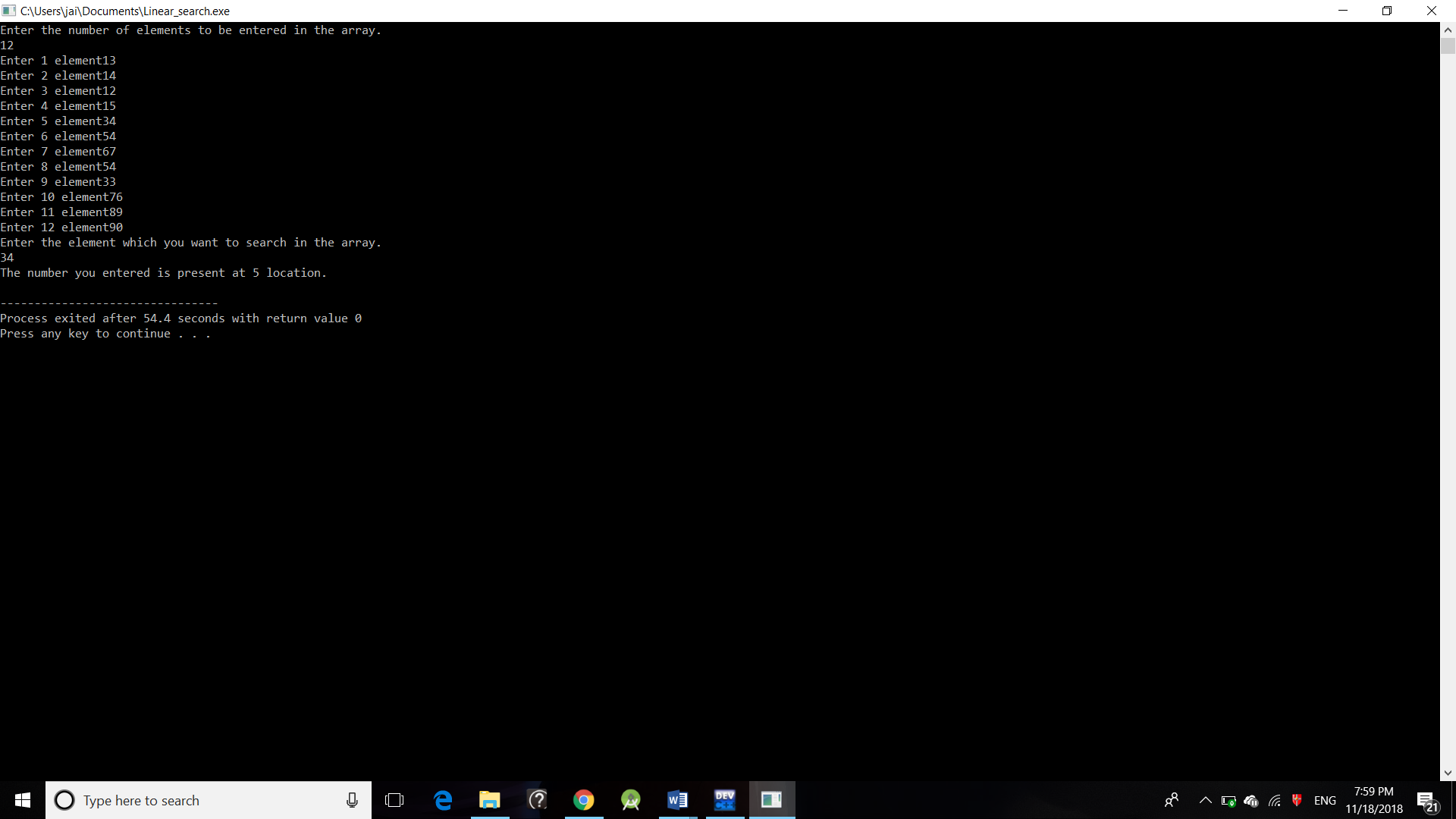
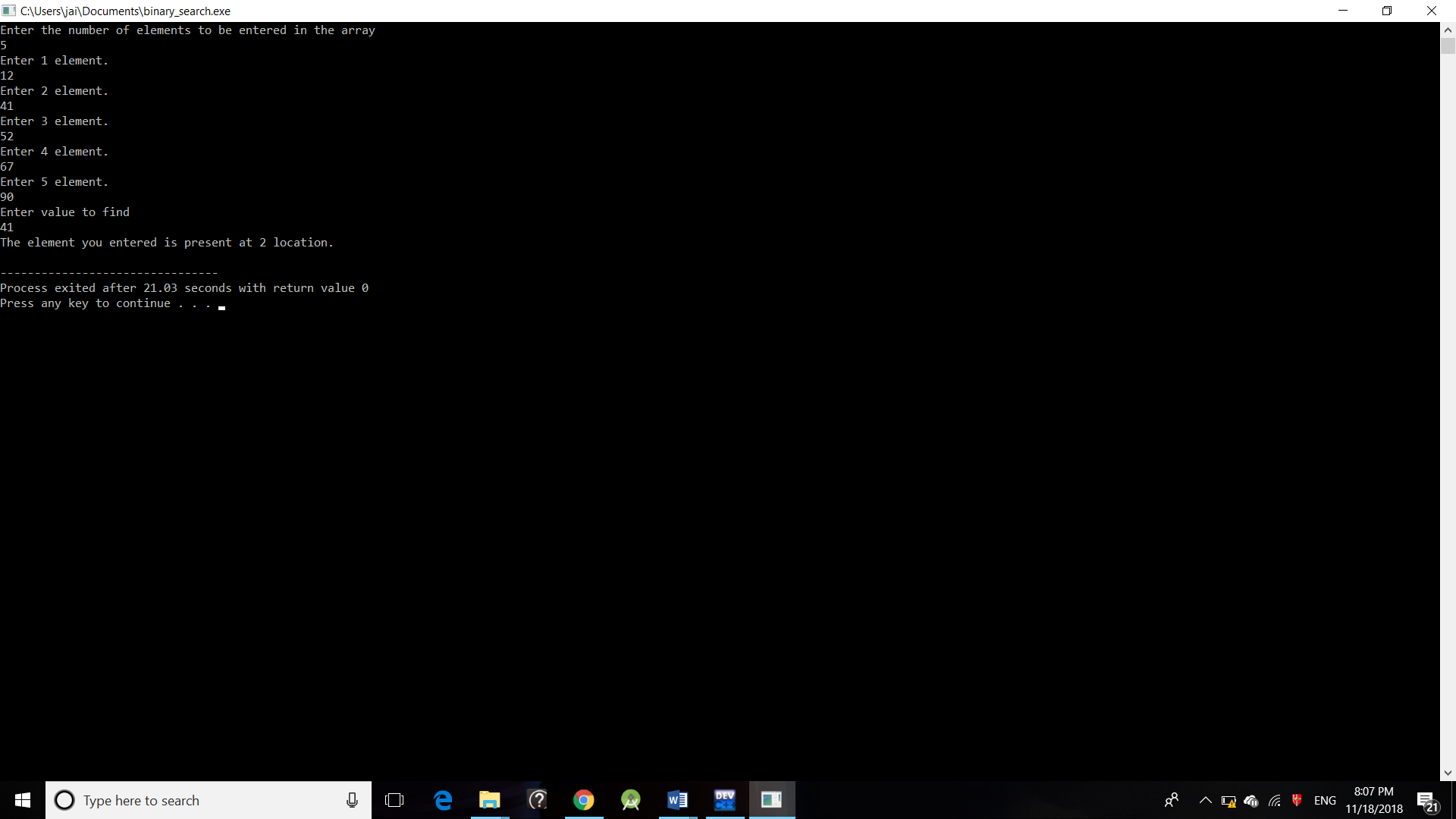
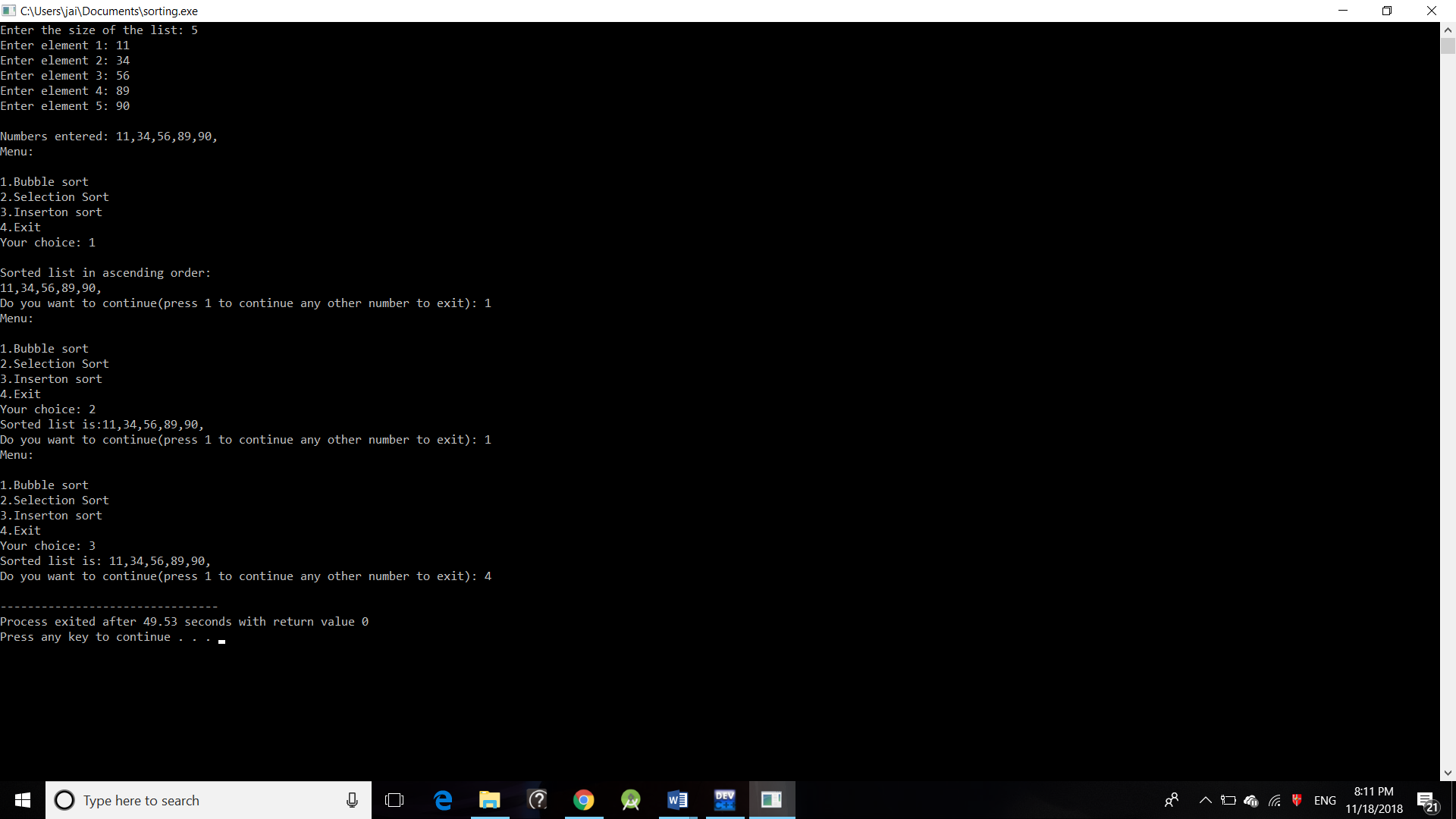
***OUTPUT:***



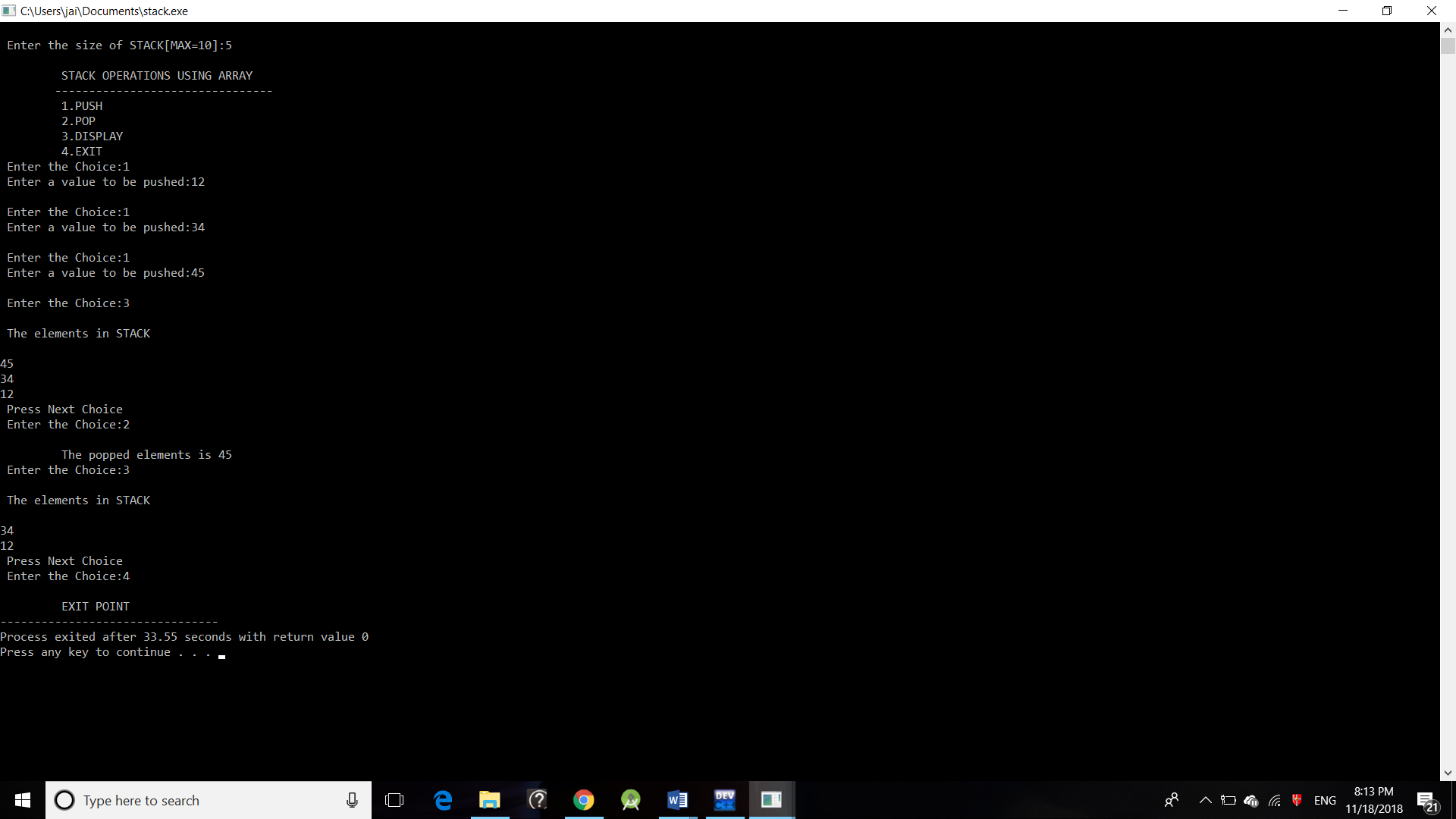
***OUTPUT:***



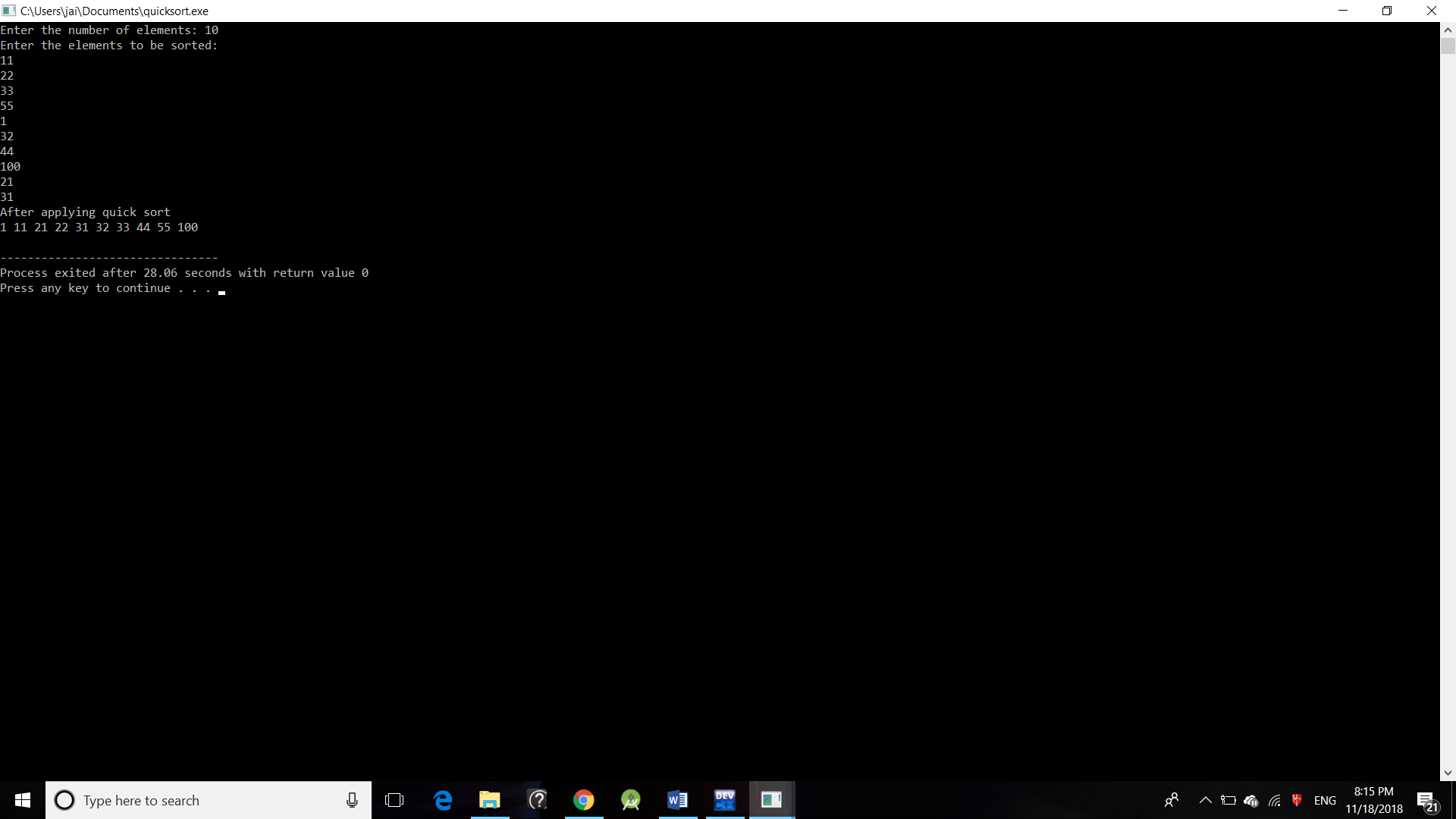
***OUTPUT:***



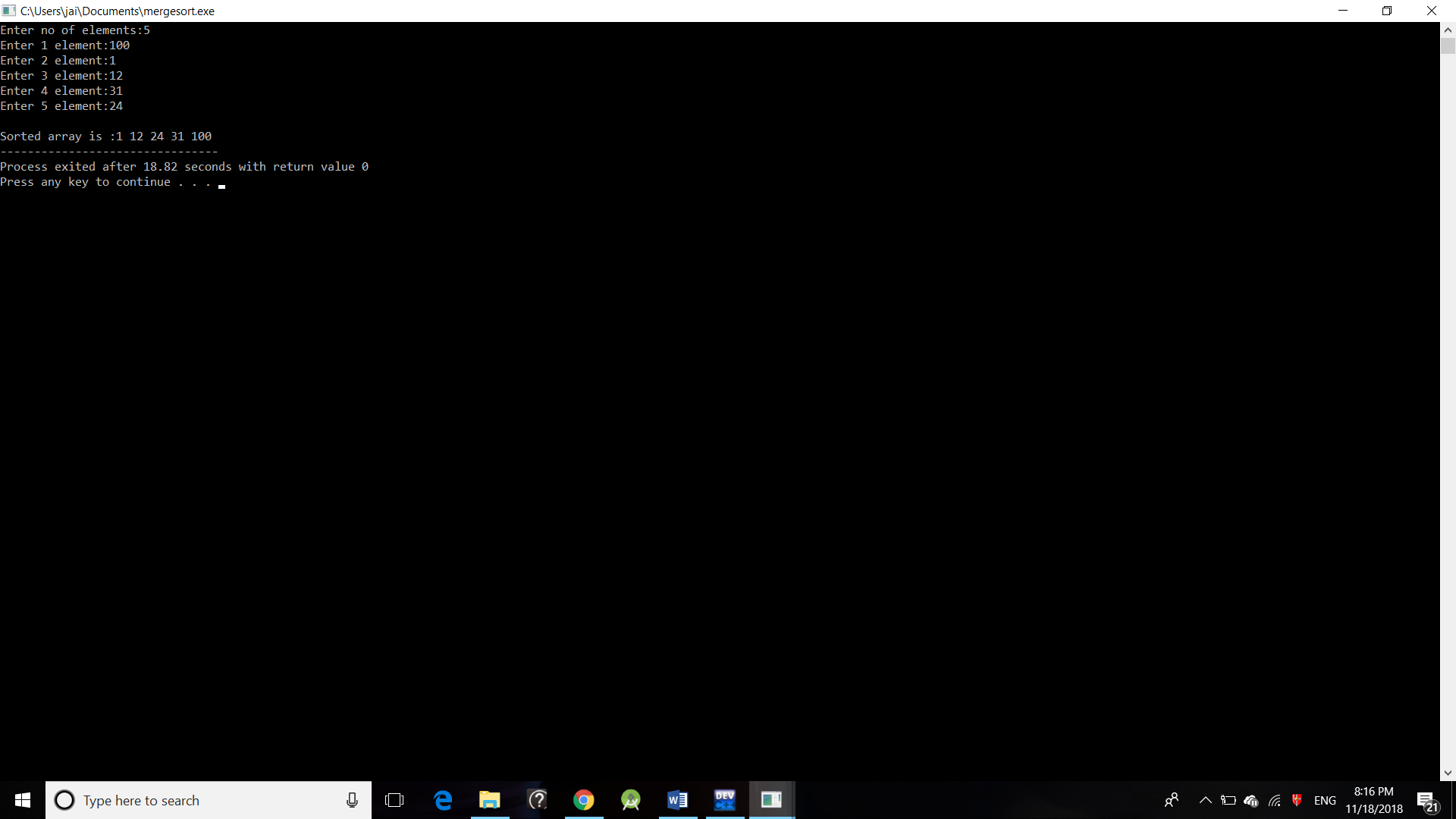
***OUTPUT:***



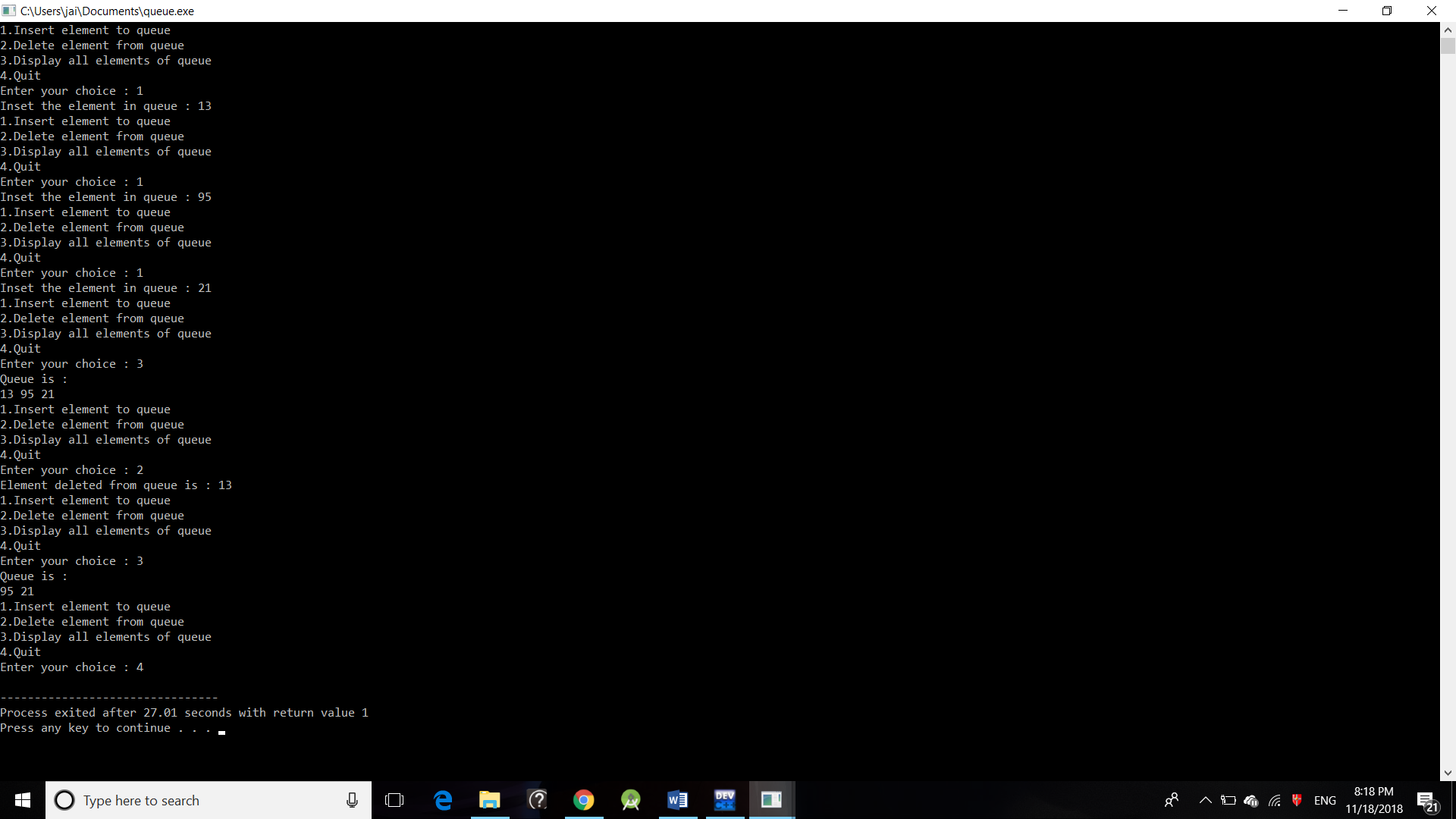
***OUTPUT:***



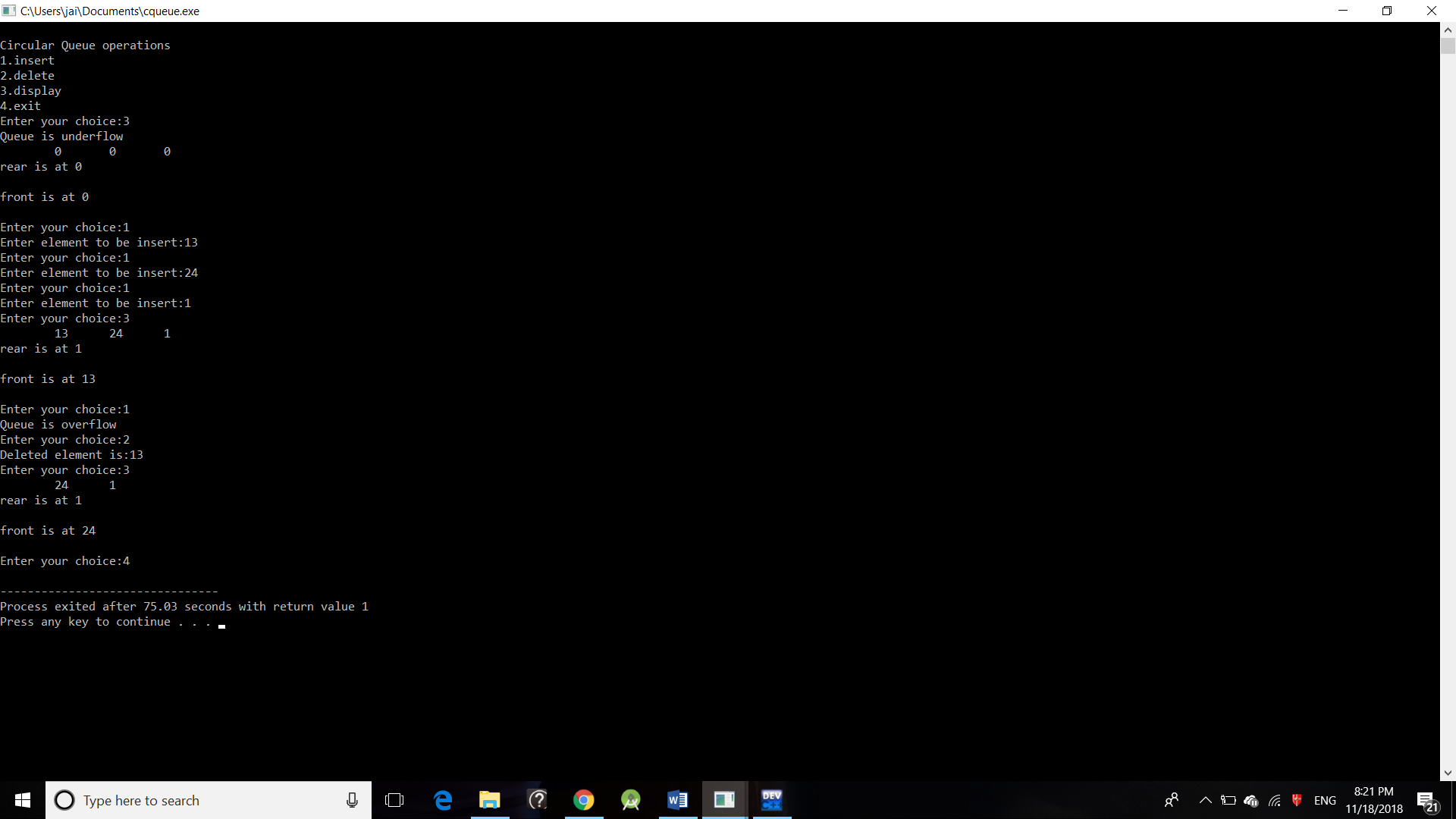
***OUTPUT:***



***OUTPUT:***



***OUTPUT:***



***Practical – 9***

***Aim –*** Write a program to implement Linked list.

***Code -***

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

void main()

{

struct node

{

int num;

struct node \*ptr;

};

typedef struct node NODE;

NODE \*head, \*first, \*temp = 0;

int count = 0;

int choice = 1;

first = 0;

while (choice)

{

head = (NODE \*)malloc(sizeof(NODE));

printf("Enter the data item\n");

scanf("%d", &head-> num);

if (first != 0)

{

temp->ptr = head;

temp = head;

}

else

{

first = temp = head;

}

fflush(stdin);

printf("Do you want to continue(Type 0 or 1)?\n");

scanf("%d", &choice);

}

temp->ptr = 0;

/\* reset temp to the beginning \*/

temp = first;

printf("\n status of the linked list is\n");

while (temp != 0)

{

printf("%d=>", temp->num);

count++;

temp = temp -> ptr;

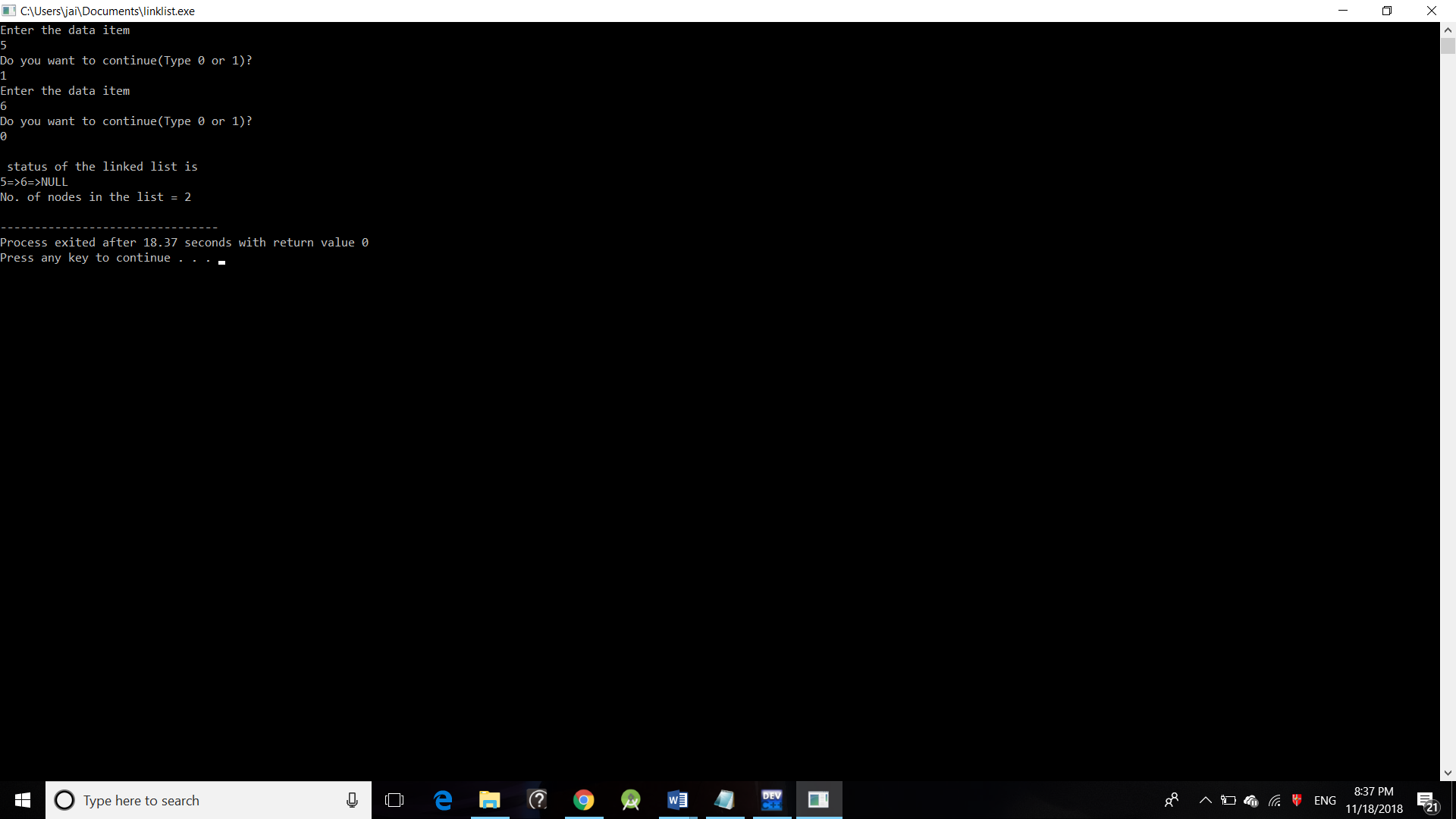
}

printf("NULL\n");

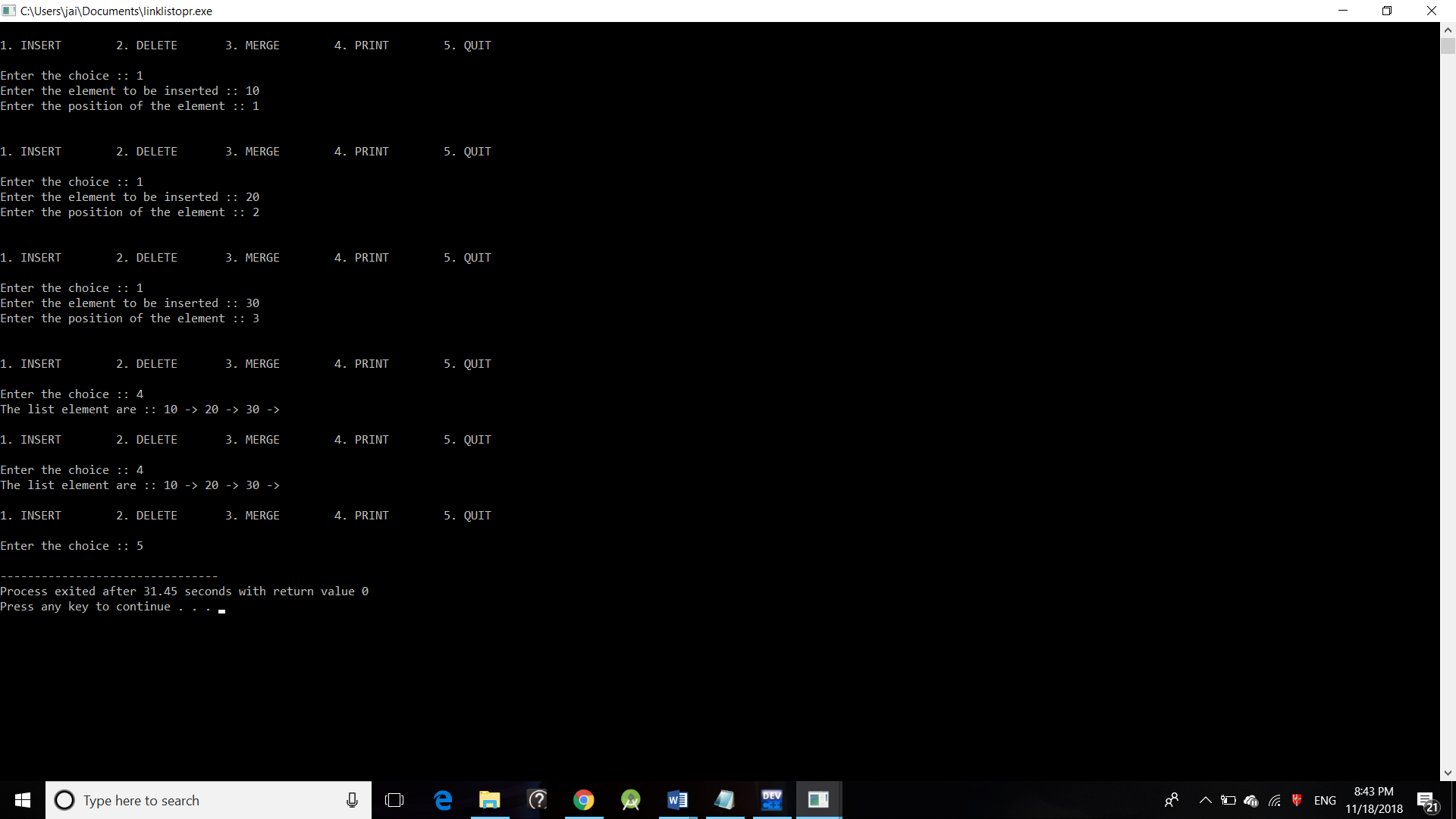
printf("No. of nodes in the list = %d\n", count);

}

***OUTPUT:***



***OUTPUT:***



***Practical – 10***

***Aim –*** Write a program to perform operations on a single Linked list.

***Code -***

#include<stdio.h>

#include<stdlib.h>

struct Node;

typedef struct Node \* PtrToNode;

typedef PtrToNode List;

typedef PtrToNode Position;

struct Node

{

int e;

Position next;

};

void Insert(int x, List l, Position p)

{

Position TmpCell;

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

if(TmpCell == NULL)

printf("Memory out of space\n");

else

{

TmpCell->e = x;

TmpCell->next = p->next;

p->next = TmpCell;

}

}

int isLast(Position p)

{

return (p->next == NULL);

}

Position FindPrevious(int x, List l)

{

Position p = l;

while(p->next != NULL && p->next->e != x)

p = p->next;

return p;

}

void Delete(int x, List l)

{

Position p, TmpCell;

p = FindPrevious(x, l);

if(!isLast(p))

{

TmpCell = p->next;

p->next = TmpCell->next;

free(TmpCell);

}

else

printf("Element does not exist!!!\n");

}

void Display(List l)

{

printf("The list element are :: ");

Position p = l->next;

while(p != NULL)

{

printf("%d -> ", p->e);

p = p->next;

}

}

void Merge(List l, List l1)

{

int i, n, x, j;

Position p;

printf("Enter the number of elements to be merged :: ");

scanf("%d",&n);

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

{

p = l1;

scanf("%d", &x);

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

p = p->next;

Insert(x, l1, p);

}

printf("The new List :: ");

Display(l1);

printf("The merged List ::");

p = l;

while(p->next != NULL)

{

p = p->next;

}

p->next = l1->next;

Display(l);

}

int main()

{

int x, pos, ch, i;

List l, l1;

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

l->next = NULL;

List p = l;

printf("LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");

do

{

printf("\n\n1. INSERT\t 2. DELETE\t 3. MERGE\t 4. PRINT\t 5. QUIT\n\nEnter the choice :: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

p = l;

printf("Enter the element to be inserted :: ");

scanf("%d",&x);

printf("Enter the position of the element :: ");

scanf("%d",&pos);

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

{

p = p->next;

}

Insert(x,l,p);

break;

case 2:

p = l;

printf("Enter the element to be deleted :: ");

scanf("%d",&x);

Delete(x,p);

break;

case 3:

l1 = (struct Node \*) malloc(sizeof(struct Node));

l1->next = NULL;

Merge(l, l1);

break;

case 4:

Display(l);

break;

}

}

while(ch<5);

return 0;

}

***Practical – 11***

***Aim –*** Write a program to implement Linked list as a stack

***Code –***

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*top,\*top1,\*temp;

int topelement();

void push(int data);

void pop();

void empty();

void display();

void destroy();

void stack\_count();

void create();

int count = 0;

int main()

{

int no, ch, e;

printf("\n 1 - Push");

printf("\n 2 - Pop");

printf("\n 3 - Top");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Dipslay");

printf("\n 7 - Stack Count");

printf("\n 8 - Destroy stack");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

push(no);

break;

case 2:

pop();

break;

case 3:

if (top == NULL)

printf("No elements in stack");

else

{

e = topelement();

printf("\n Top element : %d", e);

}

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

stack\_count();

break;

case 8:

destroy();

break;

default :

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

break;

}

}

}

/\* Create empty stack \*/

void create()

{

top = NULL;

}

/\* Count stack elements \*/

void stack\_count()

{

printf("\n No. of elements in stack : %d", count);

}

/\* Push data into stack \*/

void push(int data)

{

if (top == NULL)

{

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

top->ptr = NULL;

top->info = data;

}

else

{

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

temp->ptr = top;

temp->info = data;

top = temp;

}

count++;

}

/\* Display stack elements \*/

void display()

{

top1 = top;

if (top1 == NULL)

{

printf("Stack is empty");

return;

}

while (top1 != NULL)

{

printf("%d ", top1->info);

top1 = top1->ptr;

}

}

/\* Pop Operation on stack \*/

void pop()

{

top1 = top;

if (top1 == NULL)

{

printf("\n Error : Trying to pop from empty stack");

return;

}

else

top1 = top1->ptr;

printf("\n Popped value : %d", top->info);

free(top);

top = top1;

count--;

}

/\* Return top element \*/

int topelement()

{

return(top->info);

}

/\* Check if stack is empty or not \*/

void empty()

{

if (top == NULL)

printf("\n Stack is empty");

else

printf("\n Stack is not empty with %d elements", count);

}

/\* Destroy entire stack \*/

void destroy()

{

top1 = top;

while (top1 != NULL)

{

top1 = top->ptr;

free(top);

top = top1;

top1 = top1->ptr;

}

free(top1);

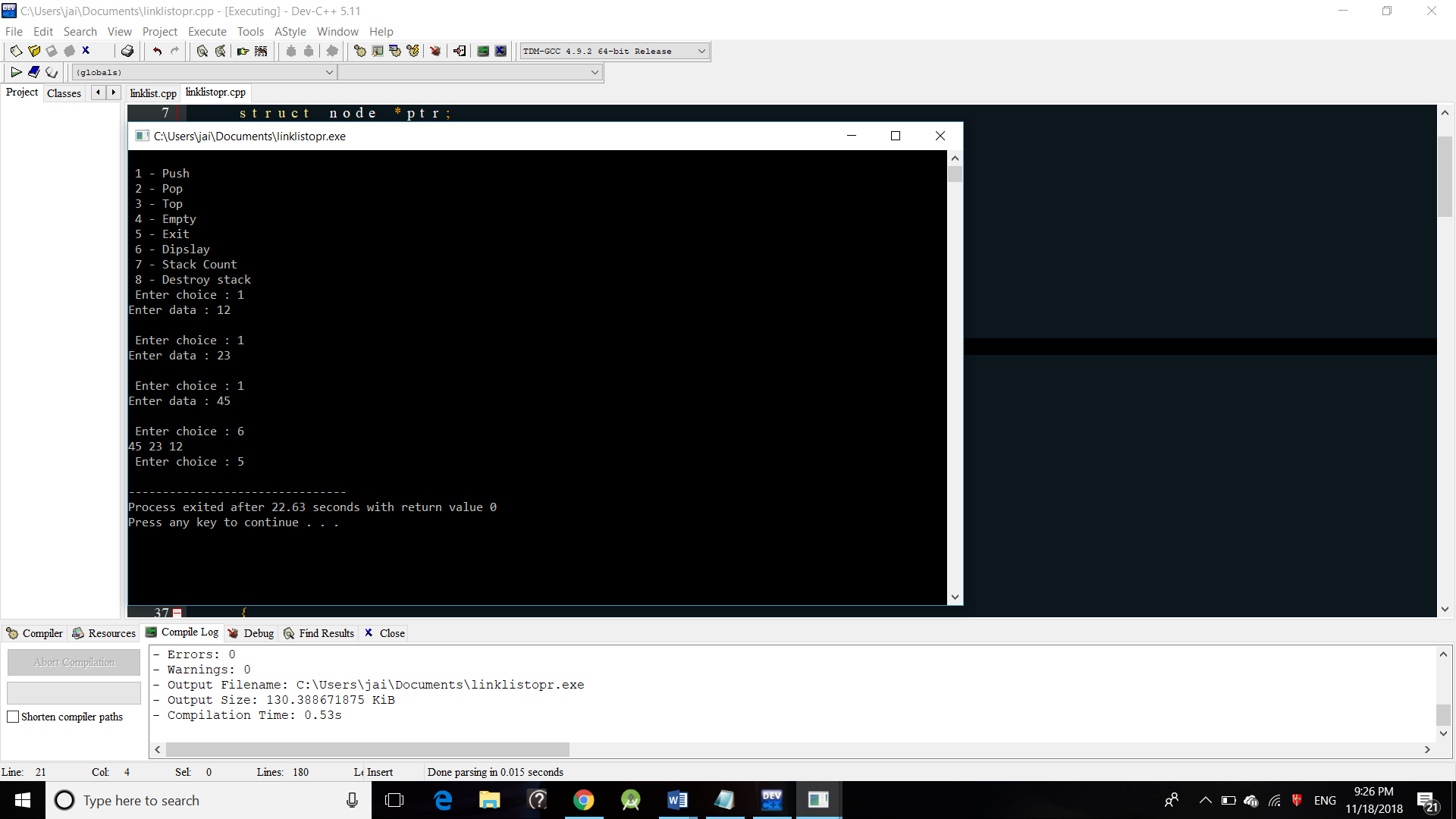
top = NULL;

printf("\n All stack elements destroyed");

count = 0;

}

***OUTPUT:***



***Practical – 12***

***Aim –*** Write a program to implement Linked list as a queue.

***Code –***

#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;

void 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;

}

}

}

/\* Create an empty queue \*/

void create()

{

front = rear = NULL;

}

/\* Returns queue size \*/

void queuesize()

{

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

}

/\* Enqueing the queue \*/

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++;

}

/\* Displaying the queue elements \*/

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);

}

/\* Dequeing the queue \*/

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--;

}

/\* Returns the front element of queue \*/

int frontelement()

{

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

return(front->info);

else

return 0;

}

/\* Display if queue is empty or not \*/

void empty()

{

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

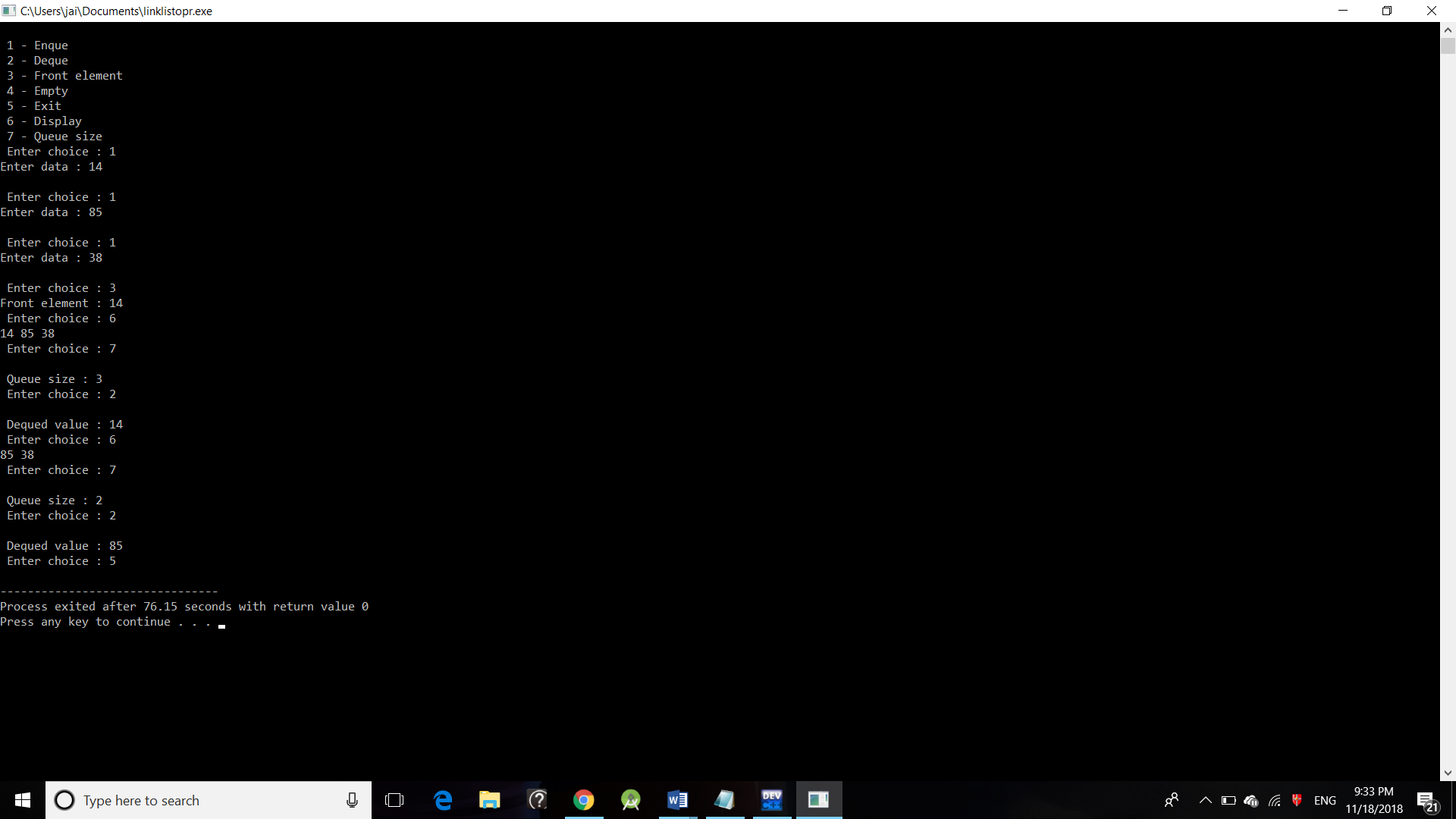
printf("\n Queue empty");

else

printf("Queue not empty");

}

***OUTPUT:***



***Practical – 13***

***Aim –***  To convert Infix expression into Postfix expression

***Code –***

#include<stdio.h>

#include<ctype.h>

char stack[20];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

}

main()

{

char exp[20];

char \*e, x;

printf("Enter the expression :: ");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c",pop());

push(\*e);

}

e++;

}

while(top != -1)

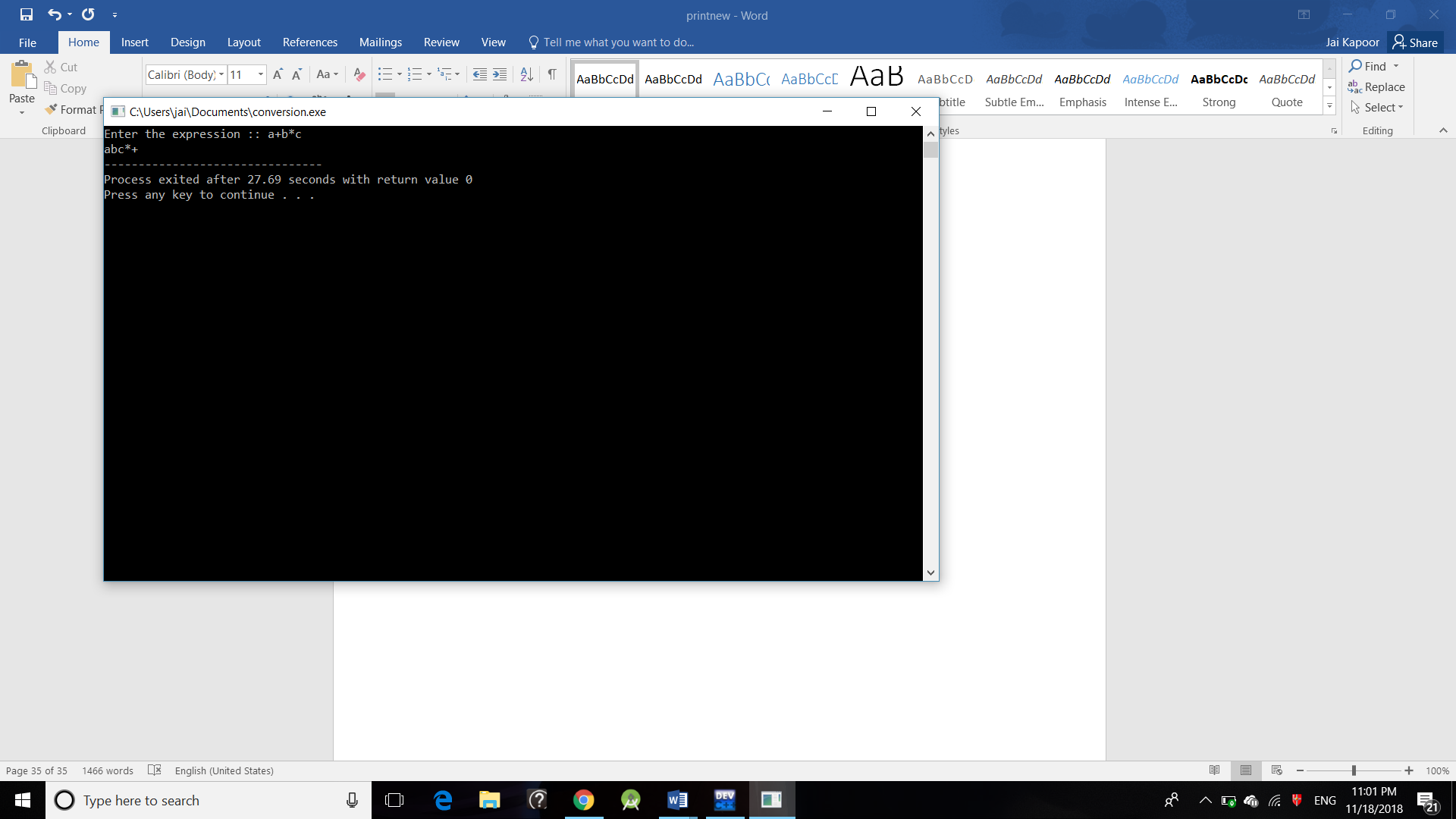
{

printf("%c",pop());

}

}

***OUTPUT:***



***Practical – 14***

***Aim –***  To perform operations on a Linear Array.

***Code –***

#include<stdio.h>

#include<stdlib.h>

int a[20],b[20],c[40];

int m,n,p,val,i,j,key,pos,temp;

void create();

void display();

void insert();

void del();

int main()

{

int choice;

do{

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

printf("1.Create\n");

printf("2.Display\n");

printf("3.Insert\n");

printf("4.Delete\n");

printf("5.Exit\n");

printf("-----------------------");

printf("\nEnter your choice:\t");

scanf("%d",&choice);

switch(choice)

{

case 1: create();

break;

case 2:

display();

break;

case 3:

insert();

break;

case 4:

del();

break;

case 5:

exit(0);

break;

default:

printf("\nInvalid choice:\n");

break;

}

}while(choice!=8);

return 0;

}

void create()

{

printf("\nEnter the size of the array elements:\t");

scanf("%d",&n);

printf("\nEnter the elements for the array:\n");

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

{

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

}

}

void display()

{

int i;

printf("\nThe array elements are:\n");

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

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

}

}

void insert()

{

printf("\nEnter the position for the new element:\t");

scanf("%d",&pos);

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

scanf("%d",&val);

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

{

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

}

a[pos]=val;

n=n+1;

}

void del()

{

printf("\nEnter the position of the element to be deleted:\t");

scanf("%d",&pos);

val=a[pos];

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

{

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

}

n=n-1;

printf("\nThe deleted element is =%d",val);

}

***OUTPUT:***

