CSA0336-DATA STRUCTURES

NAME: U.SWETHA

REG NO: 192111067

YEAR&DEPT: III YEAR,CSE

1.ODD OR EVEN:

#include<stdio.h>

int main()

{

int n;

printf("enter the number: ");

scanf("%d",&n);

if(n%2==0)

{

printf("even number");

}

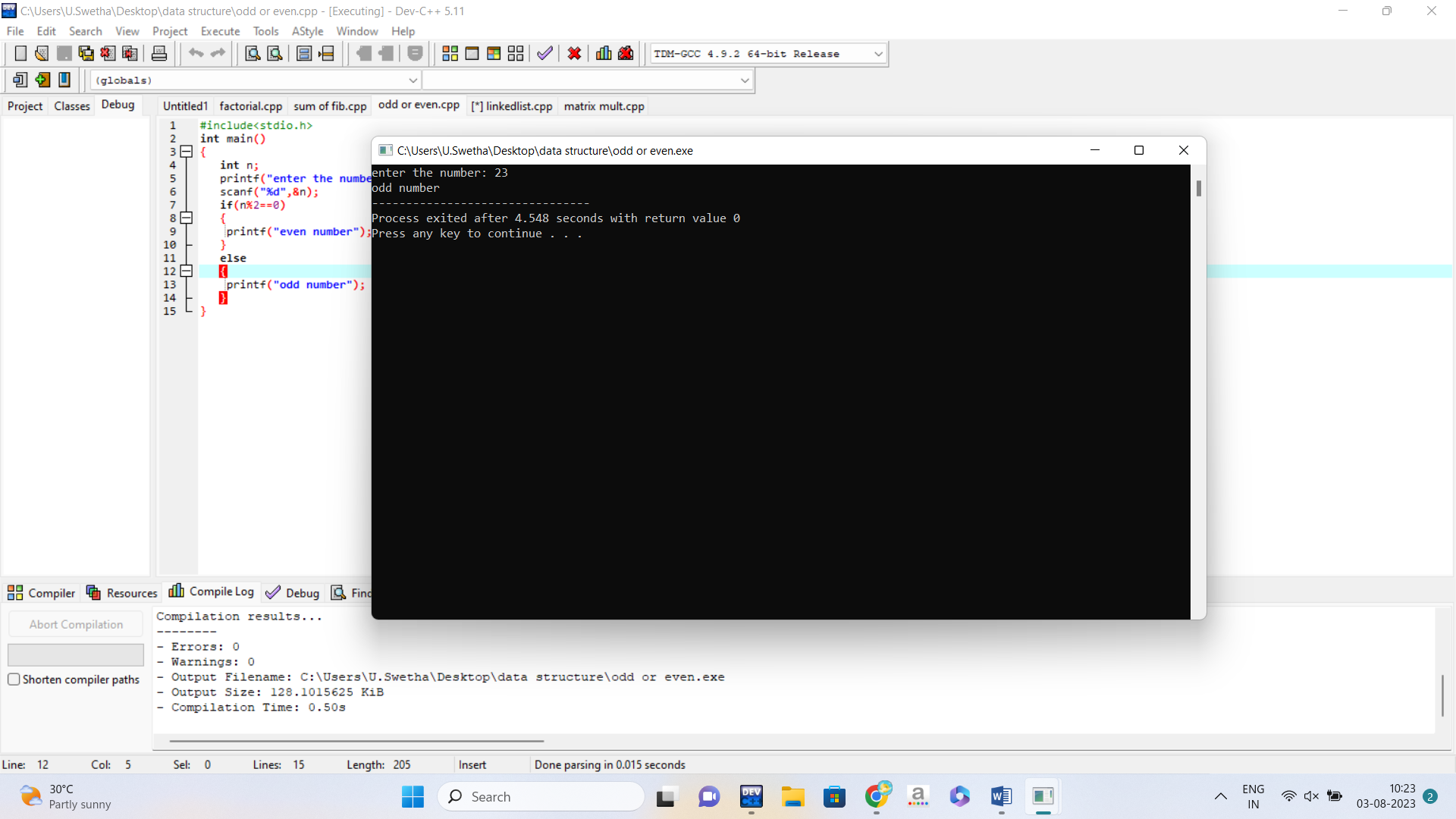
else

{

printf("odd number");

}

}



2. FIBONACCI SERIES:

#include<stdio.h>

int main()

{

int n,n1=0,n2=1,n3;

printf("enter the number :");

scanf("%d",&n);

printf("series are %d %d \n",n1,n2);

for(int i=3;i<=n;i++)

{

n3=n1+n2;

printf("%d \n",n3);

int temp;

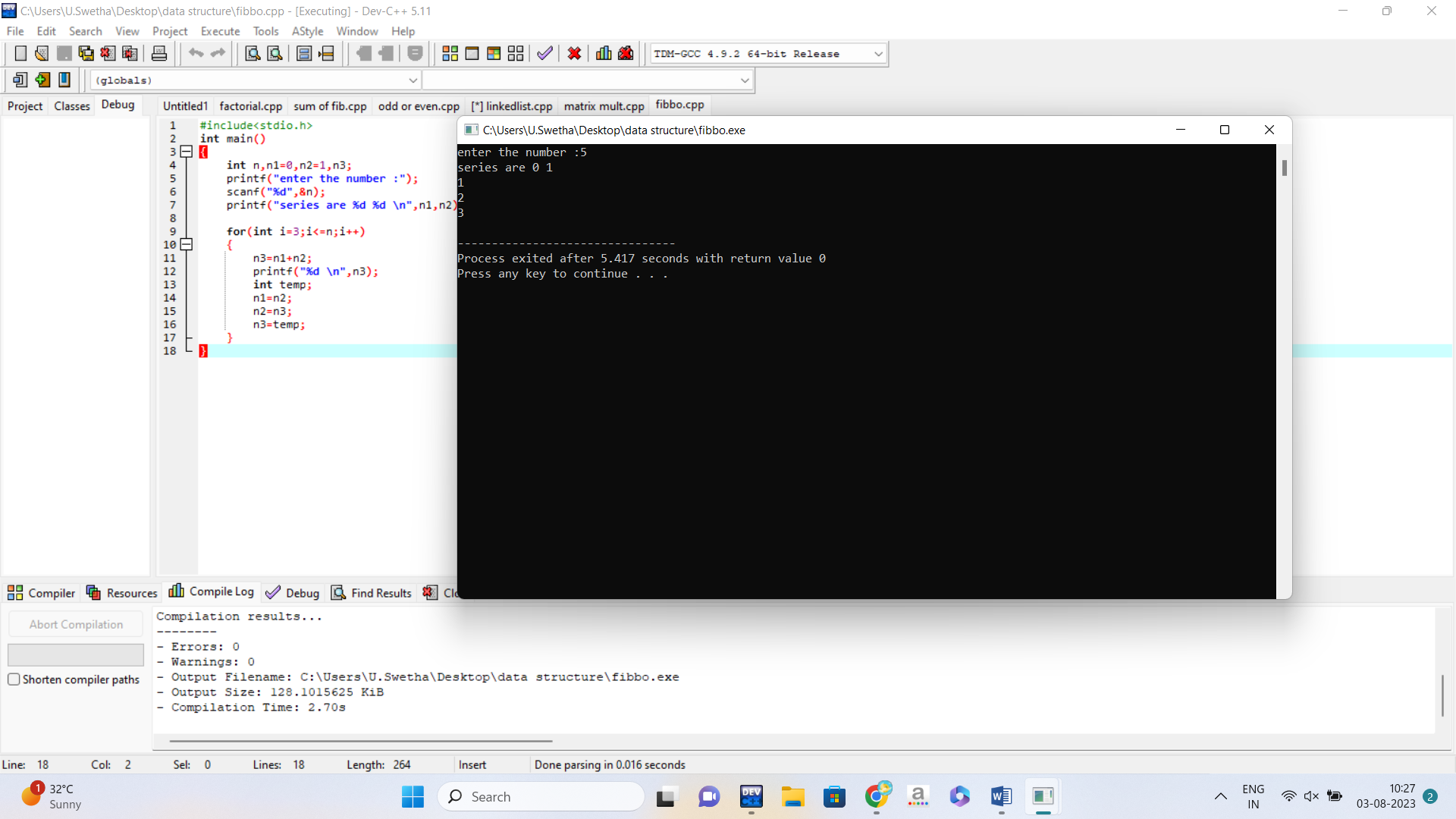
n1=n2;

n2=n3;

n3=temp;

}

}

3

3.FACTORIAL:

#include<stdio.h>

int main()

{

int n,fact=1;

printf("enter the number: ");

scanf("%d",&n);

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

{

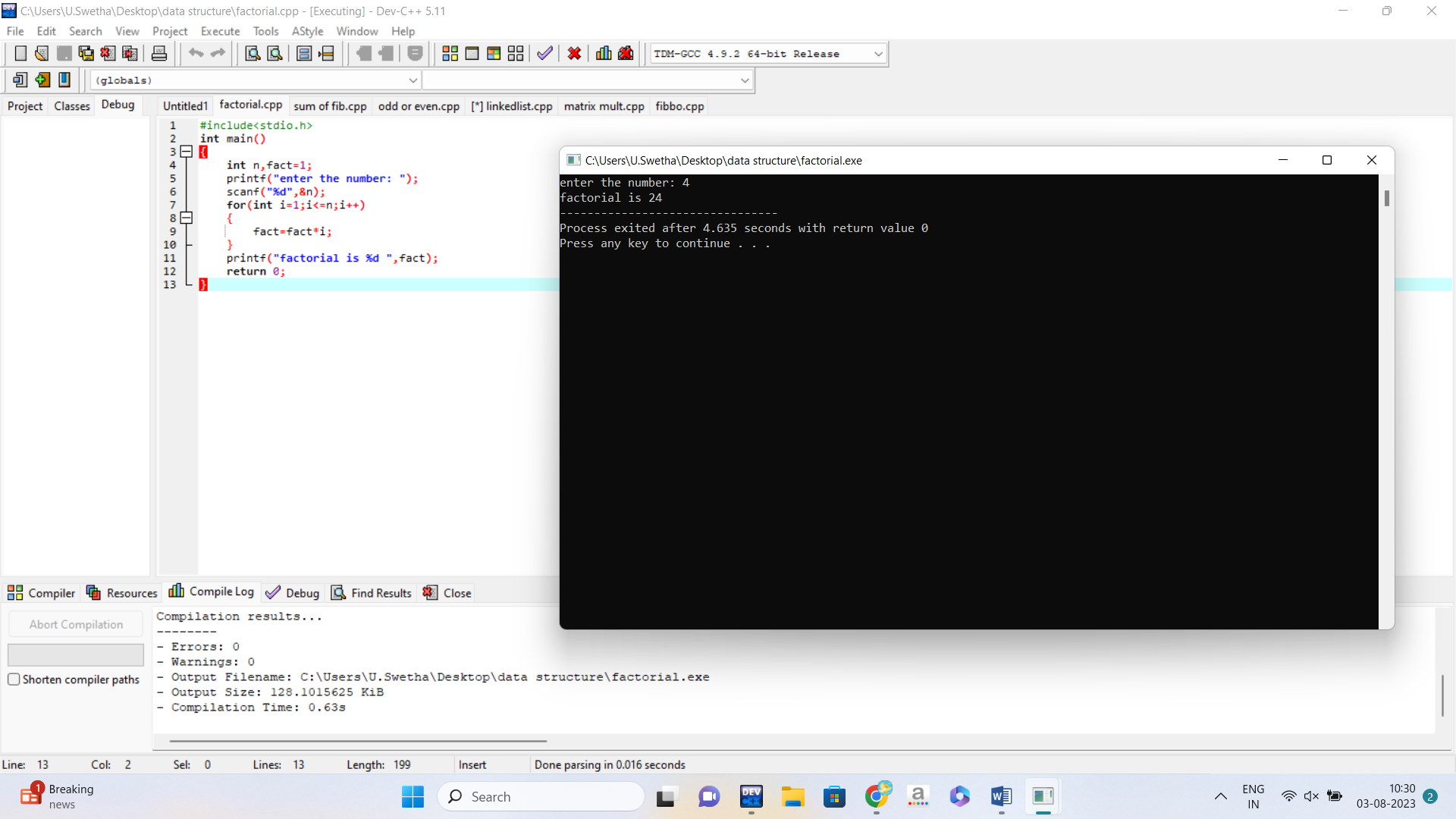
fact=fact\*i;

}

printf("factorial is %d ",fact);

return 0;

}



4. SUM OF FIBONACCI SERIES:

#include<stdio.h>

int main()

{

int n,n1=0,n2=1,n3,sum=0;

printf("enter the number :");

scanf("%d",&n);

printf("series are %d %d \n",n1,n2);

for(int i=3;i<=n;i++)

{

n3=n1+n2;

printf("%d \n",n3);

int temp;

n1=n2;

n2=n3;

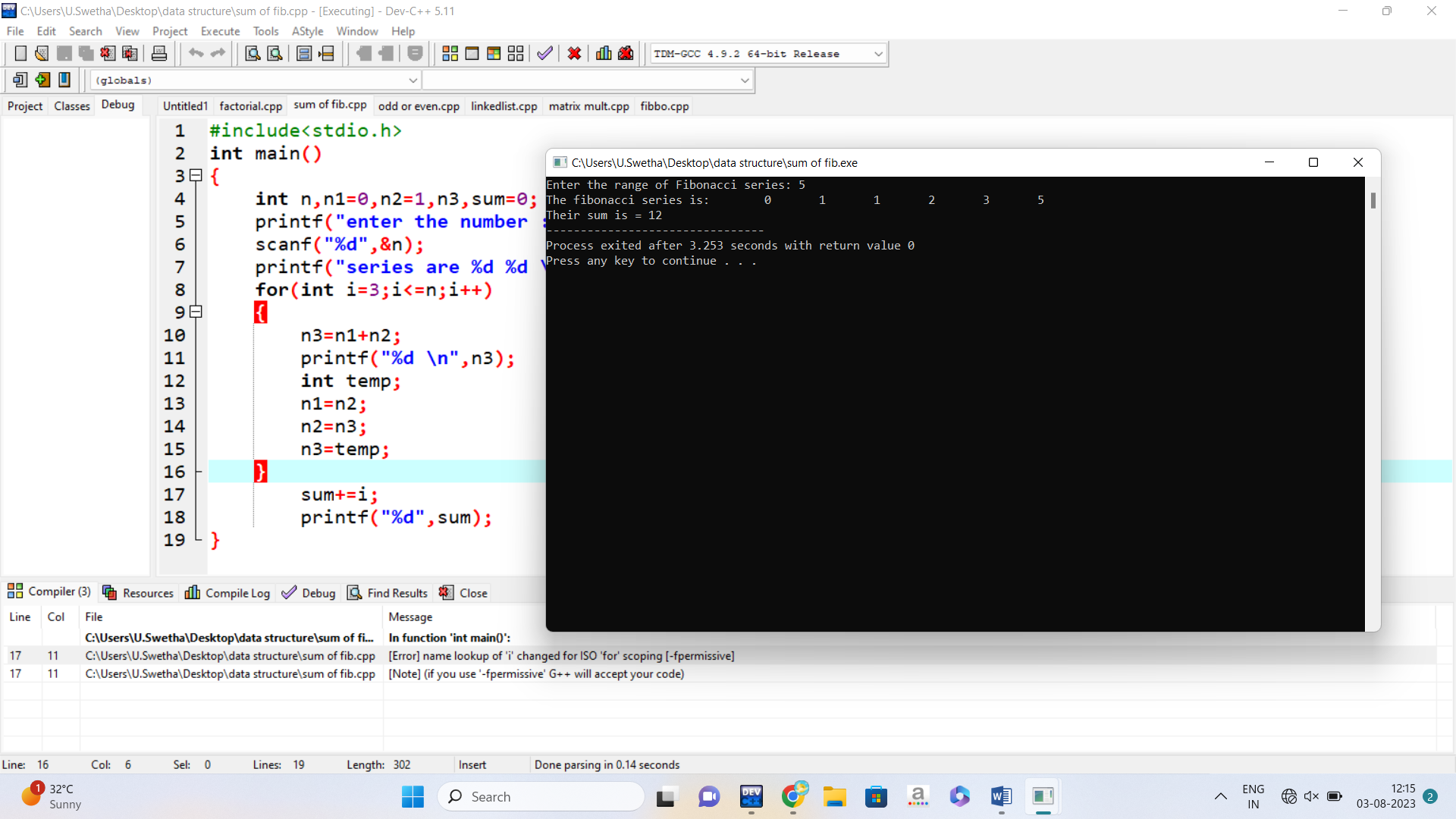
n3=temp;

}

sum+=i;

printf("%d",sum);

}



5.ARRAY OF ODD OR EVEN

#include <stdio.h>

int main()

{

int n;

printf("Enter number of elements in the array: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d elements in the array: ",n);

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

{

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

}

printf(" even numbers in the array are: ");

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

{

if(arr[i]&1==1);

else

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

}

printf("\n odd numbers in the array are: ");

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

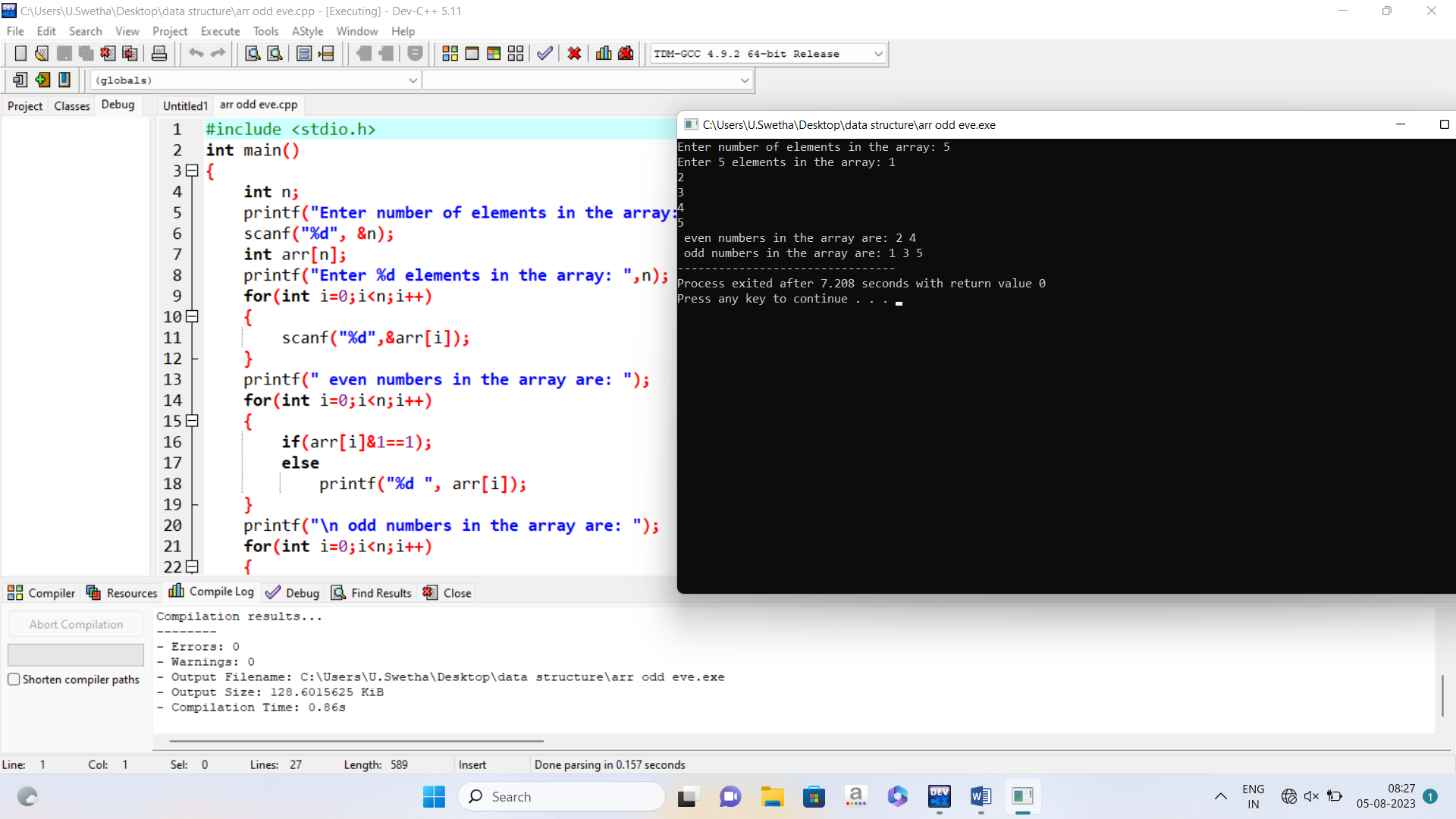
{

if(arr[i]&1==1)

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

}

}



6.LOCATION AN ELEMENT IN AN ARRAY

#include <stdio.h>

#include <conio.h>

int main()

{

int arr[10],i,n,k;

printf("Enter array size: ");

scanf("%d", &n);

printf("Enter array elements: ");

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

{

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

}

printf("Enter element to search: ");

scanf("%d", &k);

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

{

if(arr[i]==k)

{

printf("%d found at position %d", k, i+1);

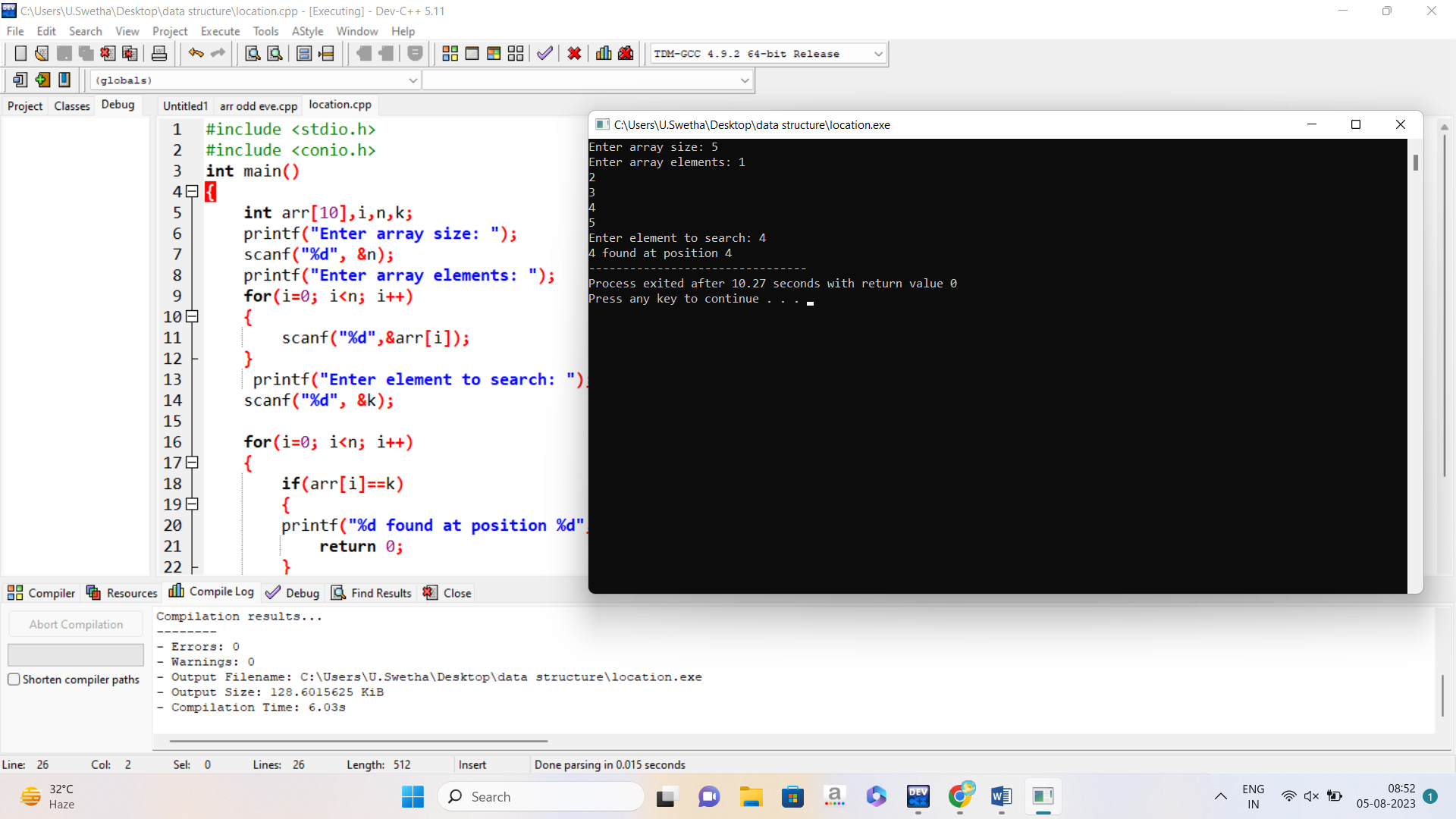
return 0;

}

}

printf("Element not found");

}



7. ROWS AND COLUMNS OF AN ARRAY

#include <stdio.h>

int main() {

int array[3][4] = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12}

};

int rows = sizeof(array) / sizeof(array[0]);

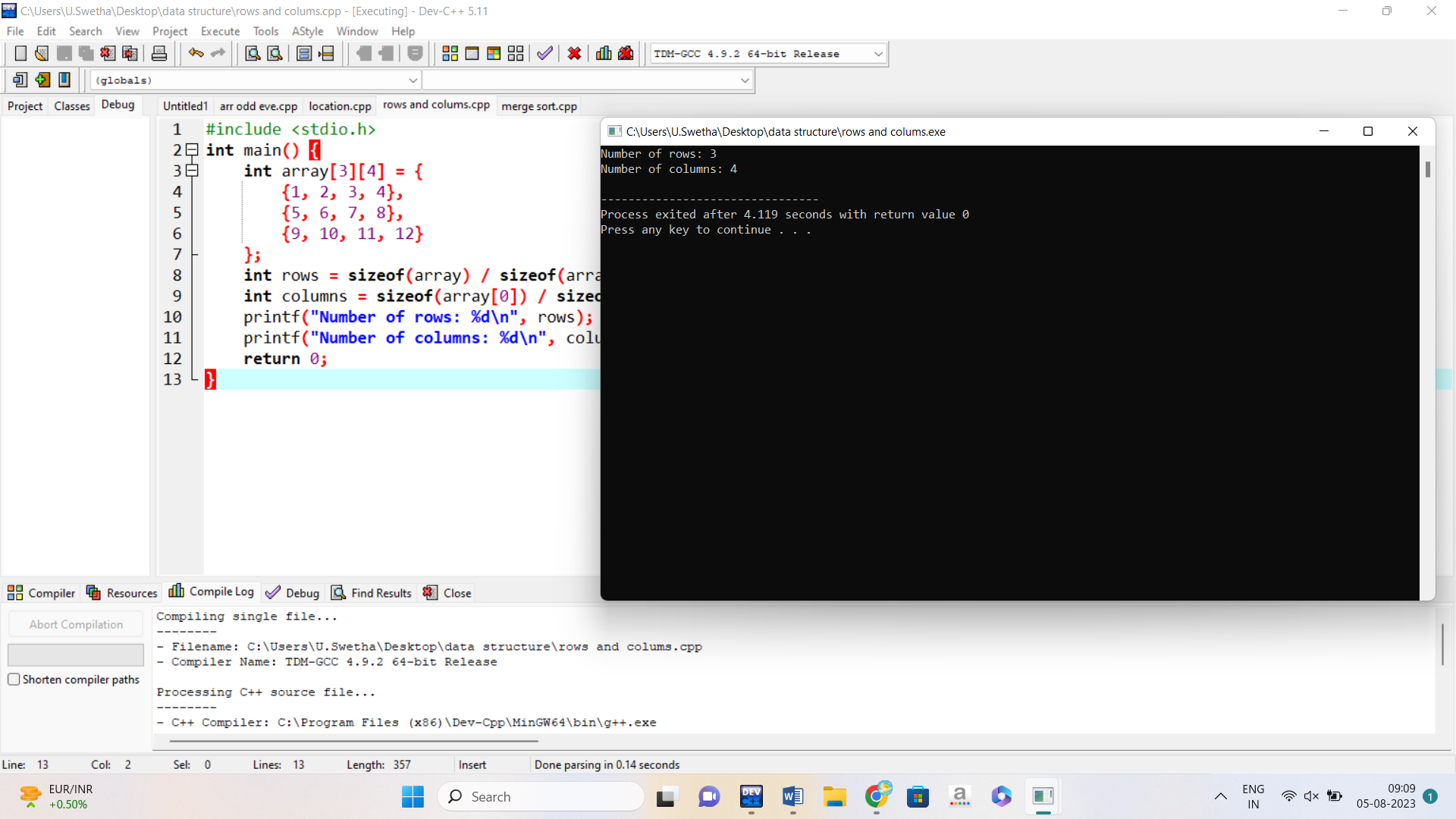
int columns = sizeof(array[0]) / sizeof(array[0][0]);

printf("Number of rows: %d\n", rows);

printf("Number of columns: %d\n", columns);

return 0;

}



8. SORT MERGING

#include <stdio.h>

#include <stdlib.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 A[], int size)

{

int i;

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

printf("%d ", A[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);

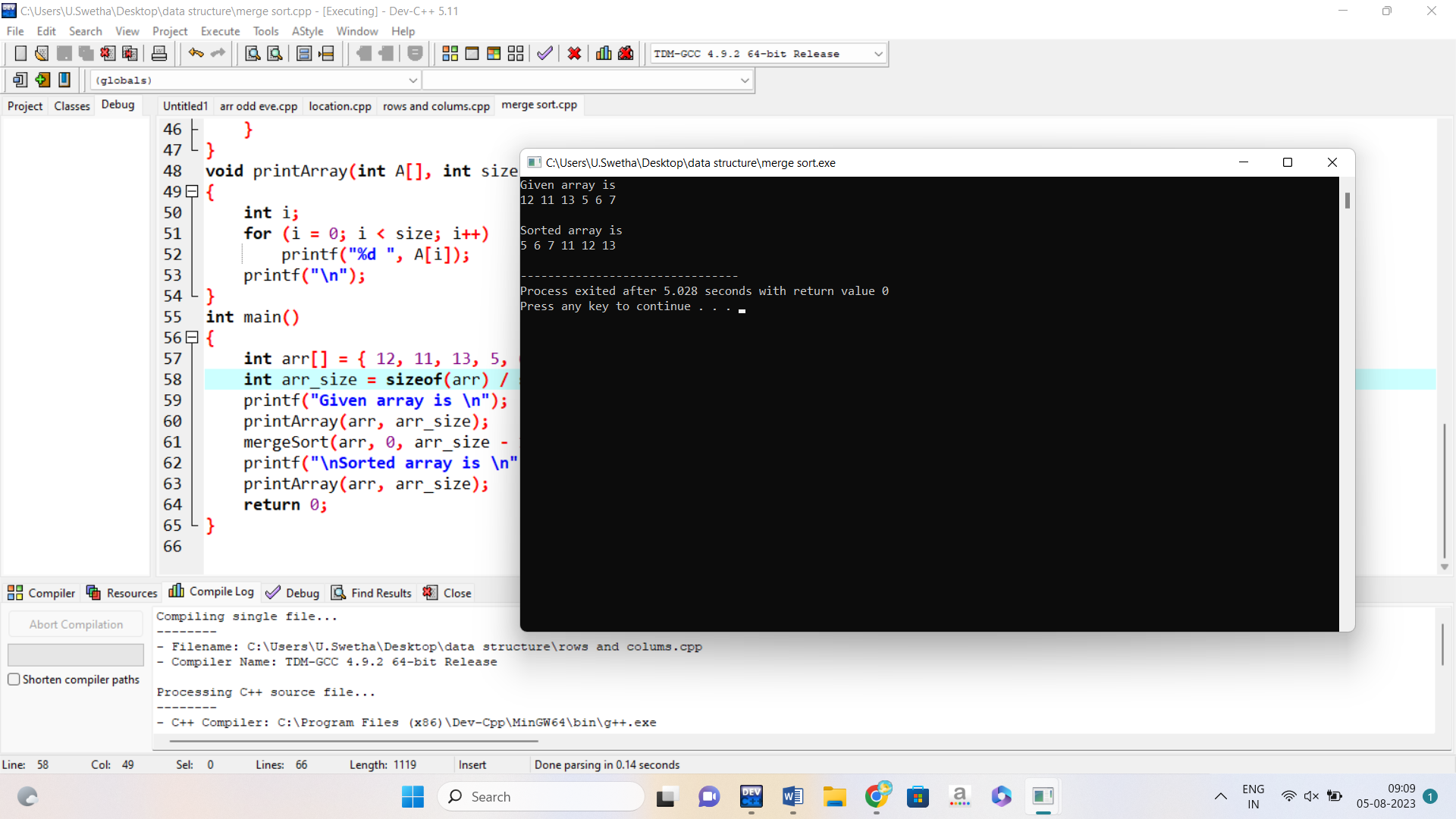
mergeSort(arr, 0, arr\_size - 1);

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

printArray(arr, arr\_size);

return 0;

}



9.DUPLICATION OF AN ARRAY

#include<stdio.h>

int main(){

int arr[100];

int n, i, j, k;

printf("Enter number of elements in array: ");

scanf("%d", &n);

printf("Enter %d elements: ", n);

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

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

}

printf("Repeating elements in the array are: ");

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

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

if(arr[i] == arr[j]){

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

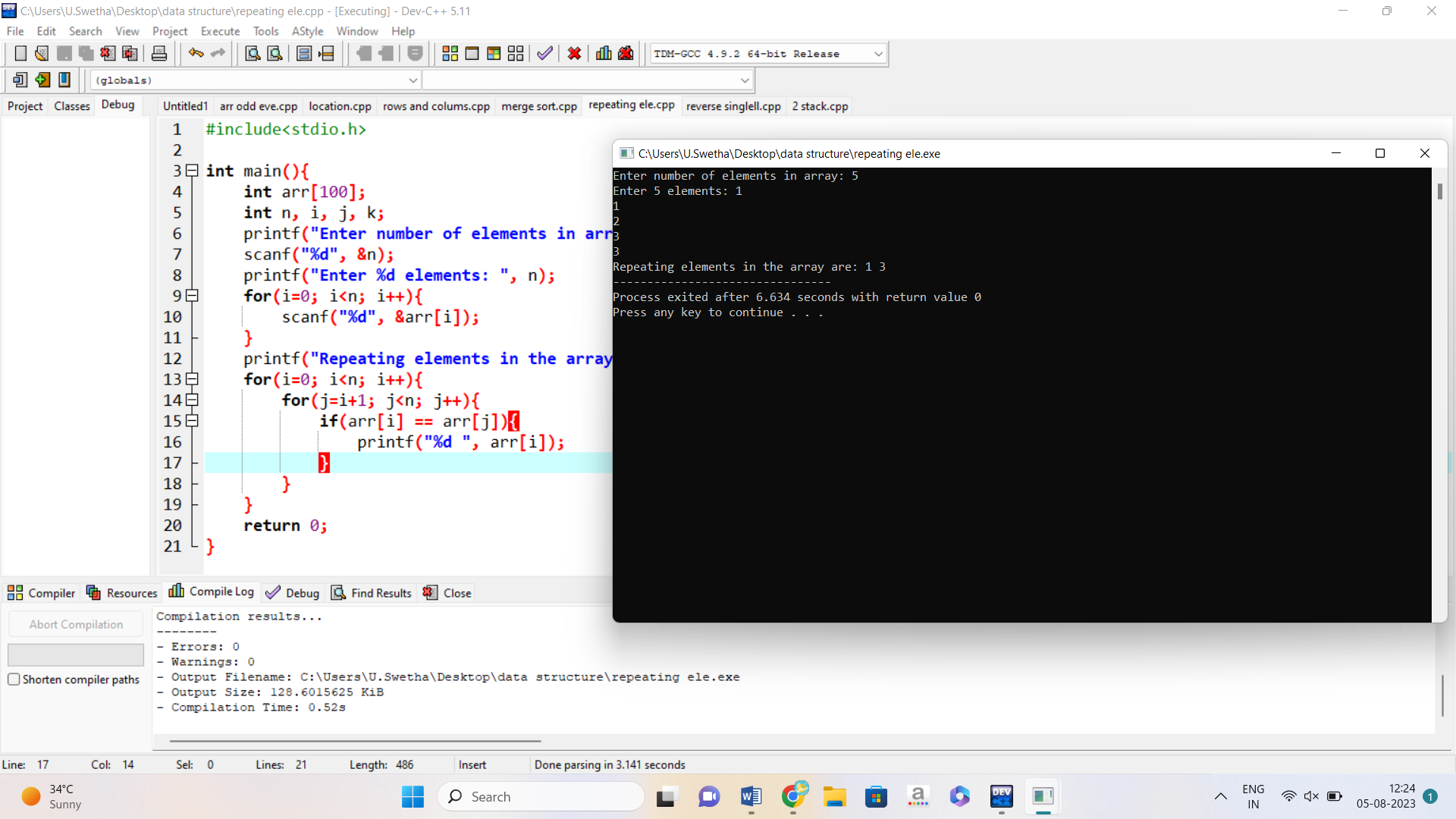
}

}

}

return 0;

}



10.MATRIX MULTIPLICATION

#include<stdio.h>

#include<stdlib.h>

int main(){

int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;

system("cls");

printf("enter the number of row=");

scanf("%d",&r);

printf("enter the number of column=");

scanf("%d",&c);

printf("enter the first matrix element=\n");

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

{

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

{

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

}

}

printf("enter the second matrix element=\n");

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

{

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

{

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

}

}

printf("multiply of the matrix=\n");

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

{

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

{

mul[i][j]=0;

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

{

mul[i][j]+=a[i][k]\*b[k][j];

}

}

}

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

{

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

{

printf("%d\t",mul[i][j]);

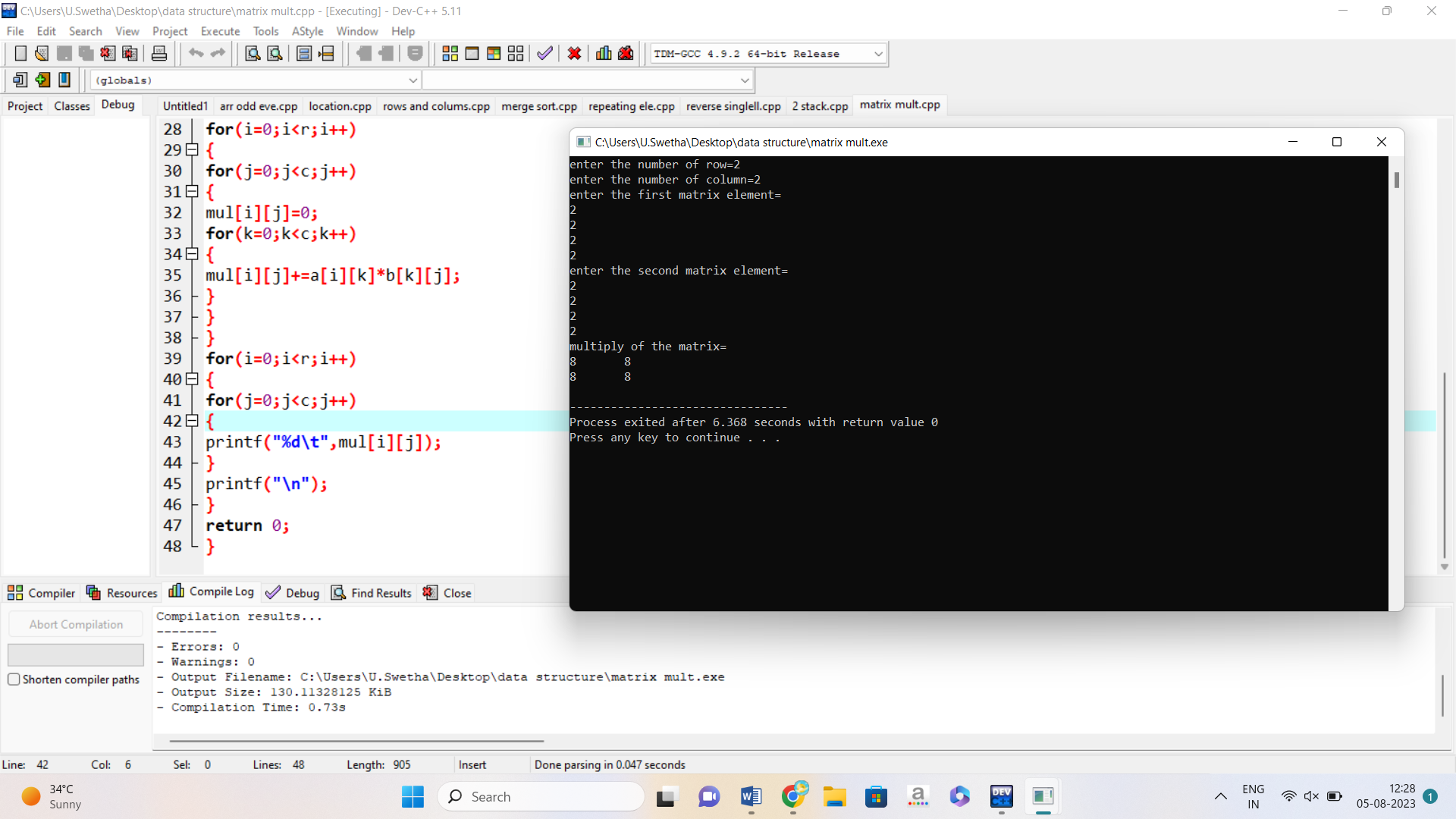
}

printf("\n");

}

return 0;

}



11. 32 BIT REVERSE A STRING

int main() {

int num, reversedNum;

printf("Enter a 32-bit signed integer: ");

scanf("%d", &num);

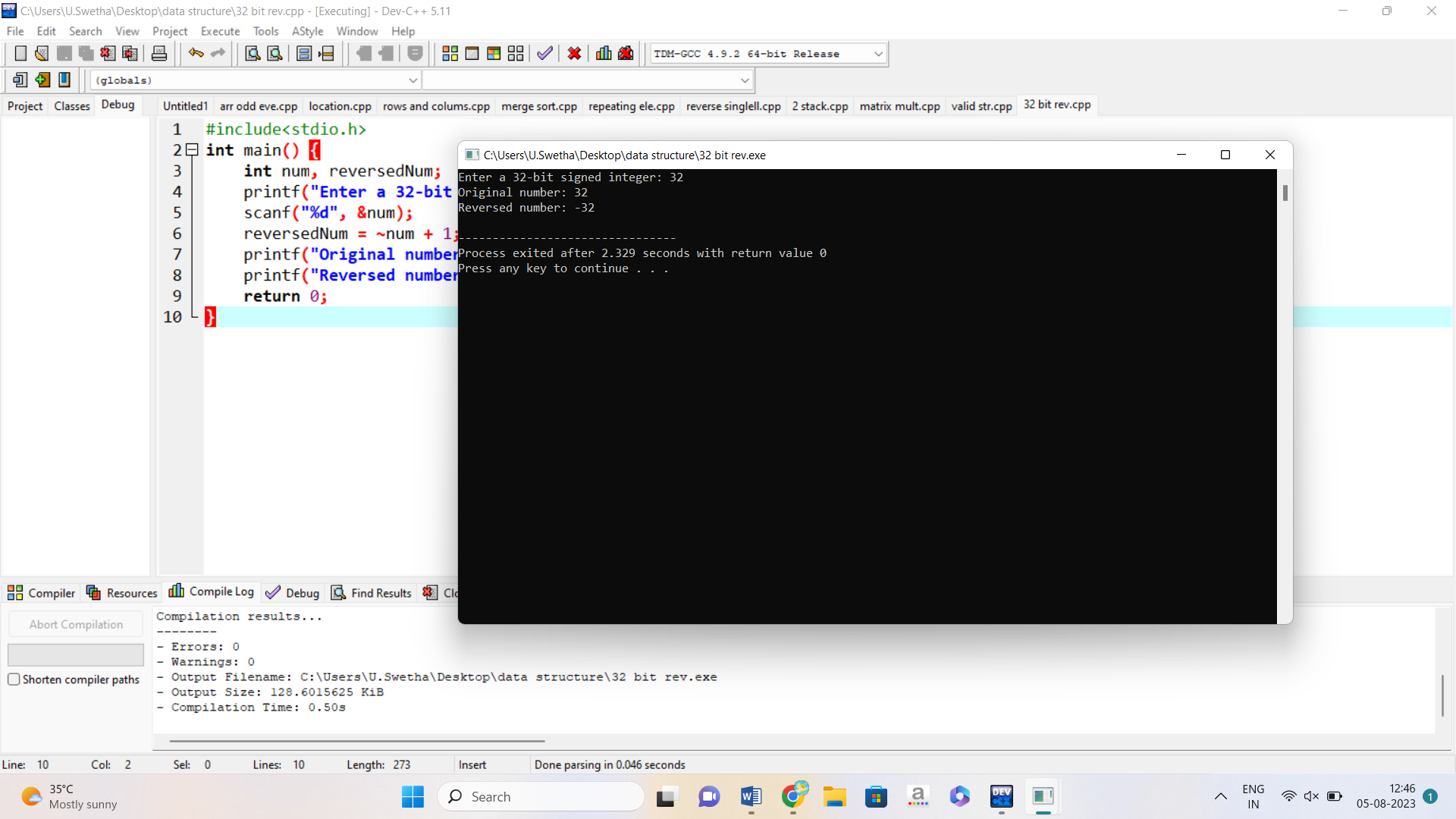
reversedNum = ~num + 1;

printf("Original number: %d\n", num);

printf("Reversed number: %d\n", reversedNum);

return 0;

}



12. LINKED LISTS

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void insertAtBeginning(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void insertAfter(struct Node\* prev\_node, int new\_data) {

if (prev\_node == NULL) {

printf("the given previous node cannot be NULL");

return;

}

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = prev\_node->next;

prev\_node->next = new\_node;

}

void insertAtEnd(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

struct Node\* last = head\_ref; // used in step 5\*/

new\_node->data = new\_data;

new\_node->next = NULL;

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

\*head\_ref = new\_node;

return;

}

while (last->next != NULL) last = last->next;

last->next = new\_node;

return;

}

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

struct Node \*temp = \*head\_ref, \*prev;

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

\*head\_ref = temp->next;

free(temp);

return;

}

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

prev = temp;

temp = temp->next;

}

if (temp == NULL) return;

prev->next = temp->next;

free(temp);

}

int searchNode(struct Node\*\* head\_ref, int key) {

struct Node\* current = \*head\_ref;

while (current != NULL) {

if (current->data == key) return 1;

current = current->next;

}

return 0;

}

void sortLinkedList(struct Node\*\* head\_ref) {

struct Node \*current = \*head\_ref, \*index = NULL;

int temp;

if (head\_ref == NULL) {

return;

} else {

while (current != NULL) {

index = current->next;

while (index != NULL) {

if (current->data > index->data) {

temp = current->data;

current->data = index->data;

index->data = temp;

}

index = index->next;

}

current = current->next;

}

}

}

void printList(struct Node\* node) {

while (node != NULL) {

printf(" %d ", node->data);

node = node->next;

}

}

int main() {

struct Node\* head = NULL;

insertAtEnd(&head, 1);

insertAtBeginning(&head, 2);

insertAtBeginning(&head, 3);

insertAtEnd(&head, 4);

insertAfter(head->next, 5);

printf("Linked list: ");

printList(head);

printf("\nAfter deleting an element: ");

deleteNode(&head, 3);

printList(head);

int item\_to\_find = 3;

if (searchNode(&head, item\_to\_find)) {

printf("\n%d is found", item\_to\_find);

} else {

printf("\n%d is not found", item\_to\_find);

}

sortLinkedList(&head);

printf("\nSorted List: ");

printList(head);

}

13.STACK USING PUSH POP AND PEEK

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#define MAX 3

int st[MAX], top=-1;

void push(int st[], int val);

int pop(int st[]);

int peek(int st[]);

void display(int st[]);

int main(int argc, char \*argv[]) {

int val, option;

do

{

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

printf("\n 1. PUSH");

printf("\n 2. POP");

printf("\n 3. PEEK");

printf("\n 4. DISPLAY");

printf("\n 5. EXIT");

printf("\n Enter your option: ");

scanf("%d", &option);

switch(option)

{

case 1:

printf("\n Enter the number to be pushed on stack: ");

scanf("%d", &val);

push(st, val);

break;

case 2:

val = pop(st);

if(val != -1)

printf("\n The value deleted from stack is: %d", val);

break;

case 3:

val = peek(st);

if(val != -1)

printf("\n The value stored at top of stack is: %d", val);

break;

case 4:

display(st);

break;

}

}while(option != 5);

return 0;

}

void push(int st[], int val)

{

if(top == MAX-1)

{

printf("\n STACK OVERFLOW");

}

else

{

top++;

st[top] = val;

}

}

int pop(int st[])

{

int val;

if(top == -1)

{

printf("\n STACK UNDERFLOW");

return -1;

}

else

{

val = st[top];

top--;

return val;

}

}

void display(int st[])

{

int i;

if(top == -1)

{

printf("\n STACK IS EMPTY");

}

else

{

for(i=top;i>=0;i--)

printf("\n %d",st[i]);

printf("\n");

}

}

int peek(int st[])

{

if(top == -1)

{

printf("\n STACK IS EMPTY");

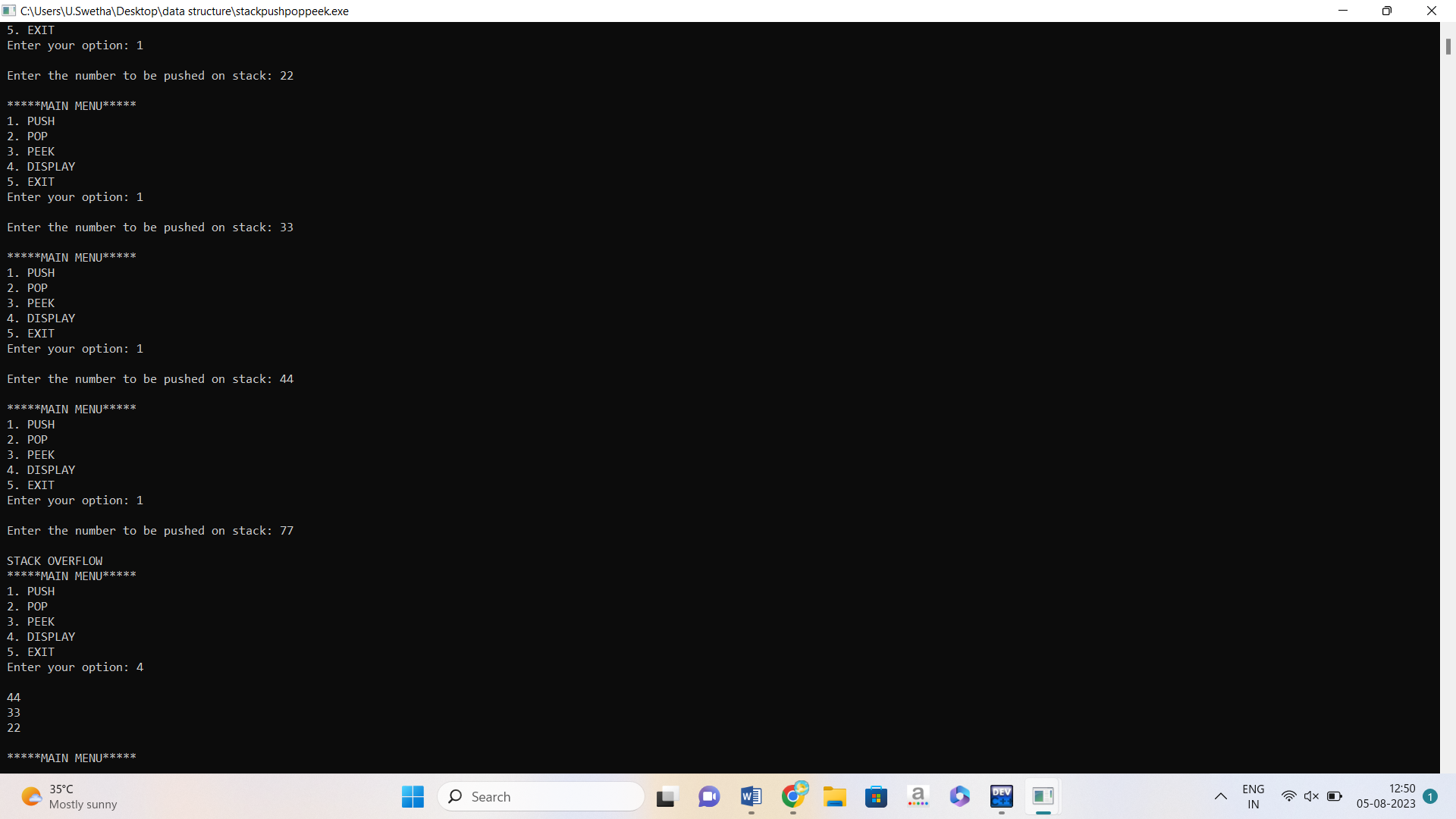
return -1;

}

else

return (st[top]);

}



14. STACK USING LINKED LISTS

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head = NULL;

void push(int val)

{

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

newNode->data = val;

newNode->next = head;

head = newNode;

}

void pop()

{

struct node \*temp;

if(head == NULL)

printf("Stack is Empty\n");

else

{

printf("Poped element = %d\n", head->data);

temp = head;

head = head->next;

free(temp);

}

}

void printList()

{

struct node \*temp = head;

while(temp != NULL)

{

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

temp = temp->next;

}

printf("NULL\n");

}

int main()

{

push(10);

push(20);

push(30);

printf("Linked List\n");

printList();

pop();

printf("After the pop, the new linked list\n");

printList();

pop();

printf("After the pop, the new linked list\n");

printList();

return 0;

}

15.QUEUE USING ARRAY

#include <stdio.h>

#define MAX 50

void insert();

void delete();

void display();

int queue\_array[MAX];

int rear = - 1;

int front = - 1;

main()

{

int choice;

while (1)

{

printf("1.Insert element to queue \n");

printf("2.Delete element from queue \n");

printf("3.Display all elements of queue \n");

printf("4.Quit \n");

printf("Enter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(1);

default:

printf("Wrong choice \n");

}

}

}

void insert()

{

int add\_item;

if (rear == MAX - 1)

printf("Queue Overflow \n");

else

{

if (front == - 1)

front = 0;

printf("Inset the element in queue : ");

scanf("%d", &add\_item);

rear = rear + 1;

queue\_array[rear] = add\_item;

}

}

void delete()

{

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

{

printf("Queue Underflow \n");

return ;

}

else

{

printf("Element deleted from queue is : %d\n", queue\_array[front]);

front = front + 1;

}

}

void display()

{

int i;

if (front == - 1)

printf("Queue is empty \n");

else

{

printf("Queue is : \n");

for (i = front; i <= rear; i++)

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

printf("\n");

}

}

16.QUEUE USING LINKED LISTS

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*front,\*rear,\*temp,\*front1;

int frontelement();

void enq(int data);

void deq();

void empty();

void display();

void create();

void queuesize();

int count = 0;

int main()

{

int no, ch, e;

printf("\n 1 - Enque");

printf("\n 2 - Deque");

printf("\n 3 - Front element");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Display");

printf("\n 7 - Queue size");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

enq(no);

break;

case 2:

deq();

break;

case 3:

e = frontelement();

if (e != 0)

printf("Front element : %d", e);

else

printf("\n No front element in Queue as queue is empty");

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

queuesize();

break;

default:

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

void create()

{

front = rear = NULL;

}

void queuesize()

{

printf("\n Queue size : %d", count);

}

void enq(int data)

{

if (rear == NULL)

{

rear = (struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = NULL;

rear->info = data;

front = rear;

}

else

{

temp=(struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = temp;

temp->info = data;

temp->ptr = NULL;

rear = temp;

}

count++;

}

void display()

{

front1 = front;

if ((front1 == NULL) && (rear == NULL))

{

printf("Queue is empty");

return;

}

while (front1 != rear)

{

printf("%d ", front1->info);

front1 = front1->ptr;

}

if (front1 == rear)

printf("%d", front1->info);

}

void deq()

{

front1 = front;

if (front1 == NULL)

{

printf("\n Error: Trying to display elements from empty queue");

return;

}

else

if (front1->ptr != NULL)

{

front1 = front1->ptr;

printf("\n Dequed value : %d", front->info);

free(front);

front = front1;

}

else

{

printf("\n Dequed value : %d", front->info);

free(front);

front = NULL;

rear = NULL;

}

count--;

}

int frontelement()

{

if ((front != NULL) && (rear != NULL))

return(front->info);

else

return 0;

}

void empty()

{

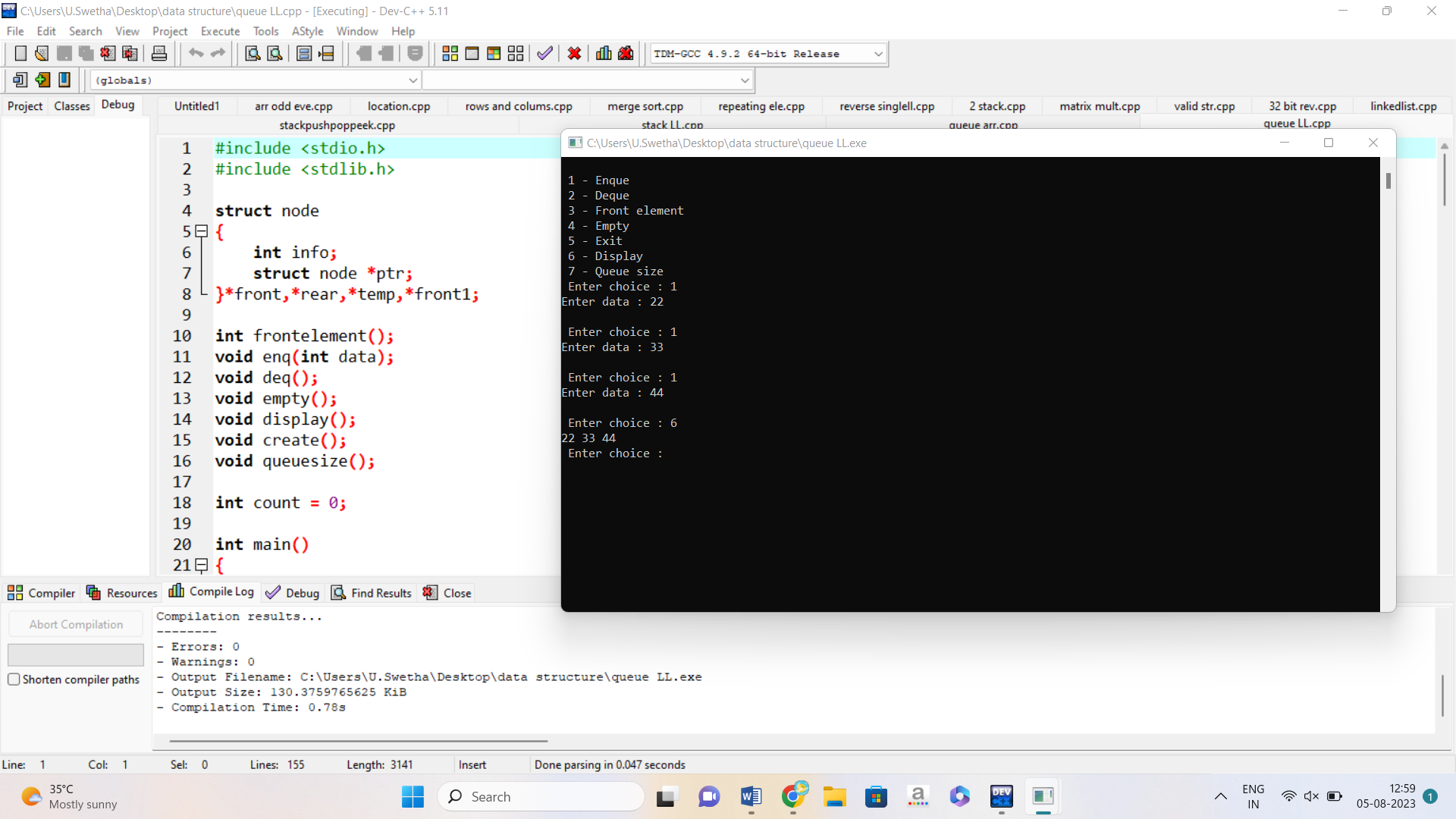
if ((front == NULL) && (rear == NULL))

printf("\n Queue empty");

else

printf("Queue not empty");

}



17. BINARY SEARCH

#include <stdio.h>

#include <stdlib.h>

struct node {

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item) {

struct node \*temp = (struct node \*)malloc(sizeof(struct node));

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void inorder(struct node \*root) {

if (root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

}

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

if (node == NULL) return newNode(key);

if (key < node->key)

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

else

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

return node;

}

struct node \*minValueNode(struct node \*node) {

struct node \*current = node;

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

current = current->left;

return current;

}

struct node \*deleteNode(struct node \*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) {

struct node \*temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

struct node \*temp = root->left;

free(root);

return temp;

}

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

root->key = temp->key;

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

}

return root;

}

int main() {

struct node \*root = NULL;

root = insert(root, 8);

root = insert(root, 3);

root = insert(root, 1);

root = insert(root, 6);

root = insert(root, 7);

root = insert(root, 10);

root = insert(root, 14);

root = insert(root, 4);

printf("Inorder traversal: ");

inorder(root);

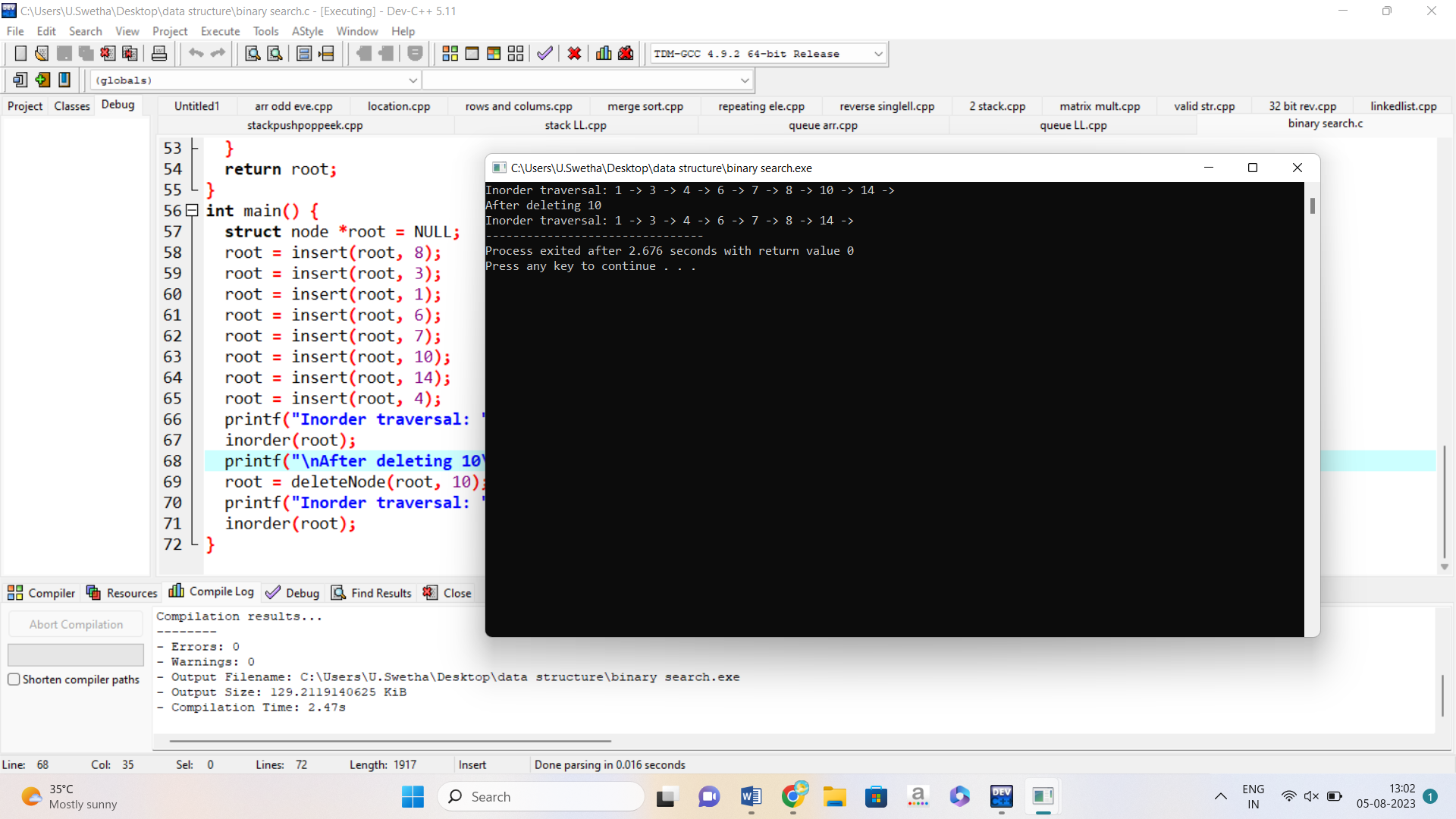
printf("\nAfter deleting 10\n");

root = deleteNode(root, 10);

printf("Inorder traversal: ");

inorder(root);

}

18

18. BINARY TREE

#include<stdio.h>

#include<stdlib.h>

struct node

{

int key;

struct node \*left;

struct node \*right;

};

struct node \*getNode(int val)

{

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->key = val;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct node \*insertNode(struct node \*root, int val)

{

if(root == NULL)

return getNode(val);

if(root->key < val)

root->right = insertNode(root->right,val);

if(root->key > val)

root->left = insertNode(root->left,val);

return root;

}

void inorder(struct node \*root)

{

if(root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

int main()

{

struct node \*root = NULL;

int data;

char ch;

do

{

printf("\nSelect one of the operations::");

printf("\n1. To insert a new node in the Binary Tree");

printf("\n2. To display the nodes of the Binary Tree(via Inorder Traversal).\n");

int choice;

scanf("%d",&choice);

switch (choice)

{

case 1 :

printf("\nEnter the value to be inserted\n");

scanf("%d",&data);

root = insertNode(root,data);

break;

case 2 :

printf("\nInorder Traversal of the Binary Tree::\n");

inorder(root);

break;

default :

printf("Wrong Entry\n");

break;

}

printf("\nDo you want to continue (Type y or n)\n");

scanf(" %c",&ch);

} while (ch == 'Y'|| ch == 'y');

return 0;

}

19. LINEAR SEARCH

#include <stdio.h>

int main()

{

int array[100], search, c, n;

printf("Enter number of elements in array\n");

scanf("%d", &n);

printf("Enter %d integer(s)\n", n);

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

scanf("%d", &array[c]);

printf("Enter a number to search\n");

scanf("%d", &search);

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

{

if (array[c] == search)

{

printf("%d is present at location %d.\n", search, c+1);

break;

}

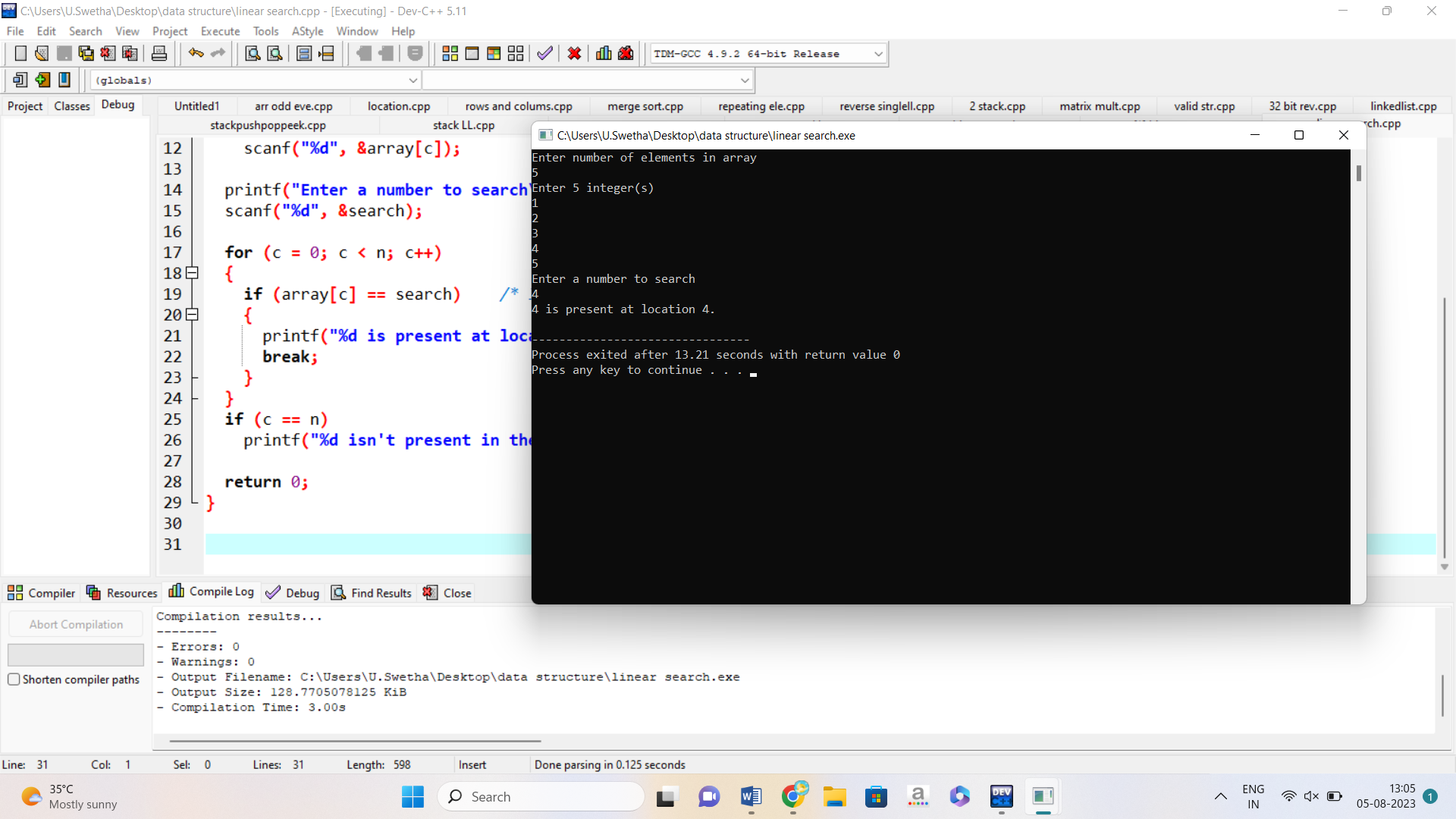
}

if (c == n)

printf("%d isn't present in the array.\n", search);

return 0;

}



20.FACTORIAL USING RECURSION

#include<stdio.h>

long int multiplyNumbers(int n);

int main() {

int n;

printf("Enter a positive integer: ");

scanf("%d",&n);

printf("Factorial of %d = %ld", n, multiplyNumbers(n));

return 0;

}

long int multiplyNumbers(int n) {

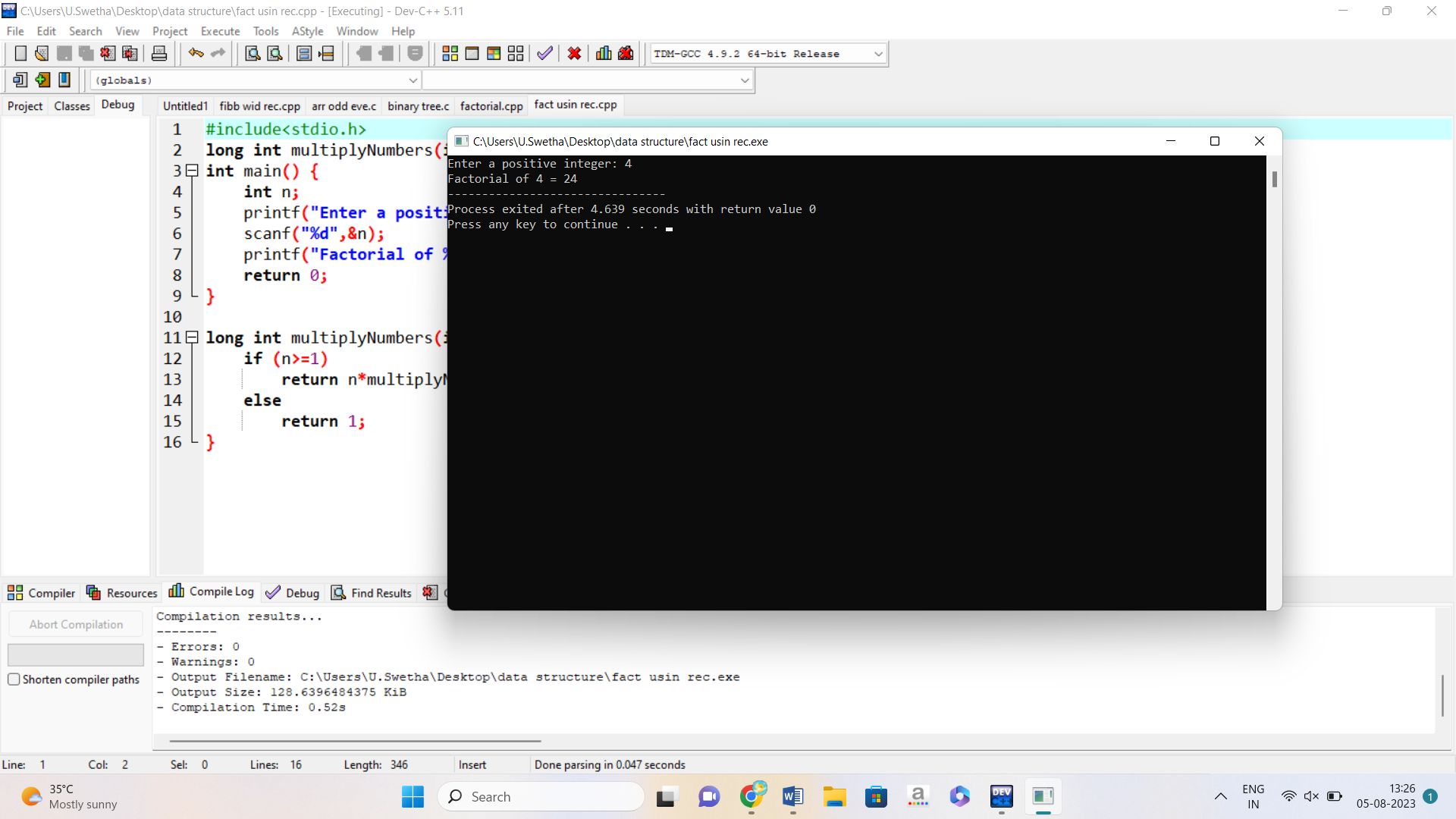
if (n>=1)

return n\*multiplyNumbers(n-1);

else

return 1;

}



21.GRAPH TRAVERSAL USING BREADTH FIRST SEARCH

#include<stdio.h>

#include<conio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v) {

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

if(a[v][i] && !visited[i])

q[++r]=i;

if(f<=r) {

visited[q[f]]=1;

bfs(q[f++]);

}

}

int main() {

int v;

int clrscr();

printf("\n Enter the number of vertices:");

scanf("%d",&n);

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

q[i]=0;

visited[i]=0;

}

printf("\n Enter graph data in matrix form:\n");

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

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

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

printf("\n Enter the starting vertex:");

scanf("%d",&v);

bfs(v);

printf("\n The node which are reachable are:\n");

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

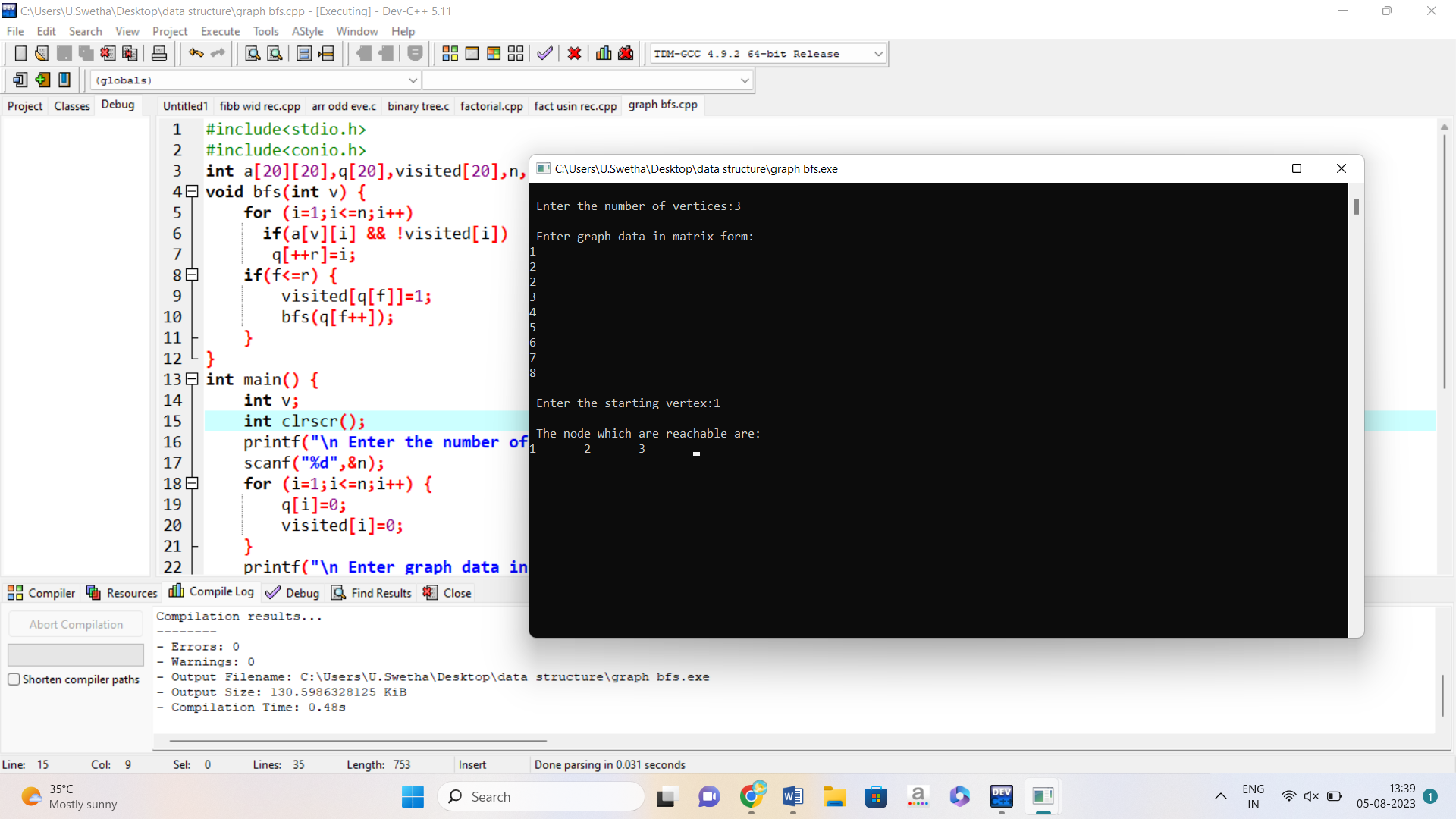
if(visited[i])

printf("%d\t",i); else

printf("\n Bfs is not possible");

getch();

}



22.GRAPH TRAVERSAL USING DEPTH FIRST SEARCH

#include<stdio.h>

#include<conio.h>

int a[20][20],reach[20],n;

void dfs(int v) {

int i;

reach[v]=1;

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

if(a[v][i] && !reach[i]) {

printf("\n %d->%d",v,i);

dfs(i);

}

}

int main() {

int i,j,count=0;

int clrscr();

printf("\n Enter number of vertices:");

scanf("%d",&n);

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

reach[i]=0;

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

a[i][j]=0;

}

printf("\n Enter the adjacency matrix:\n");

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

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

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

dfs(1);

printf("\n");

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

if(reach[i])

count++;

}

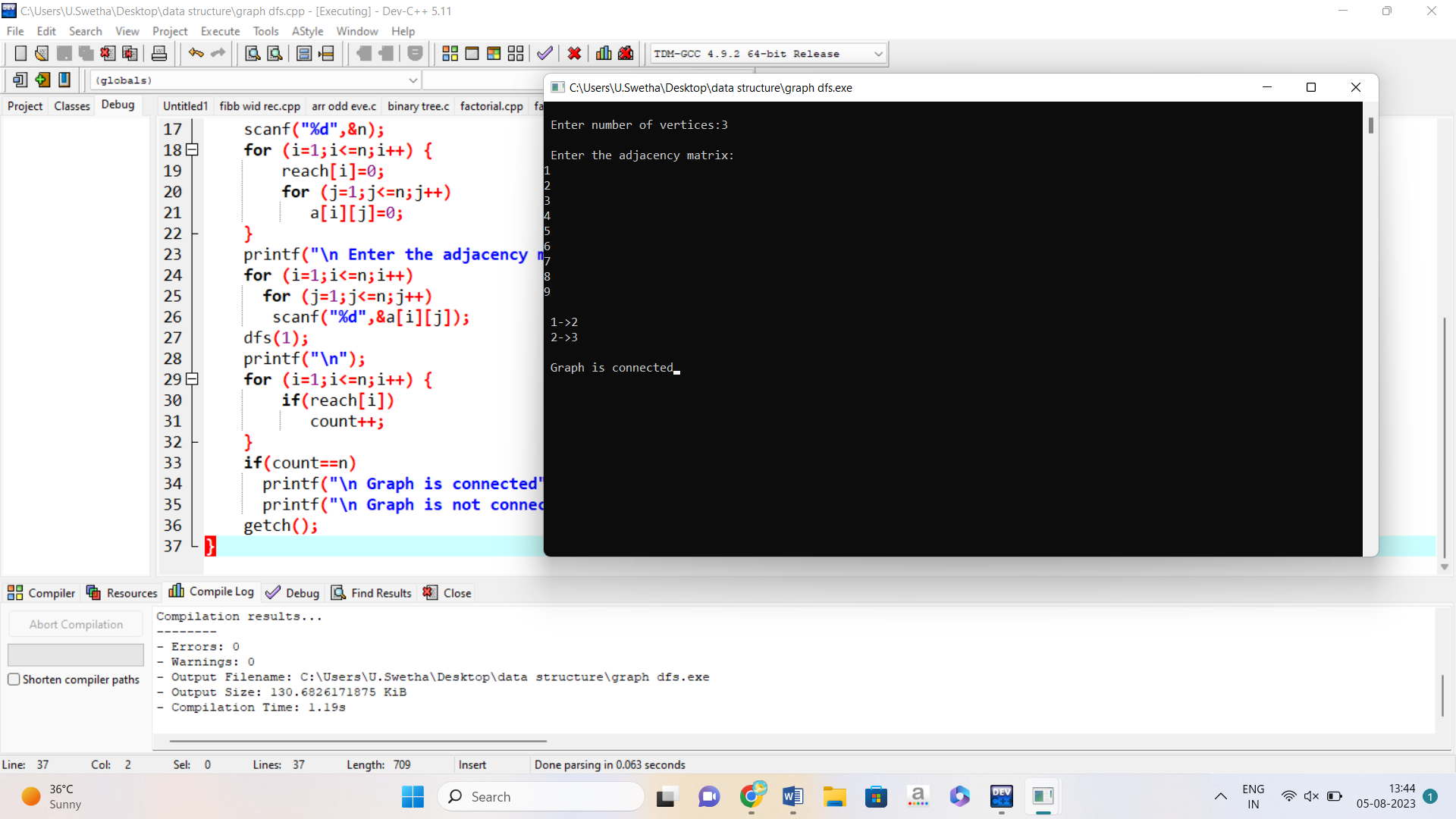
if(count==n)

printf("\n Graph is connected"); else

printf("\n Graph is not connected");

getch();

}



23.HEAP SORT

#include<stdio.h>

void heapify(int\*,int, int);

void heapsort(int\*, int);

void print\_array(int\*, int);

int main()

{

int arr[] = { 10, 30, 5, 63, 22, 12, 56, 33 };

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

printf("\nArray before sorting:\n");

print\_array(arr, n)

heapsort(arr, n);

printf("\n\nArray after sorting:\n");

print\_array(arr, n);

return 0;

}

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--)

{

int temp = arr[i];

arr[i] = arr[0];

arr[0] = temp;

heapify(arr, i, 0);

}

}

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)

{

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, n, largest);

}

}

void print\_array(int\* arr, int n)

{

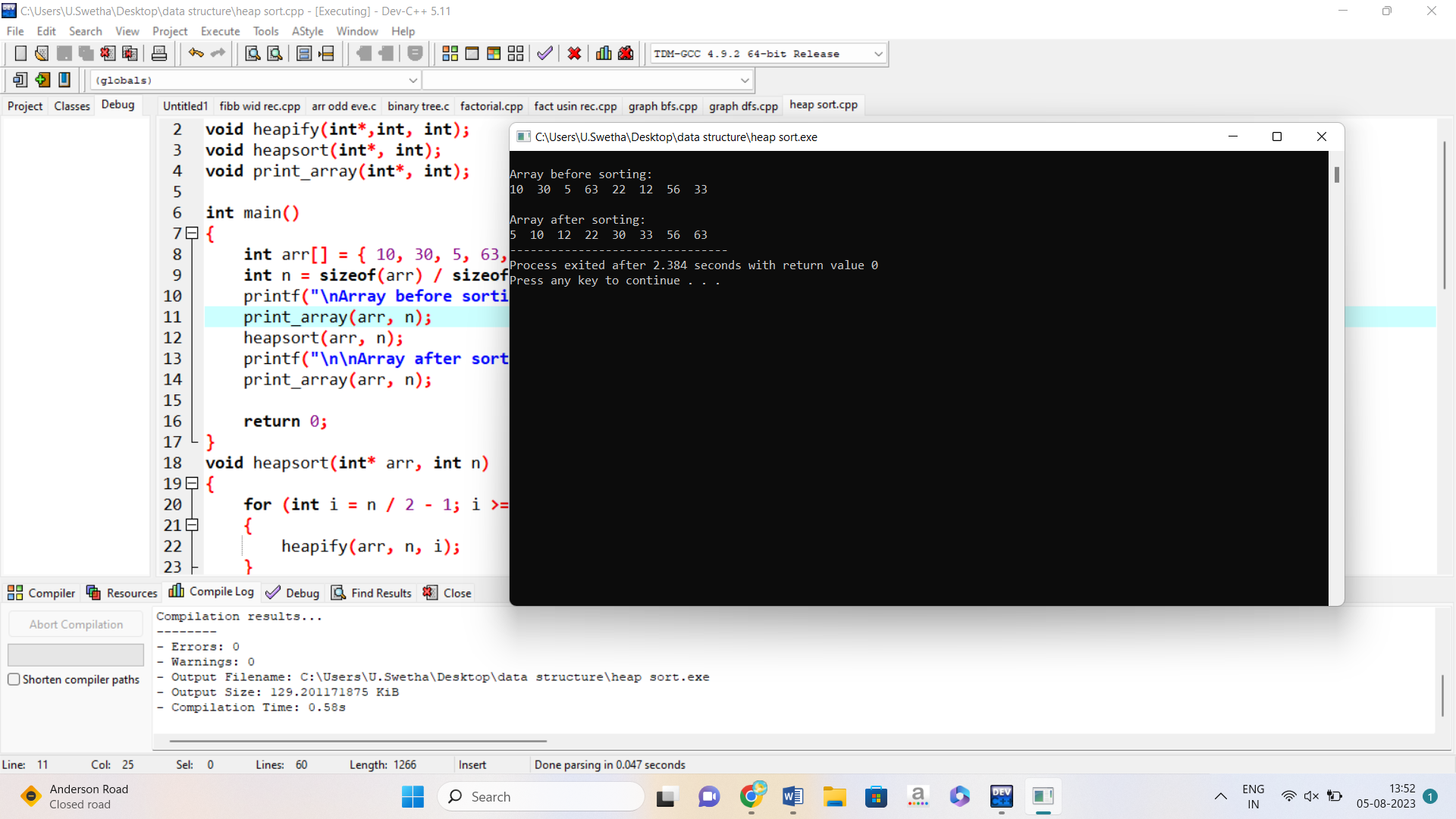
for (int i = 0; i < n; i++)

{

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

}

}



24. QUICK SORT

#include<stdio.h>

void quicksort(int number[25],int first,int last){

int i, j, pivot, temp;

if(first<last){

pivot=first;

i=first;

j=last;

while(i<j){

while(number[i]<=number[pivot]&&i<last)

i++;

while(number[j]>number[pivot])

j--;

if(i<j){

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}

}

int main(){

int i, count, number[25];

printf("How many elements are u going to enter?: ");

scanf("%d",&count);

printf("Enter %d elements: ", count);

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

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

quicksort(number,0,count-1);

printf("Order of Sorted elements: ");

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

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

return 0;

}

