**PROGRAM 1- MATRIX MULTIPLICATION**

include<stdio.h>

int main() {

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

printf("Enter the value of N (N <= 10): ");

scanf("%d", & n);

printf("Enter the elements of Matrix-A: \n");

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

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

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

}

}

printf("Enter the elements of Matrix-B: \n");

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

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

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

}

}

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

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

c[i][j] = 0;

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

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

}

}

}

printf("The product of the two matrices is: \n");

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

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

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

}

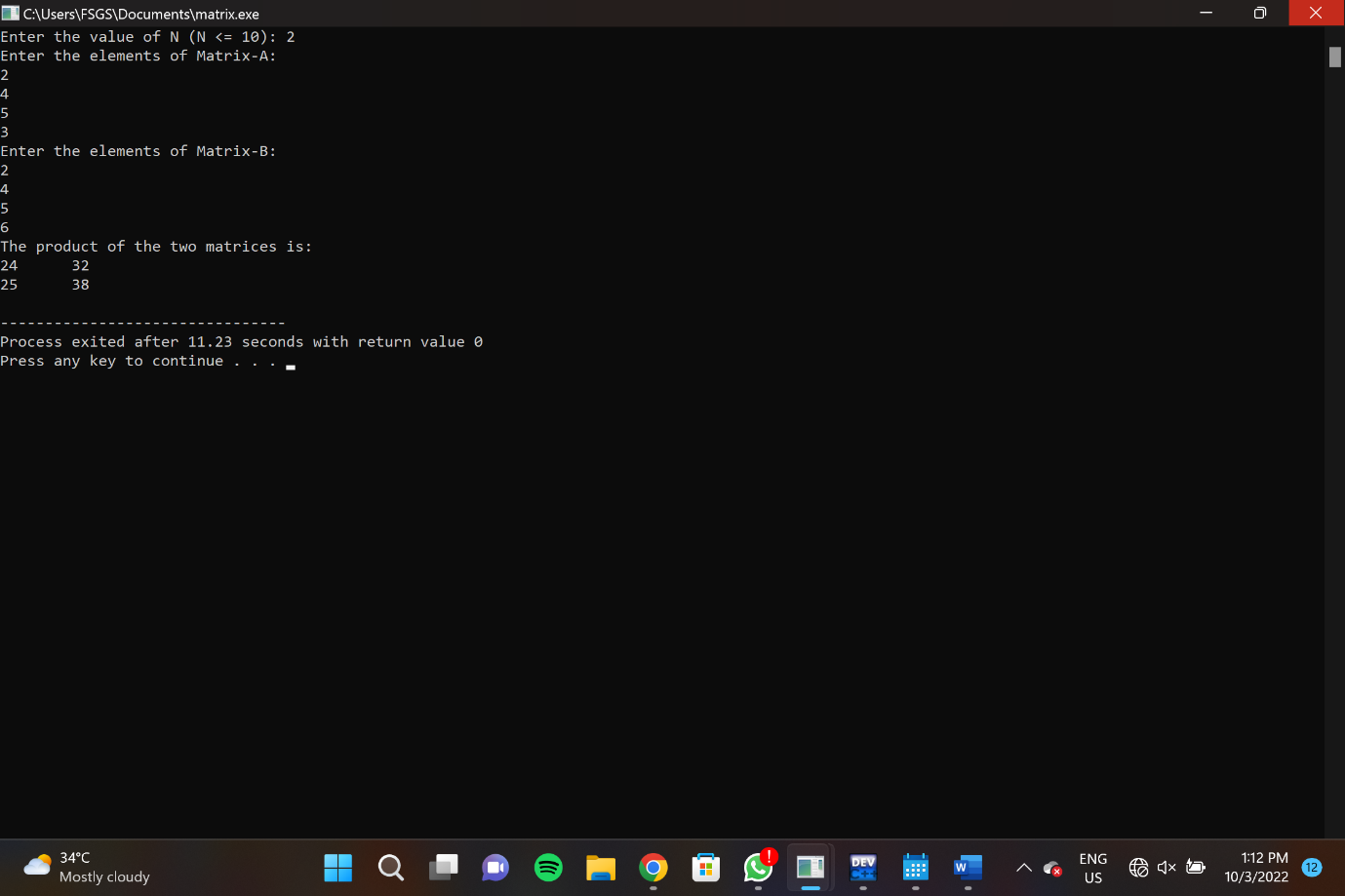
printf("\n");

}

return 0;

}

**OUTPUT**



**PROGRAM 2: ODD OR EVEN**

#include<stdio.h>

int main() {

int number;

printf("Enter any integer: ");

scanf("%d",&number);

if(number % 2 ==0)

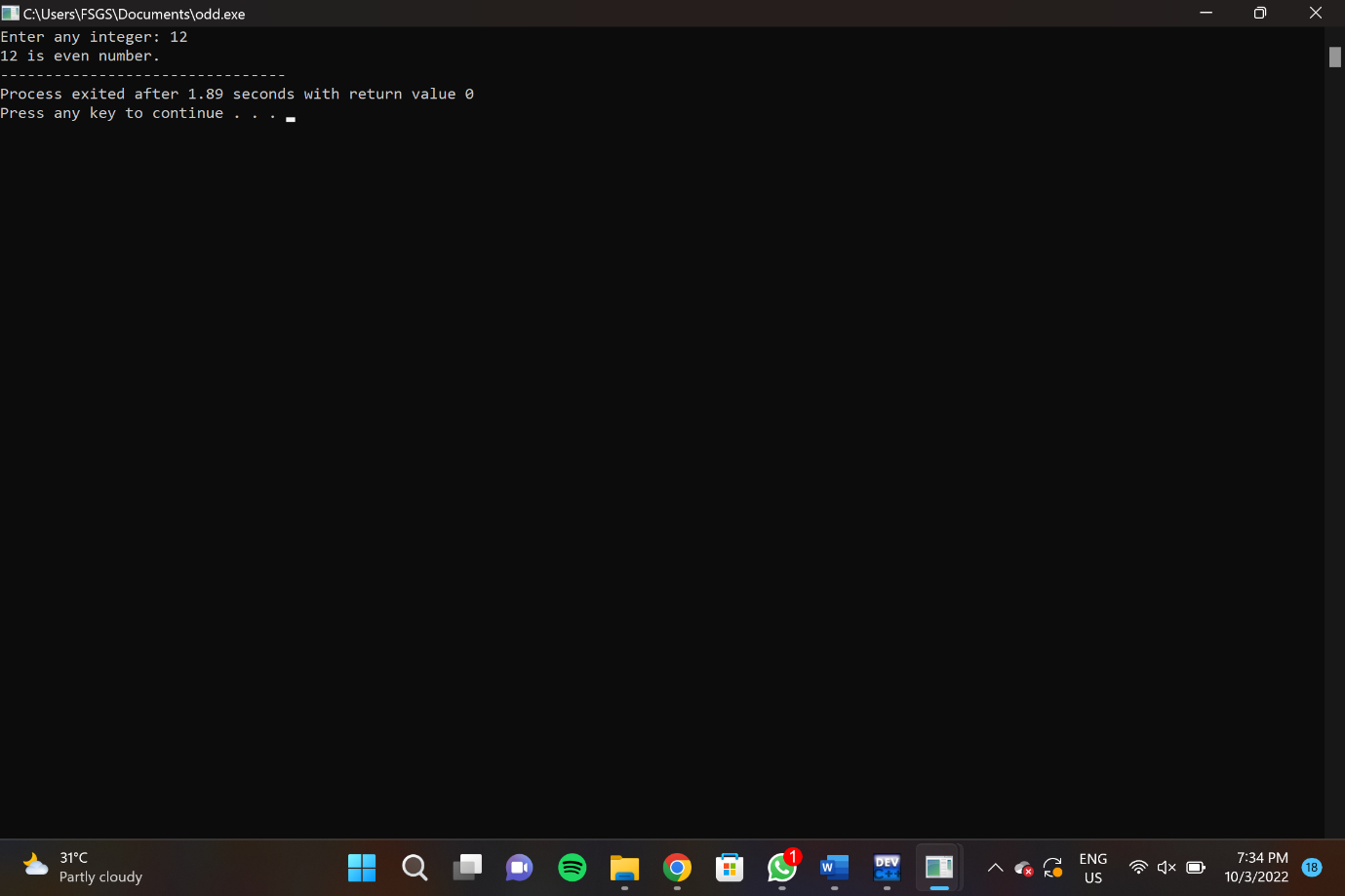
printf("%d is even number.",number); else

printf("%d is odd number.",number);

return 0;

}

**OUTPUT:**



**PROGRAM 3**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main()**

**{**

**int n, i;**

**unsigned long long factorial = 1;**

**printf("Enter a number to find factorial: ");**

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

**// show error if the user enters a negative integer**

**if (n < 0)**

**printf("Error! Please enter any positive integer number");**

**else**

**{**

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

**{**

**factorial \*= i; // factorial = factorial\*i;**

**}**

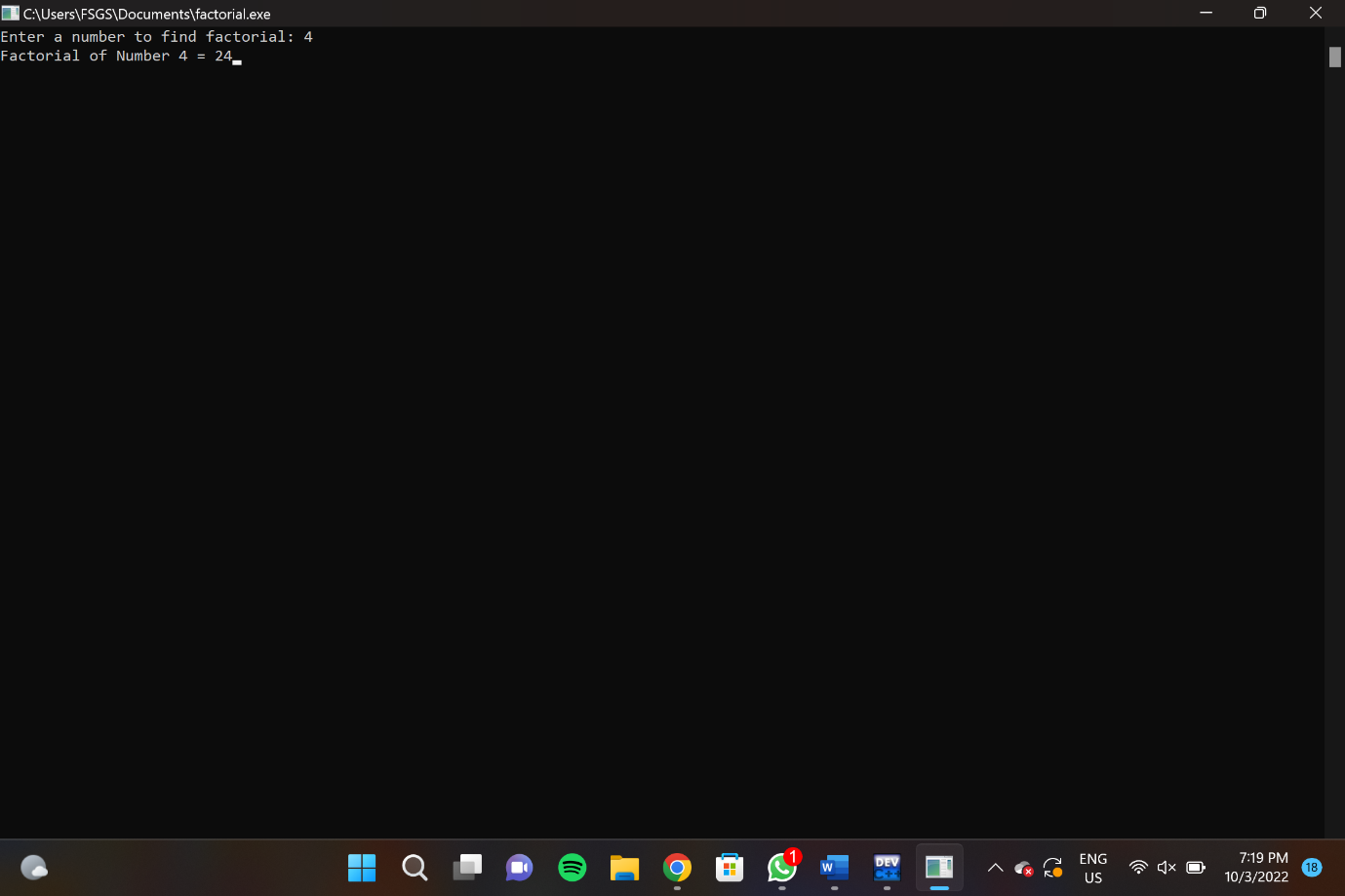
**printf("Factorial of Number %d = %llu", n, factorial);**

**}**

**getch();**

**}**

**OUTPUT:**

****

**PROGRAM 4- FIBONACCI SERIES WITHOUT RECURSION**

int main()

{

int i, n, firstTerm=0, secondTerm=1, sum=0;

printf("\nEnter number of terms required in Fibonacci Series: ");

scanf("%d",&n);

// Showing the first two term of the Fibonacci Series

printf("\nFibonacci Series is:\n\n\n %d %d ", firstTerm, secondTerm);

//start i =2

i=2;

//i starts from 2, as the first two terms of the series have already been shown

while (i<n)

{

sum=firstTerm+secondTerm;

firstTerm=secondTerm;

secondTerm=sum;

++i;

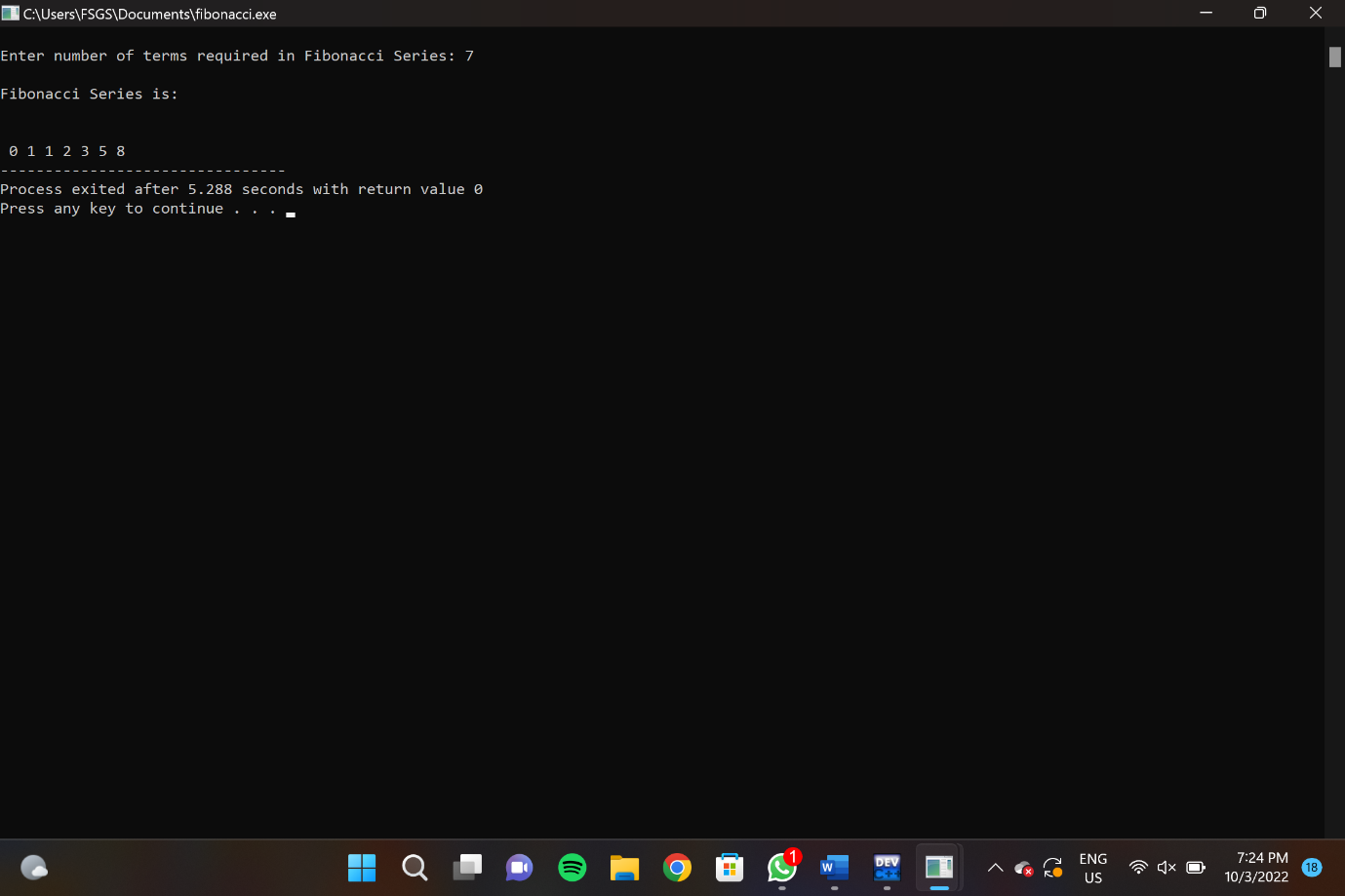
printf("%d ",sum);

}

return 0;

}

**OUTPUT:**

****

**PROGRAM 5- 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;

}

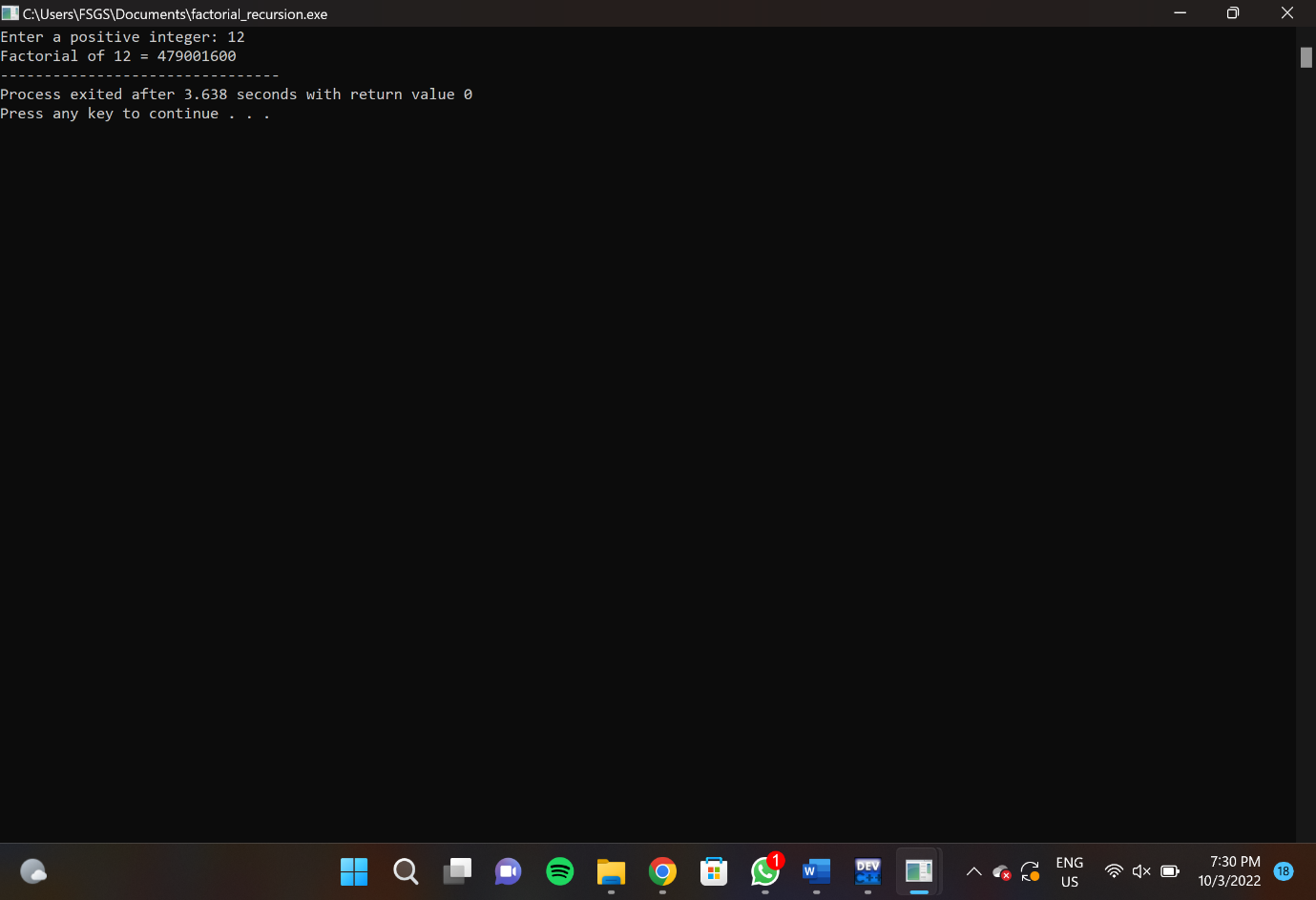
long int multiplyNumbers(int n) {

if (n>=1)

return n\*multiplyNumbers(n-1);

else

return 1;

}****

**PROGRAM 6: FIBONACCI SERIES USING RECURSION**

#include<stdio.h>

int main() {

int number;

printf("Enter any integer: ");

scanf("%d",&number);

if(number % 2 ==0)

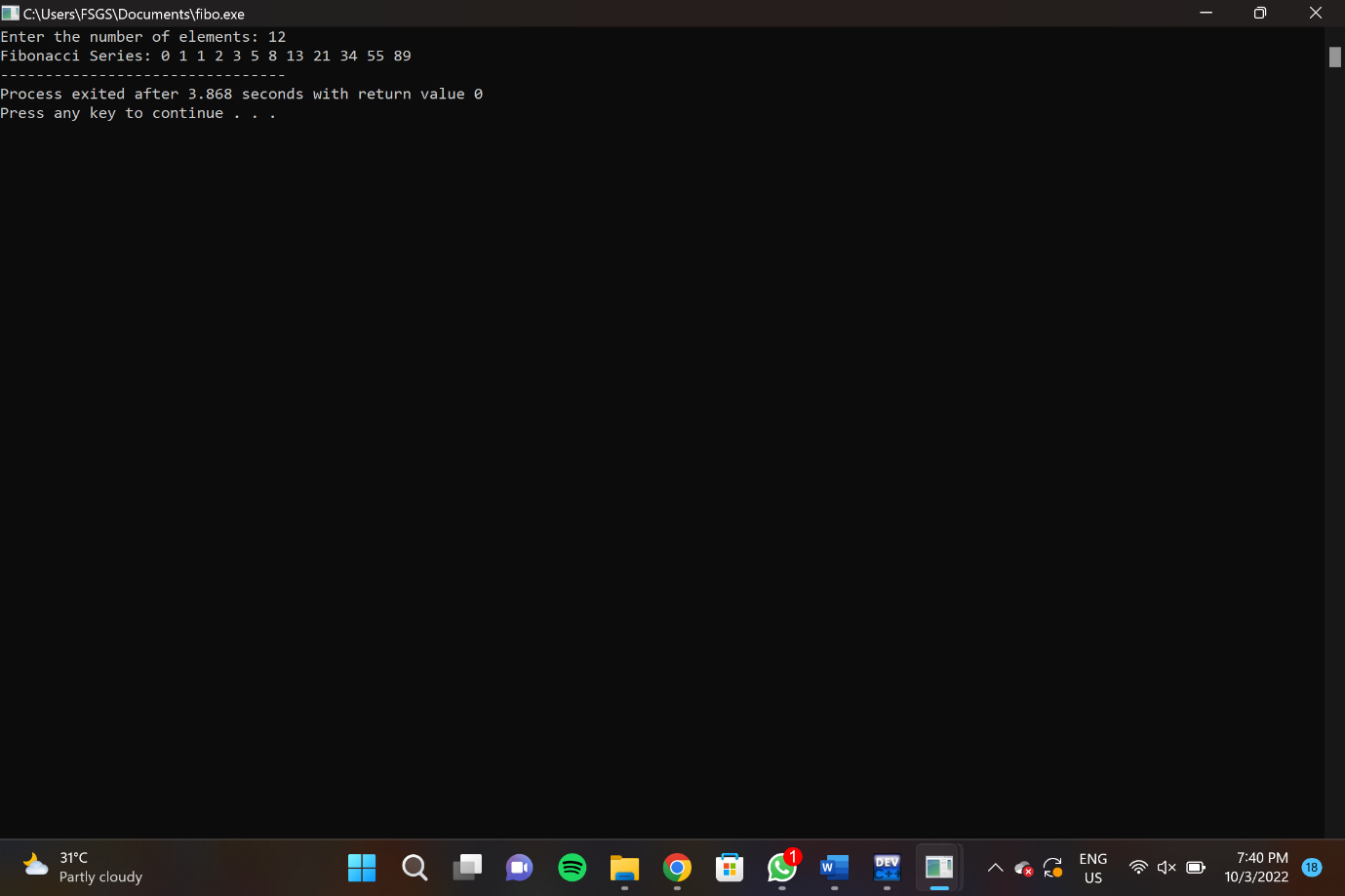
printf("%d is even number.",number); else

printf("%d is odd number.",number);

return 0;

}

**OUTPUT:**

****

**PROGRAM 7**

#include<stdio.h>

int main(){

int a[100],n,d,i,choice,ans = 1;

printf("Enter How many elements you want to insert : ");

scanf("%d",&n);

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

printf("Enter : ");

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

}

printf("\nWhat operation do you want to perform : \n");

printf("1.Insert \n2.Delete \n3.Display\n");

while(ans){

printf("Enter your Choice : ");

scanf("%d",&choice);

if(choice == 1){

int pos,e;

printf("Enter at which position do you want to insert the Element : ");

if(!(scanf("%d",&pos)) || pos <= 0 || pos > n){

printf("Invalid Input!!\n");

}

else{

printf("Enter the element : ");

scanf("%d",&e);

for(i = n - 1 ;i >= pos-1 ;i--){

a[i + 1] = a[i];

}

a[pos - 1] = e;

n++;

printf("\nthe array after insertion is : \n");

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

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

}

printf("\n");

}

}

else if(choice == 2){

int pos;

printf("At which Position do you want to Delete : ");

if(!(scanf("%d",&pos)) || pos <= 0 || pos > n){

printf("Invalid Input!!\n");

}

else{

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

a[i] = a[i+1];

}

printf("\nthe array after deletion is : \n");

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

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

}

printf("\n");

}

}

else if(choice == 3){

printf("The Elements in the array are : \n");

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

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

}

printf("\n");

}

else{

printf("Enter Invalid Input!!");

printf("\n");

}

printf("\nDo you want to Continue ? ");

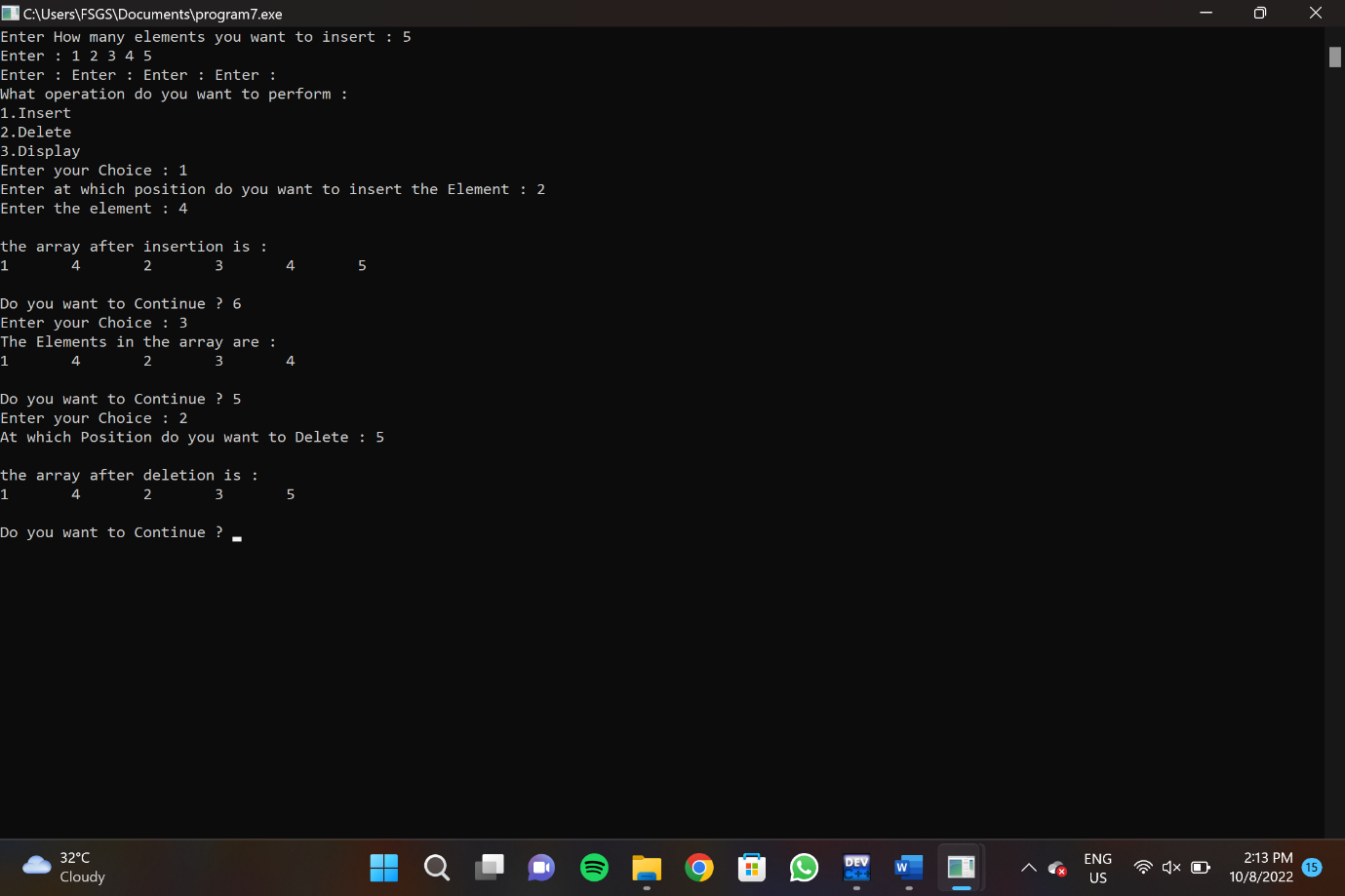
scanf("%d",&ans);

}

return 0;

}

**OUTPUT:**

****

**PROGRAM 8- C program to implement Array operations such as Insert, Delete and Display.**

#include<stdio.h>

int main(){

int a[100],n,d,i,choice,ans = 1;

printf("Enter How many elements you want to insert : ");

scanf("%d",&n);

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

printf("Enter : ");

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

}

printf("\nWhat operation do you want to perform : \n");

printf("1.Insert \n2.Delete \n3.Display\n");

while(ans){

printf("Enter your Choice : ");

scanf("%d",&choice);

if(choice == 1){

int pos,e;

printf("Enter at which position do you want to insert the Element : ");

if(!(scanf("%d",&pos)) || pos <= 0 || pos > n){

printf("Invalid Input!!\n");

}

else{

printf("Enter the element : ");

scanf("%d",&e);

for(i = n - 1 ;i >= pos-1 ;i--){

a[i + 1] = a[i];

}

a[pos - 1] = e;

n++;

printf("\nthe array after insertion is : \n");

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

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

}

printf("\n");

}

}

else if(choice == 2){

int pos;

printf("At which Position do you want to Delete : ");

if(!(scanf("%d",&pos)) || pos <= 0 || pos > n){

printf("Invalid Input!!\n");

}

else{

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

a[i] = a[i+1];

}

printf("\nthe array after deletion is : \n");

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

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

}

printf("\n");

}

}

else if(choice == 3){

printf("The Elements in the array are : \n");

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

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

}

printf("\n");

}

else{

printf("Enter Invalid Input!!");

printf("\n");

}

printf("\nDo you want to Continue ? ");

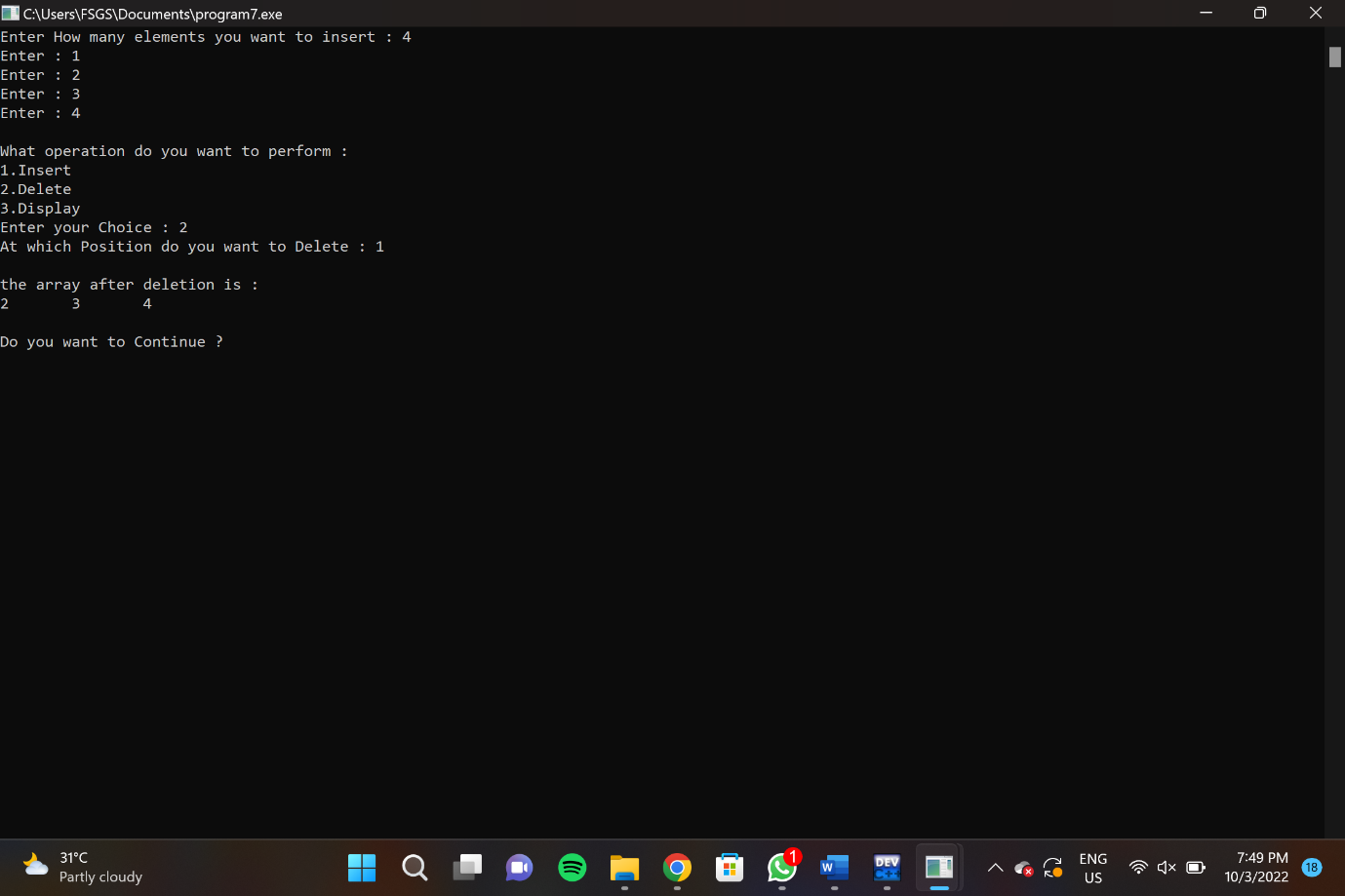
scanf("%d",&ans);

}

return 0;

}

**OUTPUT:**

****

**PROGRAM 9- LINEAR SEARCH METHOD**

#include <stdio.h>

int linearSearch(int a[], int n, int val) {

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

{

if (a[i] == val)

return i+1;

}

return -1;

}

int main() {

int a[] = {70, 40, 30, 11, 57, 41, 25, 14, 52};

int val = 41;

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

int res = linearSearch(a, n, val);

printf("The elements of the array are - ");

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

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

printf("\nElement to be searched is - %d", val);

if (res == -1)

printf("\nElement is not present in the array");

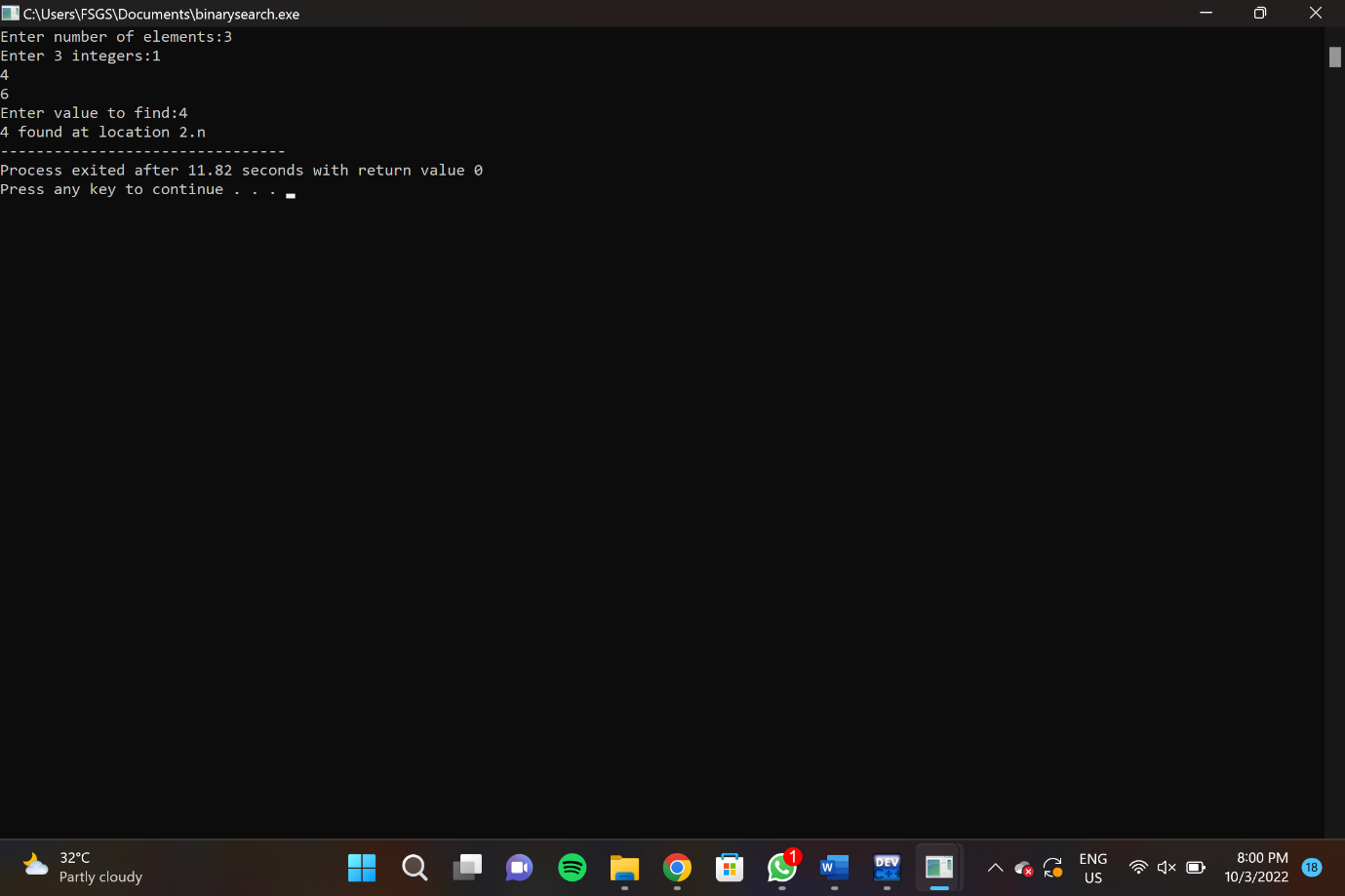
else

printf("\nElement is present at %d position of array", res);

return 0;

}

**OUTPUT:**

****

**PROGRAM 10-LINKED LIST OPERATIONS:**

#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;

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);

}

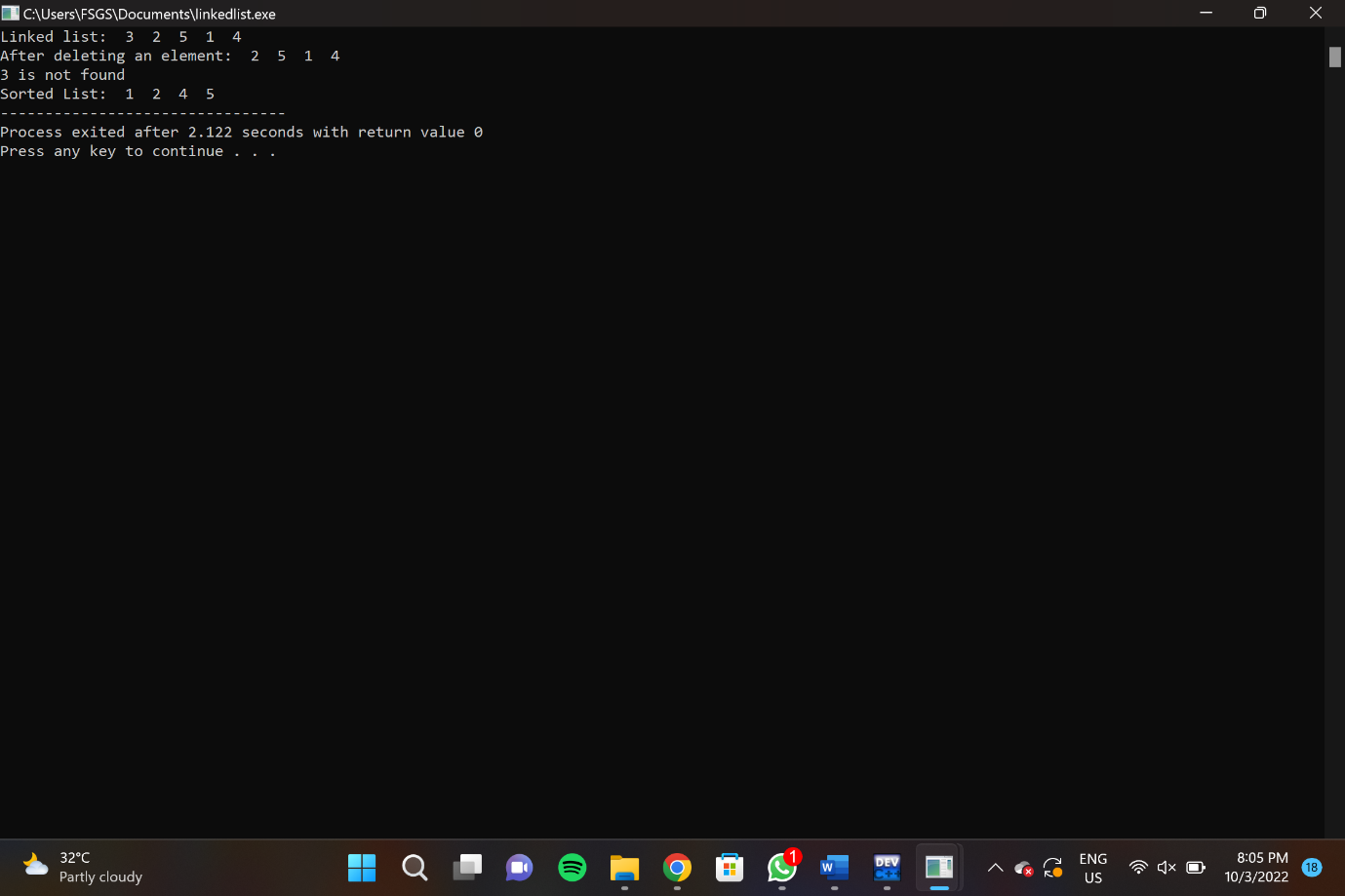
sortLinkedList(&head);

printf("\nSorted List: ");

printList(head);

}

**OUTPUT:**

****

**PROGRAM 11-STACK OPERATIONS**

#include<stdio.h>

#include<stdlib.h>

#define Size 4

int Top=-1, inp\_array[Size];

void Push();

void Pop();

void show();

int main()

{

int choice;

while(1)

{

printf("\nOperations performed by Stack");

printf("\n1.Push the element\n2.Pop the element\n3.Show\n4.End");

printf("\n\nEnter the choice:");

scanf("%d",&choice);

switch(choice)

{

case 1: Push();

break;

case 2: Pop();

break;

case 3: show();

break;

case 4: exit(0);

default: printf("\nInvalid choice!!");

}

}

}

void Push()

{

int x;

if(Top==Size-1)

{

printf("\nOverflow!!");

}

else

{

printf("\nEnter element to be inserted to the stack:");

scanf("%d",&x);

Top=Top+1;

inp\_array[Top]=x;

}

}

void Pop()

{

if(Top==-1)

{

printf("\nUnderflow!!");

}

else

{

printf("\nPopped element: %d",inp\_array[Top]);

Top=Top-1;

}

}

void show()

{

if(Top==-1)

{

printf("\nUnderflow!!");

}

else

{

printf("\nElements present in the stack: \n");

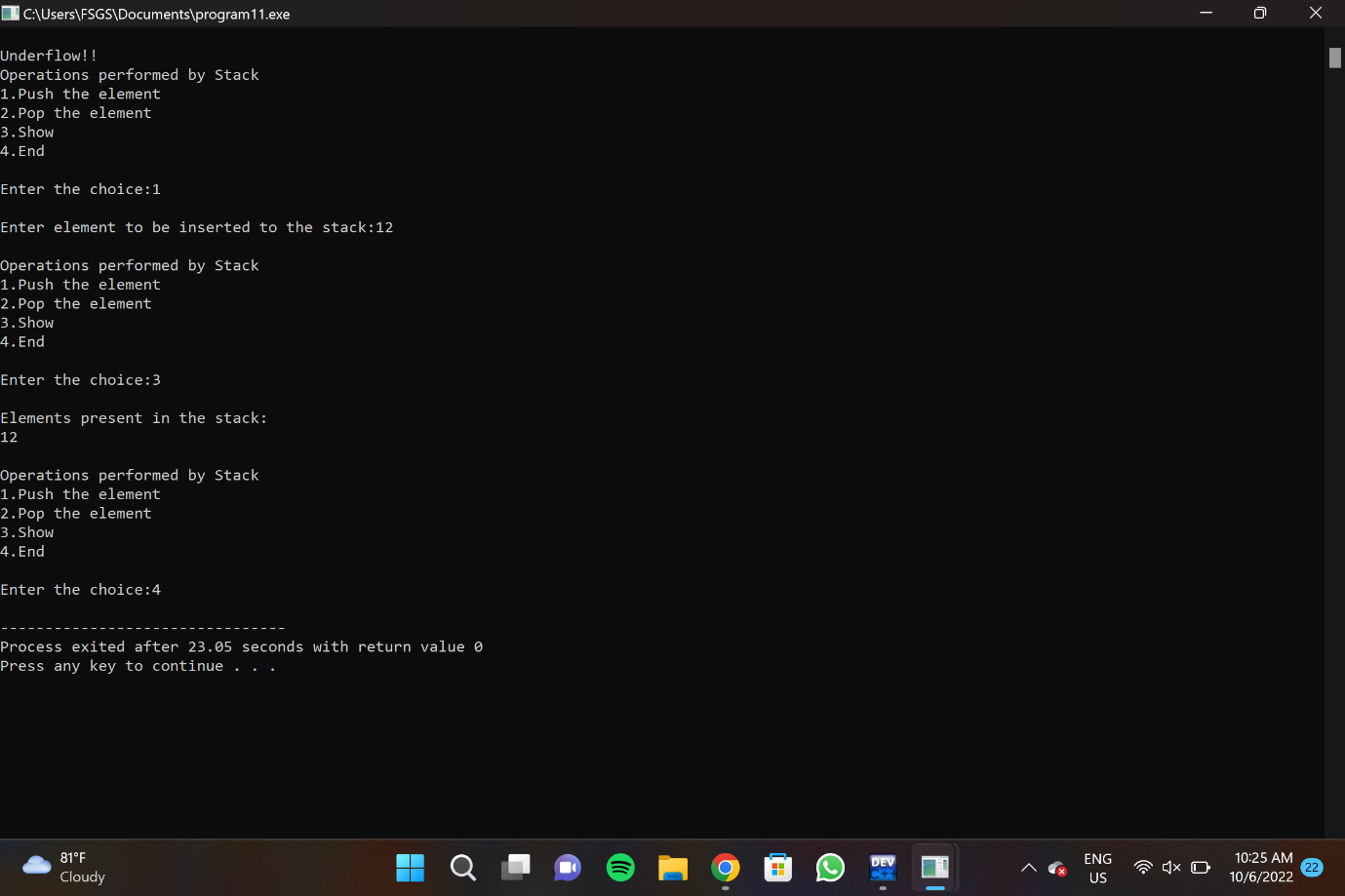
for(int i=Top;i>=0;--i)

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

}

}

**OUTPUT:**

****

**PROGRAM 12-STACK NOTATIONS**

#include <stdio.h>

int MAXSIZE = 8;

int stack[8];

int top = -1;

int isempty() {

if(top == -1)

return 1;

else

return 0;

}

int isfull() {

if(top == MAXSIZE)

return 1;

else

return 0;

}

int peek() {

return stack[top];

}

int pop() {

int data;

if(!isempty()) {

data = stack[top];

top = top - 1;

return data;

} else {

printf("Could not retrieve data, Stack is empty.\n");

}

}

int push(int data) {

if(!isfull()) {

top = top + 1;

stack[top] = data;

} else {

printf("Could not insert data, Stack is full.\n");

}

}

int main() {

push(3);

push(5);

push(9);

push(1);

push(12);

push(15);

printf("Element at top of the stack: %d\n" ,peek());

printf("Elements: \n");

while(!isempty()) {

int data = pop();

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

}

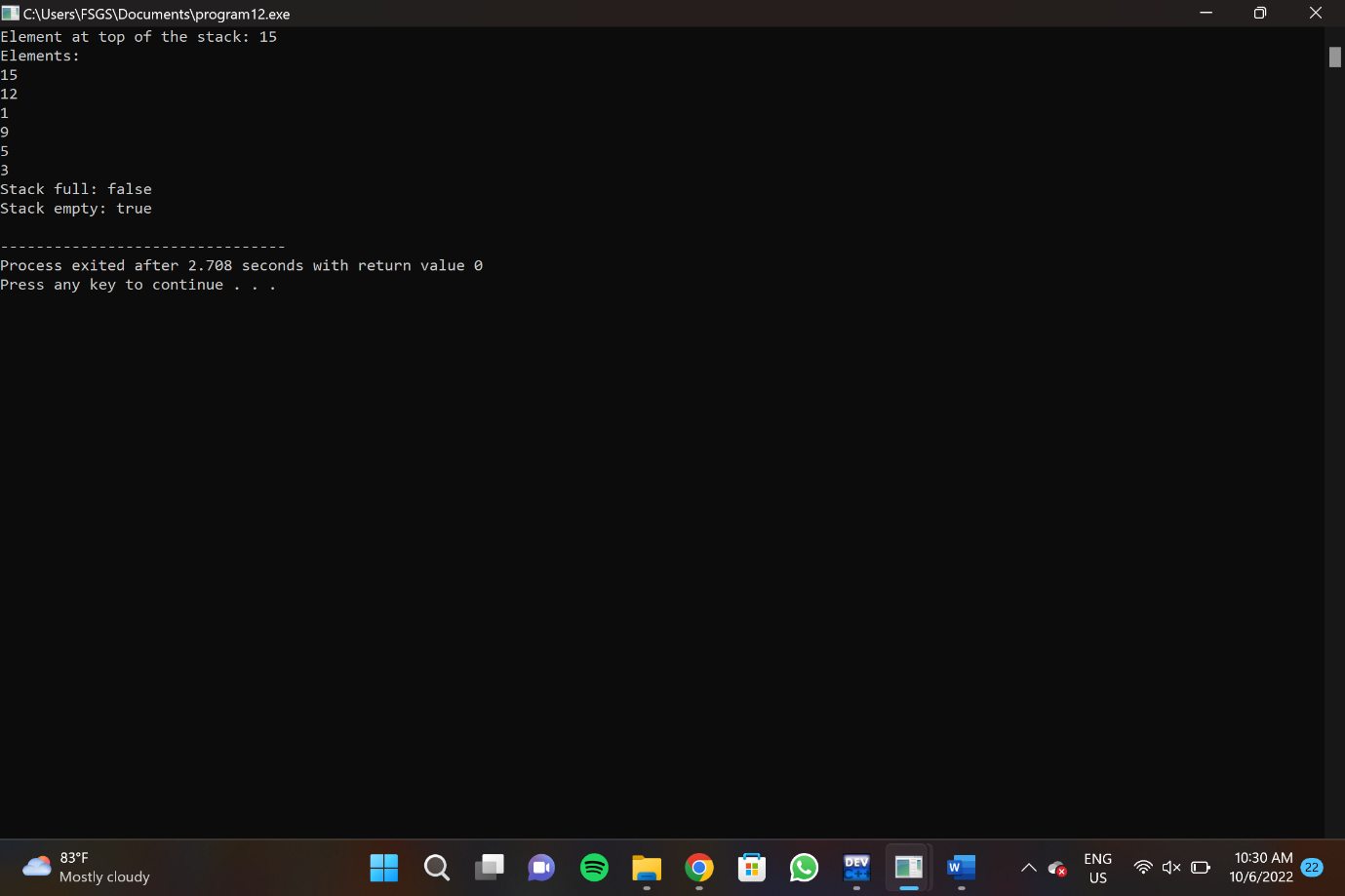
printf("Stack full: %s\n" , isfull()?"true":"false");

printf("Stack empty: %s\n" , isempty()?"true":"false");

return 0;

}

**OUTPUT:**

****

**PROGRAM 13:**

#include <stdio.h>

#include<conio.h>

#include<stdlib.h>

#define MAX 50

void enqueue();

void dequeue();

void display();

int queue\_array[MAX];

int rear = - 1;

int front = - 1;

int main()

{

int choice;

while (1)

{

printf("\n1.Insert \n");

printf("2.Delete \n");

printf("3.Display \n");

printf("4.Quit \n");

printf("Enter your choice : ");

scanf("%d", &choice);

switch(choice)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(1);

default:

printf("Wrong choice \n");

}

}

}

void enqueue()

{

int item;

if(rear == MAX - 1)

printf("Queue Overflow \n");

else

{

if(front== - 1)

front = 0;

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

scanf("%d", &item);

rear = rear + 1;

queue\_array[rear] = item;

}

}

void dequeue()

{

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++)

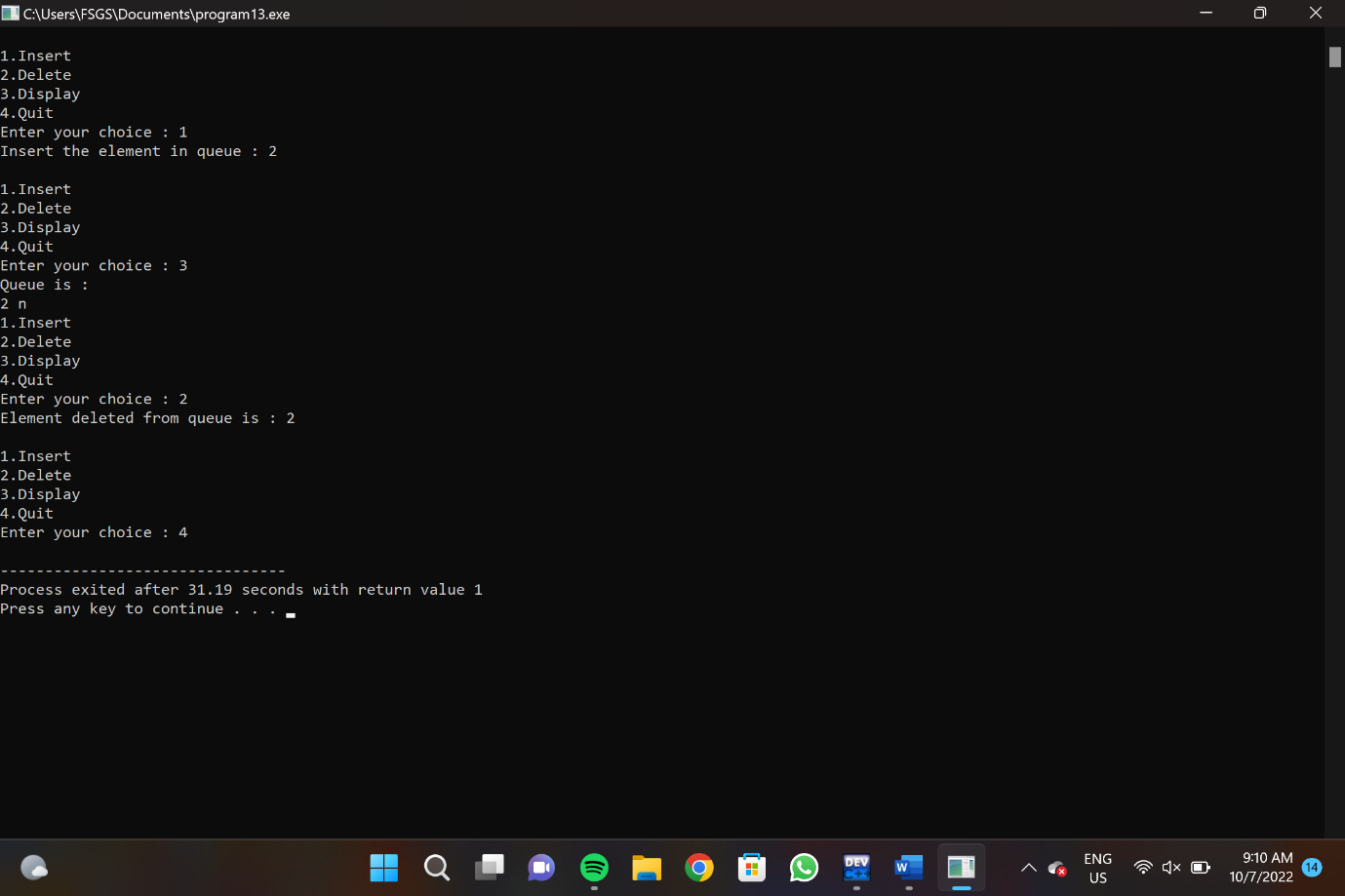
printf("%d ", queue\_array[i]);

printf("n");

}

}

**OUTPUT:**

****

**PROGRAM 14**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*leftChild;

struct node \*rightChild;

};

struct node \*root = NULL;

void insert(int data) {

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

struct node \*current;

struct node \*parent;

tempNode->data = data;

tempNode->leftChild = NULL;

tempNode->rightChild = NULL;

if(root == NULL) {

root = tempNode;

} else {

current = root;

parent = NULL;

while(1) {

parent = current;

if(data < parent->data) {

current = current->leftChild;

if(current == NULL) {

parent->leftChild = tempNode;

return;

}

}

else {

current = current->rightChild;

if(current == NULL) {

parent->rightChild = tempNode;

return;

}

}

}

}

}

struct node\* search(int data) {

struct node \*current = root;

printf("Visiting elements: ");

while(current->data != data) {

if(current != NULL)

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

if(current->data > data) {

current = current->leftChild;

}

else {

current = current->rightChild;

}

if(current == NULL) {

return NULL;

}

}

return current;

}

void pre\_order\_traversal(struct node\* root) {

if(root != NULL) {

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

pre\_order\_traversal(root->leftChild);

pre\_order\_traversal(root->rightChild);

}

}

void inorder\_traversal(struct node\* root) {

if(root != NULL) {

inorder\_traversal(root->leftChild);

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

inorder\_traversal(root->rightChild);

}

}

void post\_order\_traversal(struct node\* root) {

if(root != NULL) {

post\_order\_traversal(root->leftChild);

post\_order\_traversal(root->rightChild);

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

}

}

int main() {

int i;

int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

for(i = 0; i < 7; i++)

insert(array[i]);

i = 31;

struct node \* temp = search(i);

if(temp != NULL) {

printf("[%d] Element found.", temp->data);

printf("\n");

}else {

printf("[ x ] Element not found (%d).\n", i);

}

i = 15;

temp = search(i);

if(temp != NULL) {

printf("[%d] Element found.", temp->data);

printf("\n");

}else {

printf("[ x ] Element not found (%d).\n", i);

}

printf("\nPreorder traversal: ");

pre\_order\_traversal(root);

printf("\nInorder traversal: ");

inorder\_traversal(root);

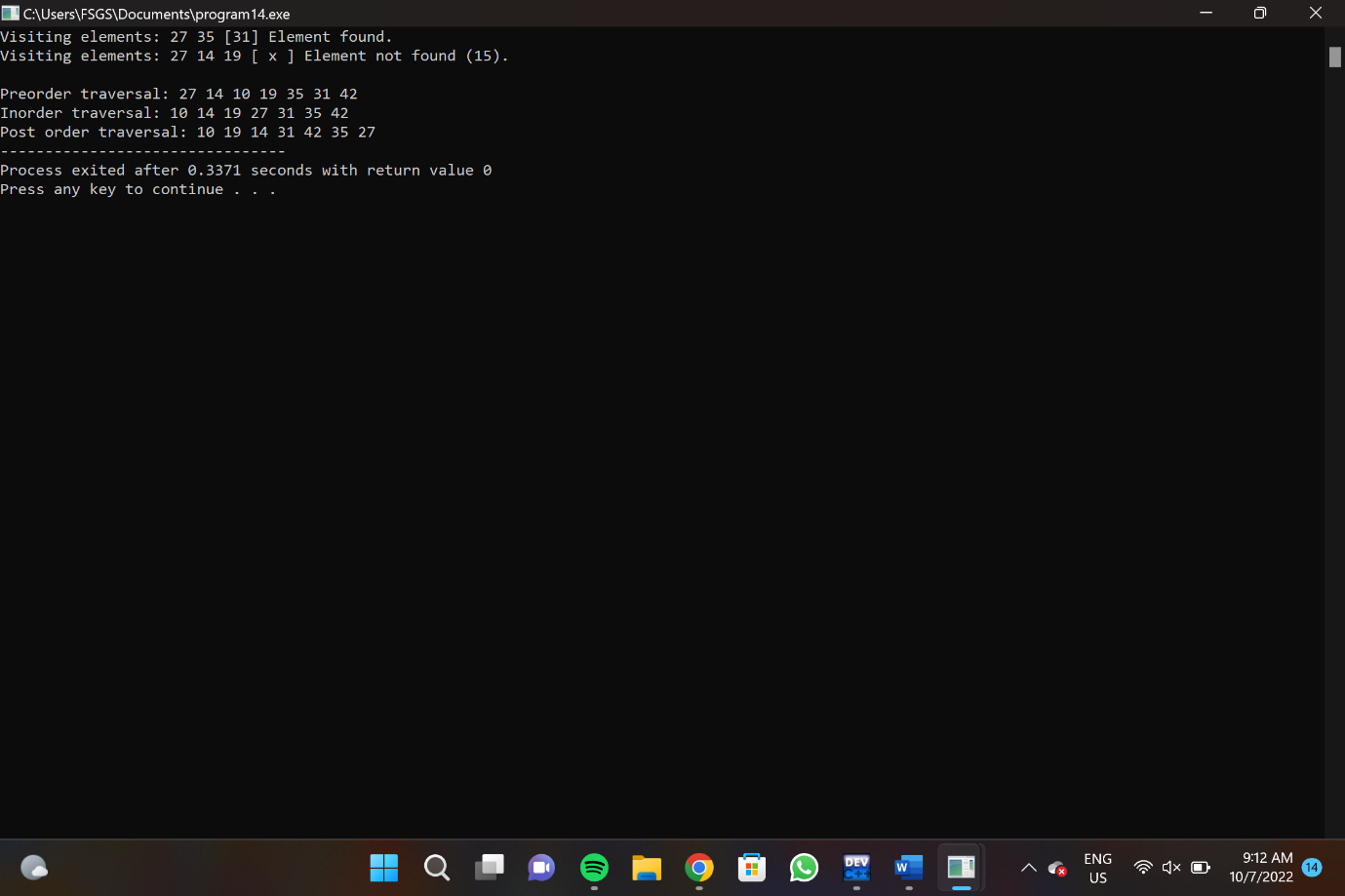
printf("\nPost order traversal: ");

post\_order\_traversal(root);

return 0;

}

**OUTPUT:**

****

**PROGRAM 15**

#include <stdio.h>

#include<stdlib.h>

#define TABLE\_SIZE 10

int h[TABLE\_SIZE]={NULL};

void insert()

{

int key,index,i,flag=0,hkey;

printf("\nenter a value to insert into hash table\n");

scanf("%d",&key);

hkey=key%TABLE\_SIZE;

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

{

index=(hkey+i)%TABLE\_SIZE;

if(h[index] == NULL)

{

h[index]=key;

break;

}

}

if(i == TABLE\_SIZE)

printf("\nelement cannot be inserted\n");

}

void search()

{

int key,index,i,flag=0,hkey;

printf("\nenter search element\n");

scanf("%d",&key);

hkey=key%TABLE\_SIZE;

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

{

index=(hkey+i)%TABLE\_SIZE;

if(h[index]==key)

{

printf("value is found at index %d",index);

break;

}

}

if(i == TABLE\_SIZE)

printf("\n value is not found\n");

}

void display()

{

int i;

printf("\nelements in the hash table are \n");

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

printf("\nat index %d \t value = %d",i,h[i]);

}

main()

{

int opt,i;

while(1)

{

printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");

scanf("%d",&opt);

switch(opt)

{

case 1:

insert();

break;

case 2:

display();

break;

case 3:

search();

break;

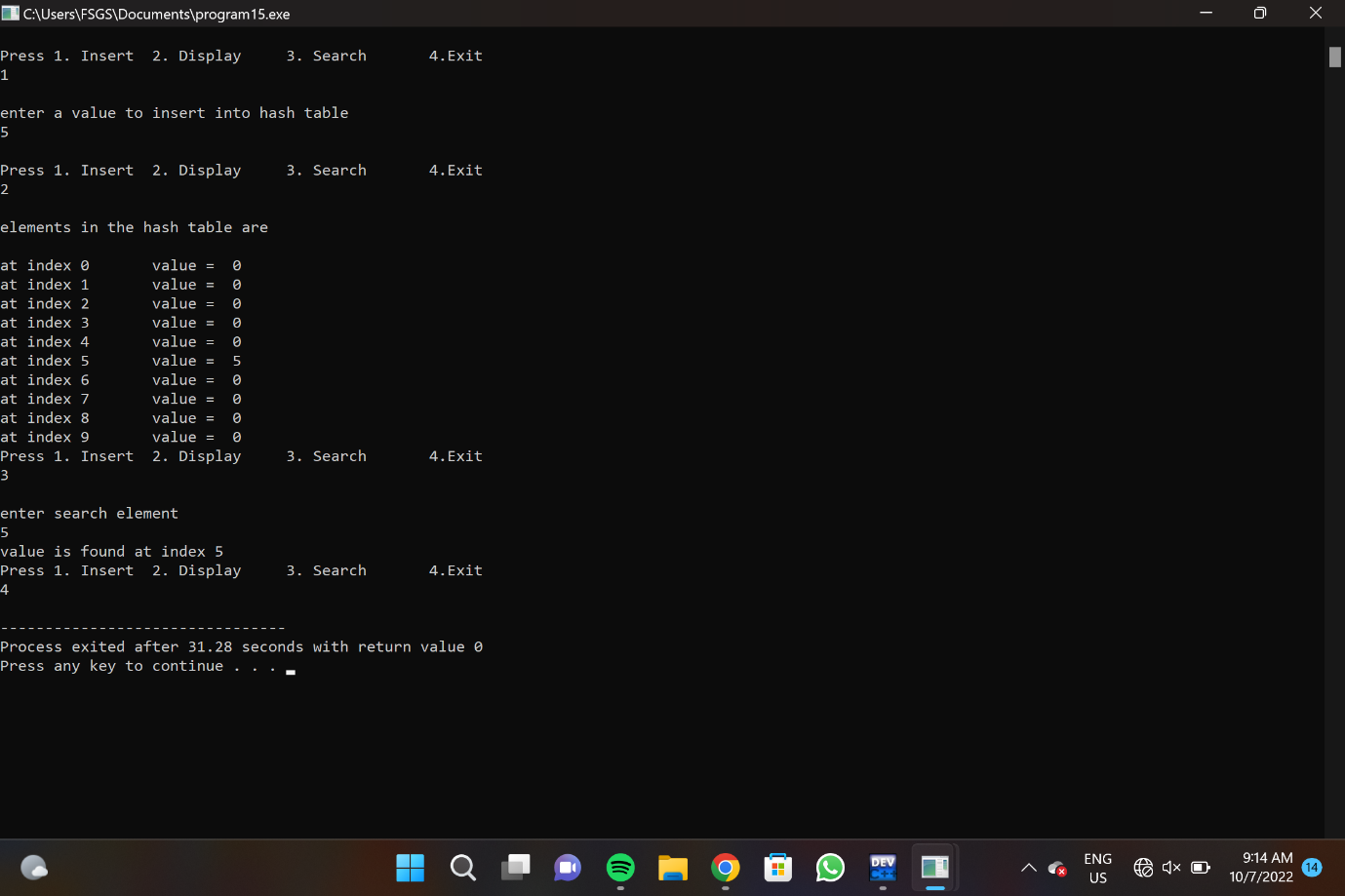
case 4:exit(0);

}

}

}

**OUTPUT:**

****

**PROGRAM 16**

#include<stdio.h>

#include<conio.h>

int main()

{

int a[10],i,j,k,n;

int clrscr( );

printf("How many elements you want to sort?\n");

scanf("%d",&n);

printf("\nEnter the Elements into an array:\n");

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

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

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

{

k=a[i];

for(j= i-1; j>=0 && k<a[j]; j--)

a[j+1]=a[j];

a[j+1]=k;

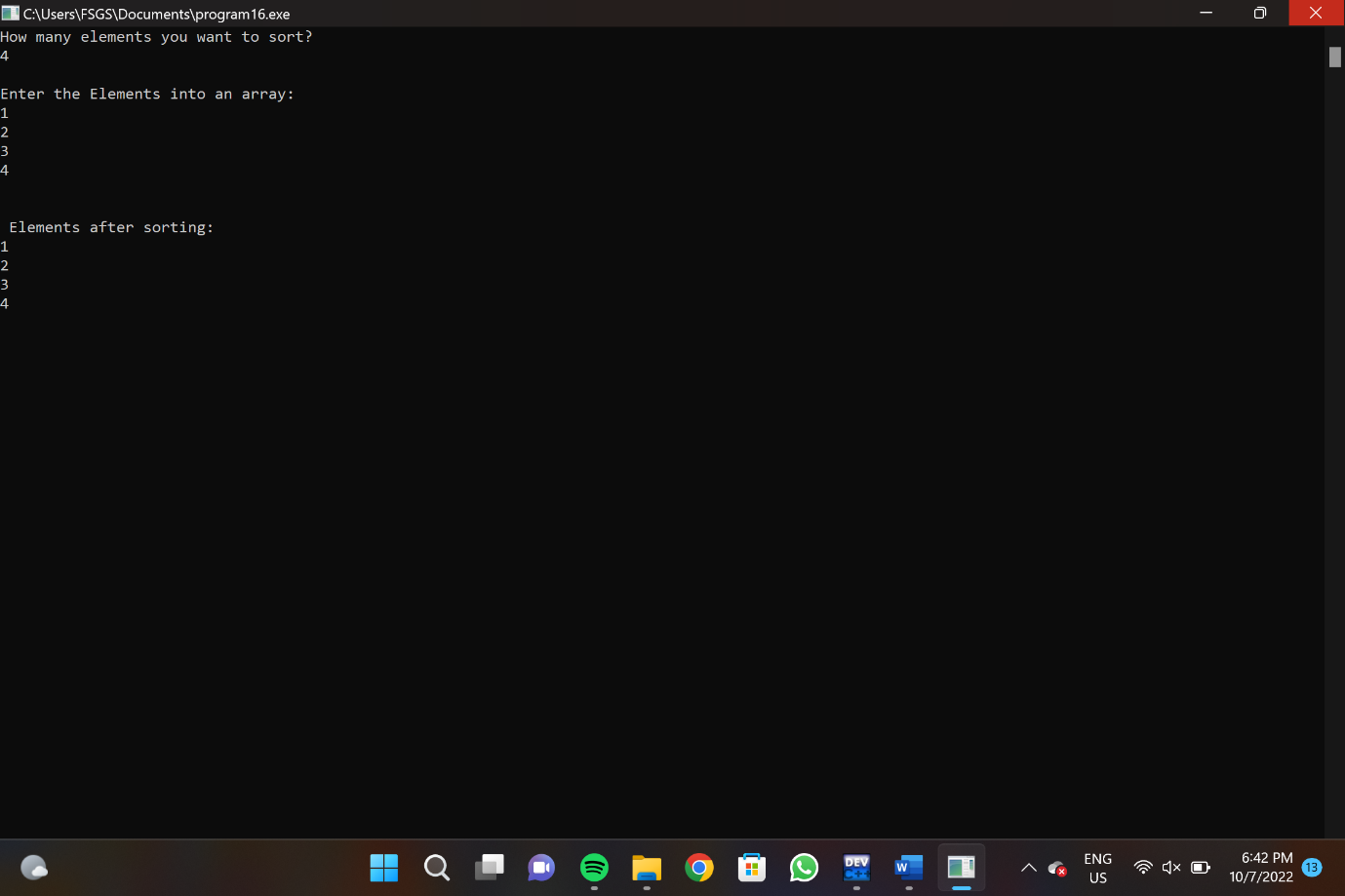
} printf("\n\n Elements after sorting: \n");

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

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

getch( );

}

**OUTPUT:**

**PROGRAM 17**

#include <stdio.h>

#define max 10

int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };

int b[10];

void merging(int low, int mid, int high) {

int l1, l2, i;

for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {

if(a[l1] <= a[l2])

b[i] = a[l1++];

else

b[i] = a[l2++];

}

while(l1 <= mid)

b[i++] = a[l1++];

while(l2 <= high)

b[i++] = a[l2++];

for(i = low; i <= high; i++)

a[i] = b[i];

}

void sort(int low, int high) {

int mid;

if(low < high) {

mid = (low + high) / 2;

sort(low, mid);

sort(mid+1, high);

merging(low, mid, high);

} else {

return;

}

}

int main() {

int i;

printf("List before sorting\n");

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

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

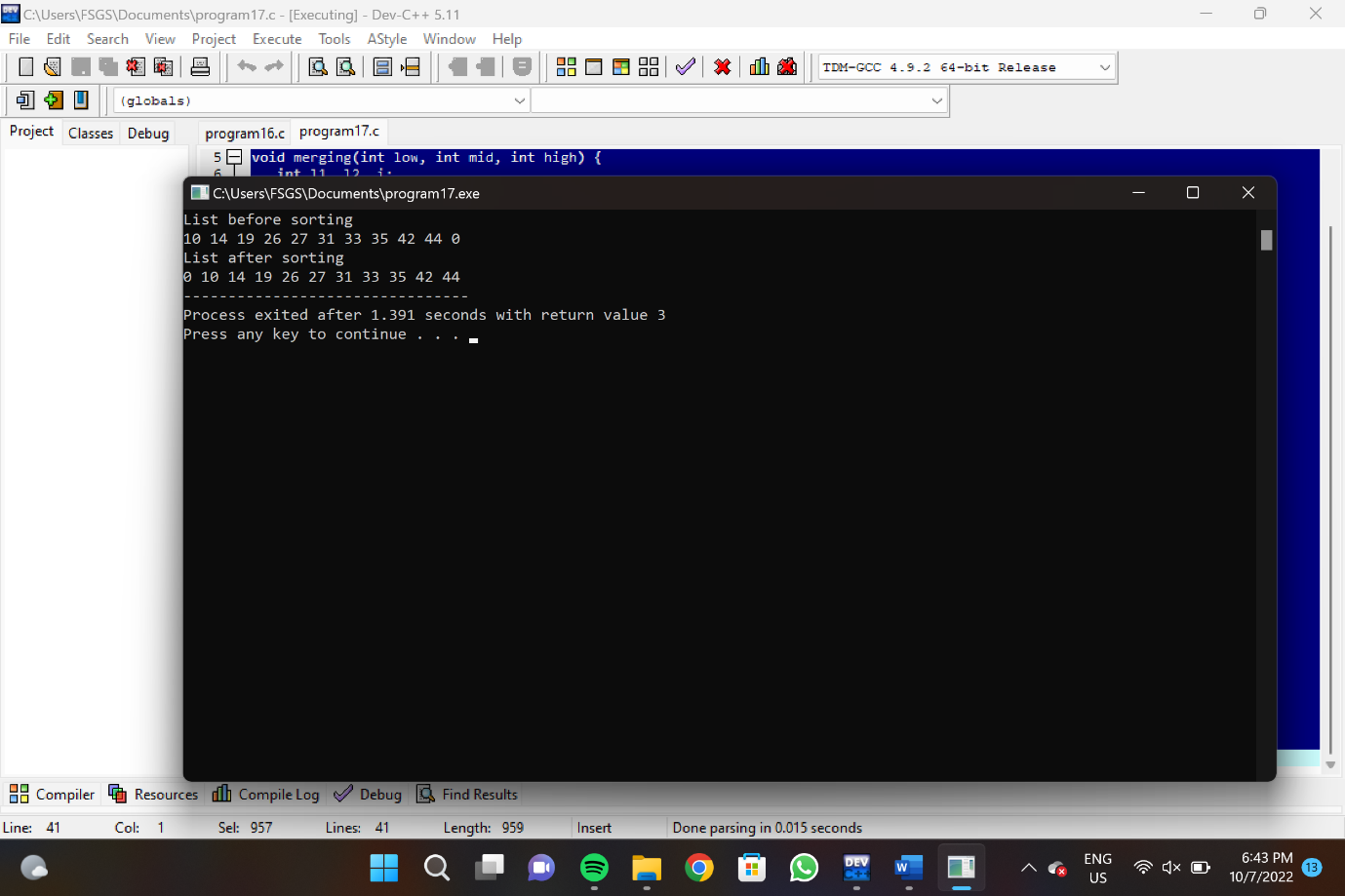
sort(0, max);

printf("\nList after sorting\n");

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

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

}

**OUTPUT:**

**PROGRAM 18**

#include<stdio.h>

void quicksort(int[ ],int,int);

int main()

{

int low, high, pivot, t, n, i, j, a[10];

int clrscr();

printf("\nHow many elements you want to sort ? ");

scanf("%d",&n);

printf("\Enter elements for an array:");

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

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

low=0;

high=n-1;

quicksort(a,low,high);

printf("\After Sorting the elements are:");

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

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

}

void quicksort(int a[ ],int low,int high)

{

int pivot,t,i,j;

if(low<high)

{

pivot=a[low];

i=low+1;

j=high;

while(1)

{

while(pivot>a[i]&&i<=high)

i++;

while(pivot<a[j]&&j>=low)

j--;

if(i<j)

{

t=a[i];

a[i]=a[j];

a[j]=t;

}

else

break;

}

a[low]=a[j];

a[j]=pivot;

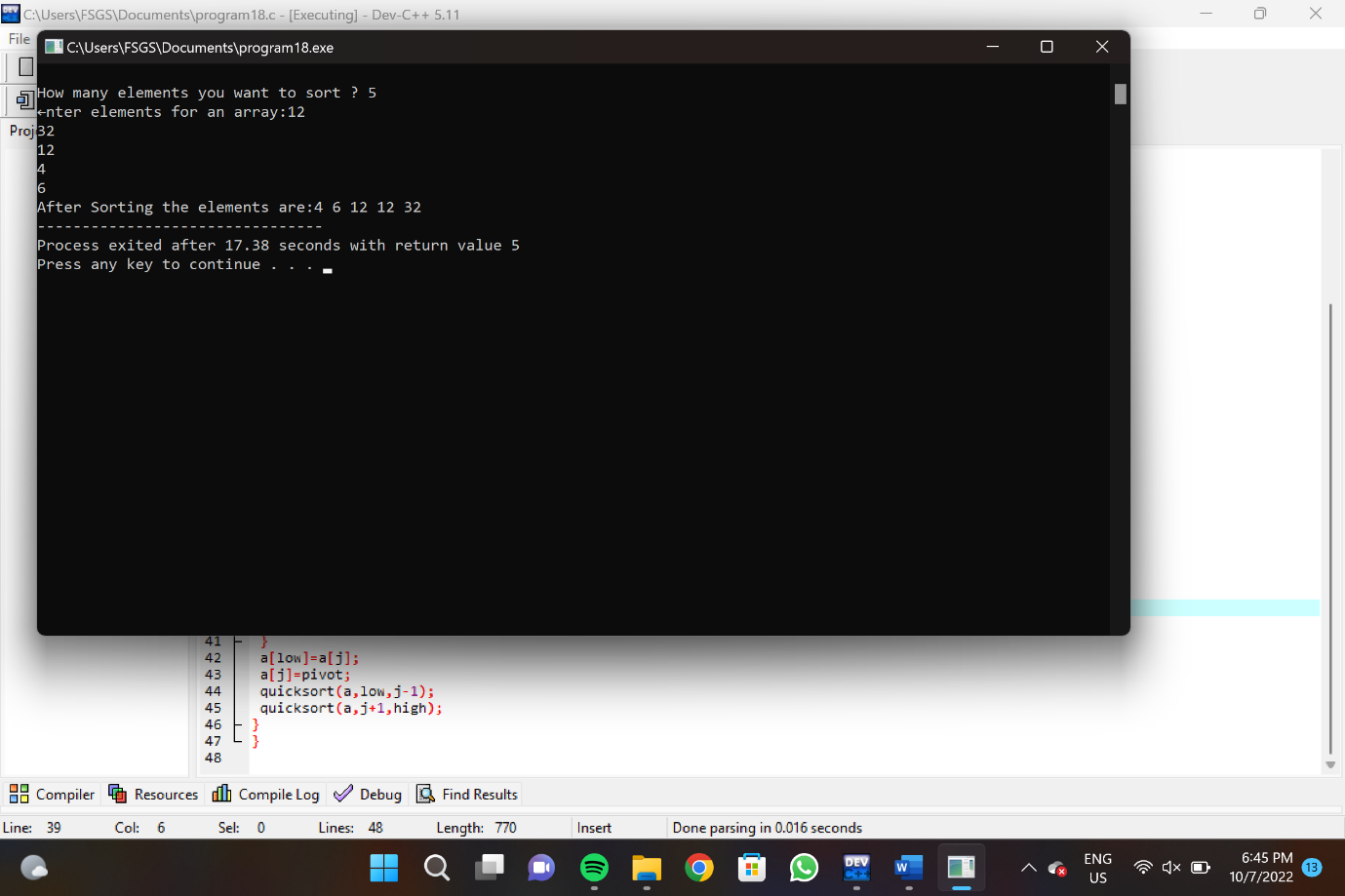
quicksort(a,low,j-1);

quicksort(a,j+1,high);

}

}

**OUTPUT:**

****

**PROGRAM 19**

#include <stdio.h>

void heapify(int a[], int n, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && a[left] > a[largest])

largest = left;

if (right < n && a[right] > a[largest])

largest = right;

if (largest != i) {

int temp = a[i];

a[i] = a[largest];

a[largest] = temp;

heapify(a, n, largest);

}

}

void heapSort(int a[], int n)

{

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

heapify(a, n, i);

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

int temp = a[0];

a[0] = a[i];

a[i] = temp;

heapify(a, i, 0);

}

}

void printArr(int arr[], int n)

{

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

{

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

printf(" ");

}

}

int main()

{

int a[] = {48, 10, 23, 43, 28, 26, 1};

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

printf("Before sorting array elements are - \n");

printArr(a, n);

heapSort(a, n);

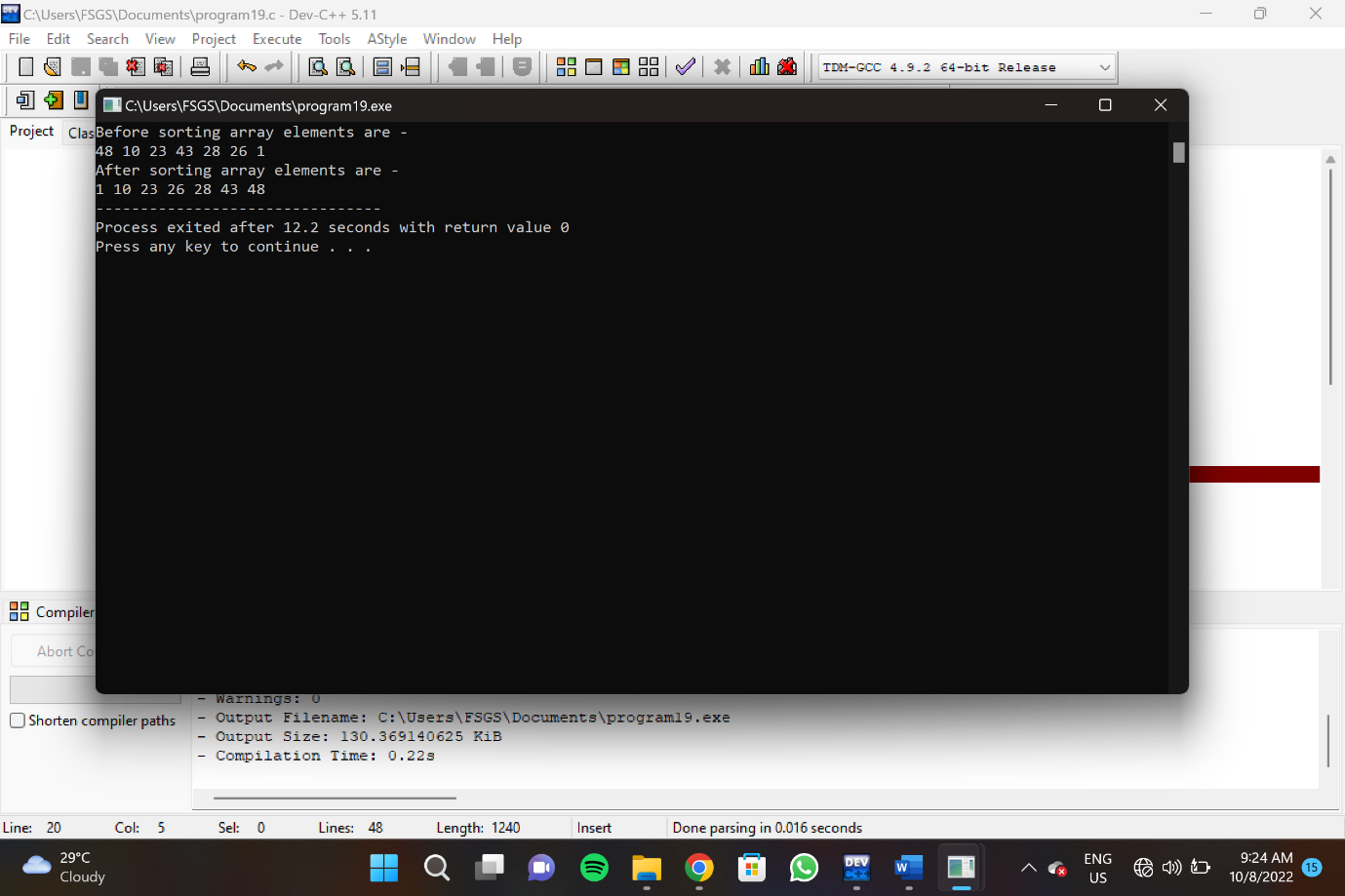
printf("\nAfter sorting array elements are - \n");

printArr(a, n);

return 0;

}

**OUTPUT:**

****

**PROGRAM 20**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\* left;

struct node\* right;

int ht;

};

struct node\* root = NULL;

struct node\* create(int);

struct node\* insert(struct node\*, int);

struct node\* delet(struct node\*, int);

struct node\* search(struct node\*, int);

struct node\* rotate\_left(struct node\*);

struct node\* rotate\_right(struct node\*);

int balance\_factor(struct node\*);

int height(struct node\*);

void inorder(struct node\*);

void preorder(struct node\*);

void postorder(struct node\*);

int main()

{

int user\_choice, data;

char user\_continue = 'y';

struct node\* result = NULL;

while (user\_continue == 'y' || user\_continue == 'Y')

{

printf("\n\n------- AVL TREE --------\n");

printf("\n1. Insert");

printf("\n2. Delete");

printf("\n3. Search");

printf("\n4. Inorder");

printf("\n5. Preorder");

printf("\n6. Postorder");

printf("\n7. EXIT");

printf("\n\nEnter Your Choice: ");

scanf("%d", &user\_choice);

switch(user\_choice)

{

case 1:

printf("\nEnter data: ");

scanf("%d", &data);

root = insert(root, data);

break;

case 2:

printf("\nEnter data: ");

scanf("%d", &data);

root = delet(root, data);

break;

case 3:

printf("\nEnter data: ");

scanf("%d", &data);

result = search(root, data);

if (result == NULL)

{

printf("\nNode not found!");

}

else

{

printf("\n Node found");

}

break;

case 4:

inorder(root);

break;

case 5:

preorder(root);

break;

case 6:

postorder(root);

break;

case 7:

printf("\n\tProgram Terminated\n");

return 1;

default:

printf("\n\tInvalid Choice\n");

}

printf("\n\nDo you want to continue? ");

scanf(" %c", &user\_continue);

}

return 0;

}

struct node\* create(int data)

{

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

if (new\_node == NULL)

{

printf("\nMemory can't be allocated\n");

return NULL;

}

new\_node->data = data;

new\_node->left = NULL;

new\_node->right = NULL;

return new\_node;

}

struct node\* rotate\_left(struct node\* root)

{

struct node\* right\_child = root->right;

root->right = right\_child->left;

right\_child->left = root;

root->ht = height(root);

right\_child->ht = height(right\_child);

return right\_child;

}

struct node\* rotate\_right(struct node\* root)

{

struct node\* left\_child = root->left;

root->left = left\_child->right;

left\_child->right = root;

root->ht = height(root);

left\_child->ht = height(left\_child);

return left\_child;

}

int balance\_factor(struct node\* root)

{

int lh, rh;

if (root == NULL)

return 0;

if (root->left == NULL)

lh = 0;

else

lh = 1 + root->left->ht;

if (root->right == NULL)

rh = 0;

else

rh = 1 + root->right->ht;

return lh - rh;

}

int height(struct node\* root)

{

int lh, rh;

if (root == NULL)

{

return 0;

}

if (root->left == NULL)

lh = 0;

else

lh = 1 + root->left->ht;

if (root->right == NULL)

rh = 0;

else

rh = 1 + root->right->ht;

if (lh > rh)

return (lh);

return (rh);

}

struct node\* insert(struct node\* root, int data)

{

if (root == NULL)

{

struct node\* new\_node = create(data);

if (new\_node == NULL)

{

return NULL;

}

root = new\_node;

}

else if (data > root->data)

{

root->right = insert(root->right, data);

if (balance\_factor(root) == -2)

{

if (data > root->right->data)

{

root = rotate\_left(root);

}

else

{

root->right = rotate\_right(root->right);

root = rotate\_left(root);

}

}

}

else

{

root->left = insert(root->left, data);

if (balance\_factor(root) == 2)

{

if (data < root->left->data)

{

root = rotate\_right(root);

}

else

{

root->left = rotate\_left(root->left);

root = rotate\_right(root);

}

}

}

root->ht = height(root);

return root;

}

struct node \* delet(struct node \*root, int x)

{

struct node \* temp = NULL;

if (root == NULL)

{

return NULL;

}

if (x > root->data)

{

root->right = delet(root->right, x);

if (balance\_factor(root) == 2)

{

if (balance\_factor(root->left) >= 0)

{

root = rotate\_right(root);

}

else

{

root->left = rotate\_left(root->left);

root = rotate\_right(root);

}

}

}

else if (x < root->data)

{

root->left = delet(root->left, x);

if (balance\_factor(root) == -2)

{

if (balance\_factor(root->right) <= 0)

{

root = rotate\_left(root);

}

else

{

root->right = rotate\_right(root->right);

root = rotate\_left(root);

}

}

}

else

{

if (root->right != NULL)

{

temp = root->right;

while (temp->left != NULL)

temp = temp->left;

root->data = temp->data;

root->right = delet(root->right, temp->data);

if (balance\_factor(root) == 2)

{

if (balance\_factor(root->left) >= 0)

{

root = rotate\_right(root);

}

else

{

root->left = rotate\_left(root->left);

root = rotate\_right(root);

}

}

}

else

{

return (root->left);

}

}

root->ht = height(root);

return (root);

}

struct node\* search(struct node\* root, int key)

{

if (root == NULL)

{

return NULL;

}

if(root->data == key)

{

return root;

}

if(key > root->data)

{

search(root->right, key);

}

else

{

search(root->left, key);

}

}

void inorder(struct node\* root)

{

if (root == NULL)

{

return;

}

inorder(root->left);

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

inorder(root->right);

}

void preorder(struct node\* root)

{

if (root == NULL)

{

return;

}

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

preorder(root->left);

preorder(root->right);

}

void postorder(struct node\* root)

{

if (root == NULL)

{

return;

}

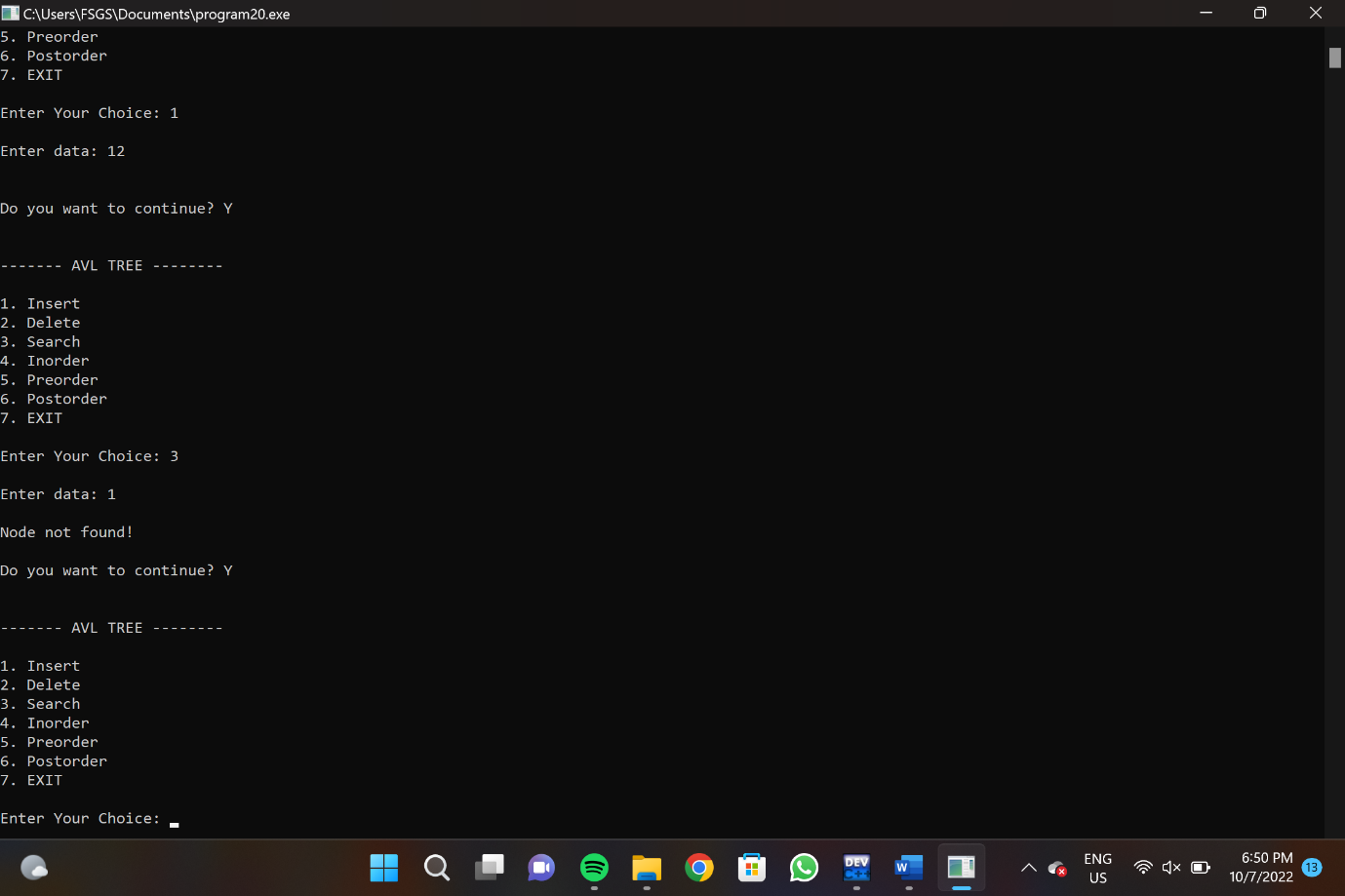
postorder(root->left);

postorder(root->right);

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

}

**OUTPUT:**

****

**PROGRAM 21**

#include<stdio.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;

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. Not all nodes are reachable");

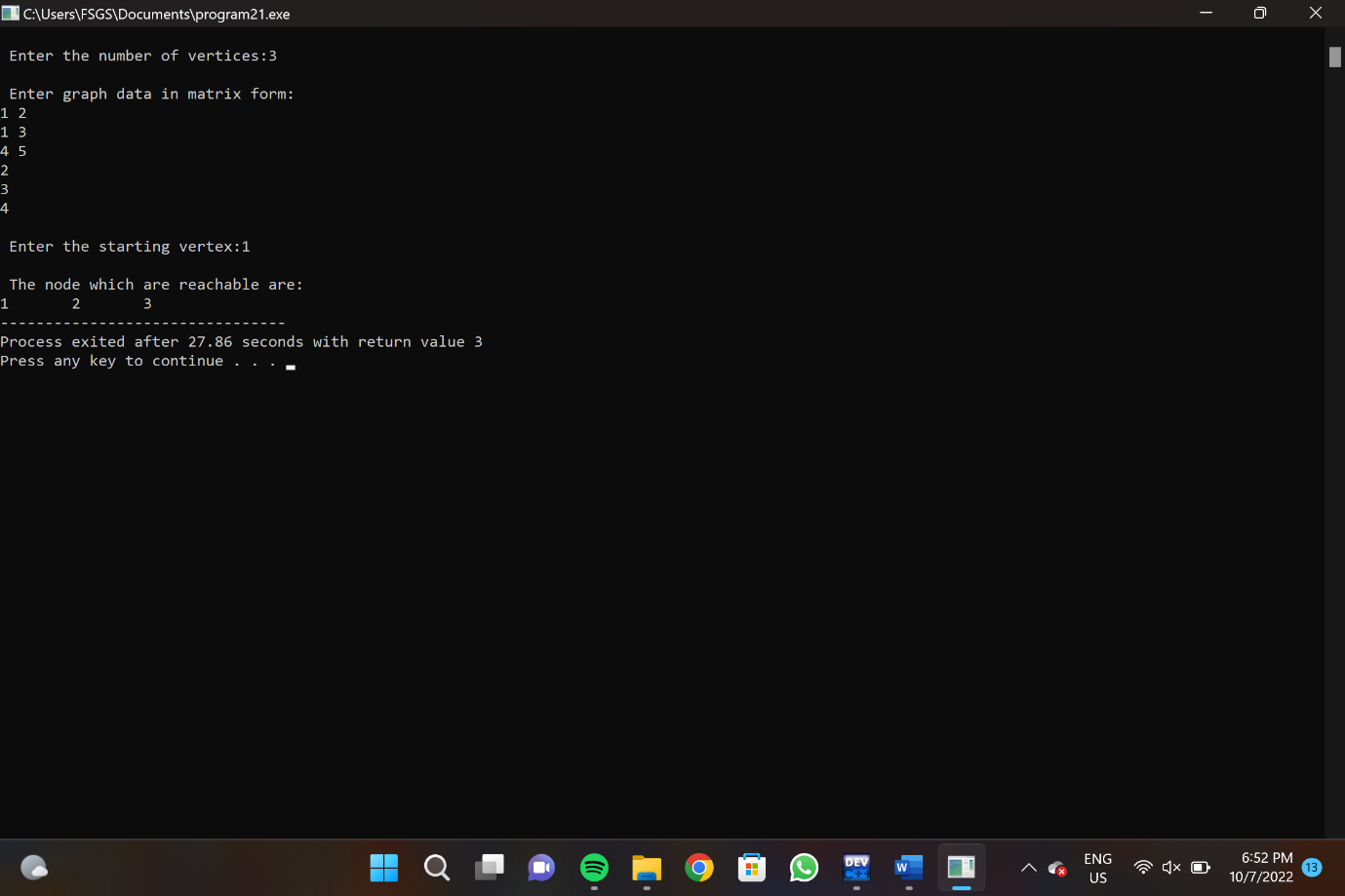
break;

}

}

}

**OUTPUT:**

****

**PROGRAM 22**

#include<stdio.h>

void DFS(int);

int G[10][10],visited[10],n;

int main()

{

int i,j;

printf("Enter number of vertices:");

scanf("%d",&n);

printf("Enter adjecency matrix of the graph:\n");

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

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

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

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

visited[i]=0;

DFS(0);

}

void DFS(int i)

{

int j;

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

visited[i]=1;

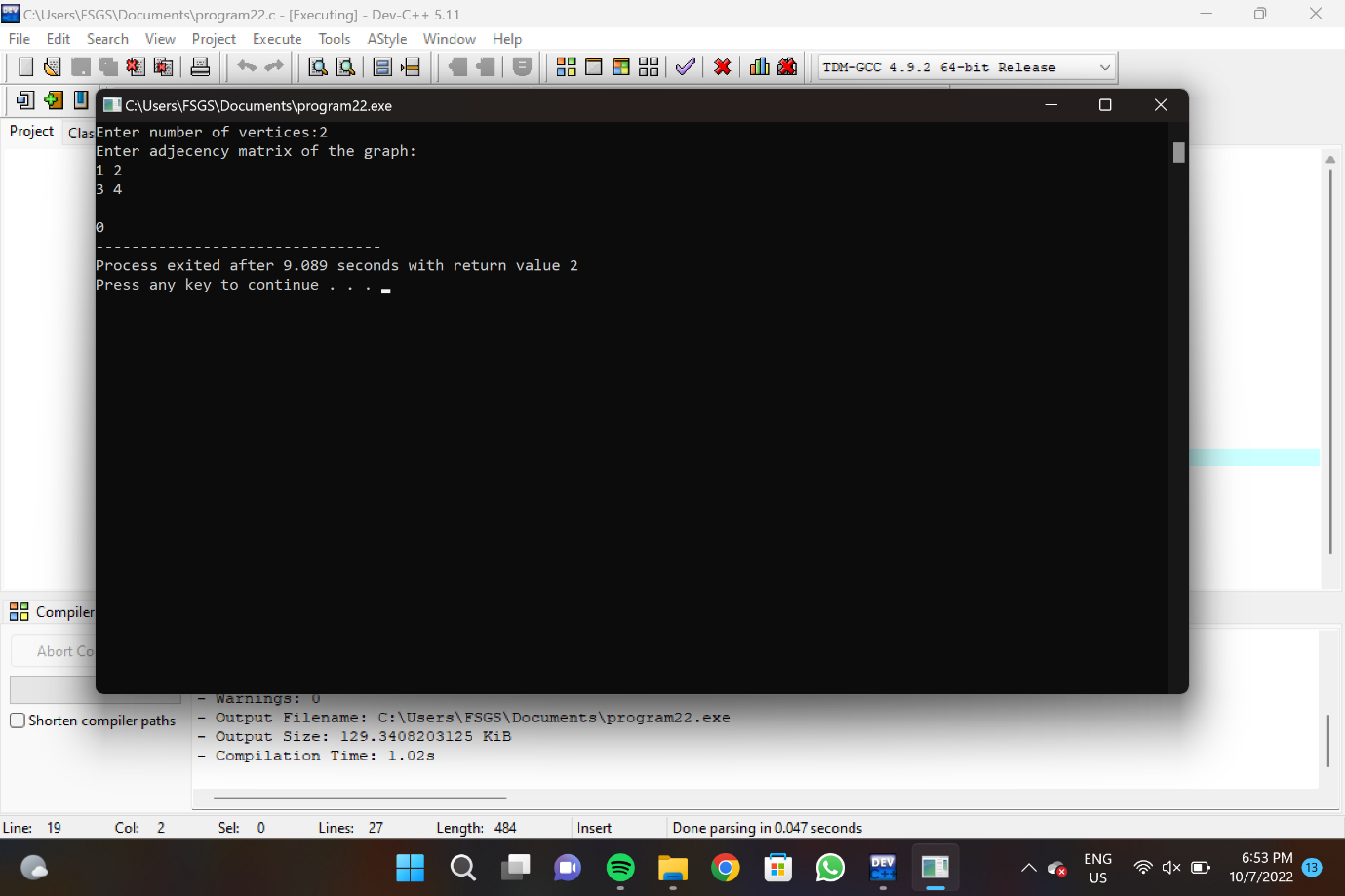
for(j=0;j<n;j++)

if(!visited[j]&&G[i][j]==1)

DFS(j);

}

**OUTPUT:**

****

**PROGRAM 23**

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()

{

int G[MAX][MAX],i,j,n,u;

printf("Enter no. of vertices:");

scanf("%d",&n);

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

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

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

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

printf("\nEnter the starting node:");

scanf("%d",&u);

dijkstra(G,n,u);

return 0;

}

void dijkstra(int G[MAX][MAX],int n,int startnode)

{

int cost[MAX][MAX],distance[MAX],pred[MAX];

int visited[MAX],count,mindistance,nextnode,i,j;

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

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

if(G[i][j]==0)

cost[i][j]=INFINITY;

else

cost[i][j]=G[i][j];

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

{

distance[i]=cost[startnode][i];

pred[i]=startnode;

visited[i]=0;

}

distance[startnode]=0;

visited[startnode]=1;

count=1;

while(count<n-1)

{

mindistance=INFINITY;

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

if(distance[i]<mindistance&&!visited[i])

{

mindistance=distance[i];

nextnode=i;

}

visited[nextnode]=1;

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

if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i])

{

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode;

}

count++;

}

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

if(i!=startnode)

{

printf("\nDistance of node%d=%d",i,distance[i]);

printf("\nPath=%d",i);

j=i;

do

{

j=pred[j];

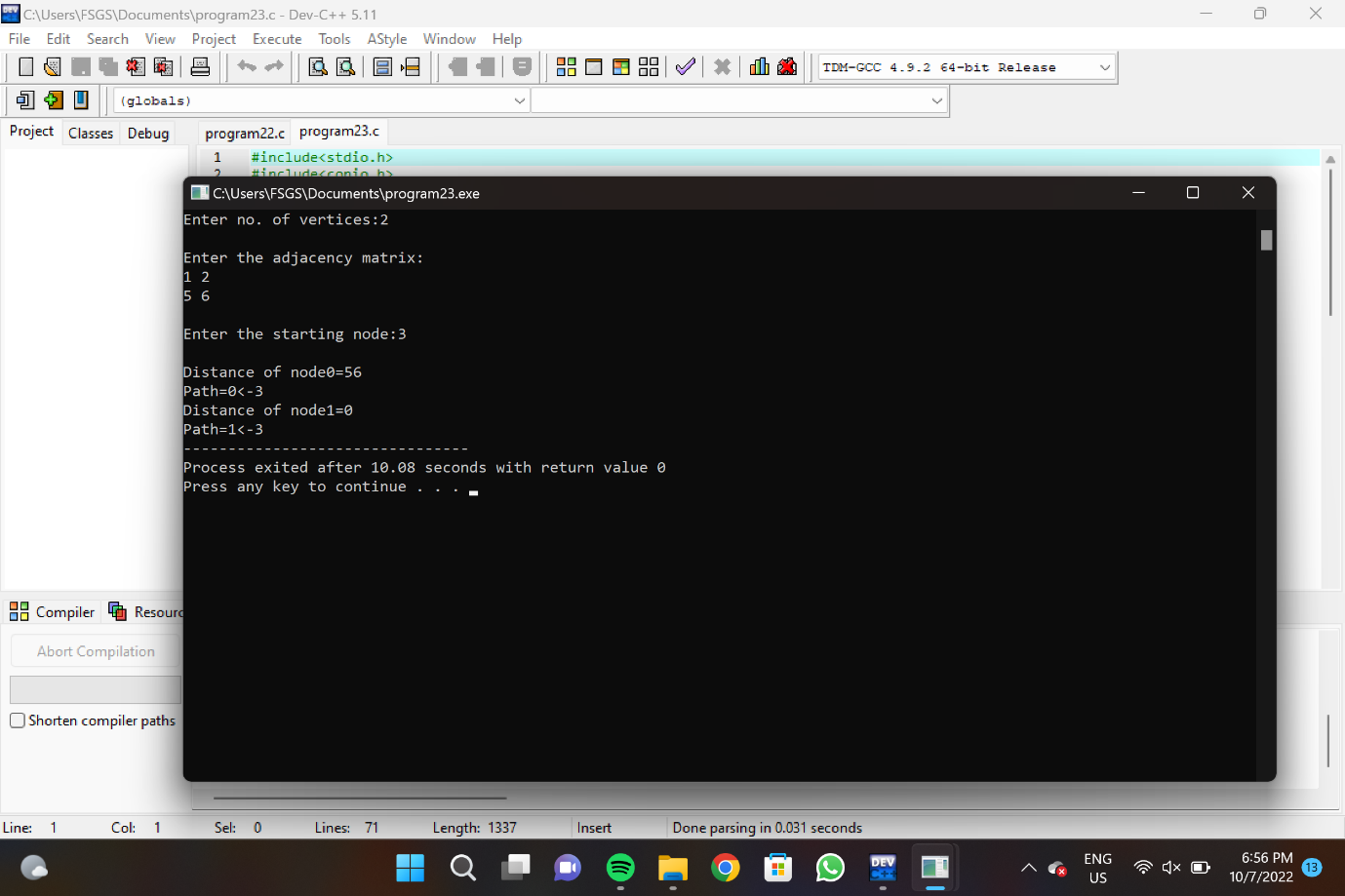
printf("<-%d",j);

}while(j!=startnode);

}

}

**OUTPUT:**

****

**PROGRAM 24**

#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()

{

int G[MAX][MAX],i,j,n,u;

printf("Enter no. of vertices:");

scanf("%d",&n);

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

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

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

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

printf("\nEnter the starting node:");

scanf("%d",&u);

dijkstra(G,n,u);

return 0;

}

void dijkstra(int G[MAX][MAX],int n,int startnode)

{

int cost[MAX][MAX],distance[MAX],pred[MAX];

int visited[MAX],count,mindistance,nextnode,i,j;

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

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

if(G[i][j]==0)

cost[i][j]=INFINITY;

else

cost[i][j]=G[i][j];

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

{

distance[i]=cost[startnode][i];

pred[i]=startnode;

visited[i]=0;

}

distance[startnode]=0;

visited[startnode]=1;

count=1;

while(count<n-1)

{

mindistance=INFINITY;

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

if(distance[i]<mindistance&&!visited[i])

{

mindistance=distance[i];

nextnode=i;

}

visited[nextnode]=1;

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

if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i])

{

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode;

}

count++;

}

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

if(i!=startnode)

{

printf("\nDistance of node%d=%d",i,distance[i]);

printf("\nPath=%d",i);

j=i;

do

{

j=pred[j];

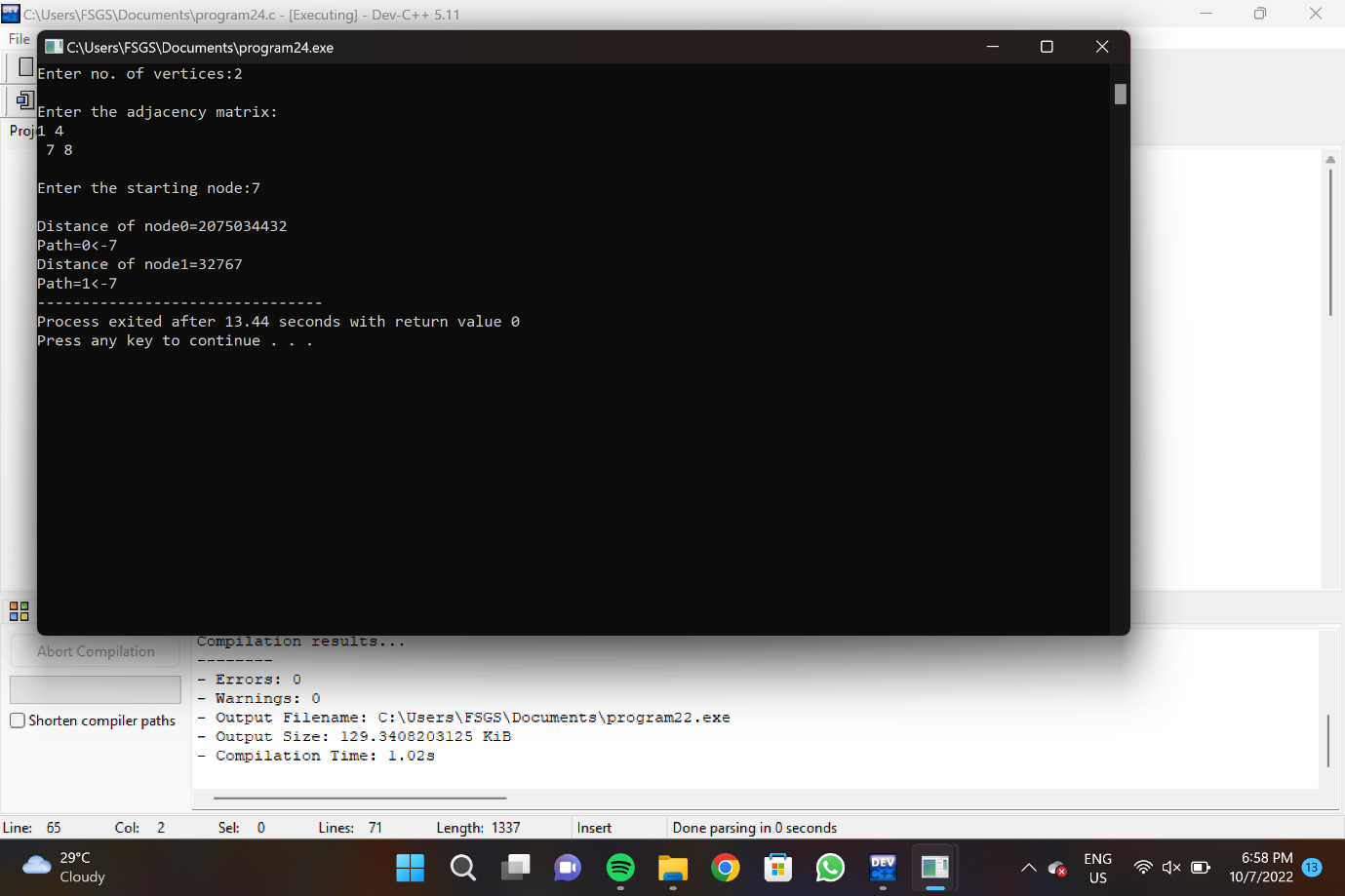
printf("<-%d",j);

}while(j!=startnode);

}

}

**OUTPUT:**

****

**PROGRAM 25**

#include <iostream>

#include <algorithm>

using namespace std;

const int MAX = 1e4 + 5;

int id[MAX], nodes, edges;

pair <long long, pair<int, int> > p[MAX];

void init()

{

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

id[i] = i;

}

int root(int x)

{

while(id[x] != x)

{

id[x] = id[id[x]];

x = id[x];

}

return x;

}

void union1(int x, int y)

{

int p = root(x);

int q = root(y);

id[p] = id[q];

}

long long kruskal(pair<long long, pair<int, int> > p[])

{

int x, y;

long long cost, minimumCost = 0;

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

{

x = p[i].second.first;

y = p[i].second.second;

cost = p[i].first;

if(root(x) != root(y))

{

minimumCost += cost;

union1(x, y);

}

}

return minimumCost;

}

int main()

{

int x, y;

long long weight, cost, minimumCost;

init();

cout <<"Enter Nodes and edges:";

cin >> nodes >> edges;

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

{

cout<<"Enter the value of X, Y and edges:";

cin >> x >> y >> weight;

p[i] = make\_pair(weight, make\_pair(x, y));

}

sort(p, p + edges);

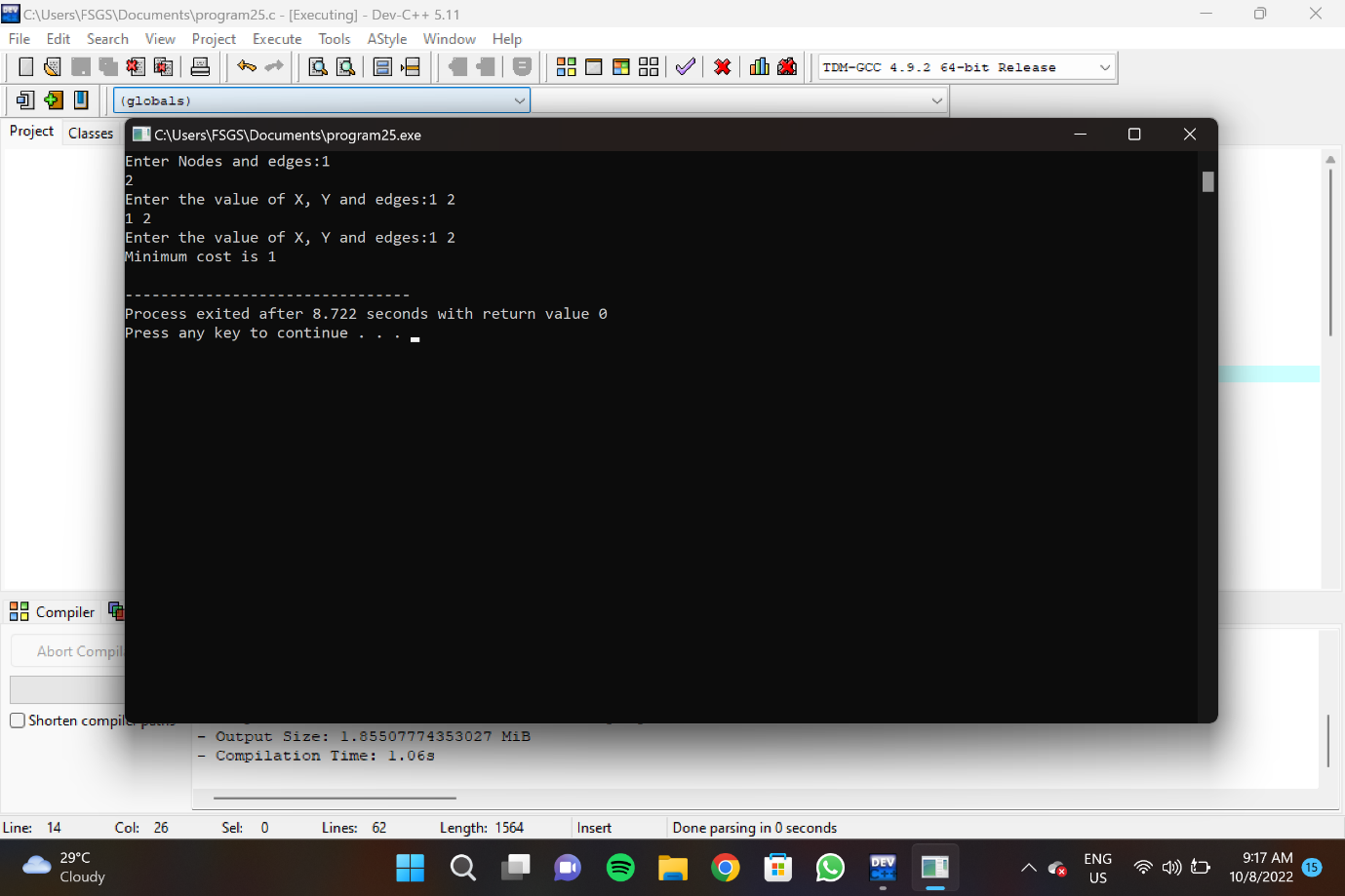
minimumCost = kruskal(p);

cout <<"Minimum cost is "<< minimumCost << endl;

return 0;

}

**OUTPUT:**

****