K.UDAYKUMAR

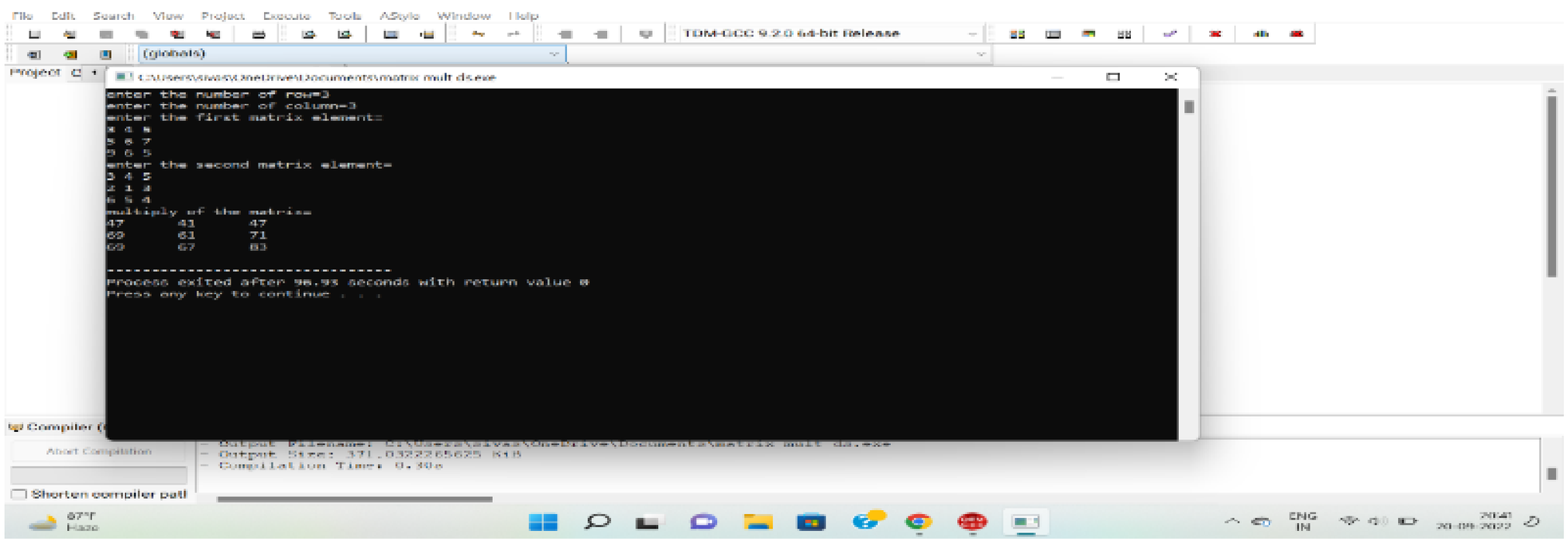
192110467

CSA0373

1.Program

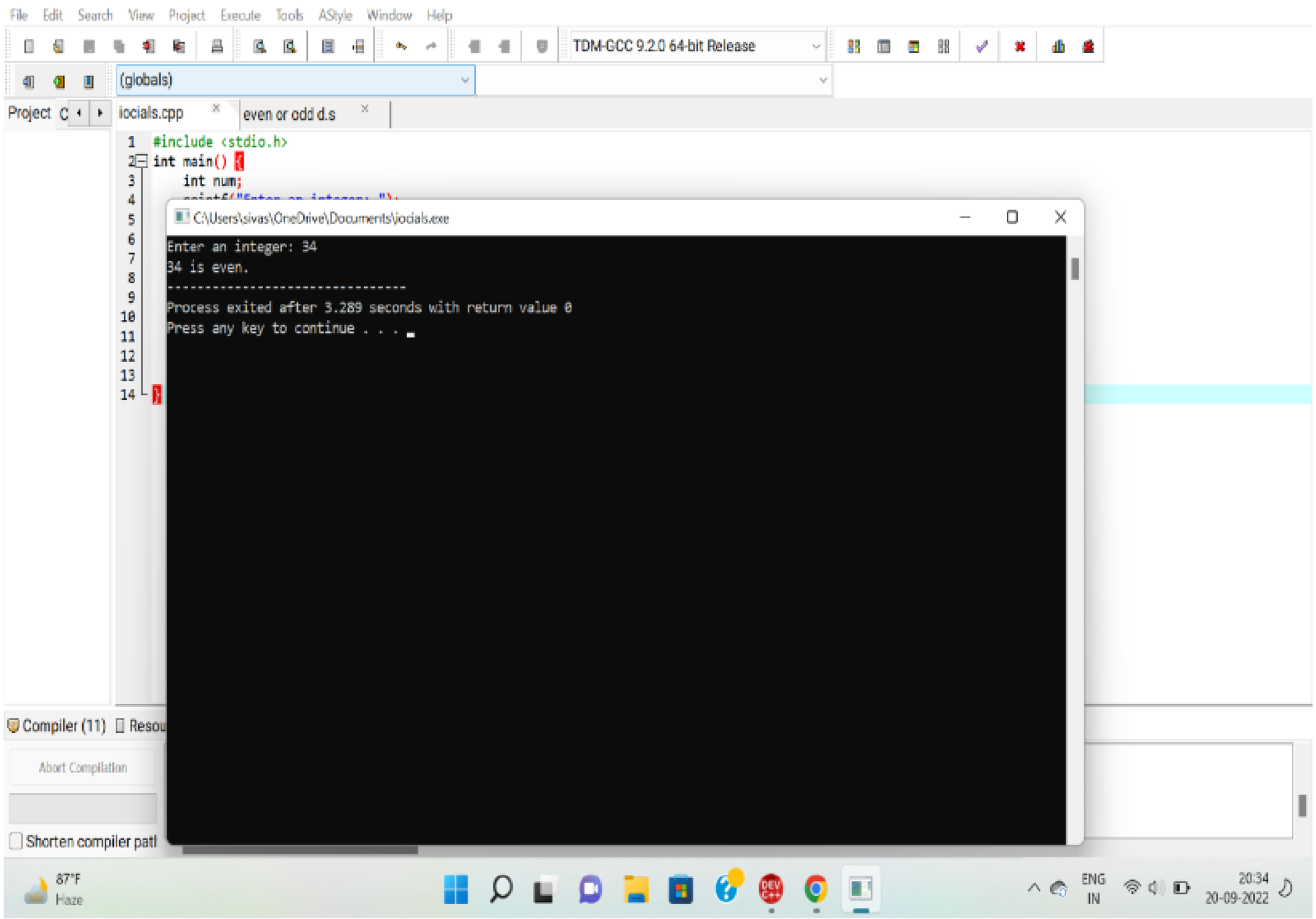
```
#include<stdio.h>
#include<stdlib.h>
int main(){
int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;
system("cls");
printf("enter the number of row=");
scanf("%d",&r);
printf("enter the number of column=");
scanf("%d",&c);
printf("enter the first matrix element=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{   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 printing result
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
printf("%d\t",mul[i][j]);
}
printf("\n");
}   return 0;
}
```



2.Program even or odd

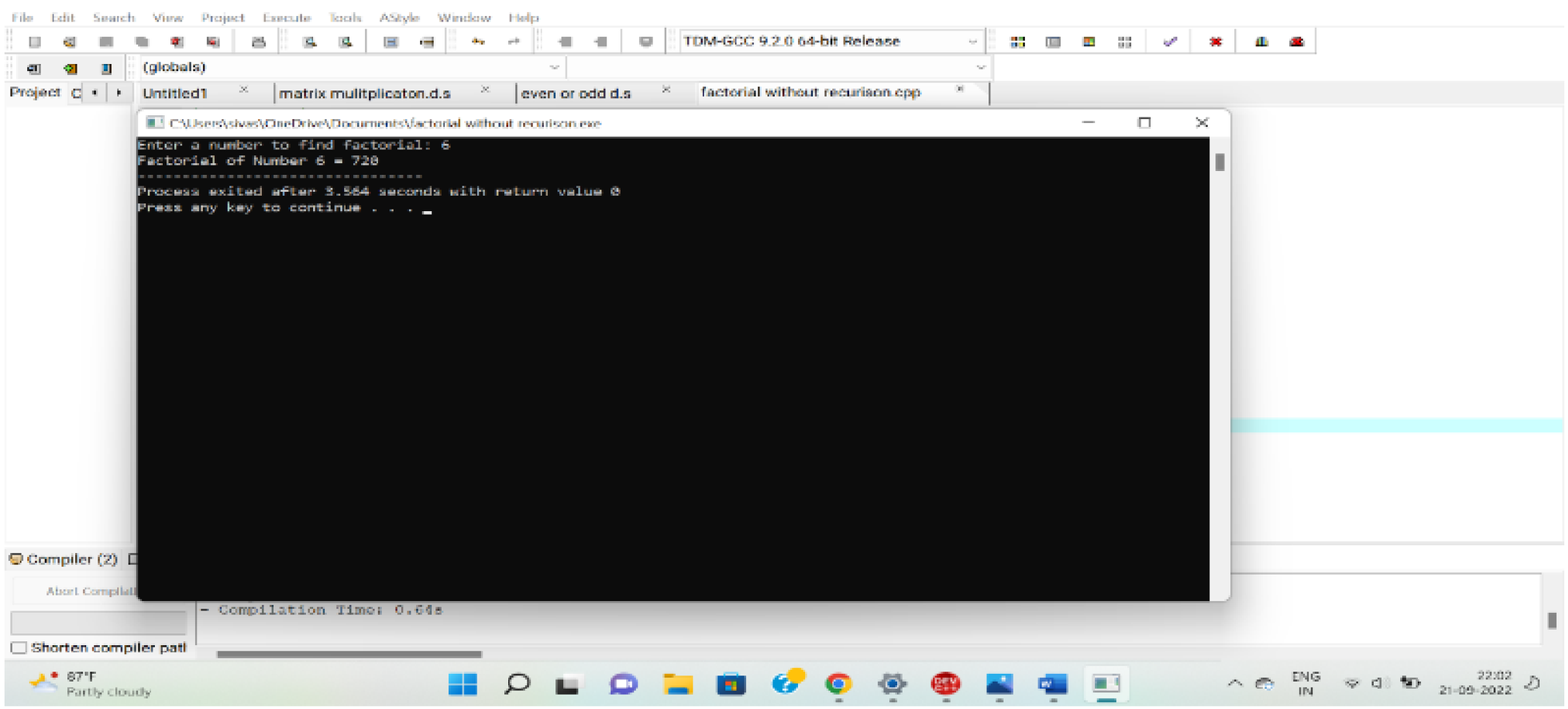
```
#include<stdio.h>

int main(){

    int num;

    printf("Enter an integer: ");

    scanf("%d",&num);

    if(num%2==0)

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

    else

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


    return 0;

}
```



3.Program Factorial without recurison

```
#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);

if (n < 0)

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

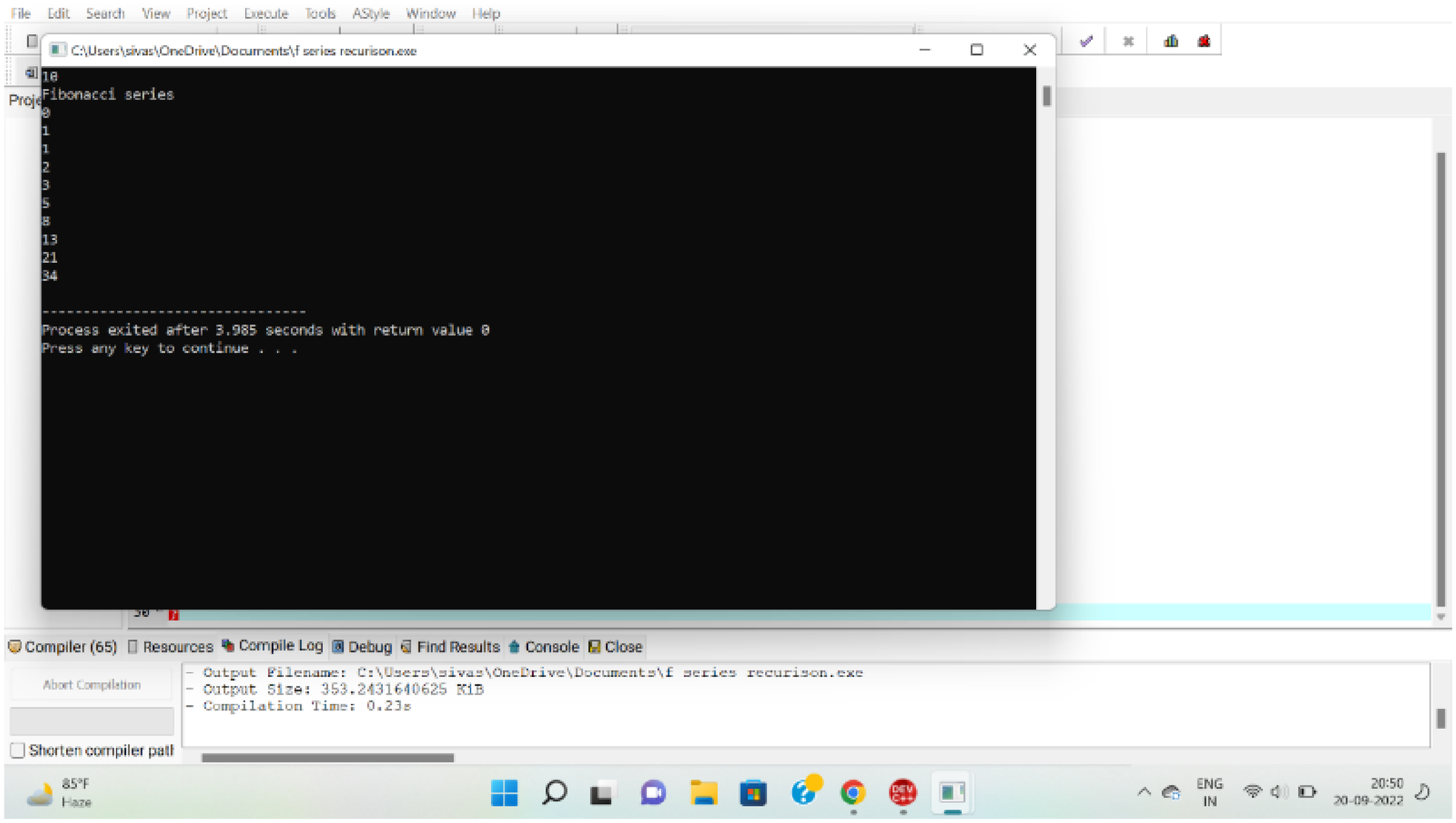
```
else
  {
     for(i=1;i<=n;++i)
     {
        factorial*=i;              //factorial=factorial*i;
     }
     printf("Factorial of Number %d = %llu",n,factorial);
  }
```



4.Program

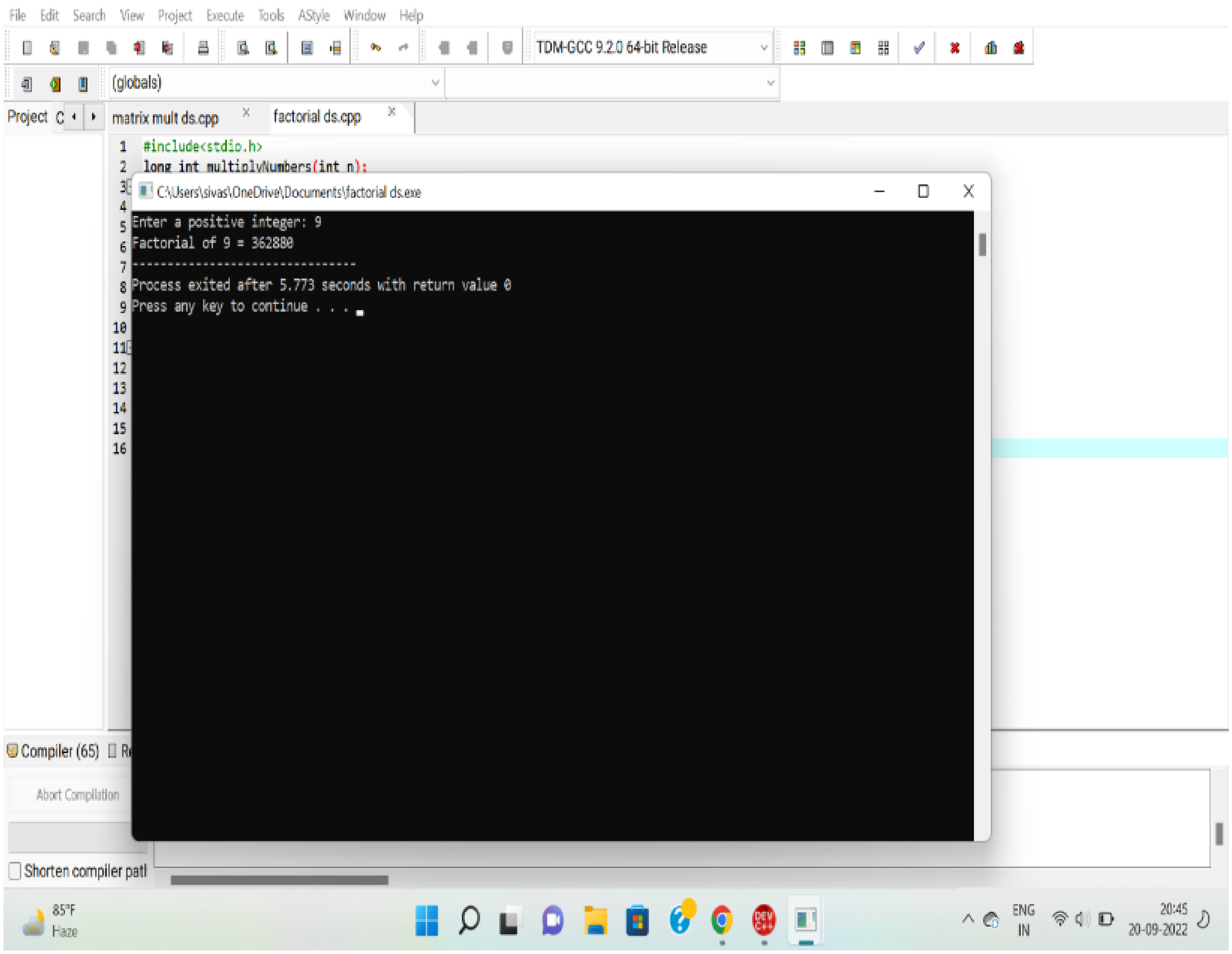
```
#include<stdio.h>


int Fibonacci(int);

int main()
{
   int n, i = 0, c;


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

```
printf("Fibonacci series\n");


for(c=1;c<=n;c++)
{
   printf("%d\n",Fibonacci(i));
   i++;
   }


   return 0;
}
 int Fibonacci(int n)
{
   if(n==0)
      return 0;
   else if(n==1)
      return 1;
   else
      return(Fibonacci(n-1)+Fibonacci(n-2));
}
```



5.Program

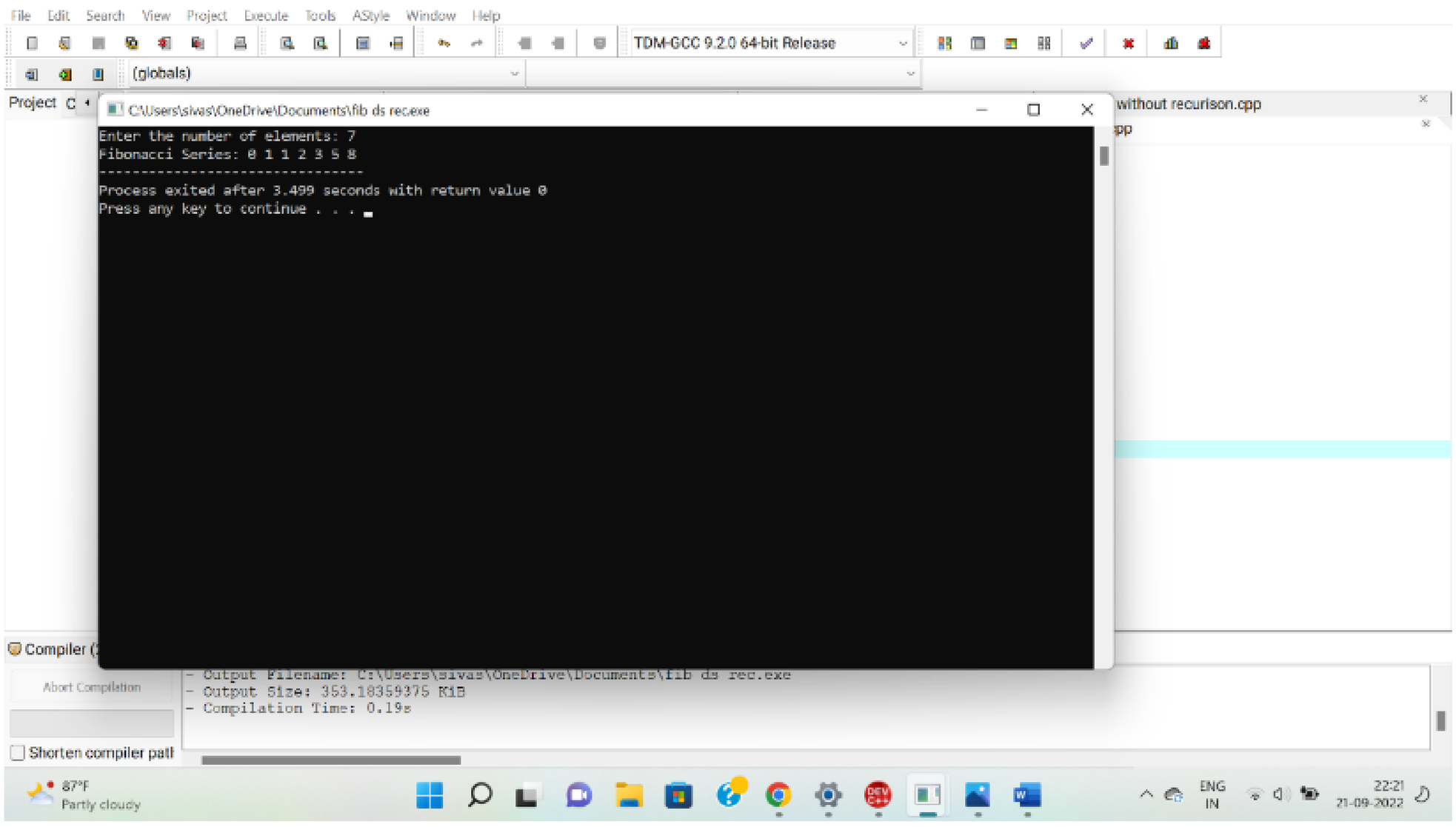
```
#include<stdio.h>

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;

}
```



6.Program

```
#include<stdio.h>

void printFibonacci(int n){

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

    if(n>0){

        n3 = n1 + n2;

        n1 = n2;

        n2 = n3;

        printf("%d ",n3);

        printFibonacci(n-1);

}
```

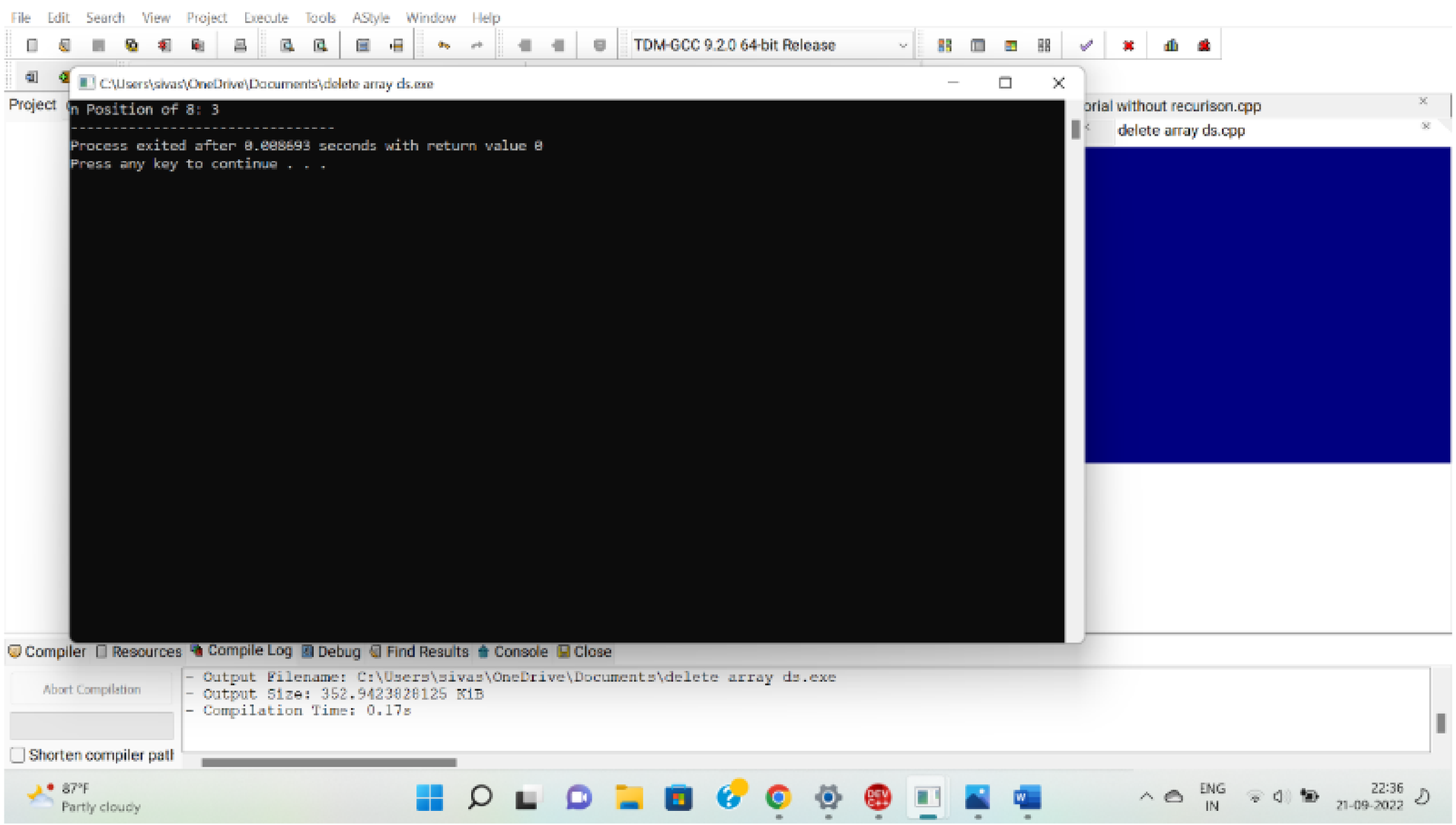
```
}
int main(){
    int n;
    printf("Enter the number of elements: ");
    scanf("%d",&n);
    printf("Fibonacci Series: ");
   printf("%d%d",0,1);
   printFibonacci(n-2);
 return 0;
```



7.program search,insert,delete

```
#include<stdio.h>

int findElement(int array[], int size, int keyToBeSearched)
{
int i;
for(i = 0; i < size; i++)
if(array[i] == keyToBeSearched)
return i;
```

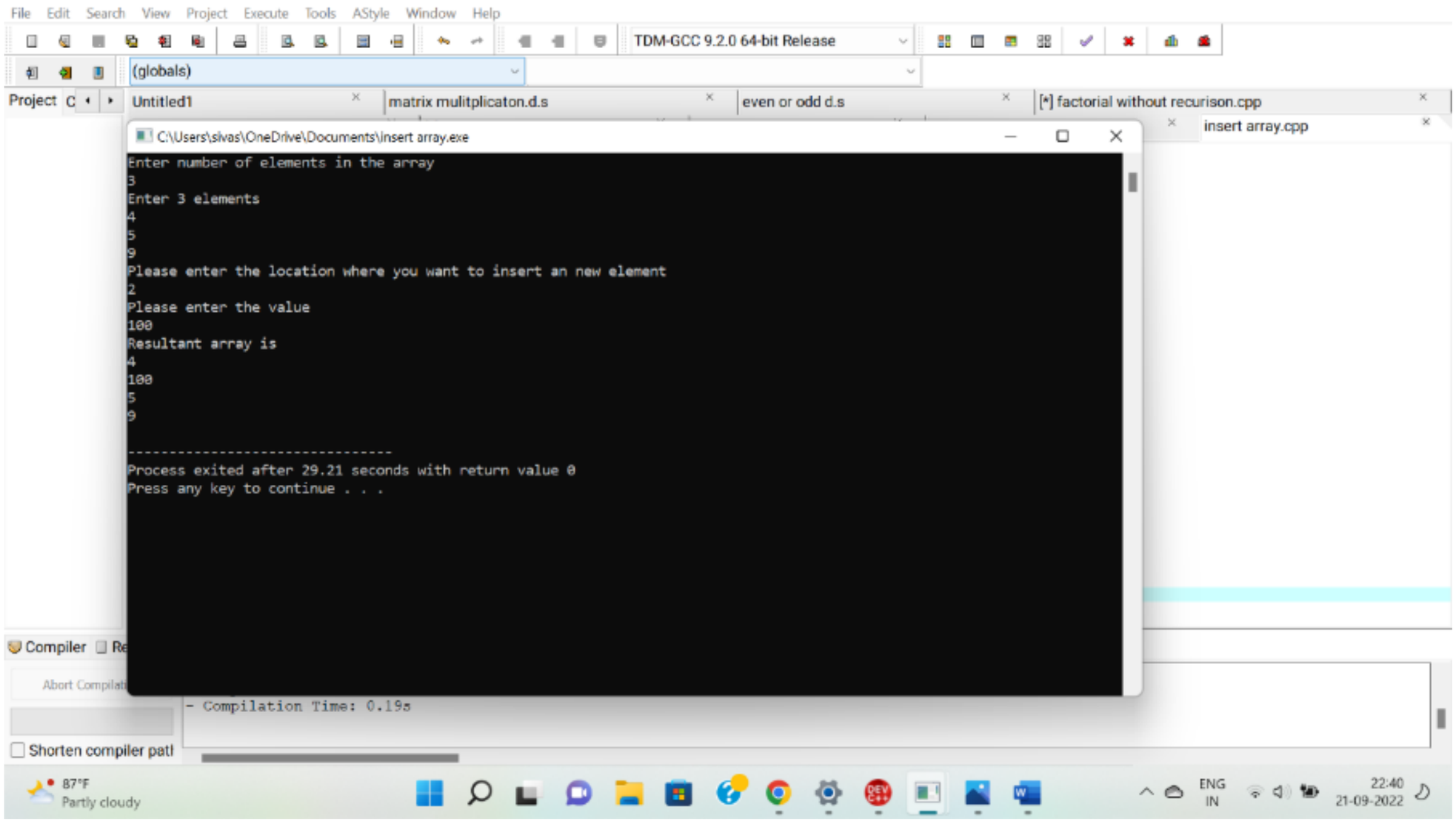
```
return -1;
}
int main()
{
int array[] = {2,3,8,9,7};
int size = sizeof(array) / sizeof(array[0]);
int keyToBeSearched = 8;
int pos = findElement(array, size, keyToBeSearched);
if(pos==-1){
printf("n Element %d not found",keyToBeSearched);
}
else{
printf("n Position of %d: %d",keyToBeSearched,pos+1);
}
return 0;
}
```



Insert

```
#include<stdio.h>
```

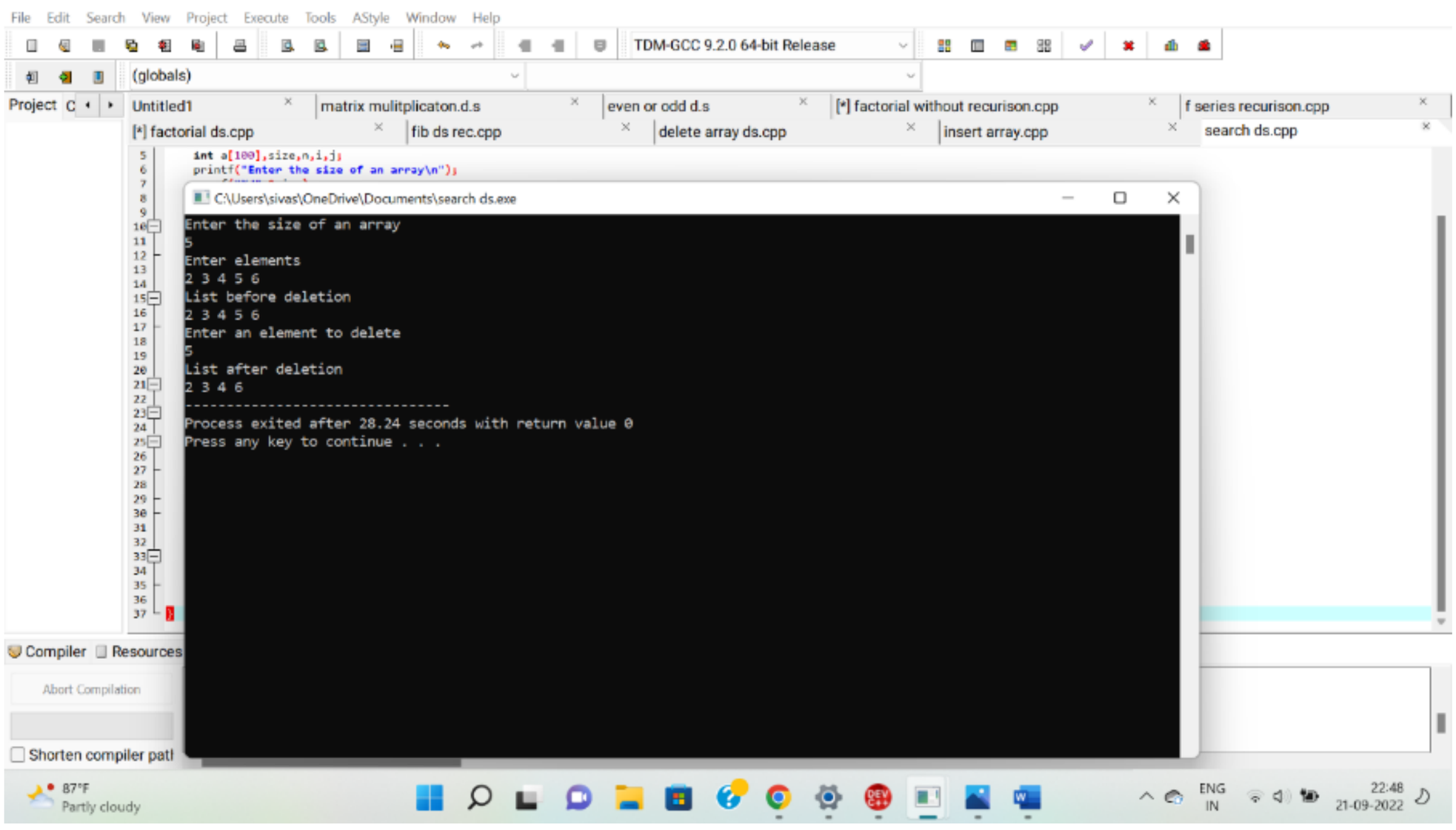
```
int main()
{ int array[50], position, c, n, value;
     printf("Enter number of elements in the array\n");
    scanf("%d", &n);
 printf("Enter %d elements\n", n);
       for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
 printf("Please enter the location where you want to insert an new element\n");
scanf("%d", &position);
printf("Please enter the value\n");
scanf("%d", &value);
for (c = n - 1; c >= position - 1; c--)
    array[c+1] = array[c];
 array[position-1] = value;
  printf("Resultant array is\n");
       for (c = 0; c <= n; c++)
         printf("%d\n", array[c]);
       return 0;
}
```



Delete

```
#include<stdio.h>

#include<stdlib.h>

int main()

{

    int a[100],size,n,i,j;

    printf("Enter the size of an array\n");

    scanf("%d",&size);

    printf("Enter elements\n");

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

    {

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

    }

    printf("List before deletion\n");

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

    {

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

    }
```

```
printf("\nEnter an element to delete\n");
scanf("%d",&n);
for(i=0;i<size;i++)
{
    if(a[i]==n)
    {
        for(j=i;j<(size-1);j++)
        {
            a[j]=a[j+1];
        }
        break;
    }
}
printf("List after deletion\n");
for(i=0;i<(size-1);i++)
    {
        printf("%d ",a[i]);
    }
    return 0;
}
```



8.Program linear search

#include<stdio.h>

int main()

{

in#t a[20],i,x,n;

printf("How many elements?");

scanf("%d",&n);

printf("Enter array elements:n");

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

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

printf("nEnter element to search:");

scanf("%d",&x);

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

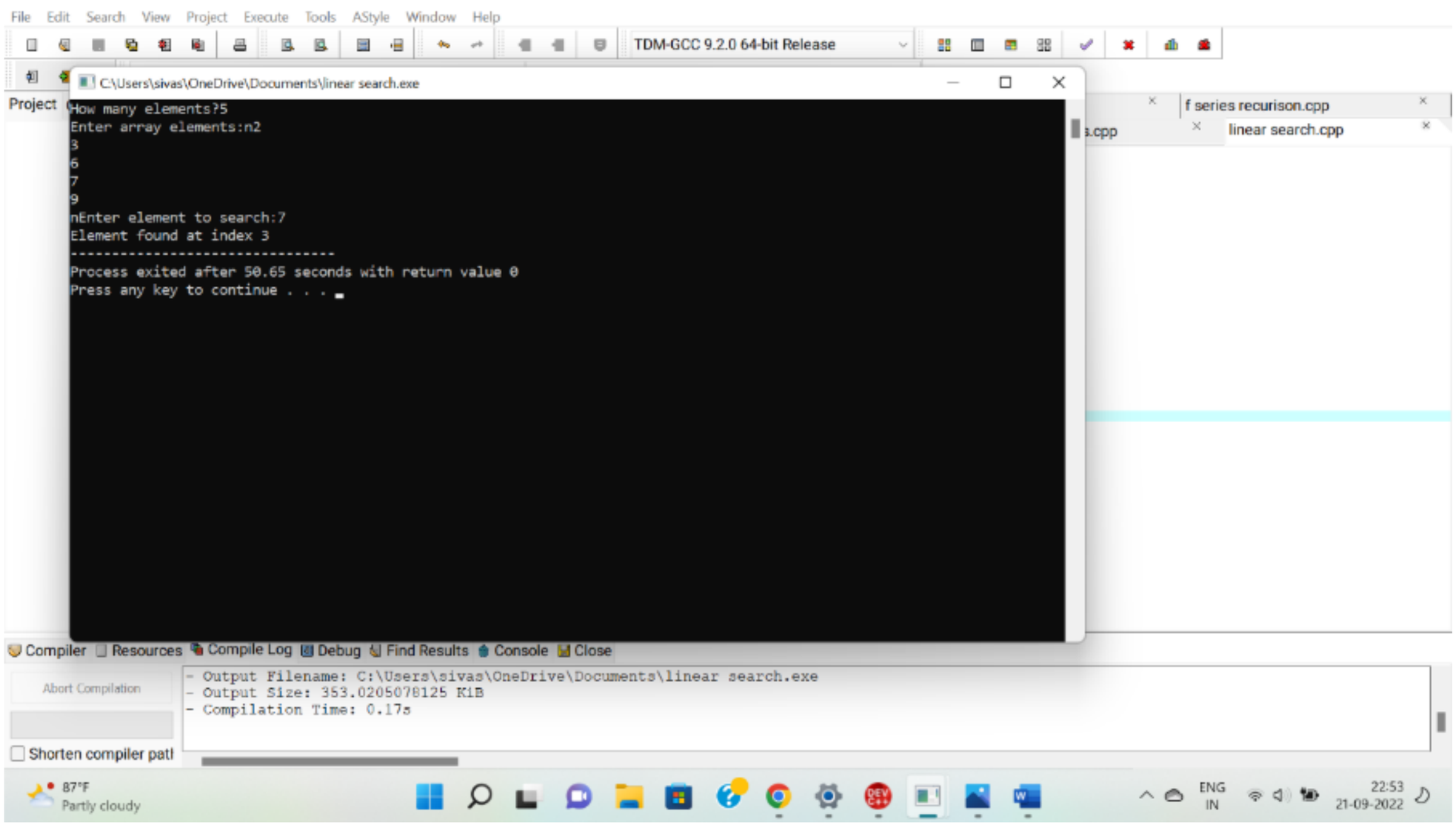
if(a[i]==x)

break;

if(i<n)

printf("Element found at index %d",i);

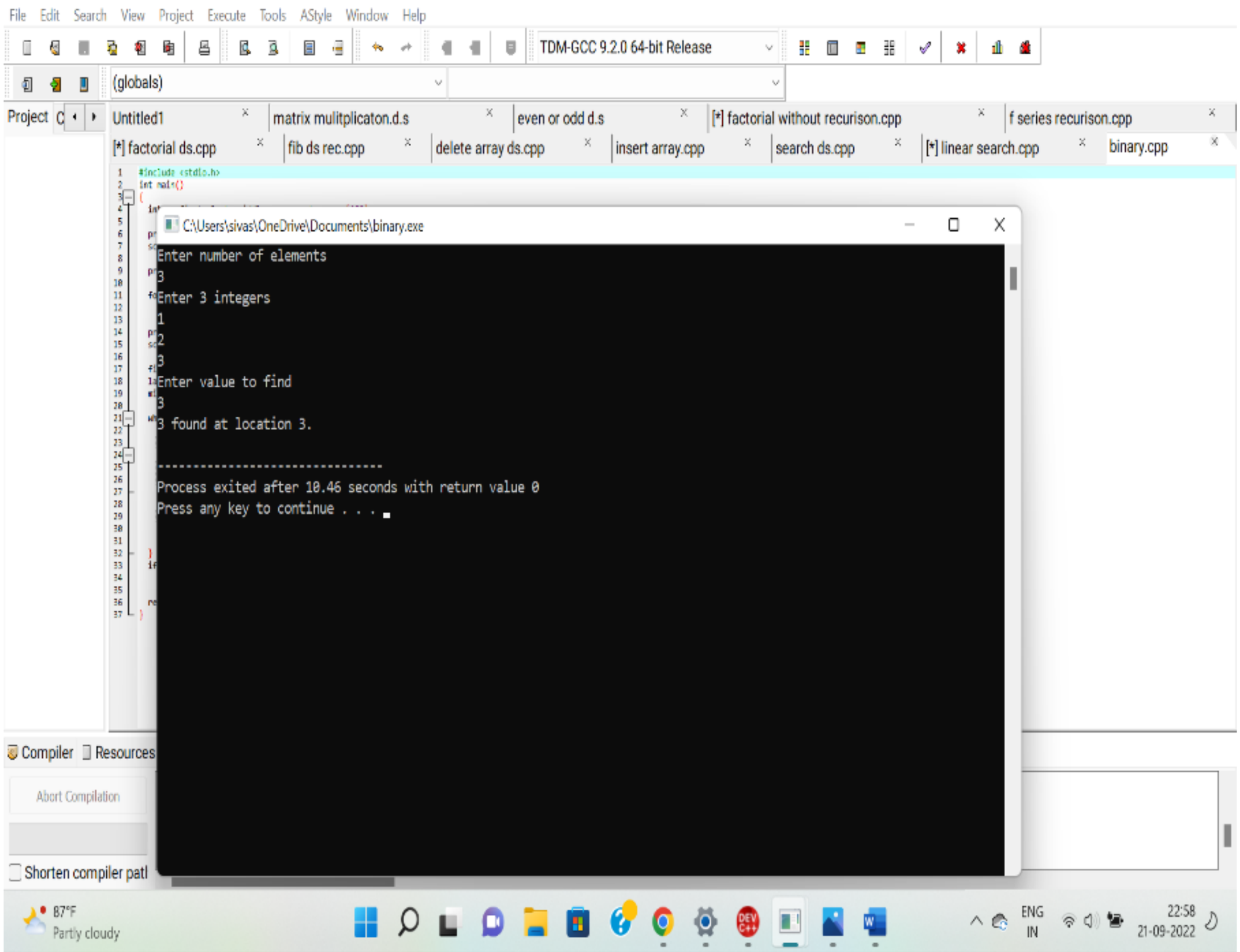
```
else
printf("Element not found");


return 0;
}
```



9.program binary search

```
#include<stdio.h>

int main()

{

  int c, first, last, middle, n, search, array[100];

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

  scanf("%d", &n);

 printf("Enter %d integers\n", n);

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

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

  printf("Enter value to find\n");

  scanf("%d", &search);
```

```
first = 0;

last = n - 1;

middle = (first+last)/2;


while (first <= last) {

  if (array[middle] < search)

    first = middle + 1;

   else if (array[middle] == search) {

    printf("%d found at location %d.\n", search, middle+1);

    break;

   }

   else

    last = middle - 1;

middle = (first + last)/2;

 }

  if (first > last)

    printf("Not found! %d isn't present in the list.\n", search);

  return 0;

}
```



10.Program linked list

```
#include<stdio.h>

#include<stdlib.h>


// Create a node

struct Node{

  int data;

  struct Node* next;

};


// Insert at the beginning

void insertAtBeginning(struct Node** head_ref, int new_data){

  // Allocate memory to a node
```

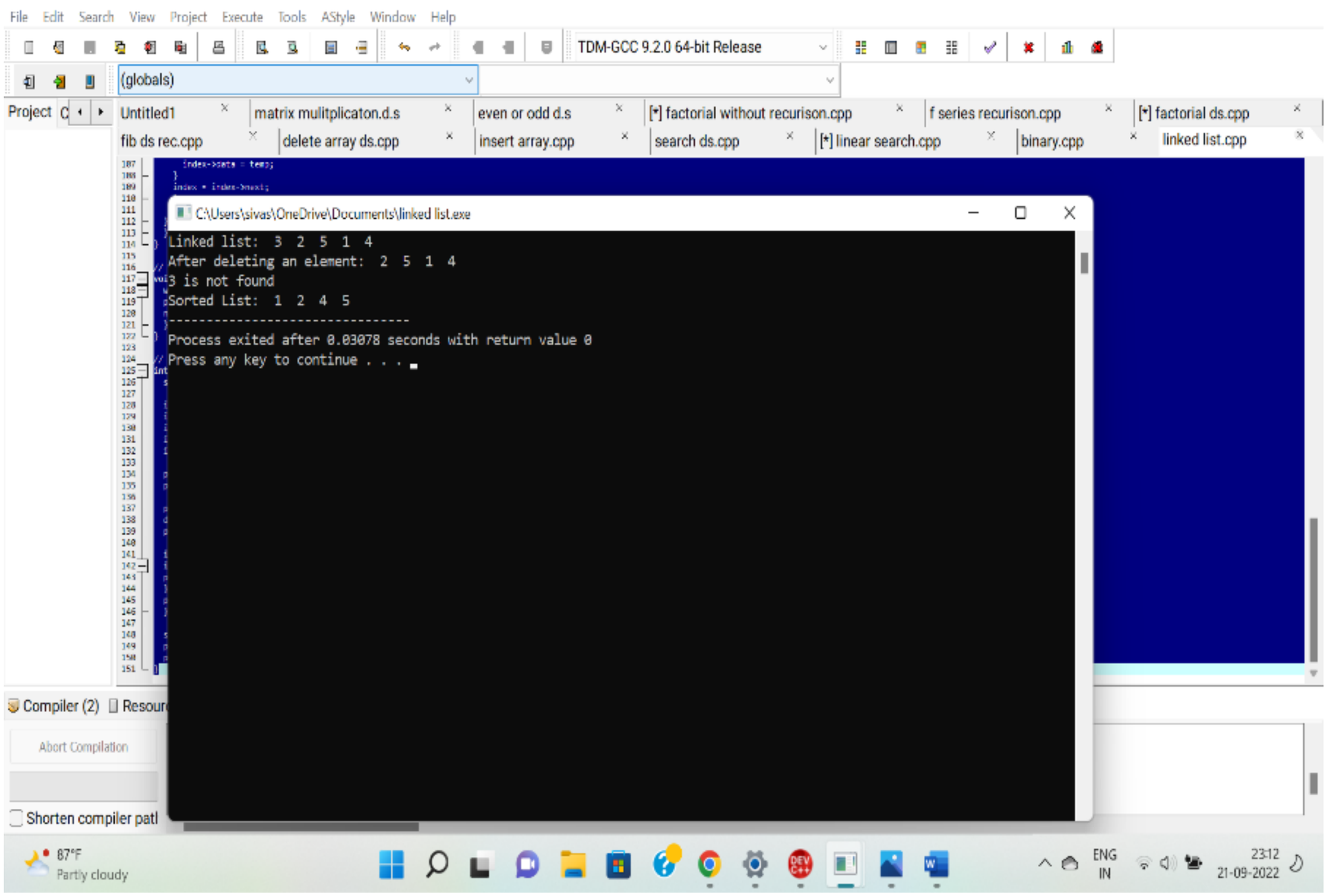
```
struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

// insert the data
new_node->data = new_data;

new_node->next = (*head_ref);
  // Move head to new node
  (*head_ref) = new_node;
}

// Insert a node after a 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;
}

// Insert the the end
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;

}


// Delete a node

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;

  }

// Find the key to be deleted

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

prev = temp;

temp = temp->next;

}
```

```
// If the key is not present
if (temp == NULL) return;

// Remove the node
prev->next = temp->next;
  free(temp);
}

// Search a node
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;
}

// Sort the linked list
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 points to the node next to current
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;
 }
 }
}

// Print the linked list
void printList(struct Node* node) {
  while (node != NULL) {
  printf(" %d ",node->data);
  node = node->next;
  }
}

// Driver program
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);
}
```



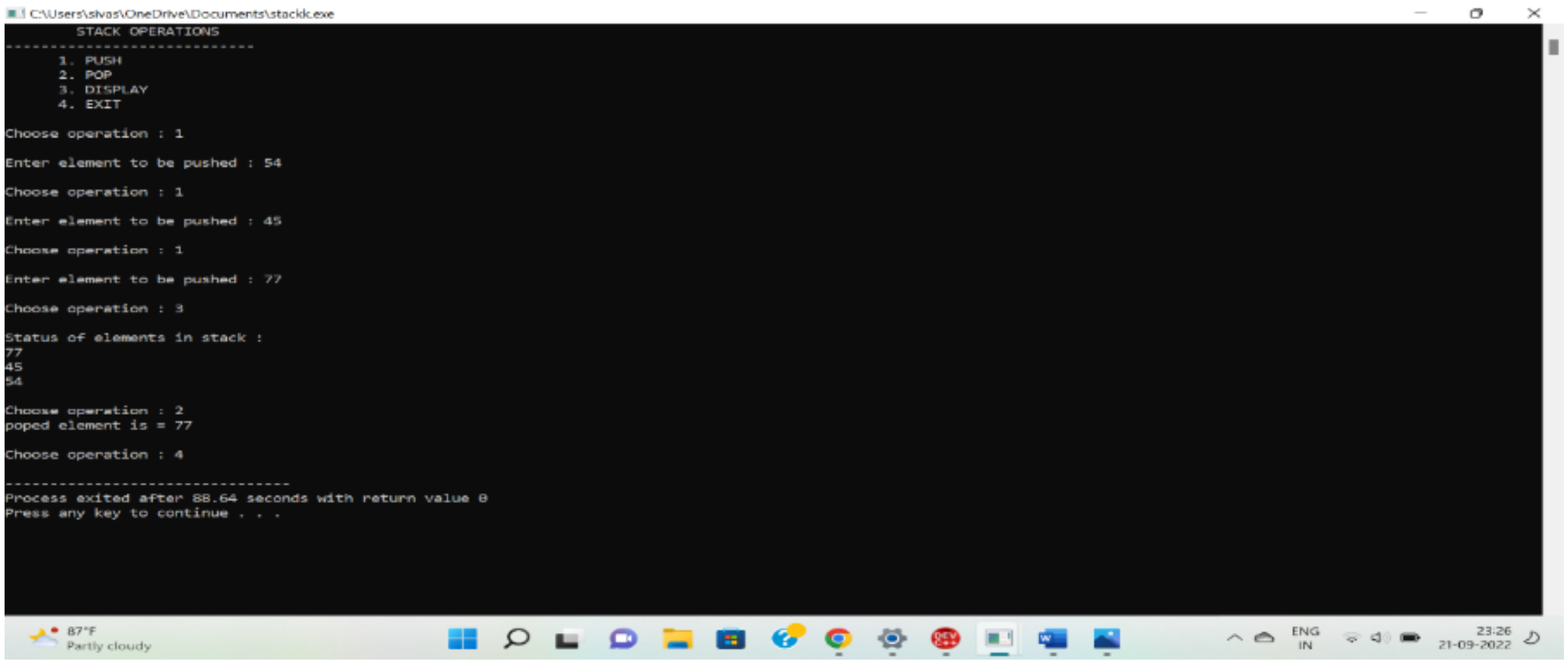
11.Program push,pop,peek

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 5
struct stack
{
    int stk[MAXSIZE];
    int top;
};
typedef struct stack ST;
ST s;
/*Function to add an element to stack */
void push ()
{
    int num;
```

```
if(s.top==(MAXSIZE-1))
{
    printf("StackisFull\n");
    return;
}
else
    {
        printf("\nEnter element to be pushed:");
        scanf("%d",&num);
        s.top=s.top+1;
        s.stk[s.top]=num;
    }
    return;
}
/*Function to delete an element from stack*/
int pop()
{
    int num;
    if(s.top==-1)
    {
        printf("StackisEmpty\n");
        return(s.top);
    }
else
{
    num=s.stk[s.top];
    printf("poped element is=%d\n",s.stk[s.top]);
    s.top=s.top-1;
}
```

```
        return(num);
}
/*Function to display the status of stack*/
void display()
{
        int i;
        if(s.top == -1)
        {
                printf("Stack is empty\n");
                return;
        }
        else
        {
                printf("\nStatus of elements in stack: \n");
                for(i = s.top; i >= 0; i--)
                        {
                                printf("%d\n", s.stk[i]);
                        }
        }
}
int main()
{
        int ch;
        s.top = -1;

        printf("\tSTACK OPERATIONS\n");
        printf("--------------------\n");
        printf("      1. PUSH\n");
        printf("      2. POP\n");
```

```
printf("      3. DISPLAY\n");
printf("      4. EXIT\n");
//printf("-----------------\n");
while(1)
{
    printf("\nChoose operation: ");
scanf("%d",&ch);
switch(ch)
{
    case 1:
        push();
    break;
    case 2:
        pop();
    break;
    case 3:
        display();
    break;
    case 4:
        exit(0);
    default:
        printf("Invalid operation \n");
}
    }
    return 0;
}
```



12.Program queue operations

```
#include<stdio.h>

#include<stdlib.h>

#include<limits.h>


// Queue capacity

#define CAPACITY 100


/**

* Global queue declaration.

*/

int queue[CAPACITY];

unsigned int size  = 0;

unsigned int rear  = CAPACITY - 1;   // Initally assumed that rear is at end

unsigned int front = 0;



/* Function declaration for various operations on queue */
```

```
int enqueue(int data);

int dequeue();

int isFull();

int isEmpty();

int getRear();

int getFront();

/* Driver function */

int main()

{

    int ch, data;

    /* Run indefinitely until user manually terminates */

    while (1)

    {

        /* Queue menu */

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

        printf("  QUEUE ARRAY IMPLEMENTATION PROGRAM  \n");

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

        printf("1. Enqueue\n");

        printf("2. Dequeue\n");

        printf("3. Size\n");

        printf("4. Get Rear\n");

        printf("5. Get Front\n");

        printf("0. Exit\n");

        printf("-------------------------\n");
```

```
printf("Select an option: ");

scanf("%d",&ch);


/* Menu control switch */
switch (ch)
{
    case 1:
        printf("\nEnter data to enqueue: ");
        scanf("%d",&data);

        // Enqueue function returns 1 on success
        // otherwise 0
        if (enqueue(data))
            printf("Element added to queue.");
    else
        printf("Queue is full.");

    break;

case 2:
    data = dequeue();
    // on success dequeue returns element removed
    // otherwise returns INT_MIN
    if (data == INT_MIN)
        printf("Queue is empty.");
    else
```

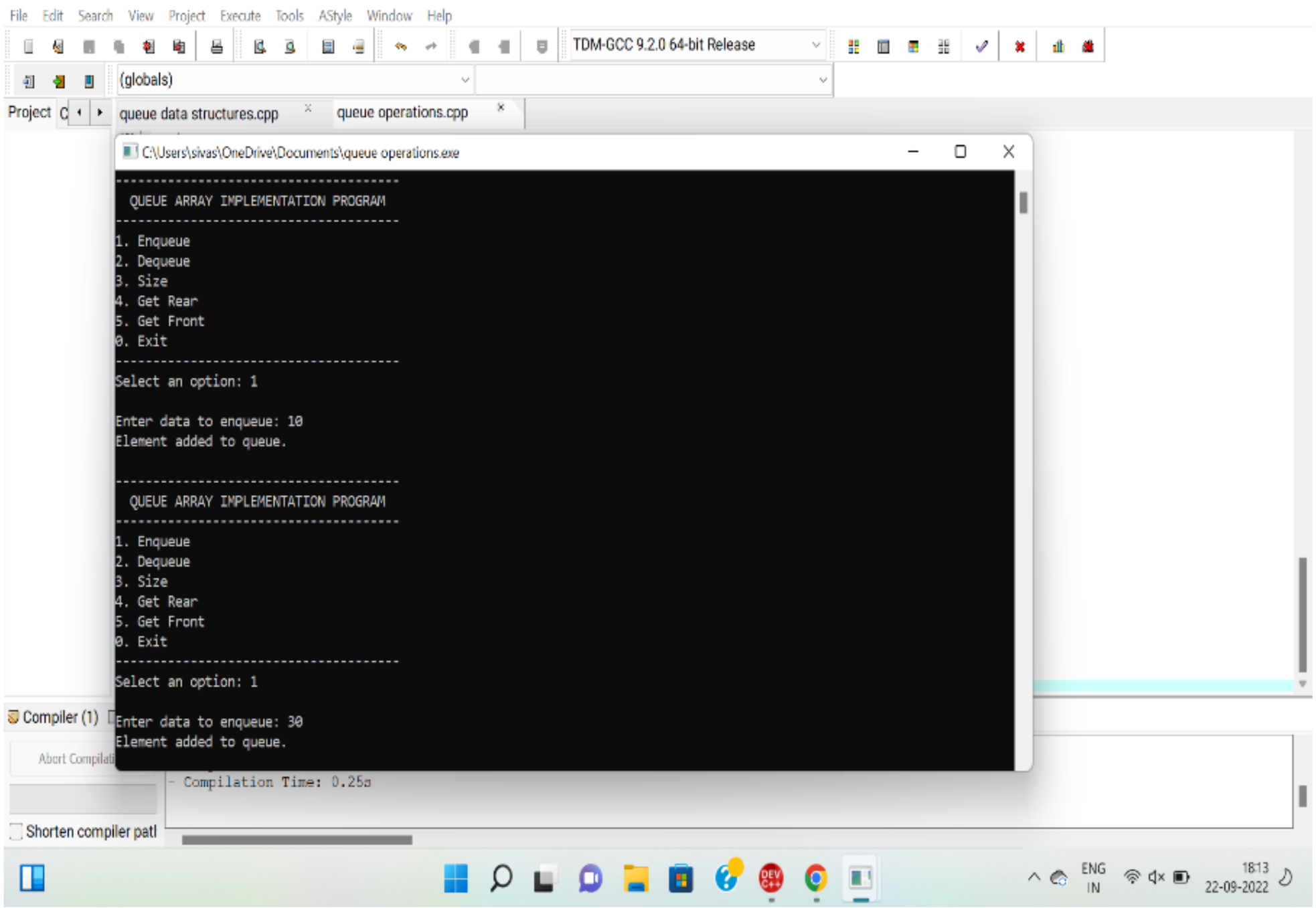
```
        printf("Data => %d",data);

    break;

case 3:
// isEmpty() function returns 1 if queue is emtpy
// otherwise returns 0
if (isEmpty())
    printf("Queue is empty.");
else
    printf("Queue size => %d", size);


break;

case 4:


    if (isEmpty())
        printf("Queue is empty.");
    else
        printf("Rear => %d",getRear());


    break;

case 5:


    if (isEmpty())
        printf("Queue is empty.");
    else
```

```
            printf("Front => %d", getFront());

        break;

    case 0:
        printf("Exiting from app.\n");
            exit(0);

        default:
            printf("Invalid choice, please input number between (0-5).");
            break;
    }

    printf("\n\n");
}
}



/**
* Enqueue/Insert an element to the queue.
*/
int enqueue(int data)
{
    // Queue is full throw Queue out of capacity error.
    if (isFull())
    {
        return 0;
    }
```

```
    // Ensure rear never crosses array bounds
    rear = (rear + 1) % CAPACITY;

    // Increment queue size
    size++;

    // Enqueue new element to queue
    queue[rear] = data;

    // Successfully enqueued element to queue
    return 1;
}

/**
 * Dequeue/Remove an element from the queue.
 */
int dequeue()
{
    int data = INT_MIN;

    // Queue is empty, throw Queue underflow error
    if (isEmpty())
    {
        return INT_MIN;
    }

    // Dequeue element from queue
```

```
    data = queue[front];

    // Ensure front never crosses array bounds
    front = (front + 1) % CAPACITY;

    // Decrease queue size
    size--;

    return data;
}

/**
 * Checks if queue is full or not. It returns 1 if queue is full,
 * overwise returns 0.
 */
int isFull()
{
    return (size == CAPACITY);
}

/**
 * Checks if queue is empty or not. It returns 1 if queue is empty,
 * otherwise returns 0.
 */
int isEmpty()
{
    return (size == 0);
```

```
}



/**
 * Gets, front of the queue. If queue is empty return INT_MAX otherwise
 * returns front of queue.
 */
int getFront()
{
    return (isEmpty())
            ? INT_MIN
            : queue[front];
}
/**
 * Gets, rear of the queue. If queue is empty return INT_MAX otherwise
 * returns rear of queue.
 */
int getRear()
{
    return (isEmpty())
            ? INT_MIN
            : queue[rear];
}
```



13.program tree

```
#include<stdio.h>

#include<stdlib.h>

#include<limits.h>


//Queue capacity

#define CAPACITY 100


/**

*Global queue declaration.

*/

int queue[CAPACITY];

unsigned int size  = 0;

unsigned int rear  = CAPACITY - 1;   //Initally assumed that rear is at end

unsigned int front = 0;
```

```
/* Function declaration for various operations on queue */
int enqueue(int data);
int dequeue();
int isFull();
int isEmpty();
int getRear();
int getFront();



/* Driver function */
int main()
{
    int ch, data;


    /* Run indefinitely until user manually terminates */
    while (1)
    {
        /* Queue menu */
        printf("-------------------------\n");
        printf("  QUEUE ARRAY IMPLEMENTATION PROGRAM  \n");
        printf("-------------------------\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Size\n");
```

```
printf("4. Get Rear\n");
printf("5. Get Front\n");
printf("0. Exit\n");
printf("-----------------------\n");
printf("Select an option: ");
scanf("%d",&ch);


/* Menu control switch */
switch (ch)
{
    case 1:
        printf("\nEnter data to enqueue: ");
        scanf("%d",&data);

// Enqueue function returns 1 on success
// otherwise 0
if (enqueue(data))
    printf("Element added to queue.");
else
    printf("Queue is full.");
    break;

case 2:
    data = dequeue();

    // on success dequeue returns element removed
```

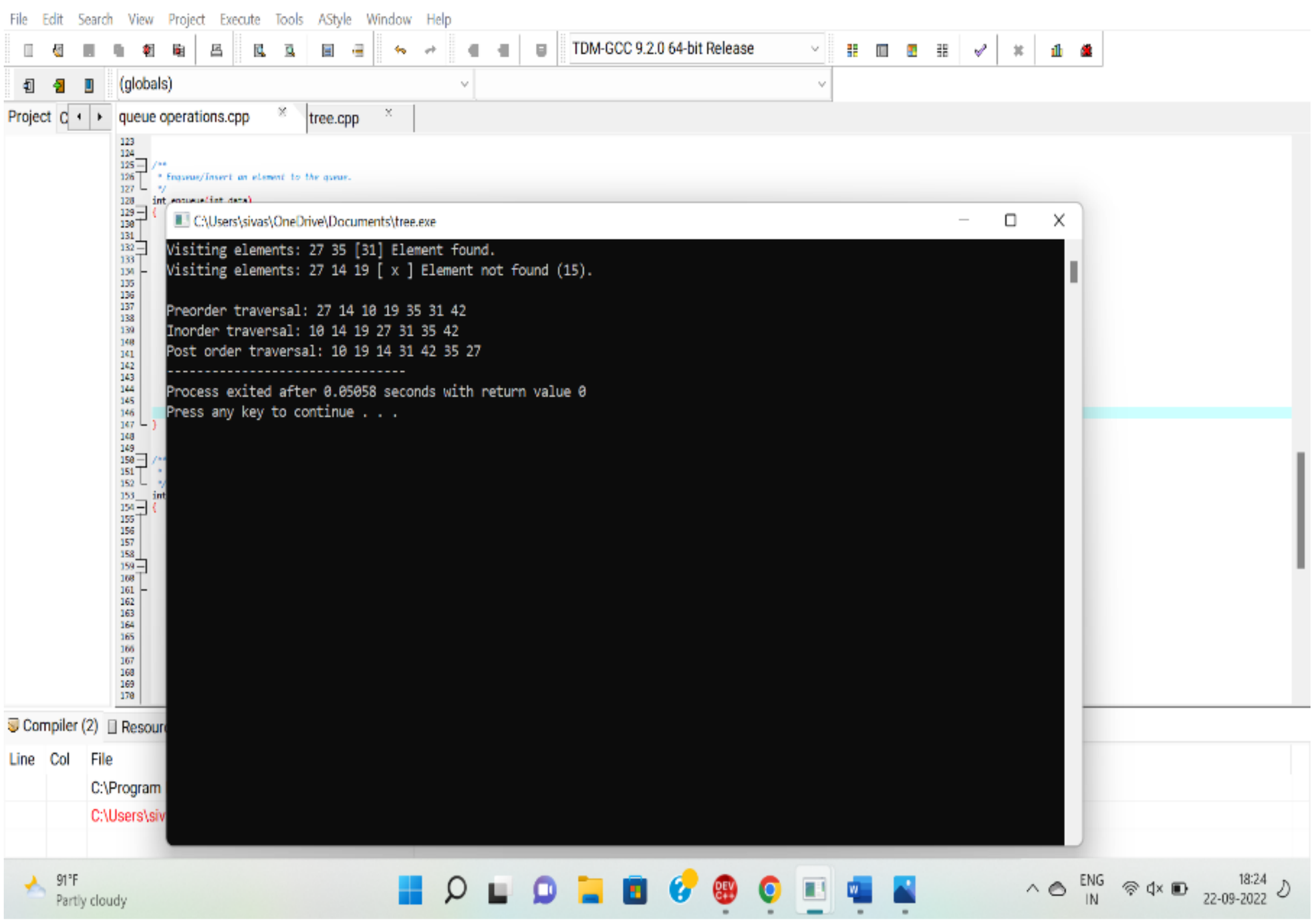
```
// otherwise returns INT_MIN
if (data == INT_MIN)
    printf("Queue is empty.");
else
    printf("Data => %d", data);

    break;

case 3:

    // isEmpty() function returns 1 if queue is emtpy
    // otherwise returns 0
    if (isEmpty())
        printf("Queue is empty.");
    else
        printf("Queue size => %d", size);

    break;

case 4:

    if (isEmpty())
        printf("Queue is empty.");
    else
        printf("Rear => %d", getRear());

    break;

case 5:
```

```
if (isEmpty())
    printf("Queue is empty.");
else
    printf("Front => %d", getFront());

    break;


case 0:
    printf("Exiting from app.\n");
    exit(0);


default:
    printf("Invalid choice, please input number between (0-5).");
    break;
        }


        printf("\n\n");
    }
}
/**
 * Enqueue/Insert an element to the queue.
 */
int enqueue(int data)
{
    // Queue is full throw Queue out of capacity error.
```

```
    if (isFull())
    {
        return 0;
    }

    // Ensure rear never crosses array bounds
    rear = (rear + 1) % CAPACITY;

    // Increment queue size
    size++;

    // Enqueue new element to queue
    queue[rear] = data;

    // Successfully enqueued element to queue
    return 1;
}



/**
 * Dequeue/Remove an element from the queue.
 */
int dequeue()
{
    int data = INT_MIN;

    // Queue is empty, throw Queue underflow error
    if (isEmpty())
    {
```

```
    return INT_MIN;
}

// Dequeue element from queue
data = queue[front];

    // Ensure front never crosses array bounds
    front = (front + 1) % CAPACITY;

    // Decrease queue size
    size--;

    return data;
}

/**
* Checks if queue is full or not. It returns 1 if queue is full,
* overwise returns 0.
*/
int isFull()
{
    return (size == CAPACITY);
}

/**
* Checks if queue is empty or not. It returns 1 if queue is empty,
* otherwise returns 0.
```

```
 */
int isEmpty()
{
    return (size == 0);
}

/**
 * Gets, front of the queue. If queue is empty return INT_MAX otherwise
 * returns front of queue.
 */
int getFront()
{
    return (isEmpty())
            ? INT_MIN
            : queue[front];
}



/**
 * Gets, rear of the queue. If queue is empty return INT_MAX otherwise
 * returns rear of queue.
 */
int getRear()
{
    return (isEmpty())
            ? INT_MIN
            : queue[rear];
}
```



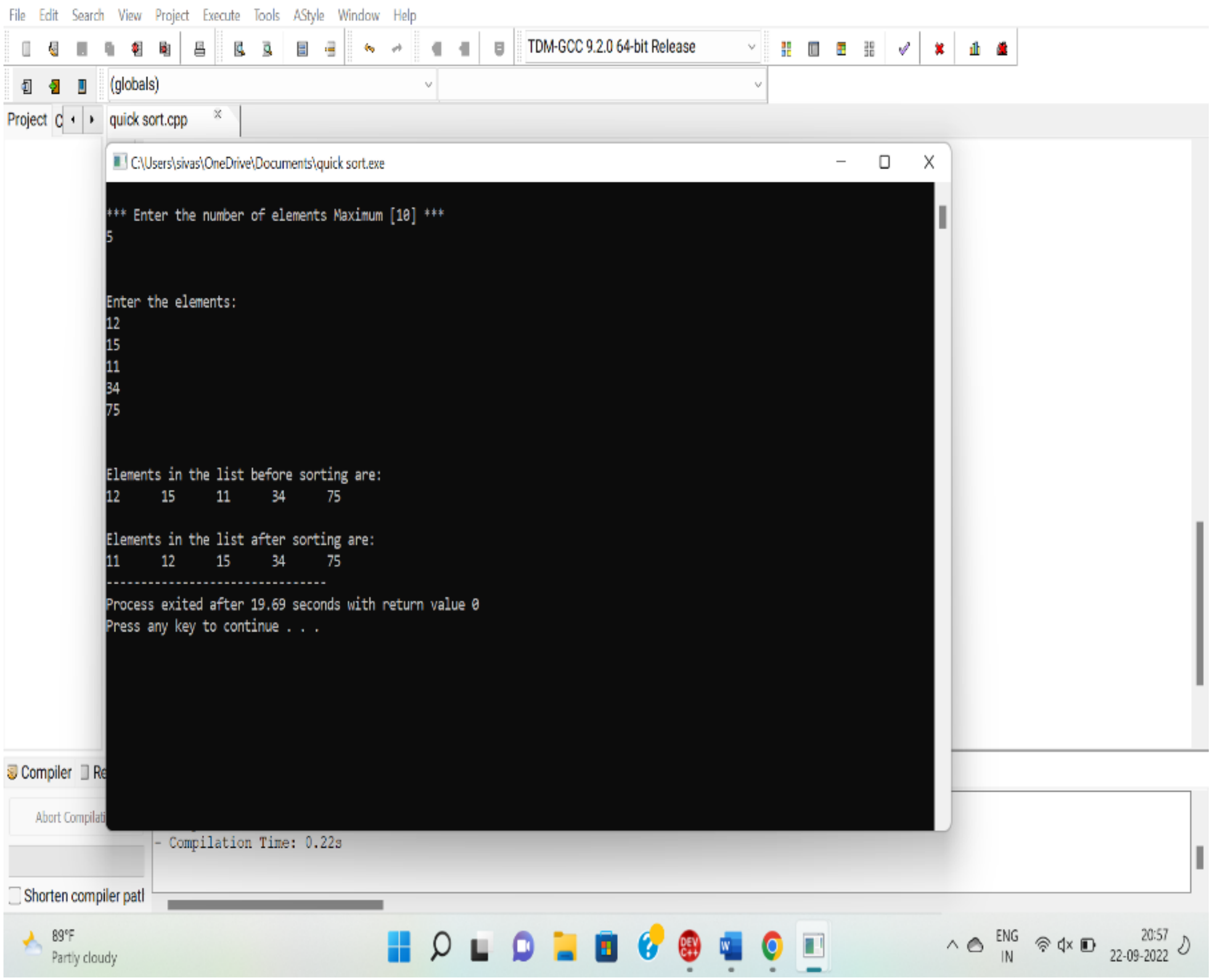
14.program quick sort

```
#include<stdio.h>

#define MAX 10


void swap(int *m,int *n)

{

   int temp;

   temp = *m;

   *m = *n;

   *n = temp;

}

int get_key_position(int x,int y)

{

   return((x+y)/2);
```

```
}

// Function for Quick Sort
void quicksort(int list[],int m,int n)
{
   int key,i,j,k;
   if( m< n)
   {
      k= get_key_position(m,n);
      swap(&list[m],&list[k]);
      key = list[m];
      i = m+1;
      j = n;
      while(i <= j)
      {
       while((i <=n) && (list[i] <= key))
            i++;
       while((j >= m) && (list[j] > key))
                    j--;
           if( i < j)
                    swap(&list[i],&list[j]);
    }
    swap(&list[m],&list[j]);
        quicksort(list,m,j-1);
        quicksort(list,j+1,n);
      }
   }

// Function to read the data
```

```
void read_data(int list[],int n)
{
   int j;
   printf("\n\nEnter the elements:\n");
   for(j=0;j<n;j++)
      scanf("%d",&list[j]);
}

//Function to print the data
void print_data(int list[],int n)
{
   int j;
   for(j=0;j<n;j++)
      printf("%d\t",list[j]);
}

int main()
{
   int list[MAX], num;
   //clrscr();
   printf("\n***Enter the number of elements Maximum [10] ***\n");
   scanf("%d",&num);
   read_data(list,num);
   printf("\n\nElements in the list before sorting are:\n");
   print_data(list,num);
   quicksort(list,0,num-1);
   printf("\n\nElements in the list after sorting are:\n");
   print_data(list,num);
   return 0;
```

}



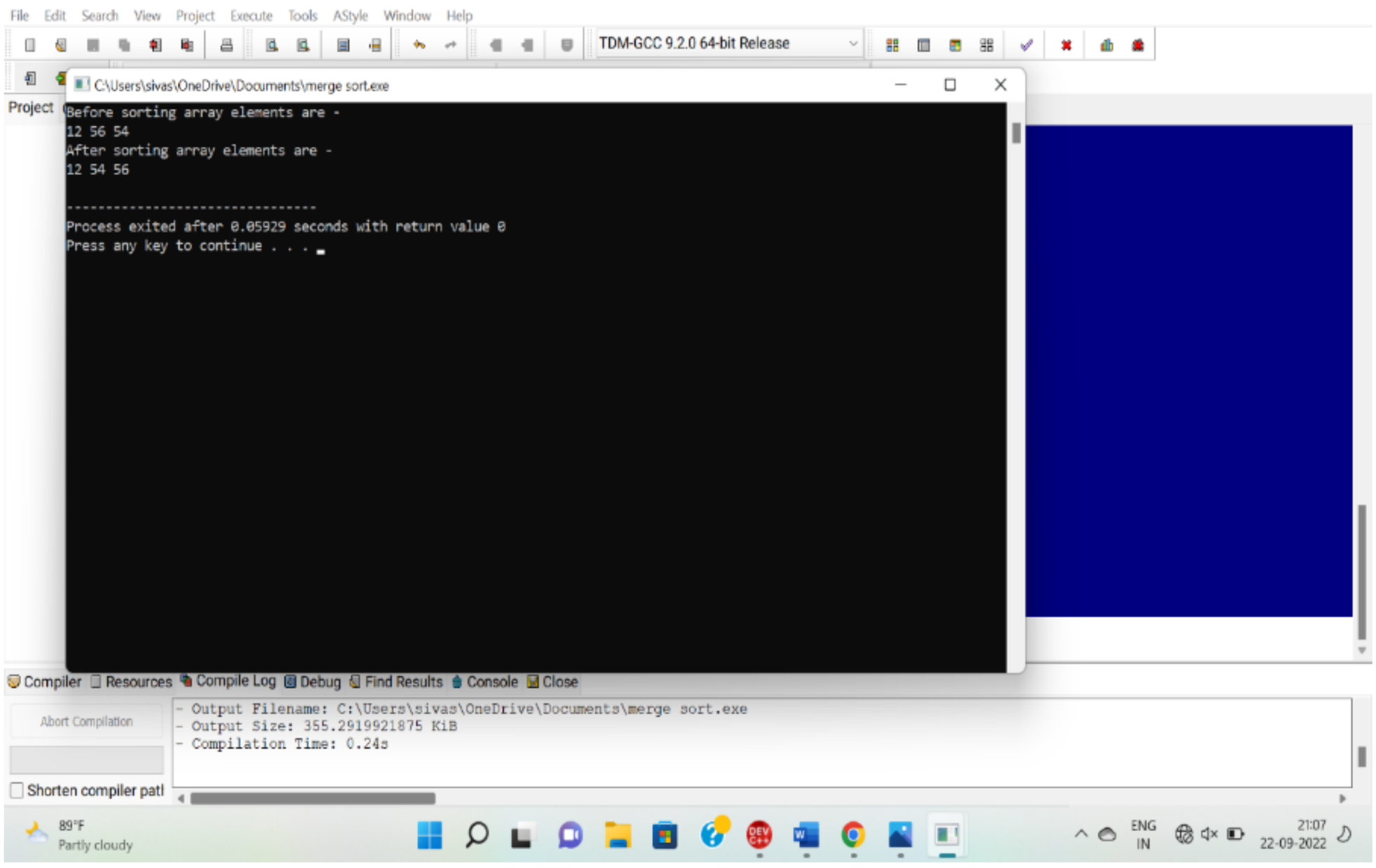
15.Program merge sort

```
#include<stdio.h>

void merge(int a[], int beg, int mid, int end)

{

int i,j,k;

int n1 = mid- beg + 1;

int n2 = end- mid;


int LeftArray[n1], RightArray[n2];

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

```
LeftArray[i] = a[beg + i];
for (int j = 0; j < n2; j++)
RightArray[j] = a[mid + 1 + j];

i = 0;
j = 0;
k = beg;

while (i < n1 && j < n2)
{
    if(LeftArray[i] <= RightArray[j])
    {
        a[k] = LeftArray[i];
        i++;
    }
    else
    {
        a[k] = RightArray[j];
        j++;
    }
    k++;
}
while (i<n1)
{
    a[k] = LeftArray[i];
    i++;
    k++;
}
```

```
while (j<n2)
{
    a[k] = RightArray[j];
    j++;
    k++;
}
}

void mergeSort(int a[], int beg, int end)
{
    if (beg < end)
    {
        int mid = (beg + end) / 2;
        mergeSort(a, beg, mid);
        mergeSort(a, mid + 1, end);
        merge(a, beg, mid, end);
    }
}

/* Function to print the array */
void printArray(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}

int main()
```

```
{
    int a[] = {12,56,54};
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArray(a, n);
    mergeSort(a,0,n-1);
    printf("After sorting array elements are - \n");
    printArray(a, n);
    return 0;
}
```

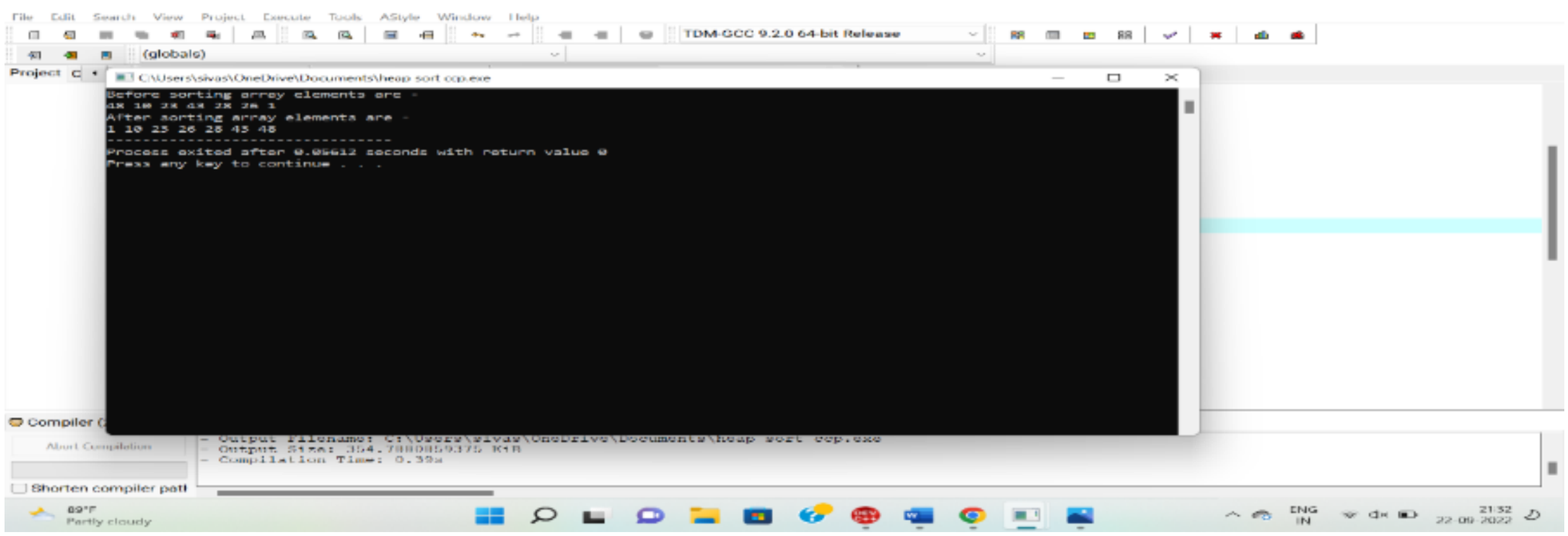


16.Program Heap sort

```
#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);
    }
}
/*Function to implement the heap sort*/
void heapSort(int a[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(a, n, i);
    // One by one extract an element from heap
    for (int i = n - 1; i >= 0; i--) {
        /* Move current root element to end*/
        // swap a[0] with a[i]
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;
        heapify(a, i, 0);
    }
}
/* function to print the array elements*/
```

```
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;
}
```

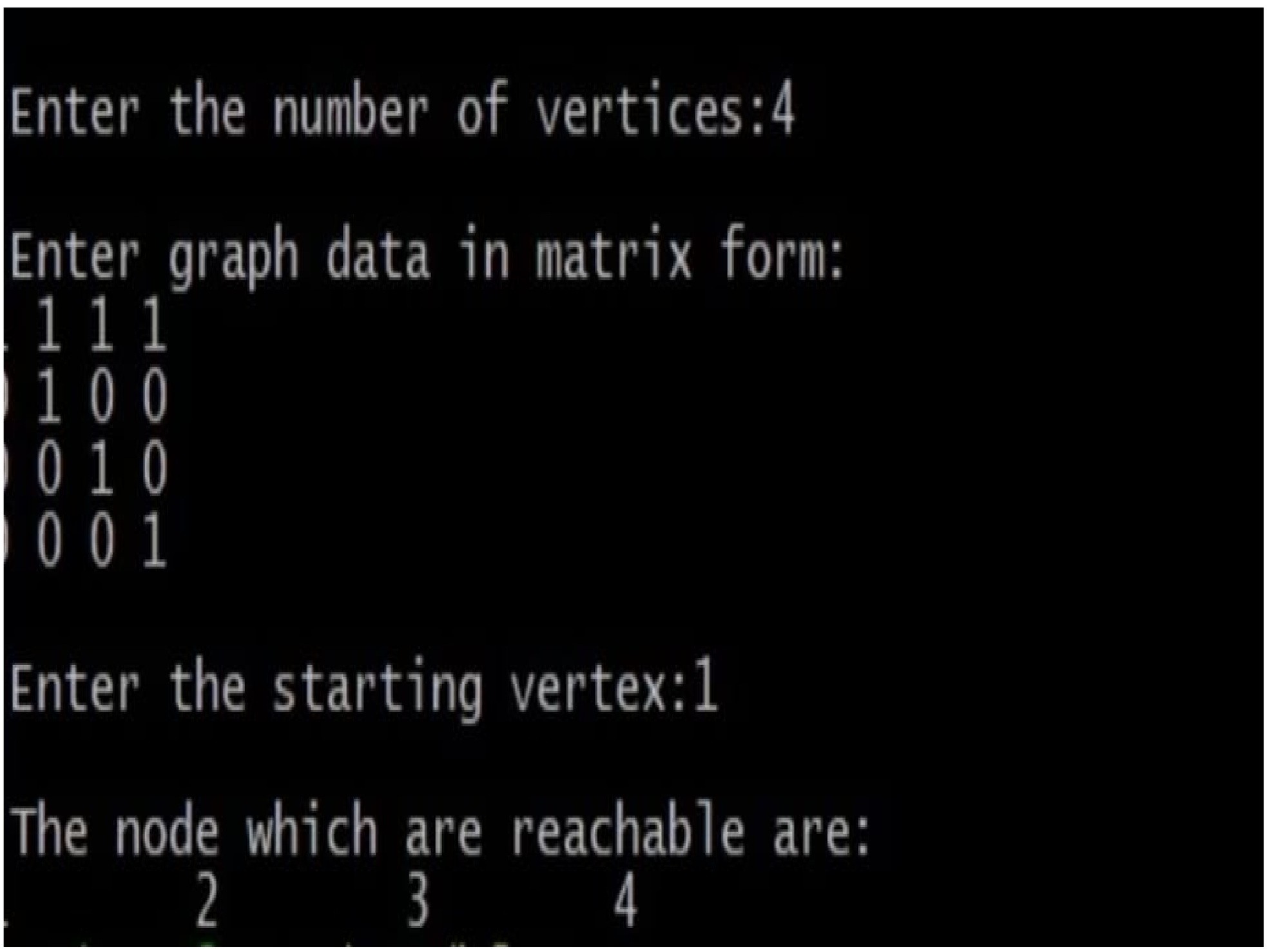


17.PROGRAM breadth first search

#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:");
```

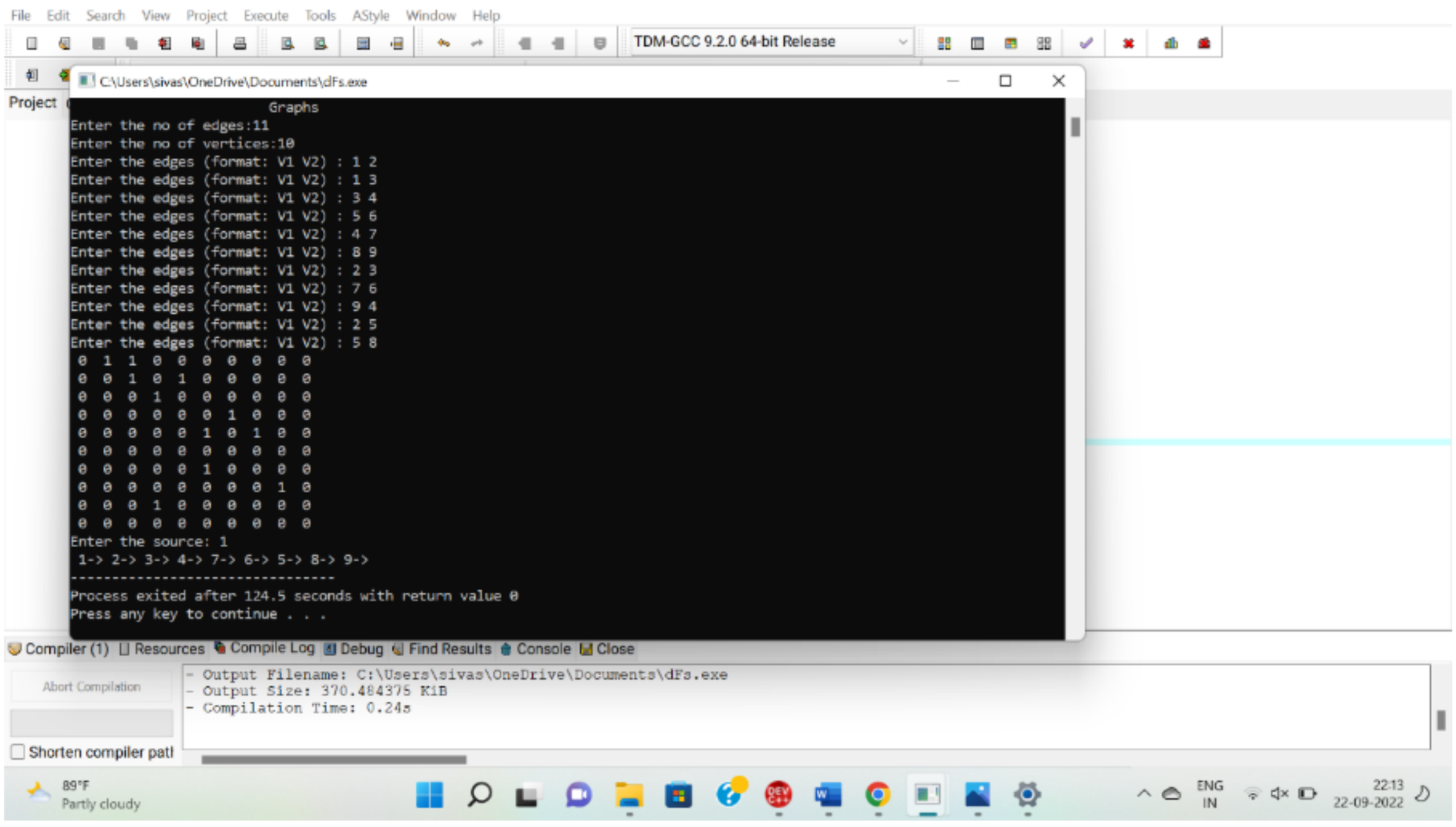
```
scanf("%d",&v);
bfs(v);
printf("\n The node which are reachable are:\n");

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;
        }
    }
```



18.Program Depth first search

```
#include<stdio.h>
#include<stdlib.h>
/*          ADJACENCY MATRIX                        */
int source,V,E,time,visited[20],G[20][20];
void DFS(int i)
{
    int j;
    visited[i]=1;
    printf("%d->",i+1);
    for(j=0;j<V;j++)
    {
        if(G[i][j]==1&&visited[j]==0)
            DFS(j);
    }
}
int main()
{
    int i,j,v1,v2;
    printf("\t\t\tGraphs\n");
    printf("Enter the no of edges:");
    scanf("%d",&E);
    printf("Enter the no of vertices:");
scanf("%d",&V);
for(i=0;i<V;i++)
{
    for(j=0;j<V;j++)
        G[i][j]=0;
```

```
}
/*    creating edges:P    */
for(i=0;i<E;i++)
{
    printf("Enter the edges (format: V1 V2) : ");
    scanf("%d%d",&v1,&v2);
    G[v1-1][v2-1]=1;

}

for(i=0;i<V;i++)
{
    for(j=0;j<V;j++)
        printf(" %d ",G[i][j]);
    printf("\n");
    }
    printf("Enter the source: ");
    scanf("%d",&source);
        DFS(source-1);
    return 0;
}
```

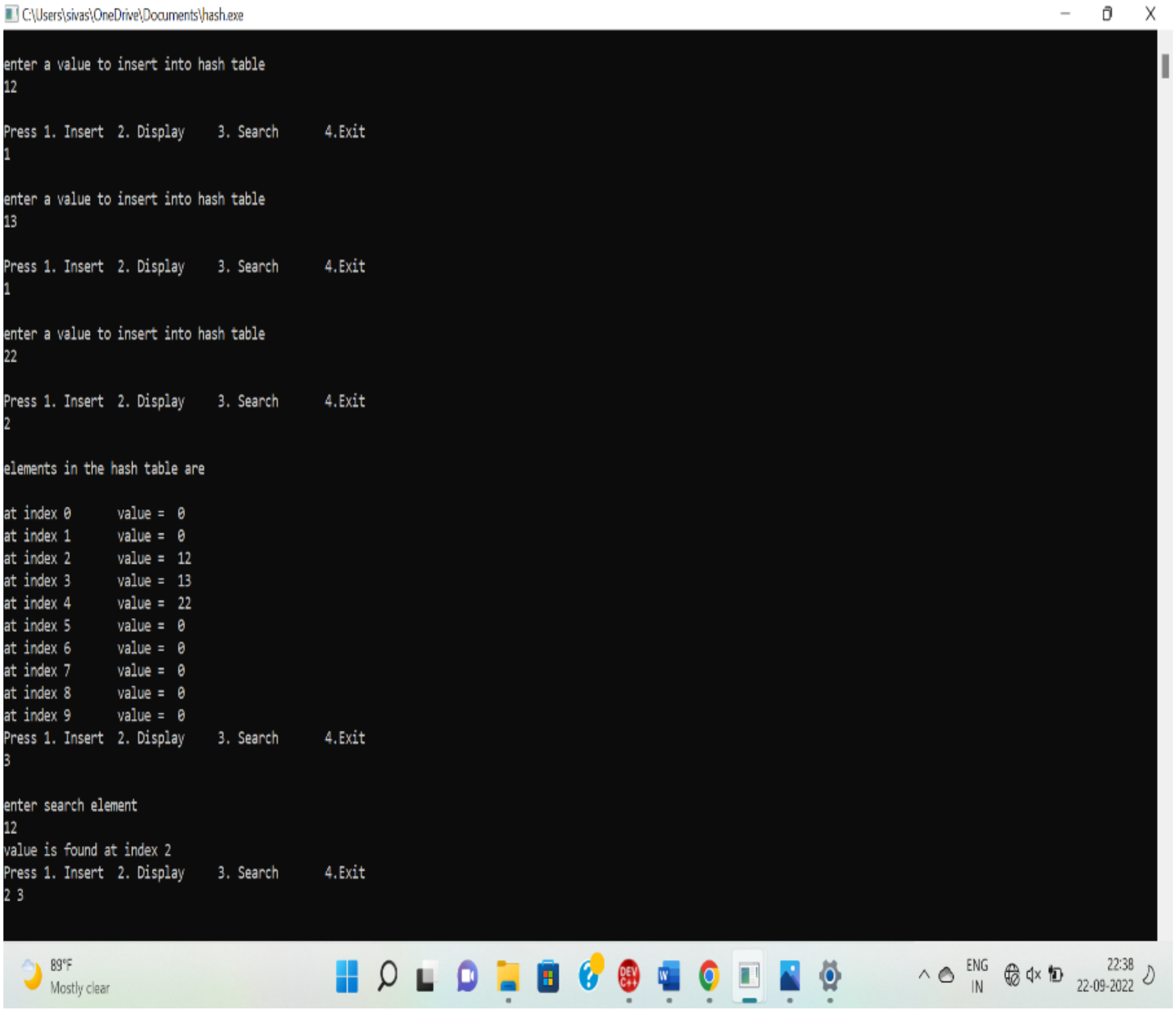


19.Program linear probaing

```
#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;
```

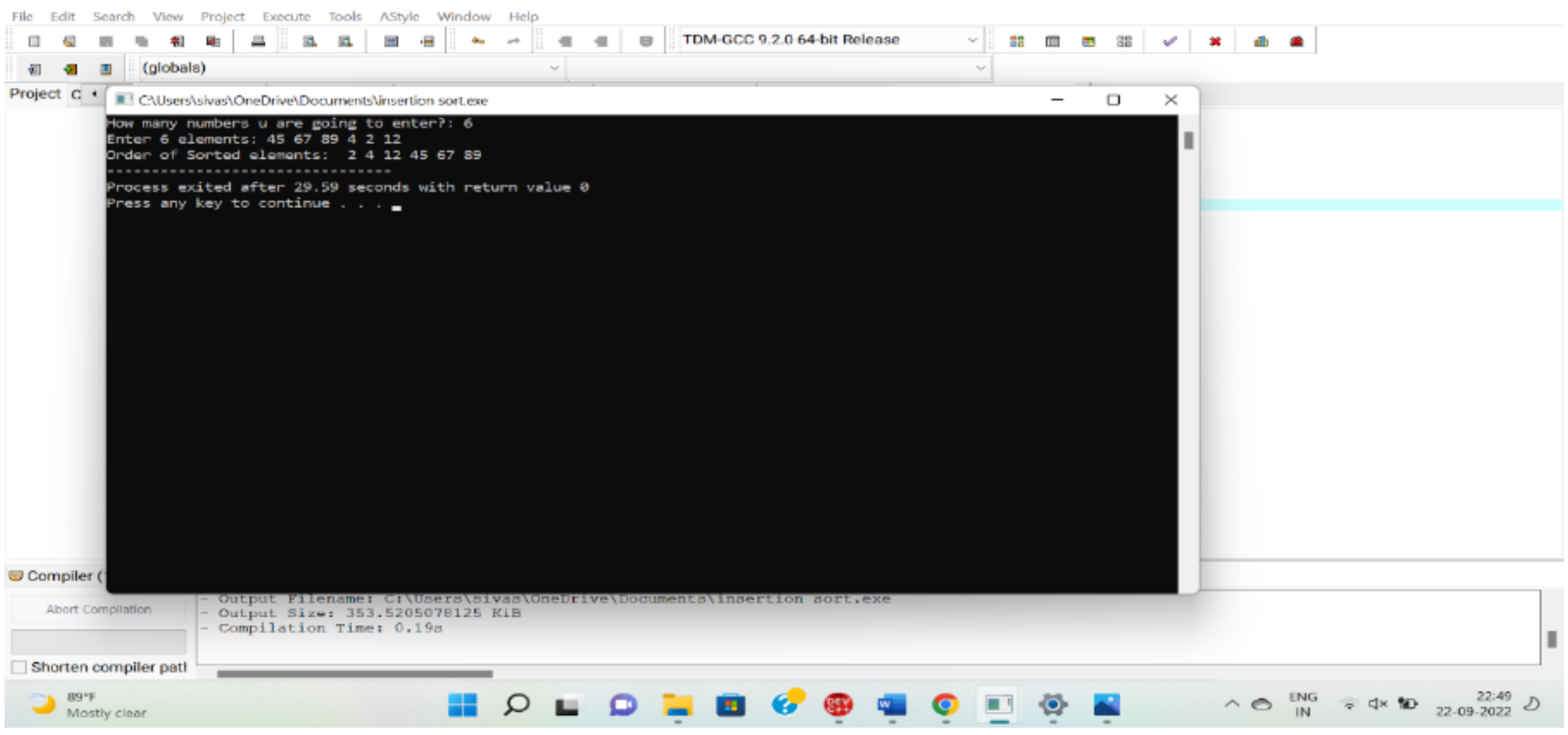
```
 printf("\nelements in the hash table are \n");
   for(i=0;i< TABLE_SIZE; i++)
 printf("\nat index %d \t value =  %d",i,h[i]);
}
main()
{
int opt,i;
while(1)
{
    printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
    scanf("%d",&opt);
    switch(opt)
    {
        case 1:
            insert();
            break;
    case 2:
        display();
        break;
    case 3:
        search();
        break;
    case 4:exit(0);
        }
    }
}
```



20.program

```
#include<stdio.h>

int main(){

   int i, j, count, temp, number[25];

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

   scanf("%d",&count);

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

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

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

   for(i=1;i<count;i++){
```

```
temp=number[i];
j=i-1;
while((temp<number[j])&&(j>=0)){
   number[j+1]=number[j];
   j=j-1;
}   number[j+1]=temp;
  }  printf("Order of Sorted elements: ");
  for(i=0;i<count;i++)
     printf(" %d",number[i]);
return 0;}
```



21.program