

**else**

**printf("Key %d not found in AVL tree.\n", key);**

**return 0;**

**}**

****

**21. Write a C program to Graph traversal using Breadth First Search**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <stdbool.h>**

**#define MAX 100**

**struct Queue {**

**int items[MAX];**

**int front;**

**int rear;**

**};**

**struct Queue\* createQueue() {**

**struct Queue\* q = (struct Queue\*)malloc(sizeof(struct Queue));**

**q->front = -1;**

**q->rear = -1;**

**return q;**

**}**

**bool isEmpty(struct Queue\* q) {**

**return q->rear == -1;**

**}**

**void enqueue(struct Queue\* q, int value) {**

**if (q->rear == MAX - 1)**

**return;**

**if (q->front == -1)**

**q->front = 0;**

**q->rear++;**

**q->items[q->rear] = value;**

**}**

**int dequeue(struct Queue\* q) {**

**int item = q->items[q->front];**

**if (q->front >= q->rear) {**

**q->front = -1;**

**q->rear = -1;**

**} else {**

**q->front++;**

**}**

**return item;**

**}**

**void bfs(int graph[MAX][MAX], int startVertex, int numVertices) {**

**bool visited[MAX] = {false};**

**struct Queue\* q = createQueue();**

**visited[startVertex] = true;**

**enqueue(q, startVertex);**

**while (!isEmpty(q)) {**

**int currentVertex = dequeue(q);**

**printf("Visited %d\n", currentVertex);**

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

**if (graph[currentVertex][i] == 1 && !visited[i]) {**

**visited[i] = true;**

**enqueue(q, i);**

**}**

**}**

**}**

**}**

**int main() {**

**int graph[MAX][MAX] = {0};**

**int numVertices = 5;**

**graph[0][1] = 1;**

**graph[0][4] = 1;**

**graph[1][0] = 1;**

**graph[1][2] = 1;**

**graph[1][3] = 1;**

**graph[1][4] = 1;**

**graph[2][1] = 1;**

**graph[3][1] = 1;**

**graph[4][0] = 1;**

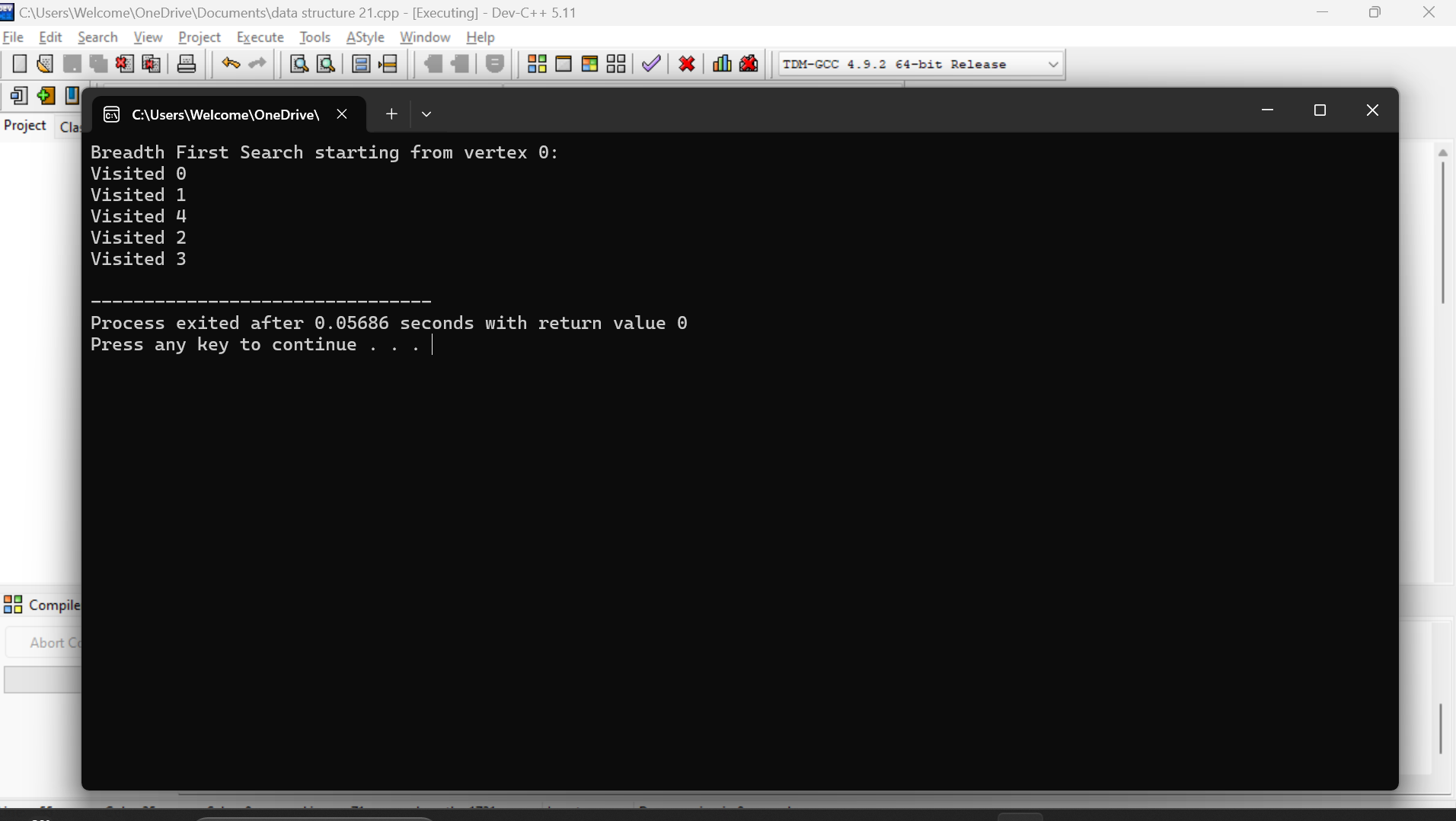
**graph[4][1] = 1;**

**printf("Breadth First Search starting from vertex 0:\n");**

**bfs(graph, 0, numVertices);**

**return 0;**

**}**

****

**22. Write a C program to Graph traversal using Depth First Search**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 100**

**int adj[MAX][MAX];**

**int visited[MAX];**

**int n;**

**void DFS(int v) {**

**visited[v] = 1;**

**printf("%d ", v);**

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

**if (adj[v][i] == 1 && !visited[i]) {**

**DFS(i);**

**}**

**}**

**}**

**void createGraph() {**

**int maxEdges, origin, dest;**

**printf("Enter the number of vertices: ");**

**scanf("%d", &n);**

**maxEdges = n \* (n - 1);**

**for (int i = 1; i <= maxEdges; i++) {**

**printf("Enter edge %d ( -1 -1 to quit ): ", i);**

**scanf("%d %d", &origin, &dest);**

**if ((origin == -1) && (dest == -1))**

**break;**

**if (origin >= n || dest >= n || origin < 0 || dest < 0) {**

**printf("Invalid edge!\n");**

**i--;**

**} else {**

**adj[origin][dest] = 1;**

**adj[dest][origin] = 1;**

**}**

**}**

**}**

**int main() {**

**int startVertex;**

**createGraph();**

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

**visited[i] = 0;**

**}**

**printf("Enter the starting vertex for DFS: ");**

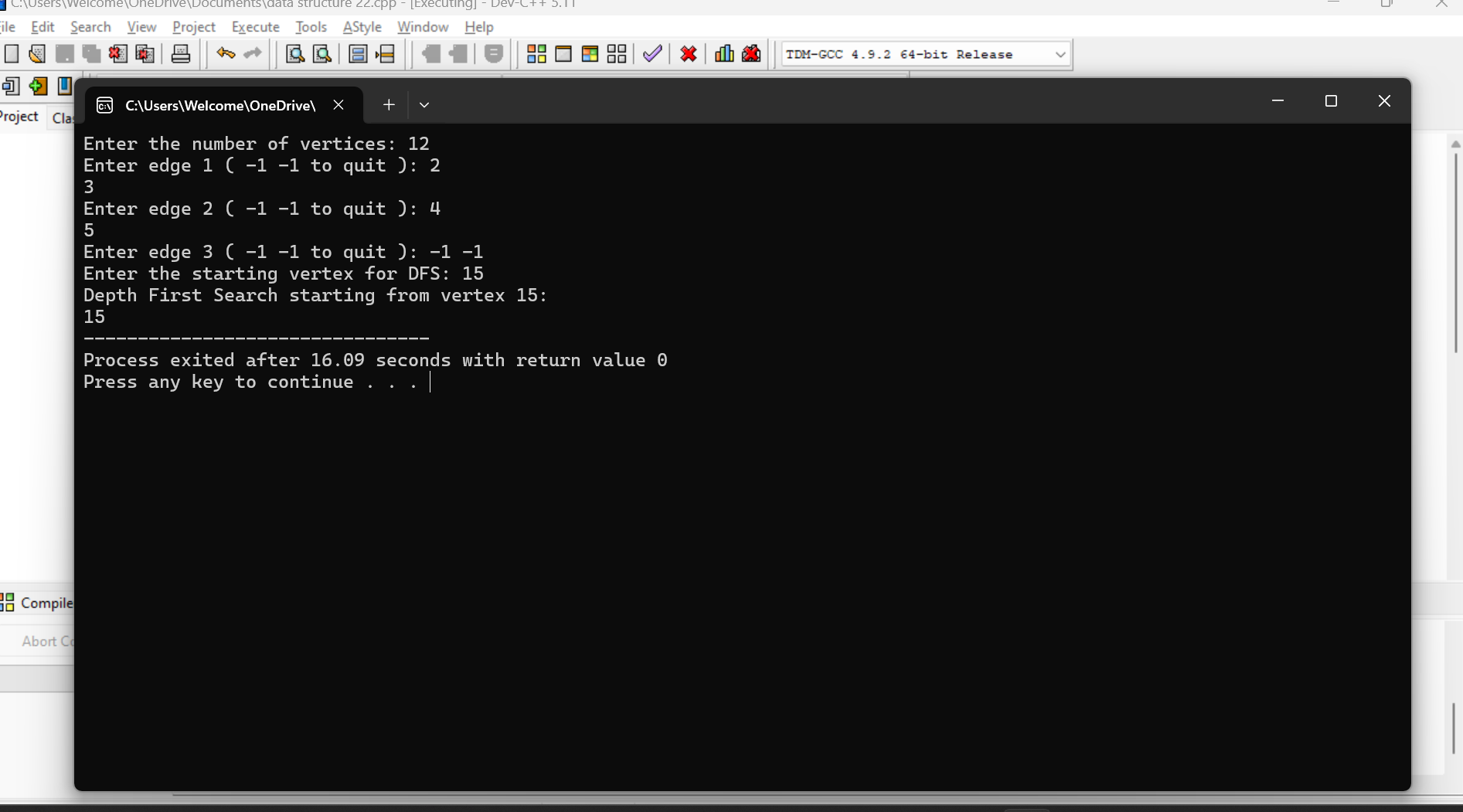
**scanf("%d", &startVertex);**

**printf("Depth First Search starting from vertex %d:\n", startVertex);**

**DFS(startVertex);**

**return 0;**

**}**

****

**23. Implementation of Shortest Path Algorithms using Dijkstra’s Algorithm**

**#include <stdio.h>**

**#include <limits.h>**

**#define MAX 100**

**#define INF INT\_MAX**

**int n;**

**int graph[MAX][MAX];**

**int minDistance(int dist[], int visited[]) {**

**int min = INF, min\_index;**

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

**if (visited[v] == 0 && dist[v] <= min) {**

**min = dist[v];**

**min\_index = v;**

**}**

**return min\_index;**

**}**

**void dijkstra(int src) {**

**int dist[MAX];**

**int visited[MAX];**

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

**dist[i] = INF;**

**visited[i] = 0;**

**}**

**dist[src] = 0;**

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

**int u = minDistance(dist, visited);**

**visited[u] = 1;**

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

**if (!visited[v] && graph[u][v] && dist[u] != INF && dist[u] + graph[u][v] < dist[v]) {**

**dist[v] = dist[u] + graph[u][v];**

**}**

**}**

**printf("Vertex \t\t Distance from Source\n");**

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

**printf("%d \t\t %d\n", i, dist[i]);**

**}**

**void createGraph() {**

**printf("Enter the number of vertices: ");**

**scanf("%d", &n);**

**printf("Enter the adjacency matrix (0 for no edge, weight for edges):\n");**

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

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

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

**if (graph[i][j] == 0 && i != j) {**

**graph[i][j] = INF;**

**}**

**}**

**}**

**}**

**int main() {**

**int startVertex;**

**createGraph();**

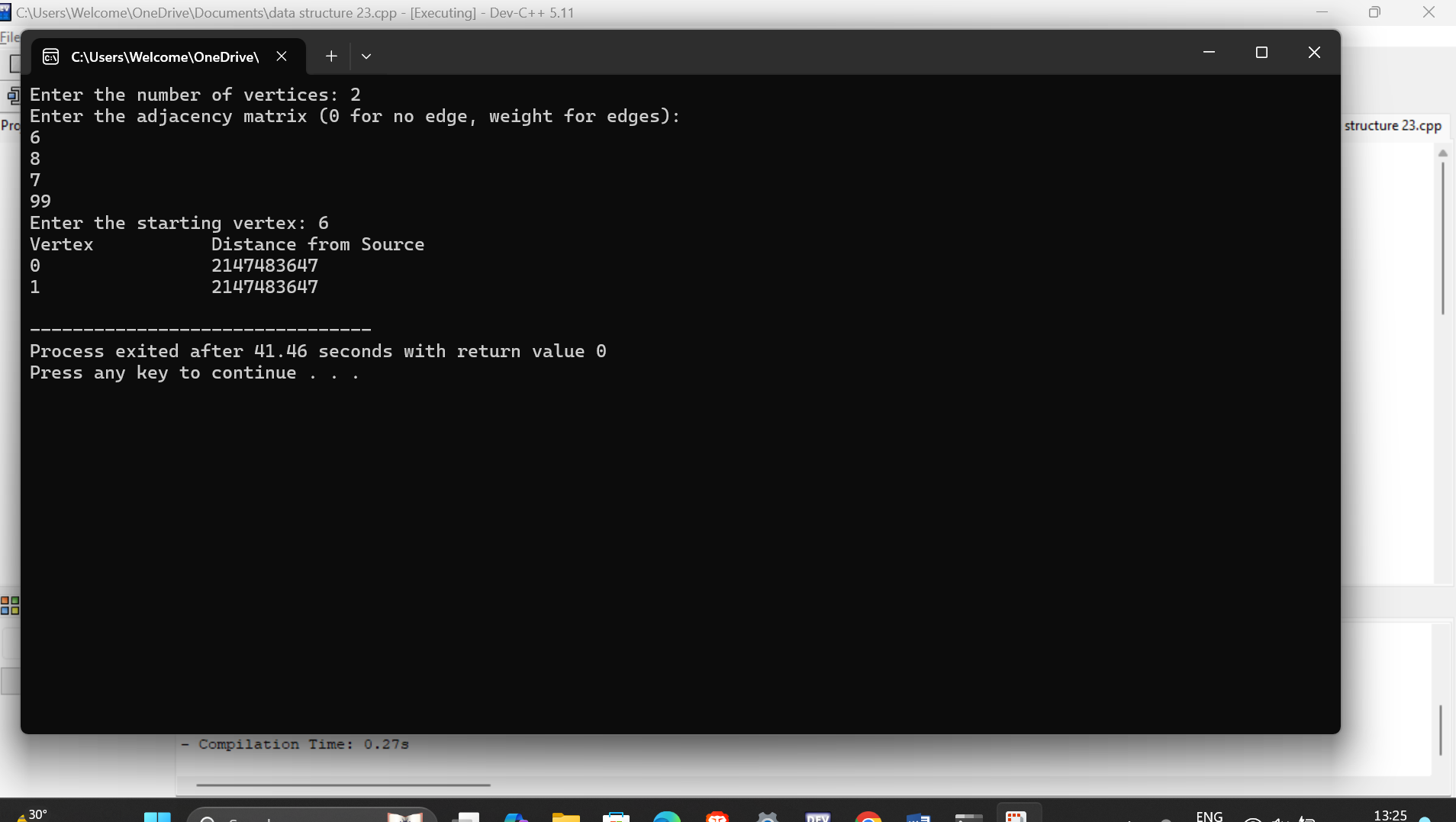
**printf("Enter the starting vertex: ");**

**scanf("%d", &startVertex);**

**dijkstra(startVertex);**

**return 0;**

**}**

****

**24. Implementation of Minimum Spanning Tree using Prim’s Algorithm**

**#include <stdio.h>**

**#include <limits.h>**

**#define MAX 100**

**#define INF INT\_MAX**

**int n;**

**int graph[MAX][MAX];**

**int minKey(int key[], int mstSet[]) {**

**int min = INF, min\_index;**

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

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

**min = key[v];**

**min\_index = v;**

**}**

**return min\_index;**

**}**

**void printMST(int parent[]) {**

**printf("Edge \tWeight\n");**

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

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

**}**

**void primMST() {**

**int parent[MAX];**

**int key[MAX];**

**int mstSet[MAX];**

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

**key[i] = INF;**

**mstSet[i] = 0;**

**}**

**key[0] = 0;**

**parent[0] = -1;**

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

**int u = minKey(key, mstSet);**

**mstSet[u] = 1;**

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

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

**parent[v] = u;**

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

**}**

**}**

**printMST(parent);**

**}**

**void createGraph() {**

**printf("Enter the number of vertices: ");**

**scanf("%d", &n);**

**printf("Enter the adjacency matrix (0 for no edge, weight for edges):\n");**

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

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

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

**}**

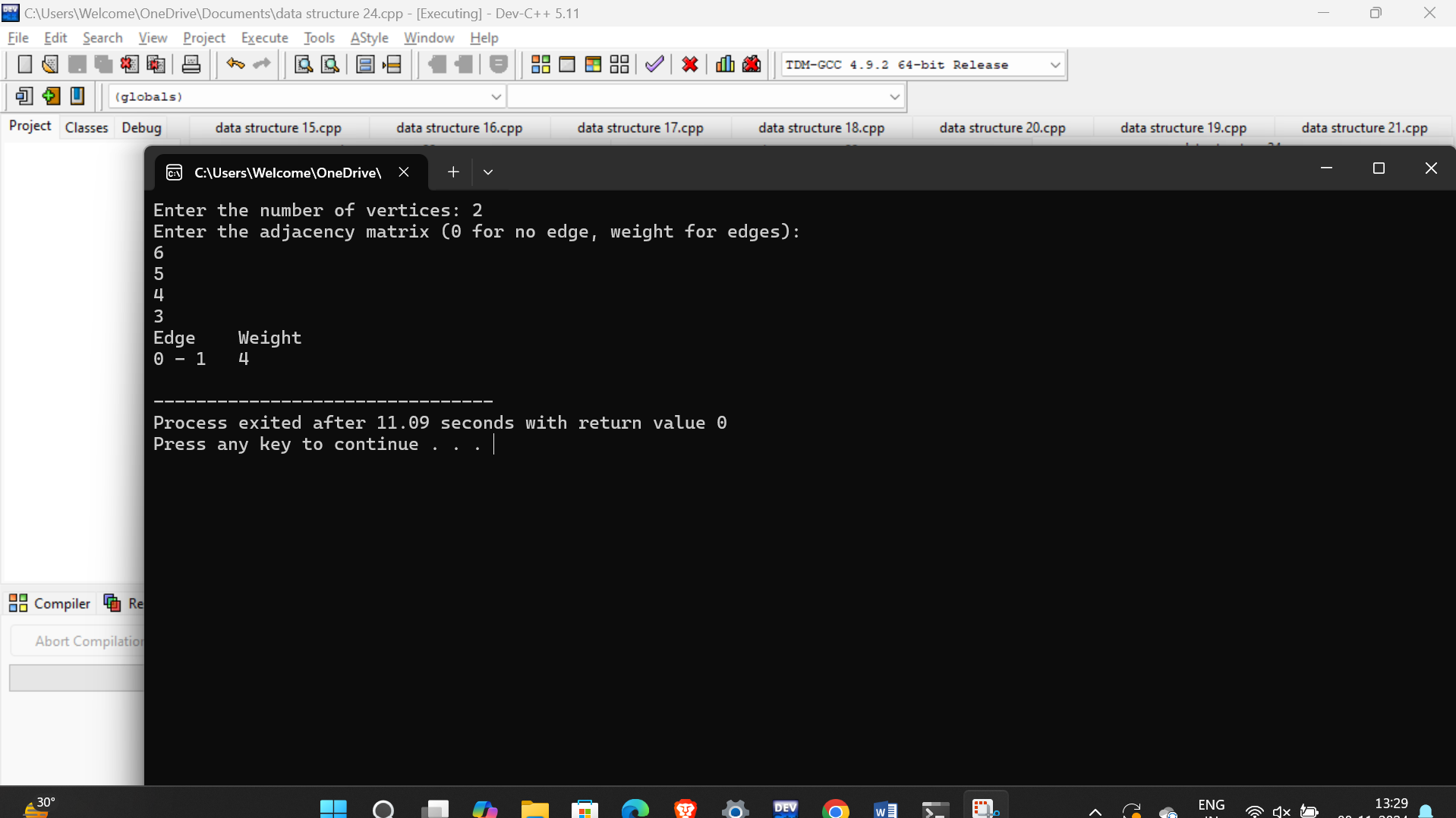
**int main() {**

**createGraph();**

**primMST();**

**return 0;**

**}**

****

**25. Implementation of Minimum Spanning Tree using Kruskal Algorithm**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 100**

**struct Edge {**

**int src, dest, weight;**

**};**

**struct Graph {**

**int V, E;**

**struct Edge edges[MAX];**

**};**

**struct Graph\* createGraph(int V, int E) {**

**struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));**

**graph->V = V;**

**graph->E = E;**

**return graph;**

**}**

**struct Subset {**

**int parent;**

**int rank;**

**};**

**int find(struct Subset subsets[], int i) {**

**if (subsets[i].parent != i)**

**subsets[i].parent = find(subsets, subsets[i].parent);**

**return subsets[i].parent;**

**}**

**void Union(struct Subset subsets[], int x, int y) {**

**int xroot = find(subsets, x);**

**int yroot = find(subsets, y);**

**if (subsets[xroot].rank < subsets[yroot].rank)**

**subsets[xroot].parent = yroot;**

**else if (subsets[xroot].rank > subsets[yroot].rank)**

**subsets[yroot].parent = xroot;**

**else {**

**subsets[yroot].parent = xroot;**

**subsets[xroot].rank++;**

**}**

**}**

**int compare(const void\* a, const void\* b) {**

**struct Edge\* a1 = (struct Edge\*)a;**

**struct Edge\* b1 = (struct Edge\*)b;**

**return a1->weight > b1->weight;**

**}**

**void KruskalMST(struct Graph\* graph) {**

**int V = graph->V;**

**struct Edge result[MAX];**

**int e = 0;**

**int i = 0;**

**qsort(graph->edges, graph->E, sizeof(graph->edges[0]), compare);**

**struct Subset\* subsets = (struct Subset\*)malloc(V \* sizeof(struct Subset));**

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

**subsets[v].parent = v;**

**subsets[v].rank = 0;**

**}**

**while (e < V - 1 && i < graph->E) {**

**struct Edge next\_edge = graph->edges[i++];**

**int x = find(subsets, next\_edge.src);**

**int y = find(subsets, next\_edge.dest);**

**if (x != y) {**

**result[e++] = next\_edge;**

**Union(subsets, x, y);**

**}**

**}**

**printf("Edge \tWeight\n");**

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

**printf("%d - %d \t%d\n", result[i].src, result[i].dest, result[i].weight);**

**free(subsets);**

**}**

**int main() {**

**int V, E;**

**printf("Enter number of vertices and edges: ");**

**scanf("%d %d", &V, &E);**

**struct Graph\* graph = createGraph(V, E);**

**printf("Enter each edge (source destination weight):\n");**

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

**scanf("%d %d %d", &graph->edges[i].src, &graph->edges[i].dest, &graph->edges[i].weight);**

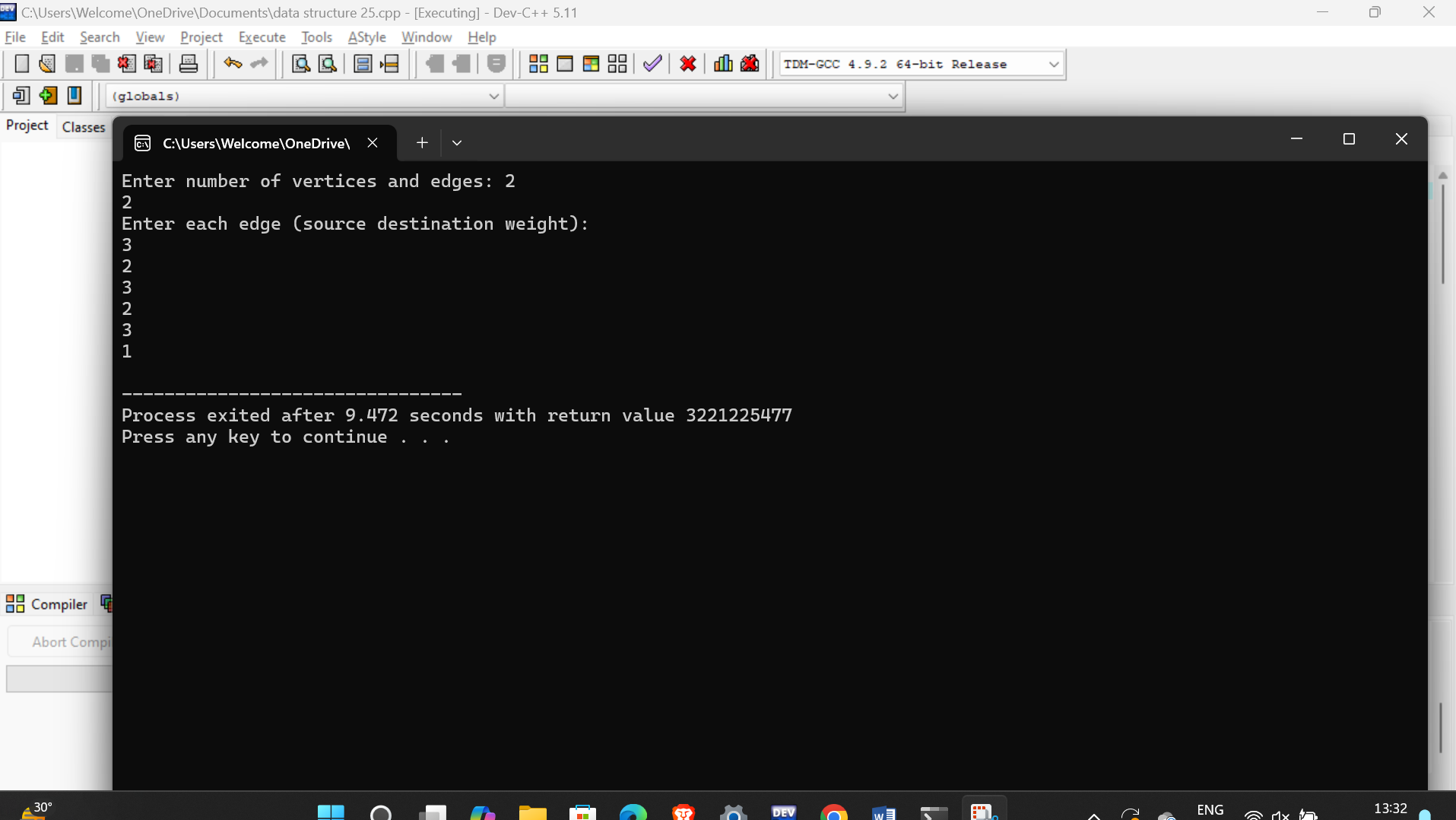
**}**

**KruskalMST(graph);**

**free(graph);**

**return 0;**

**}**

****