Program 1:

#include<stdio.h>

#include<unistd.h>

#define MAX 5

struct stack

{

int data[MAX];

int top;

};

void initialize(struct stack \*s)

{

s->top=-1;

}

int isEmpty(struct stack \*s)

{

if(s->top==-1) // return s->top==-1?1:0;

return 1;

else

return 0;

}

int isFull(struct stack \*s)

{

if(s->top==MAX-1)

return 1;

else

return 0;

}

int push(struct stack \*s,int d)

{

if(isFull(s))

return 0;

else

{

s->top++;

s->data[s->top]=d;

return 1;

}

}

void pop(struct stack \*s)

{

if(s->top==-1)

printf("\n\tStack Underflows->");

else

{

int d;

d=s->data[s->top];

s->top--; //d=s->data[s->top--];

printf("\nPopped Data is %d",d);

//return s->data[s->top--];

}

}

void peek(struct stack \*s)

{

if(s->top==-1)

printf("\n\tStack Underflows->");

else

{

int d;

d=s->data[s->top];

//s->top--; //d=s->data[s->top--];

printf("\nData at peek is %d",d);

//return s->data[s->top--];

}

}

void display(struct stack \*s)

{

int i;

if(s->top==-1)

printf("\n\tStack is Empty->");

else

{

printf("\nStack Contents ->->\n");

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

{

printf("%d\n",s->data[i]);

}

}

}

int main()

{

int ch,d;

struct stack s;

//s->top=-1;

initialize(&s);

while(1)

{

printf("\n\t\t\tMENU\n1. Push.\n2. Pop.\n3. Peek.");

printf("\n4. Display.\n5. Exit.");

printf("\n\tEnter your choice :: ");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nEnter Data to be Pushed : ");

scanf("%d",&d);

if(push(&s,d))

printf("\nPushed successfully");

else

printf("\nCannot Push..");

break;

case 2:

pop(&s);

break;

case 3:

peek(&s);

break;

case 4:

display(&s);

break;

case 5:

exit(0);

default:

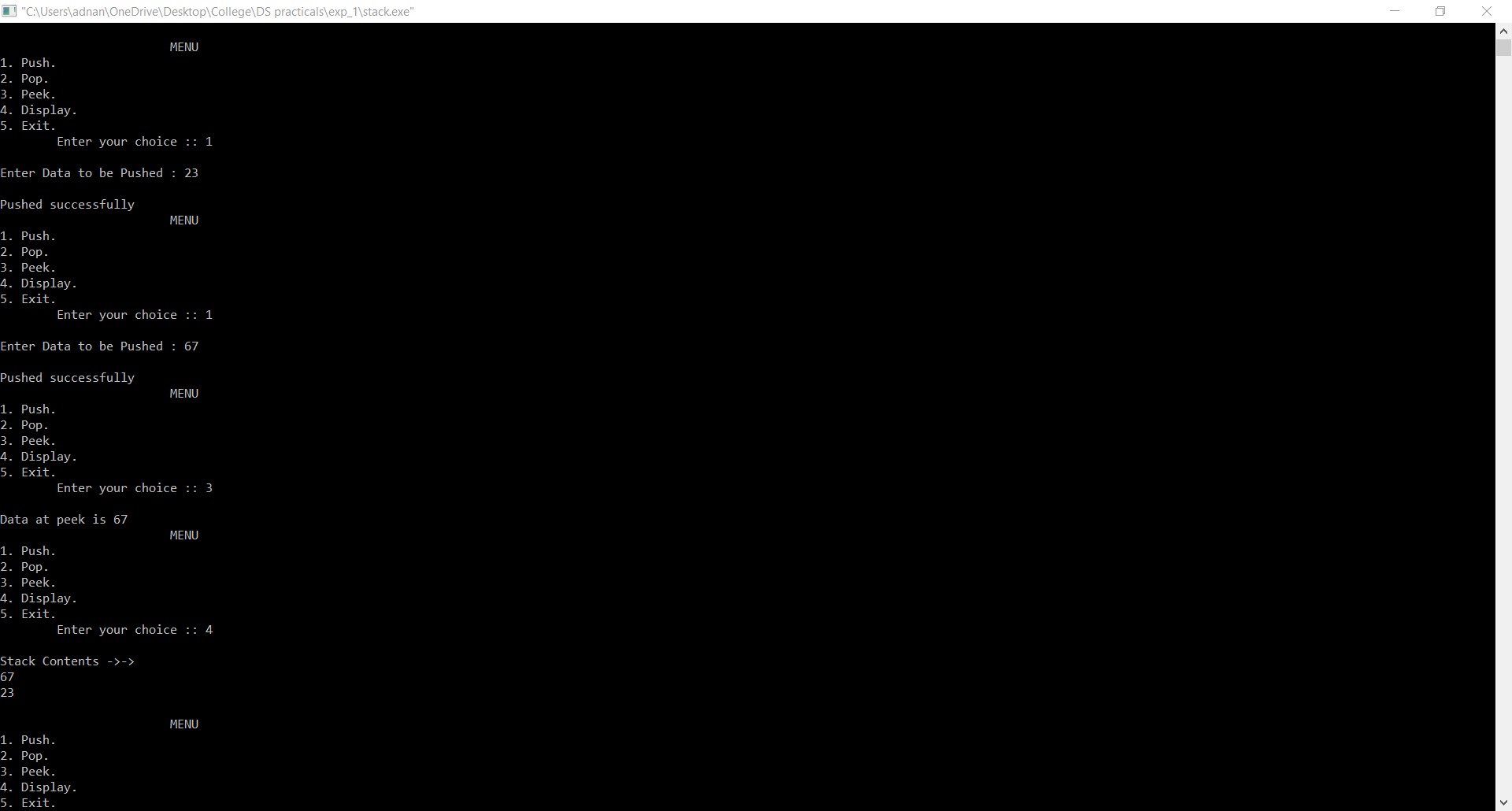
printf("\n\tPlease enter correct choice->->->->");

}

}

}

Output:





Program 2:

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define SIZE 100

char stack[SIZE];

int top = -1;

void push(char item)

{

if(top >= SIZE-1)

{

printf("\nStack Overflow.");

}

else

{

top = top+1;

stack[top] = item;

}

}

char pop()

{

char item ;

if(top <0)

{

printf("stack under flow: invalid infix expression");

getchar();

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

int is\_operator(char symbol)

{

if(symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol =='-')

{

return 1;

}

else

{

return 0;

}

}

int precedence(char symbol)

{

if(symbol == '^')

{

return(3);

}

else if(symbol == '\*' || symbol == '/')

{

return(2);

}

else if(symbol == '+' || symbol == '-')

{

return(1);

}

else

{

return(0);

}

}

void InfixToPostfix(char infix\_exp[], char postfix\_exp[])

{

int i, j;

char item;

char x;

push('(');

strcat(infix\_exp,")");

i=0;

j=0;

item=infix\_exp[i];

while(item != '\0')

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

postfix\_exp[j] = item;

j++;

}

else if(is\_operator(item) == 1)

{

x=pop();

while(is\_operator(x) == 1 && precedence(x)>= precedence(item))

{

postfix\_exp[j] = x;

j++;

x = pop();

}

push(x);

push(item);

}

else if(item == ')')

{

x = pop();

while(x != '(')

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

else

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

i++;

item = infix\_exp[i];

}

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

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

}

int main()

{

char infix[SIZE], postfix[SIZE];

printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");

printf("\nEnter Infix expression : ");

gets(infix);

InfixToPostfix(infix,postfix);

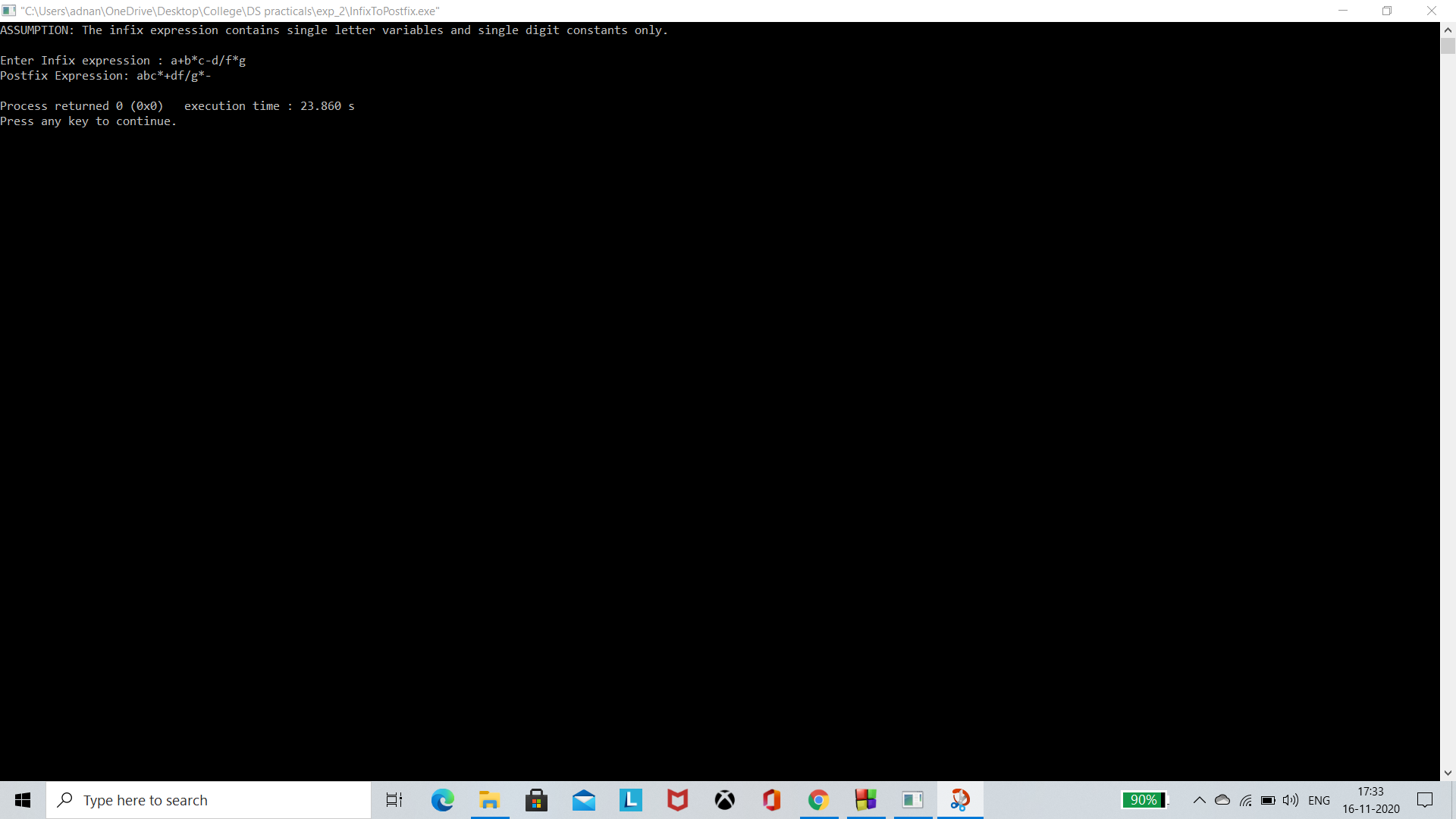
printf("Postfix Expression: ");

puts(postfix);

return 0;

}

Output:



Program 3:

#include<stdio.h>

int stack[20];

int top = -1;

void push(int x)

{

stack[++top] = x;

}

int pop()

{

return stack[top--];

}

int main()

{

char exp[20];

char \*e;

int n1,n2,n3,num;

printf("Enter the expression :: ");

scanf("%s",exp);

e = exp;

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

{

if(isdigit(\*e))

{

num = \*e - 48;

push(num);

}

else

{

n1 = pop();

n2 = pop();

switch(\*e)

{

case '+':

{

n3 = n1 + n2;

break;

}

case '-':

{

n3 = n2 - n1;

break;

}

case '\*':

{

n3 = n1 \* n2;

break;

}

case '/':

{

n3 = n2 / n1;

break;

}

}

push(n3);

}

e++;

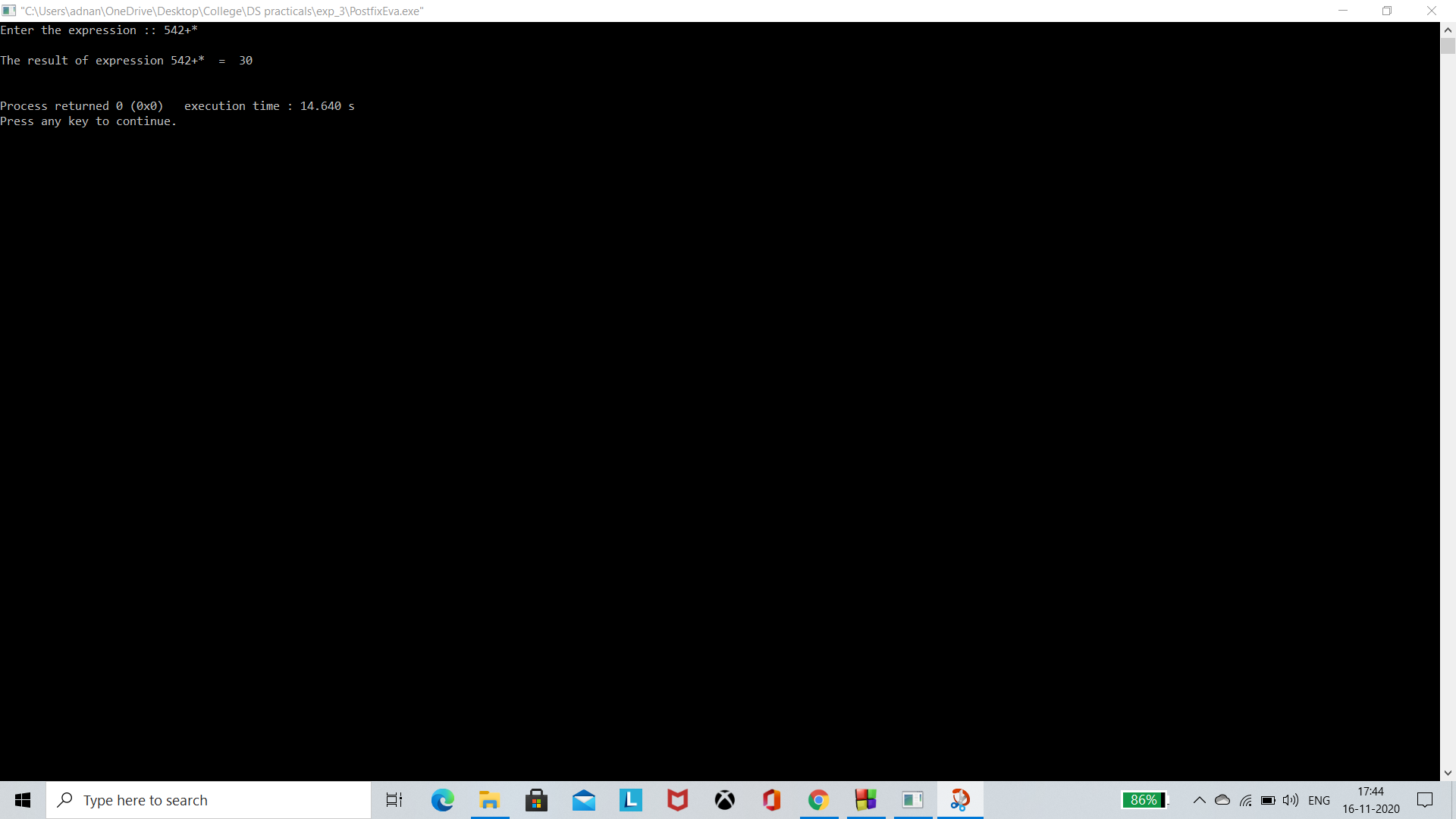
}

printf("\nThe result of expression %s = %d\n\n",exp,pop());

return 0;

}

Output:



Program 4:

#include<stdio.h>

#include<unistd.h>

#define MAX 5

struct Queue

{

int data[MAX];

int front,rear;

};

void initialize(struct Queue \*q)

{

q->front=q->rear=-1;

}

int isFull(struct Queue \*q)

{

return (q->rear==MAX-1)?1:0;

}

int isEmpty(struct Queue \*q)

{

return (q->rear==-1)?1:0;

}

int insert(struct Queue \*q,int d)

{

if(isFull(q))

return 0;

q->data[++q->rear]=d;

if(q->front==-1)

q->front=0;

return 1;

}

void display(struct Queue \*q)

{

int i;

if(isEmpty(q))

printf("\n\tQueue is Empty->");

else

{

printf("\nQueue Contents ->->\n");

printf("Queue Size : %d\nFront = %d\nRear = %d\n",MAX,q->front,q->rear);

for(i=q->front;i<=q->rear;i++)

{

printf("%d\n",q->data[i]);

}

}

}

int delete(struct Queue \*q)

{

int d;

d=q->data[q->front];

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

q->front=q->rear=-1;

else

q->front++;

return d;

}

int search(struct Queue \*q,int k)

{

int i;

for(i=q->front;i<=q->rear;i++)

if(q->data[i]==k)

return i;

return -1;

}

int main()

{

int ch,d;

struct Queue q;

initialize(&q);

while(1)

{

printf("\n\t\t\tMENU\n1. Insert.\n2. Delete.\n3. Search.");

printf("\n4. Display.\n5. Exit.");

printf("\n\tEnter your choice :: ");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nEnter Data to be Inserted : ");

scanf("%d",&d);

d=insert(&q,d);

if(d==0)

printf("Queue is full...");

else

printf("Insertion done successfully...");

break;

case 2:

if(isEmpty(&q))

printf("\nQueue is empty...");

else

printf("\nDeleted element is %d",delete(&q));

break;

case 3:

printf("\nEnter Data to be Searched : ");

scanf("%d",&d);

d=search(&q,d);

if(d==-1)

printf("\nKey is not found..");

else

printf("\nKey is found at location %d..",d);

break;

case 4:

display(&q);

break;

case 5:

exit(0);

default:

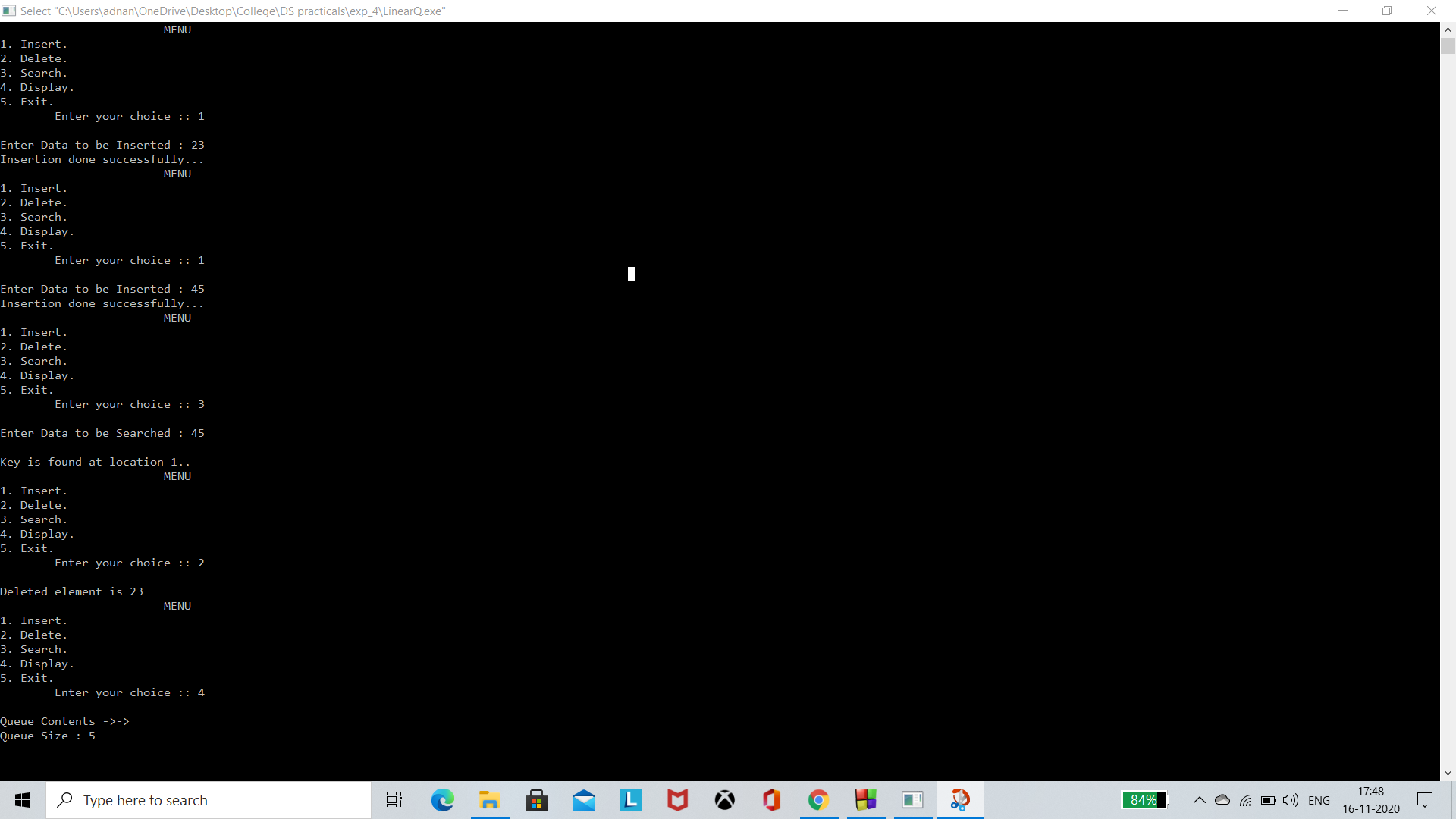
printf("\n\tPlease enter correct choice->->->->");

}

}

}

Output:



Program 5:

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#define MAX 5

struct Queue

{

int data[MAX];

int front,rear;

};

int isFull(struct Queue \*q)

{

return (q->front==((q->rear+1)%MAX))?1:0;

}

int insert(struct Queue \*q,int d)

{

if(isFull(q))

return 0;

else

{

//q->rear++;

q->rear=(q->rear+1)%MAX;

q->data[q->rear]=d;

if(q->front==-1)

q->front=0;

return 1;

}

}

void display(struct Queue \*q)

{

int i;

if(q->rear==-1)

printf("\n\tQueue is Empty->");

else

{

printf("\nQueue Contents ->->\n");

for(i=q->front;i!=q->rear;i=(i+1)%MAX)

{

printf("%d\n",q->data[i]);

}

printf("%d\n",q->data[i]);

}

}

void initialize(struct Queue \*q)

{

q->front=q->rear=-1;

}

void delete(struct Queue \*q)

{

if(q->rear==-1)

printf("\nQueue is empty..");

else

{

int d;

d=q->data[q->front];

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

q->front=q->rear=-1;

else

q->front=(q->front+1)%MAX;

printf("\nElement deleted from Queue.");

}

}

int search(struct Queue \*q,int k)

{

int i;

for(i=q->front;i!=q->rear;i=(i+1)%MAX)

if(q->data[i]==k)

return i;

if(q->data[i]==k)

return i;

return -1;

}

int main()

{

int ch,d;

struct Queue q;

initialize(&q);

while(1)

{

printf("\n\t\t\tMENU\n1. Insert.\n2. Delete.\n3. Search.");

printf("\n4. Display.\n5. Exit.");

printf("\n\tEnter your choice :: ");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nEnter Data to be Inserted : ");

scanf("%d",&d);

insert(&q,d);

break;

case 2:

delete(&q);

break;

case 3:

printf("\nEnter Data to be Searched : ");

scanf("%d",&d);

d=search(&q,d);

if(d==-1)

printf("\nKey is not found..");

else

printf("\nKey is found at location %d..",d);

break;

case 4:

display(&q);

break;

case 5:

exit(0);

default:

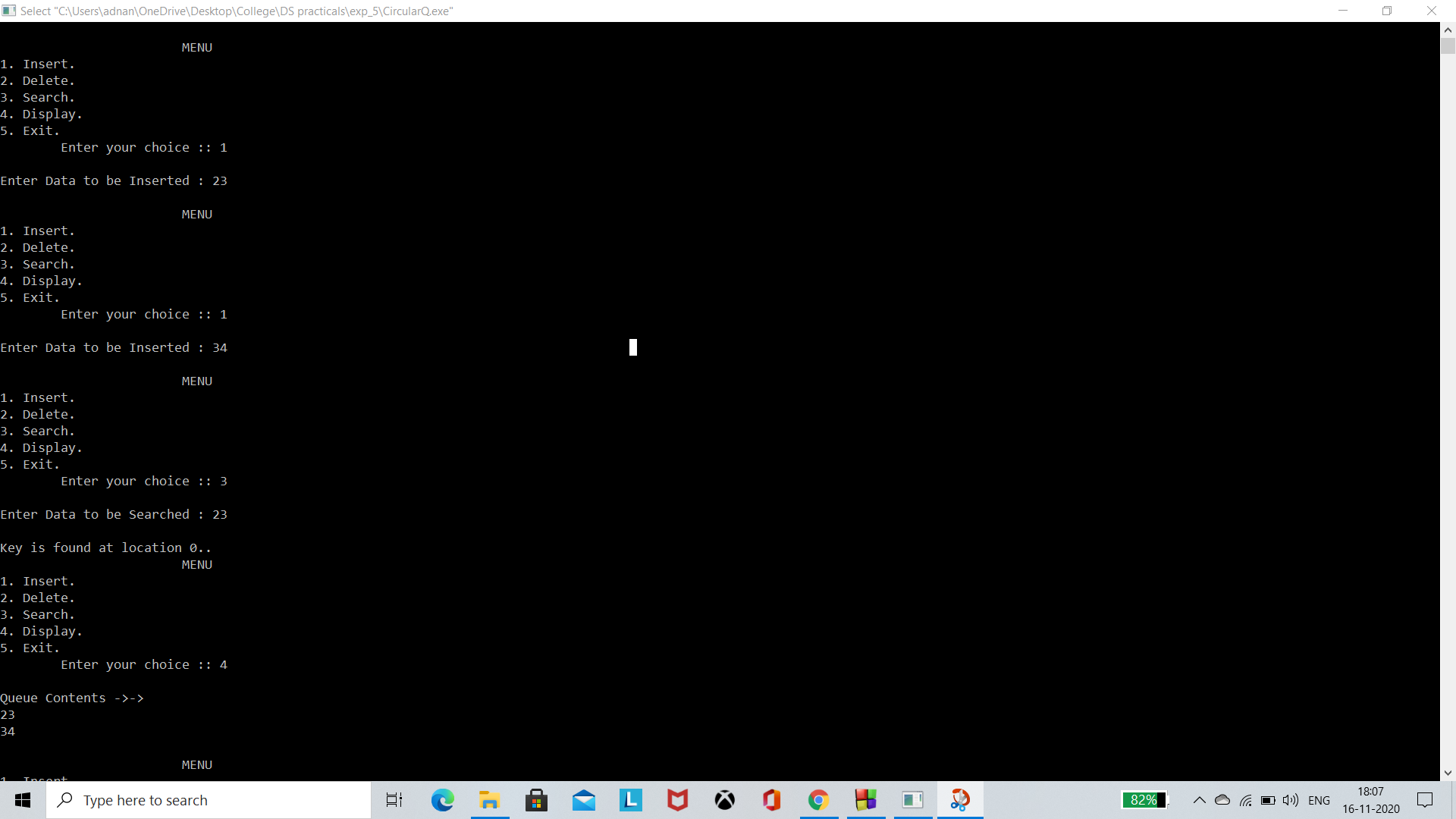
printf("\n\tPlease enter correct choice->->->->");

}

}

}

Output:



Program 6:

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

struct SLL

{

int data;

struct SLL \*next;

};

struct SLL \*insertAtEnd(struct SLL \*h,int d)

{

struct SLL \*p,\*tmp;

p=(struct SLL \*)malloc(sizeof(struct SLL));

if(p==NULL)

{

printf("\nNot enough memory to allocate.");

return h;

}

p->data=d;

p->next=NULL;

if(h==NULL)

h=p;

else

{

tmp=h;

while(tmp->next!=NULL)

tmp=tmp->next;

tmp->next=p;

}

return h;

}

struct SLL \*insertAtStart(struct SLL \*h,int d)

{

struct SLL \*p,\*tmp;

p=(struct SLL \*)malloc(sizeof(struct SLL)); //creating a new node

p->data=d; //putting data into new node

p->next=h; //making new node point to null

h=p;

return h;

}

struct SLL \*insertAfter(struct SLL \*h,int key,int d)

{

struct SLL \*p,\*tmp;

p=(struct SLL \*)malloc(sizeof(struct SLL)); //creating a new node

p->data=d; //putting data into new node

p->next=NULL; //making new node point to null

if(h==NULL)

{ // LL is empty

h=p;

}

else

{ //LL is not empty

tmp=h;

while(tmp!=NULL && tmp->data!=key ) //to traverse to the last node

tmp=tmp->next;

if(tmp!=NULL)

{

p->next=tmp->next;

tmp->next=p;

}

else

{

printf("\n\tGiven Node %d does not exist in the Linked List.",key);

free(p);

}

}

return h;

}

struct SLL \*removelast(struct SLL \*h)

{

struct SLL \*tmp,\*prev;

tmp=h;

if(h!=NULL)

{

if(h->next!=NULL)

{

while(tmp->next!=NULL)

{

prev=tmp;

tmp=tmp->next;

}

prev->next=NULL;

}

else

h=NULL;

free(tmp);

}

else

printf("\nLL is empty.");

return h;

}

struct SLL \*removefirst(struct SLL \*h)

{

struct SLL \*tmp;

tmp=h;

if(h!=NULL)

{

h=h->next;

free(tmp);

}

else

printf("\nLL is empty.");

return h;

}

struct SLL \*removeAfter(struct SLL \*h,int key)

{

struct SLL \*tmp,\*p;

tmp=h;

if(h!=NULL)

{

while(tmp!=NULL && tmp->data!=key)

tmp=tmp->next;

if(tmp!=NULL)

{

if(tmp->next!=NULL)

{

p=tmp->next;

tmp->next=p->next;

p->next=NULL;

free(p);

}

else

printf("\nGiven Node is the last Node.");

}

else

printf("\nGiven key does not exist.");

//printf("\ntmp->data=%d",tmp->dmpata);

//printf("\nprev->data=%d",prev->data);

}

else

printf("\nLL is empty.");

return h;

}

void display(struct SLL \*h)

{

struct SLL \*tmp;

tmp=h;

//modify to empty list

if(h!=NULL)

{

printf("\n\n\t\tLinked List Contents..\n");

while(tmp!=NULL)

{

printf("\t%d\n",tmp->data);

tmp=tmp->next;

}

}

else

{

printf("\nLL is empty.");

}

}

int main()

{

struct SLL \*head;

int ch,d,k;

head=NULL;

while(1)

{

printf("\n\n\n\t\t\tMENU");

printf("\n1. Insert.\n2. Insert After.\n3. Remove.\n4. Remove After.\n5. Display.\n6. Exit.");

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

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\n\tEnter Data : ");

scanf("%d",&d);

head=insertAtEnd(head,d);

break;

case 2:

printf("\n\tEnter Data : ");

scanf("%d",&d);

printf("\n\tEnter Key Data : ");

scanf("%d",&k);

head=insertAfter(head,k,d);

break;

case 3:

//head=removelast(head);

head=removefirst(head);

break;

case 4:

printf("\n\tEnter Key : ");

scanf("%d",&d);

head=removeAfter(head,d);

break;

case 5:

display(head);

break;

case 6:

return 0;

break;

default:

printf("\n\t\tPlease enter correct choice....");

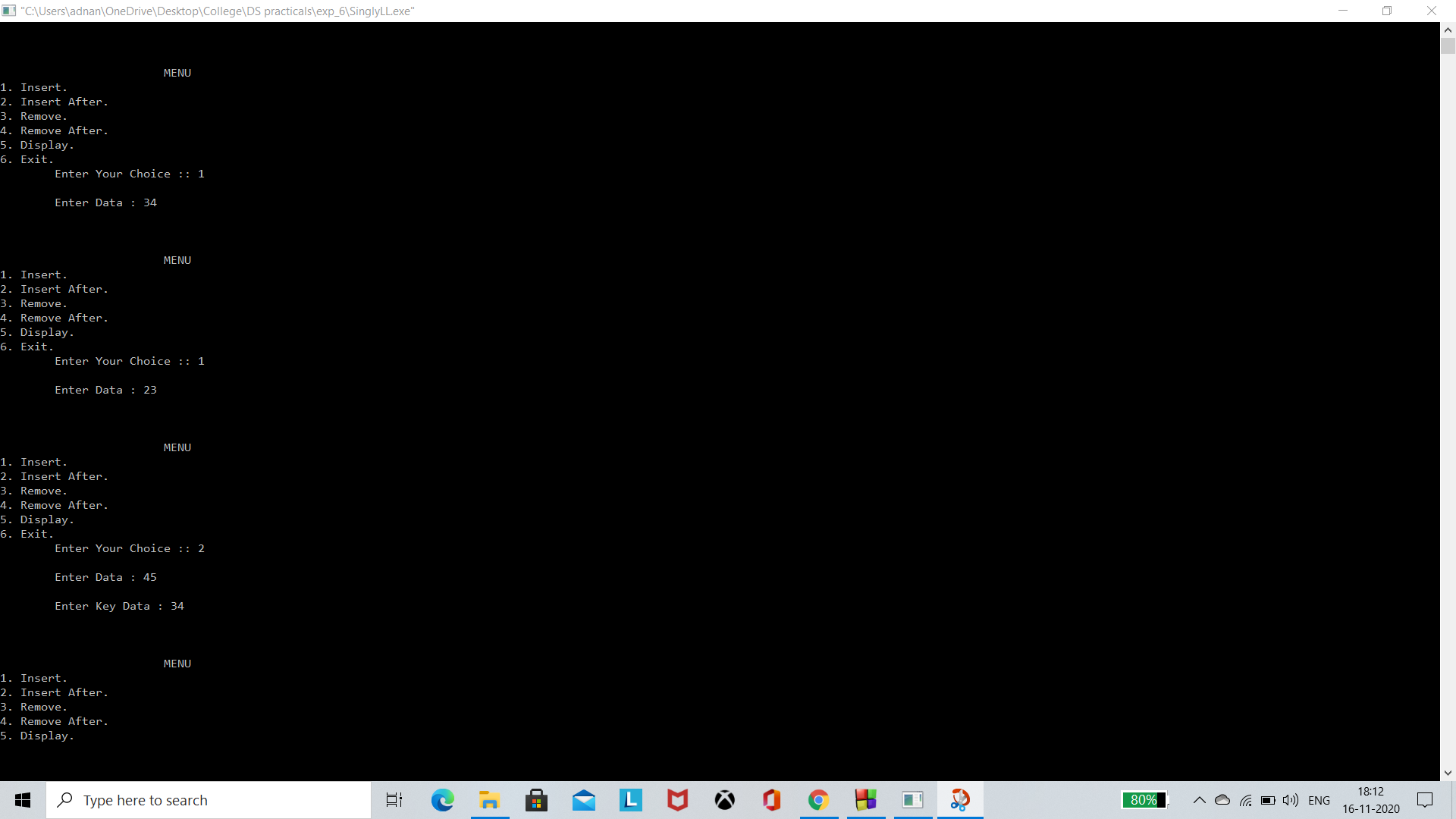
}

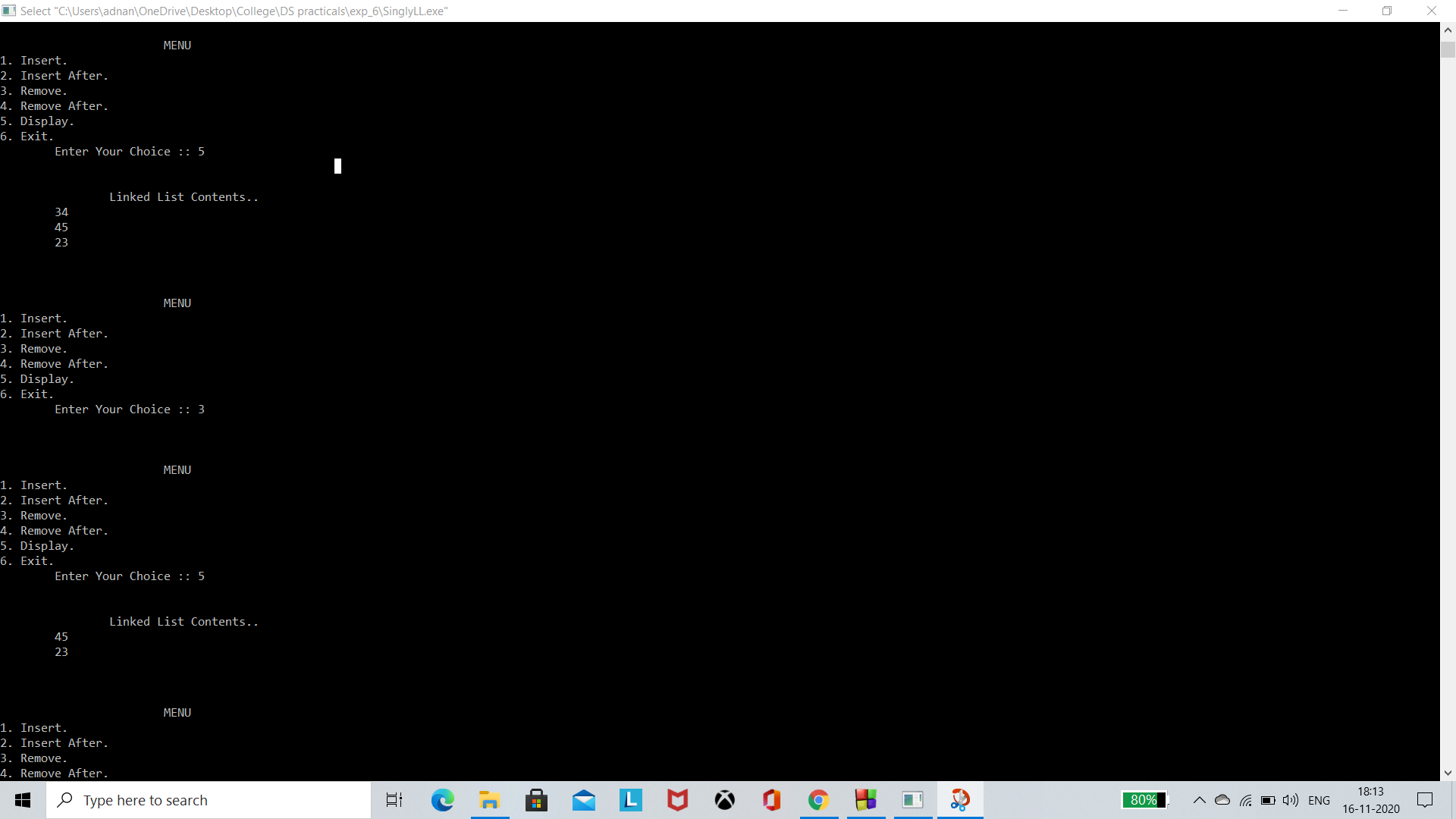
}

return 0;

}

Output:





Program 7:

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

void beginsert ();

void lastinsert ();

void randominsert();

void begin\_delete();

void last\_delete();

void random\_delete();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 7)

{

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

printf("\nChoose one option from the following list ...\n");

printf("\n===============================================\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from last\n5.Search for an element\n6.Show\n7.Exit\n");

printf("\nEnter your choice?\n");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

beginsert();

break;

case 2:

lastinsert();

break;

case 3:

begin\_delete();

break;

case 4:

last\_delete();

break;

case 5:

search();

break;

case 6:

display();

break;

case 7:

exit(0);

break;

default:

printf("Please enter valid choice..");

}

}

}

void beginsert()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter the node data?");

scanf("%d",&item);

ptr -> data = item;

if(head == NULL)

{

head = ptr;

ptr -> next = head;

}

else

{

temp = head;

while(temp->next != head)

temp = temp->next;

ptr->next = head;

temp -> next = ptr;

head = ptr;

}

printf("\nnode inserted\n");

}

}

void lastinsert()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW\n");

}

else

{

printf("\nEnter Data?");

scanf("%d",&item);

ptr->data = item;

if(head == NULL)

{

head = ptr;

ptr -> next = head;

}

else

{

temp = head;

while(temp -> next != head)

{

temp = temp -> next;

}

temp -> next = ptr;

ptr -> next = head;

}

printf("\nnode inserted\n");

}

}

void begin\_delete()

{

struct node \*ptr;

if(head == NULL)

{

printf("\nUNDERFLOW");

}

else if(head->next == head)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{ ptr = head;

while(ptr -> next != head)

ptr = ptr -> next;

ptr->next = head->next;

free(head);

head = ptr->next;

printf("\nnode deleted\n");

}

}

void last\_delete()

{

struct node \*ptr, \*preptr;

if(head==NULL)

{

printf("\nUNDERFLOW");

}

else if (head ->next == head)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

while(ptr ->next != head)

{

preptr=ptr;

ptr = ptr->next;

}

preptr->next = ptr -> next;

free(ptr);

printf("\nnode deleted\n");

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag=1;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

if(head ->data == item)

{

printf("item found at location %d",i+1);

flag=0;

}

else

{

while (ptr->next != head)

{

if(ptr->data == item)

{

printf("item found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

}

if(flag != 0)

{

printf("Item not found\n");

}

}

}

void display()

{

struct node \*ptr;

ptr=head;

if(head == NULL)

{

printf("\nnothing to print");

}

else

{

printf("\n printing values ... \n");

while(ptr -> next != head)

{

printf("%d\n", ptr -> data);

ptr = ptr -> next;

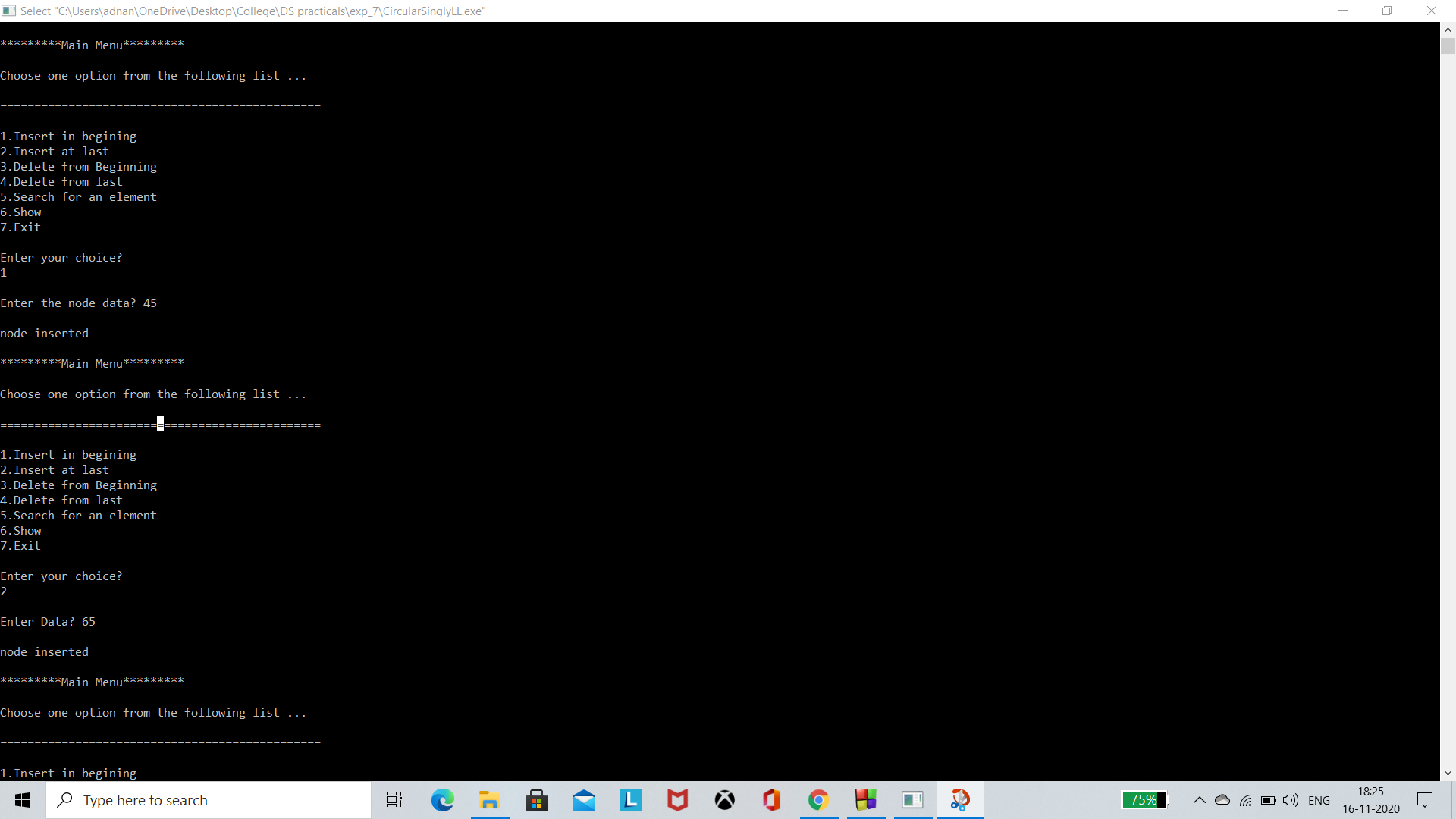
}

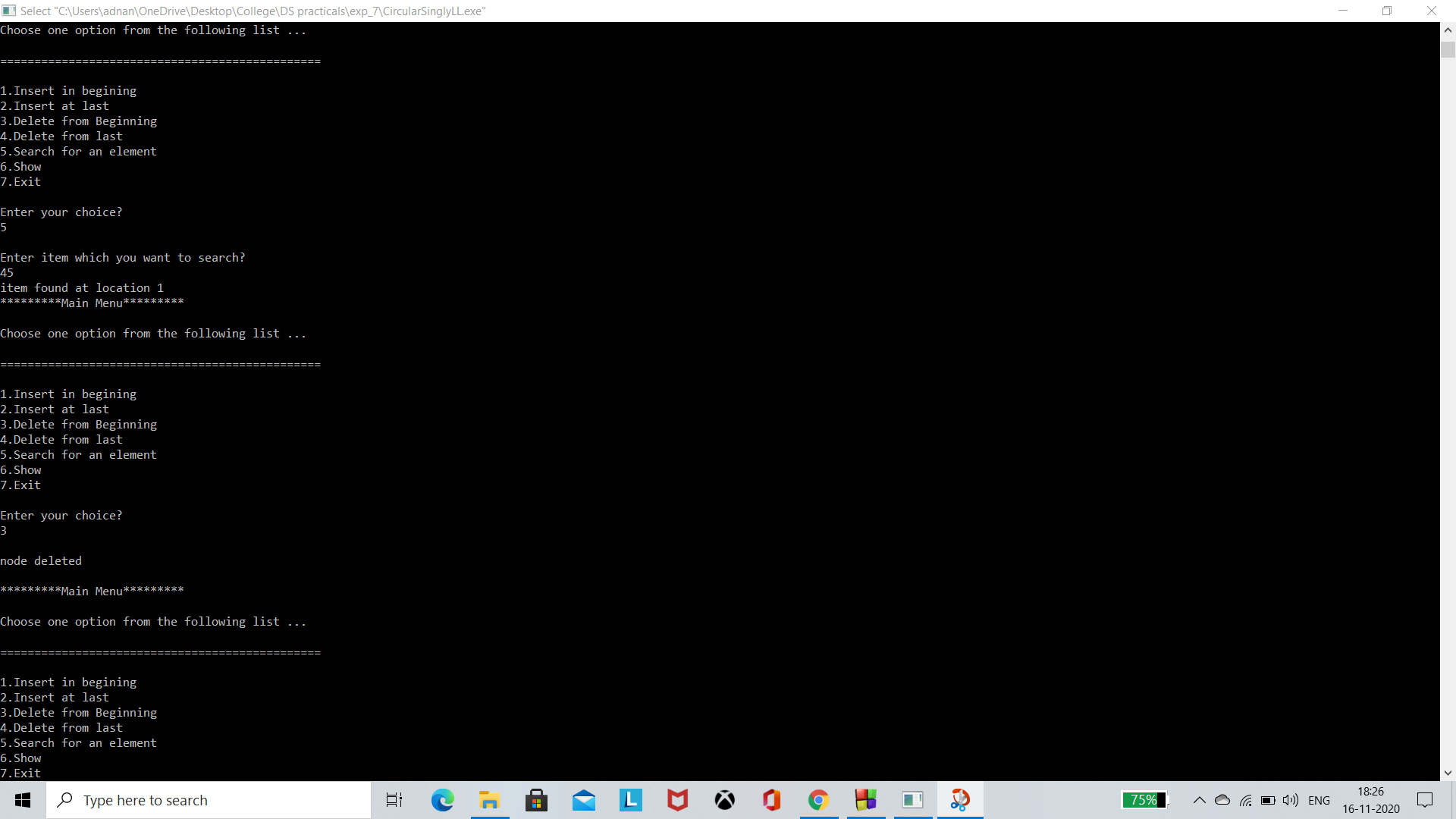
printf("%d\n", ptr -> data);

}

}

Output:





Program 8:

#include <stdio.h>

#include <stdlib.h>

void push();

void pop();

void display();

struct node

{

int val;

struct node \*next;

};

struct node \*head;

void main ()

{

int choice=0;

printf("\n\*\*\*\*\*\*\*\*\*Stack operations using linked list\*\*\*\*\*\*\*\*\*\n");

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

while(choice != 4)

{

printf("\n\nChose one from the below options...\n");

printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");

printf("\n Enter your choice \n");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("Exiting....");

break;

}

default:

{

printf("Please Enter valid choice ");

}

};

}

}

void push ()

{

int val;

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

if(ptr == NULL)

{

printf("not able to push the element");

}

else

{

printf("Enter the value");

scanf("%d",&val);

if(head==NULL)

{

ptr->val = val;

ptr -> next = NULL;

head=ptr;

}

else

{

ptr->val = val;

ptr->next = head;

head=ptr;

}

printf("Item pushed");

}

}

void pop()

{

int item;

struct node \*ptr;

if (head == NULL)

{

printf("Underflow");

}

else

{

item = head->val;

ptr = head;

head = head->next;

free(ptr);

printf("Item popped");

}

}

void display()

{

int i;

struct node \*ptr;

ptr=head;

if(ptr == NULL)

{

printf("Stack is empty\n");

}

else

{

printf("Printing Stack elements \n");

while(ptr!=NULL)

{

printf("%d\n",ptr->val);

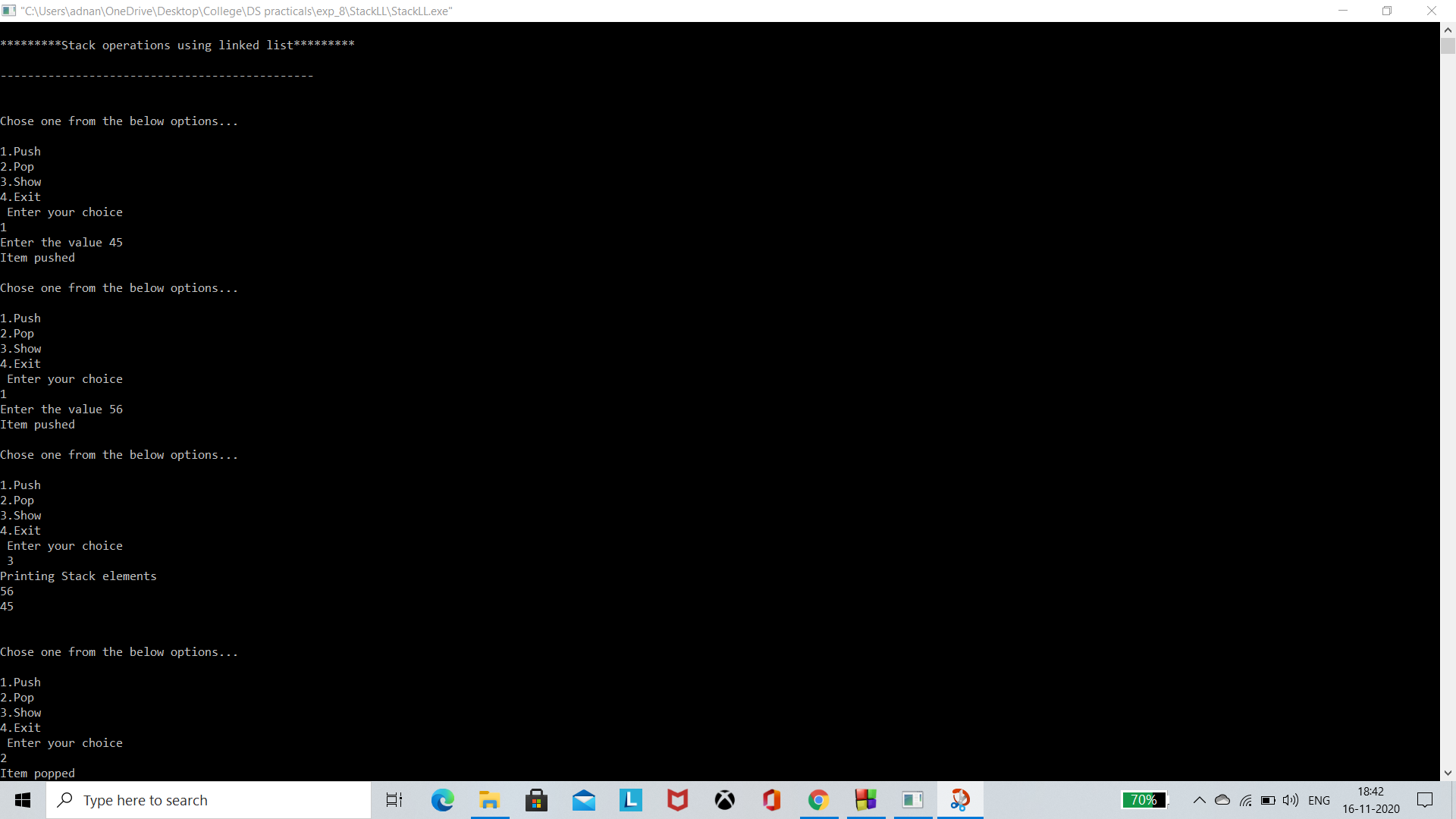
ptr = ptr->next;

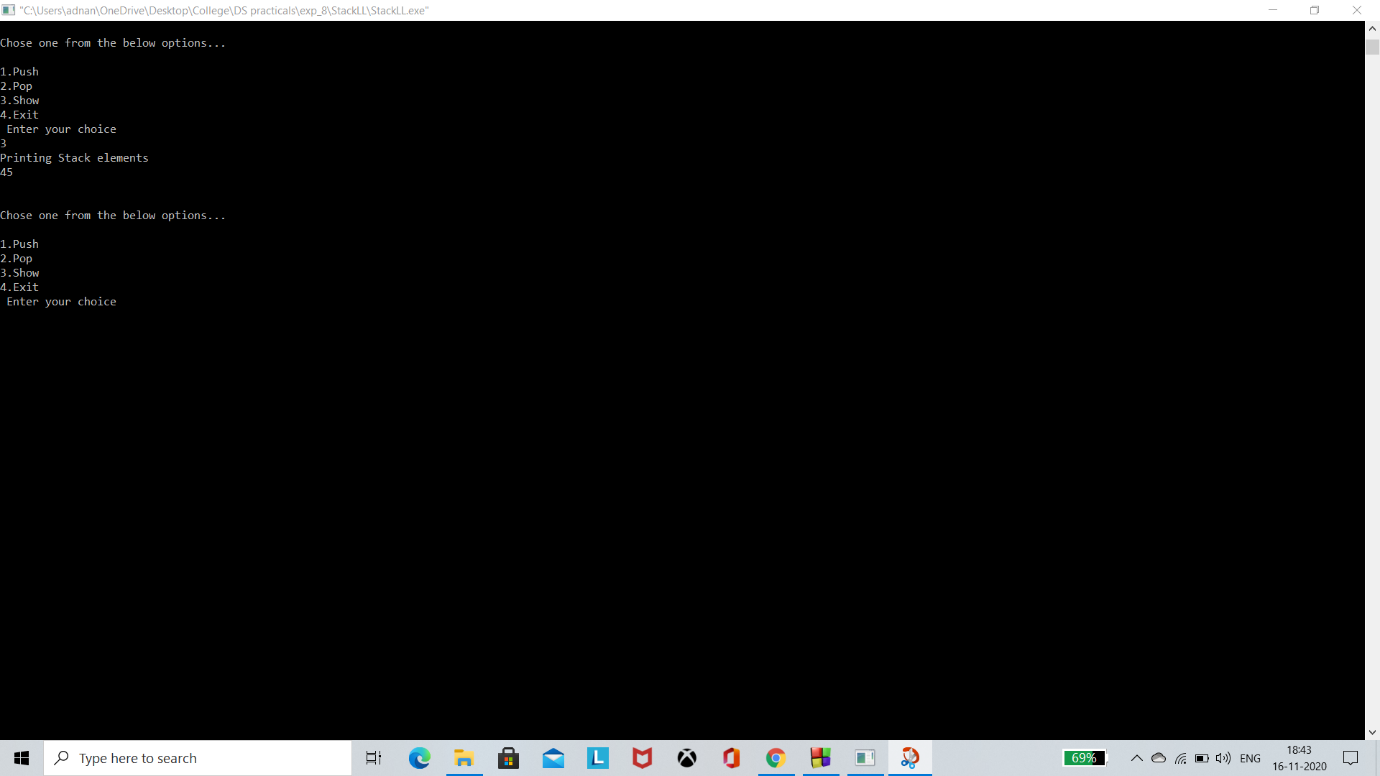
}

}

}

Output:





Program 9:

#include<stdio.h>

#include<malloc.h>

struct tree {

int info;

struct tree \*left;

struct tree \*right;

};

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

void inorder(struct tree \*);

void postorder(struct tree \*);

void preorder(struct tree \*);

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

struct tree \*search(struct tree \*);

int main(void) {

struct tree \*root;

int choice, item,item\_no;

root = NULL;

do {

do {

printf("\n \t 1. Insert in Binary Tree ");

printf("\n\t 2. Delete from Binary Tree ");

printf("\n\t 3. Inorder traversal of Binary tree");

printf("\n\t 4. Postorder traversal of Binary tree");

printf("\n\t 5. Preorder traversal of Binary tree");

printf("\n\t 6. Search and replace ");

printf("\n\t 7. Exit ");

printf("\n\t Enter choice : ");

scanf(" %d",&choice);

if(choice<1 || choice>7)

printf("\n Invalid choice - try again");

}

while (choice<1 || choice>7);

switch(choice) {

case 1:

printf("\n Enter new element: ");

scanf("%d", &item);

root= insert(root,item);

printf("\n root is %d",root->info);

printf("\n Inorder traversal of binary tree is : ");

inorder(root);

break;

case 2:

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

scanf(" %d",&item\_no);

root=delet(root,item\_no);

inorder(root);

break;

case 3:

printf("\n Inorder traversal of binary tree is : ");

inorder(root);

break;

case 4:

printf("\n Postorder traversal of binary tree is : ");

postorder(root);

break;

case 5:

printf("\n Preorder traversal of binary tree is : ");

preorder(root);

break;

case 6:

printf("\n Search and replace operation in binary tree ");

root=search(root);

break;

default:

printf("\n End of program ");

}

}

while(choice !=7);

return(0);

}

struct tree \*insert(struct tree \*root, int x) {

if(!root) {

root=(struct tree\*)malloc(sizeof(struct tree));

root->info = x;

root->left = NULL;

root->right = NULL;

return(root);

}

if(root->info > x)

root->left = insert(root->left,x); else {

if(root->info < x)

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

}

return(root);

}

void inorder(struct tree \*root) {

if(root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

return;

}

void postorder(struct tree \*root) {

if(root != NULL) {

postorder(root->left);

postorder(root->right);

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

}

return;

}

void preorder(struct tree \*root) {

if(root != NULL) {

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

preorder(root->left);

preorder(root->right);

}

return;

}

/\* FUNCTION TO DELETE A NODE FROM A BINARY TREE \*/

struct tree \*delet(struct tree \*ptr,int x) {

struct tree \*p1,\*p2;

if(!ptr) {

printf("\n Node not found ");

return(ptr);

} else {

if(ptr->info < x) {

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

/\*return(ptr);\*/

} else if (ptr->info >x) {

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

return ptr;

} else

{

if(ptr->info == x)

{

if(ptr->left == ptr->right)

{

free(ptr);

return(NULL);

} else if(ptr->left==NULL)

{

p1=ptr->right;

free(ptr);

return p1;

} else if(ptr->right==NULL)

{

p1=ptr->left;

free(ptr);

return p1;

} else {

p1=ptr->right;

p2=ptr->right;

while(p1->left != NULL)

p1=p1->left;

p1->left=ptr->left;

free(ptr);

return p2;

}

}

}

}

return(ptr);

}

struct tree \*search(struct tree \*root) {

int no,i,ino;

struct tree \*ptr;

ptr=root;

printf("\n Enter the element to be searched :");

scanf(" %d",&no);

fflush(stdin);

while(ptr) {

if(no>ptr->info)

ptr=ptr->right; else if(no<ptr->info)

ptr=ptr->left; else

break;

}

if(ptr) {

printf("\n Element %d which was searched is found and is = %d",no,ptr->info);

printf("\n Do you want replace it, press 1 for yes : ");

scanf(" %d",&i);

if(i==1) {

printf("\n Enter new element :");

scanf(" %d",&ino);

ptr->info=ino;

} else

printf("\n\t It's okay");

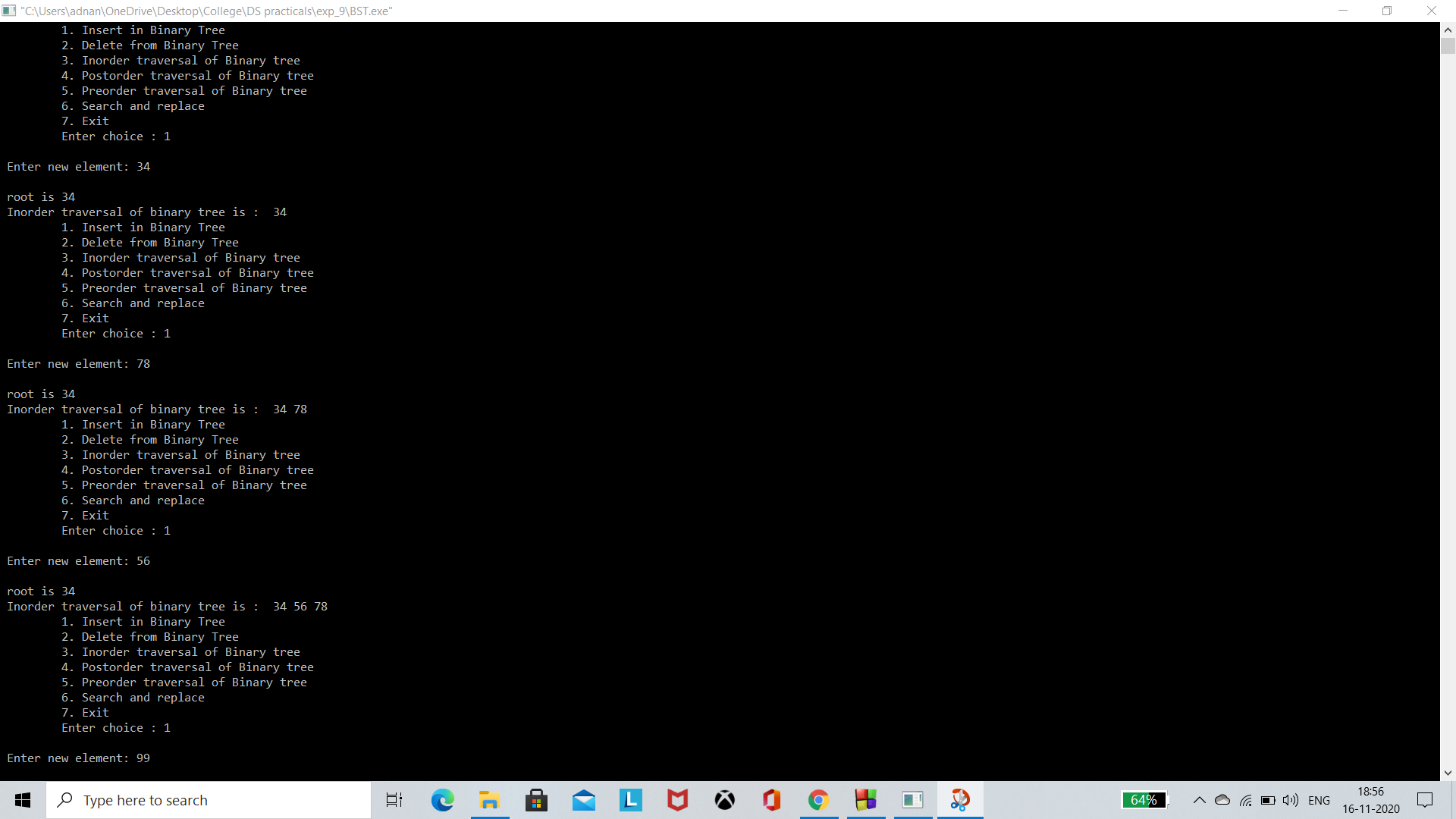
} else

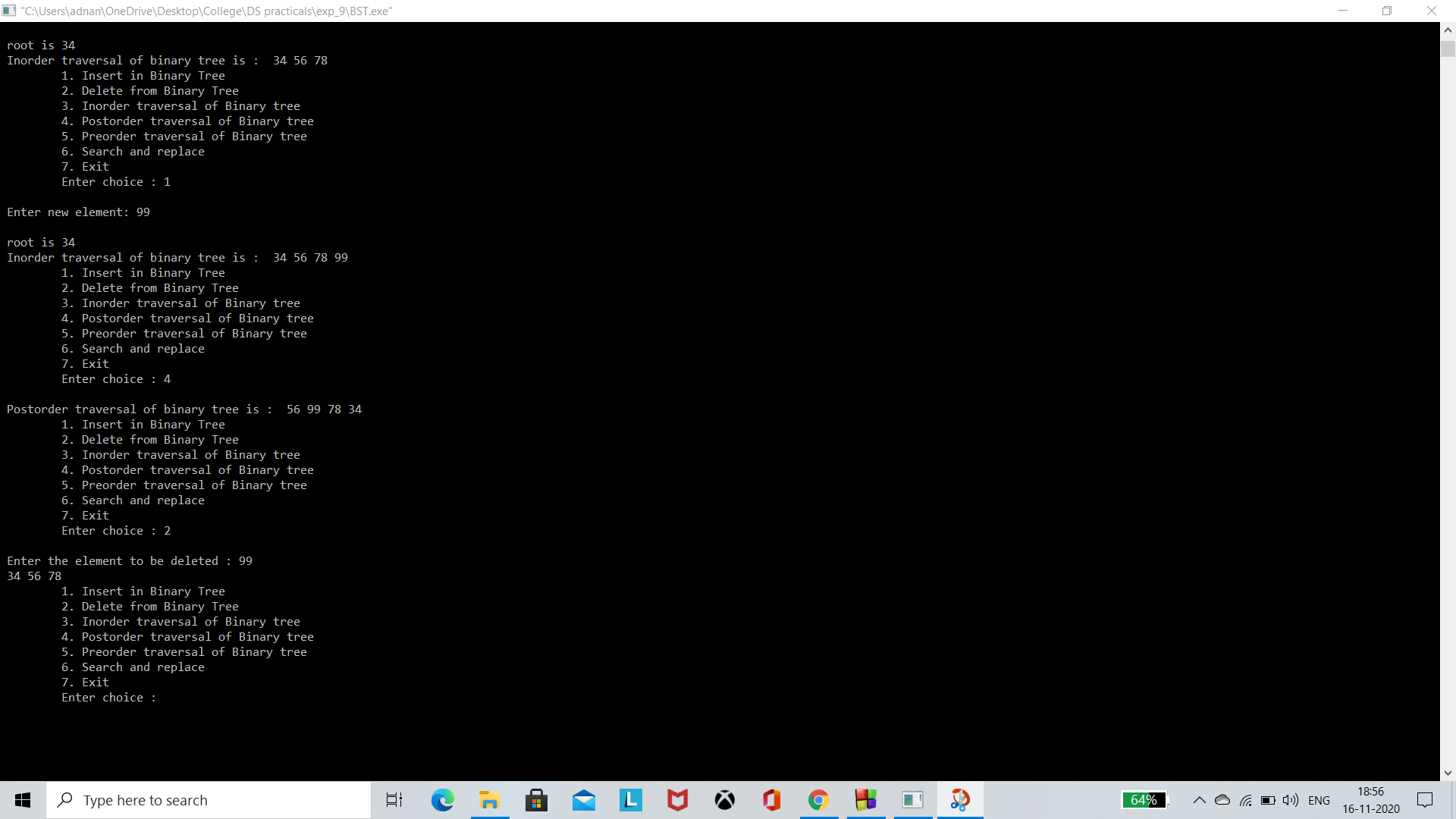
printf("\n Element %d does not exist in the binary tree",no);

return(root);

}

Output:





Program 10:

#include<stdio.h>

#include<unistd.h>

#define MAX 20

struct stack

{

int data[MAX];

int top;

};

struct Queue

{

int data[MAX];

int front,rear;

};

void push(struct stack \*s,int d)

{

if(s->top==MAX-1)

printf("\n\tStack Overflows->");

else

{

s->top++;

s->data[s->top]=d;

}

}

int pop(struct stack \*s)

{

if(s->top==-1)

printf("\n\tStack Underflows->");

else

return s->data[s->top--];

}

void initialize(struct stack \*s)

{

s->top=-1;

}

void insert(struct Queue \*q,int d)

{

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

printf("\n\tQueue is Full->");

else

{

q->rear++;

q->data[q->rear]=d;

if(q->front==-1)

q->front=0;

}

}

void initializeQ(struct Queue \*q)

{

q->front=q->rear=-1;

}

int delete(struct Queue \*q)

{

if(q->rear==-1)

printf("\nQueue is empty..");

else

{

int d;

d=q->data[q->front];

q->front++;

if(q->front>q->rear)

q->front=q->rear=-1;

return d;

}

}

void dfs(int a[][10],int n)

{

struct stack s;

int visited[10];

int i,j,v;

initialize(&s);

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

visited[i]=0; //setting all unvisited.

visited[0]=1; //visit first vertex

printf("\n\t\tDFS Traversal :\n0\t");

push(&s,0);

while(s.top!=-1) // until stack empty,

{

v=-1;

// get an unvisited vertex adjacent to stack top

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

if(a[s.data[s.top]][j]==1 && visited[j]==0)

{

v=j;

break;

}

if(v==-1) // if no such vertex,

v=pop(&s);

else // if it exists,

{

visited[v]=1; // mark it

printf("%d\t",v); // display it

push(&s,v); // push it

}

} // end while

}

void bfs(int a[][10],int n)

{

struct Queue q;

int visited[10];

int i,j,v;

initializeQ(&q);

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

visited[i]=0; //setting all unvisited.

visited[0]=1; //visit first vertex

printf("\n\t\tBFS Traversal :\n0\t");

insert(&q,0);

while(q.front!=-1) // until queue empty,

{

v=delete(&q);

// get an unvisited vertex adjacent to stack top

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

if(a[v][j]==1 && visited[j]==0)

{

visited[j]=1;

printf("%d\t",j);

insert(&q,j);

}

} // end while

}

int main()

{

int n,i,s,ch,j,a[10][10];

char c,dummy;

while(1)

{

printf("\n\n\t\tMENU\n1. Make Graph.\n2. DFS.\n3. BFS. \n4. Exit.");

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

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nENTER THE NUMBER OF VERTICES :: " );

scanf("%d",&n);

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

{

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

{

printf("\nEnter 1 if %d has a Edge with %d else 0 : ", i+1, j+1);

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

}

}

printf("\n\nTHE ADJACENCY MATRIX IS\n" );

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

{

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

{

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

}

printf("\n");

}

break;

case 2:

dfs(a,n);

break;

case 3:

bfs(a,n);

break;

case 4:

exit(0);

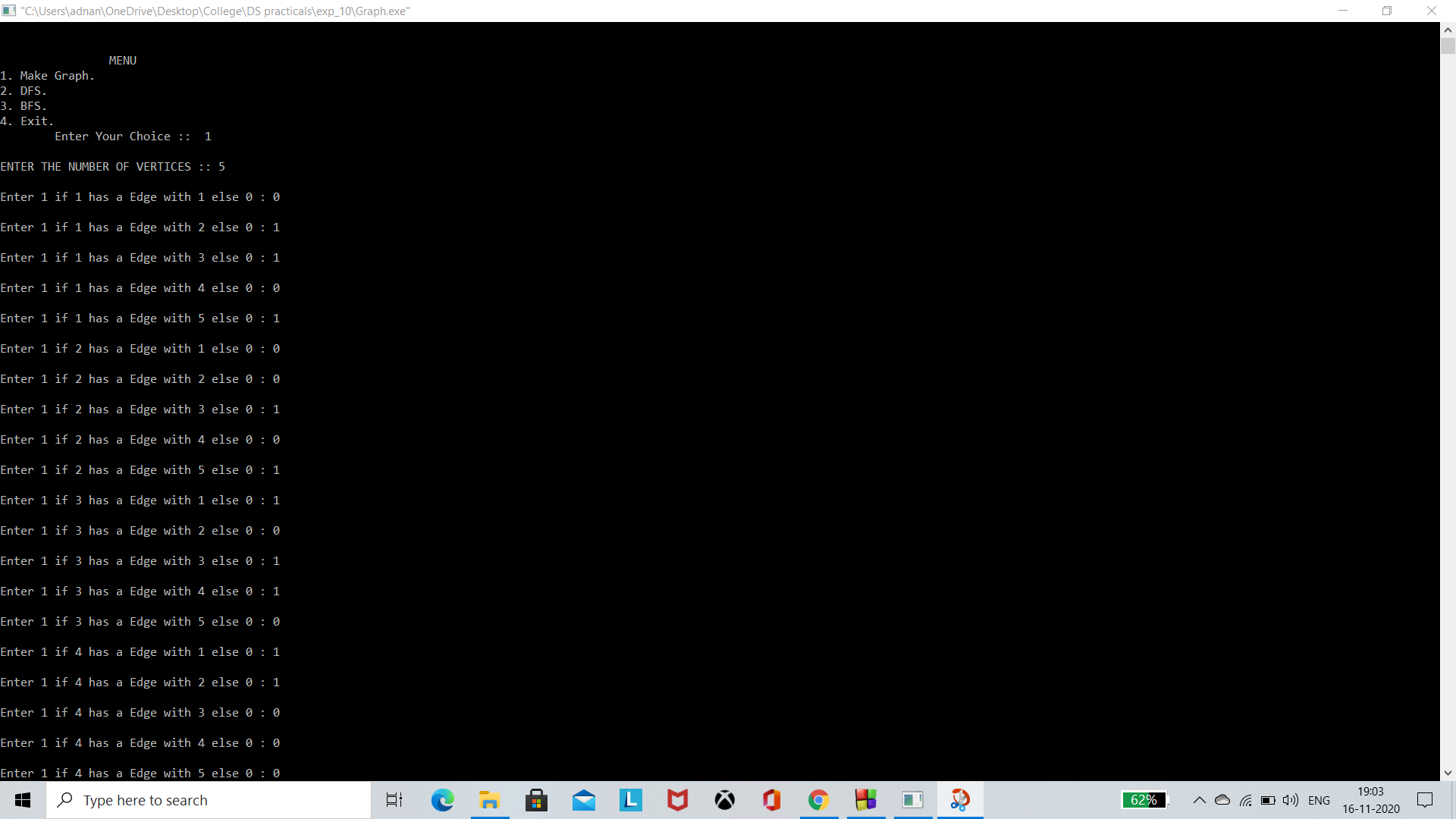
break;

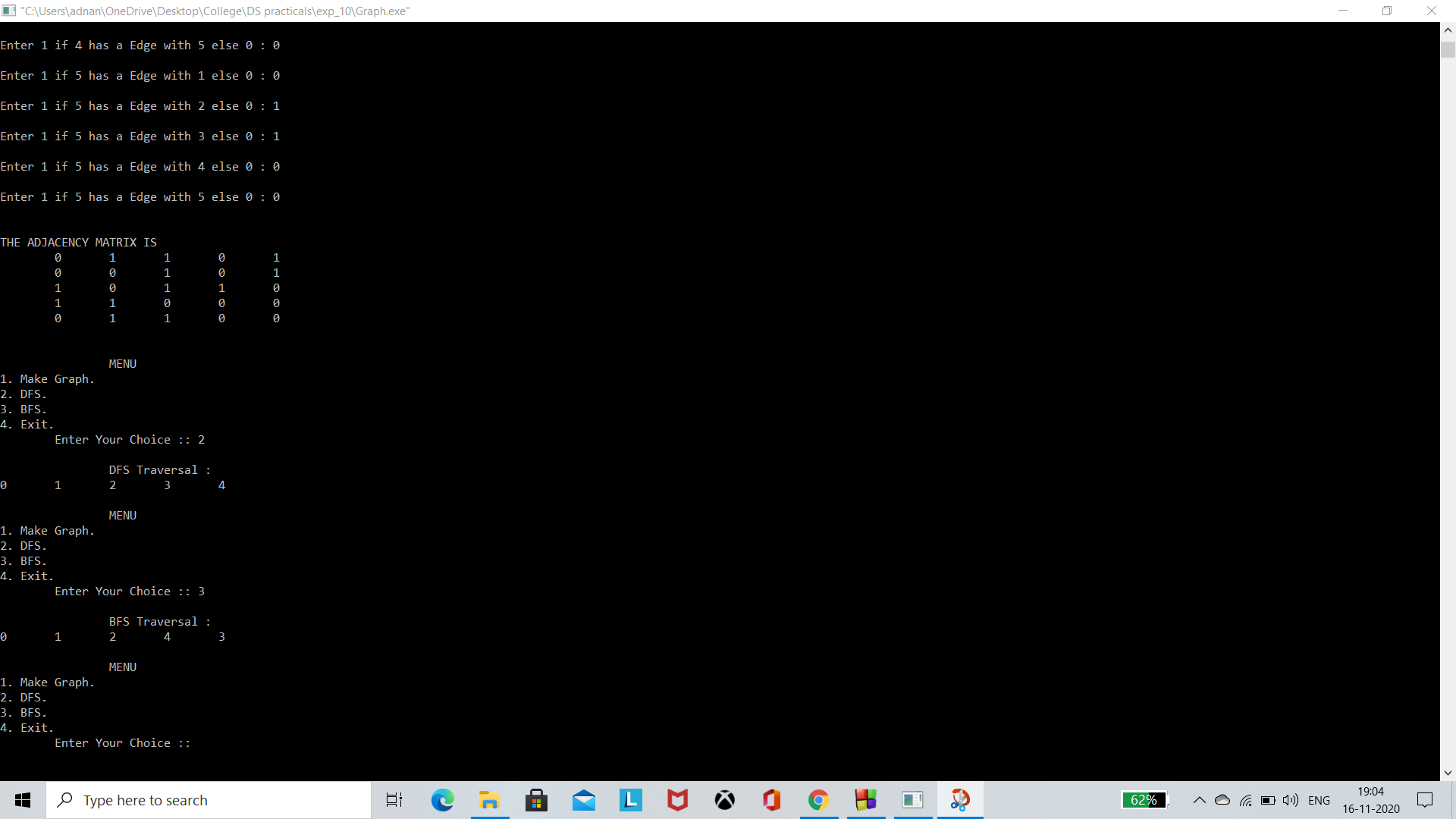
}

}

}

Output:





Program Quick Sort:

#include<stdio.h>

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

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

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high- 1; j++)

{

if (arr[j] < pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(int arr[], int size)

{

int i;

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

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

printf("\n");

}

int main()

{

int ch;

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

scanf("%d",&ch);

int arr[ch];

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

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

quickSort(arr, 0, ch-1);

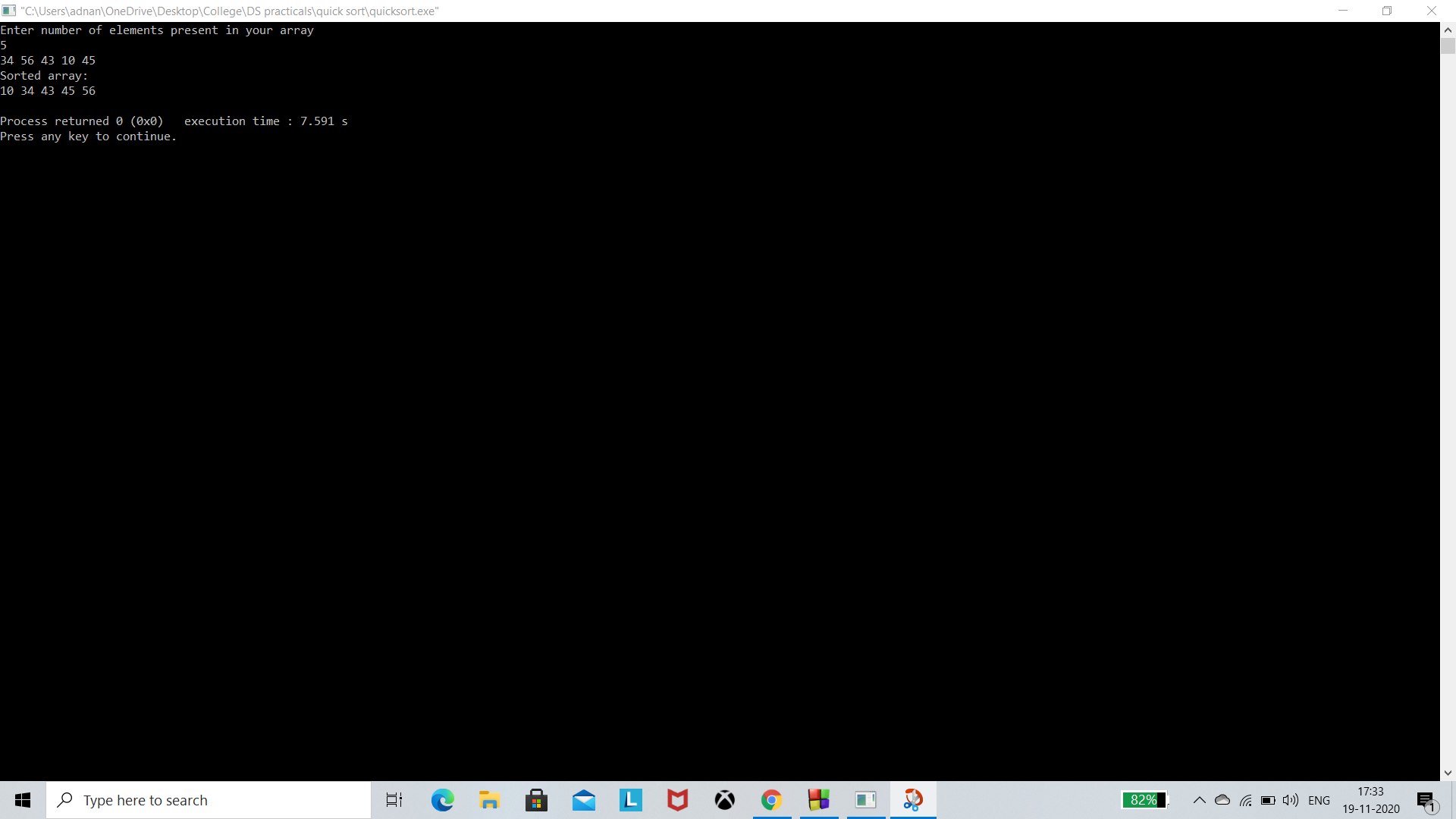
printf("Sorted array: \n");

printArray(arr, ch-1);

return 0;

}

Output:



Program Binary Search:

#include <stdio.h>

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

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

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high- 1; j++)

{

if (arr[j] < pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int binarySearch(int arr[], int l, int r, int x)

{

while (l <= r) {

int m = (l + r)/ 2;

if (arr[m] == x)

return m;

if (arr[m] < x)

l = m + 1;

else

r = m - 1;

}

return 1;

}

int main(void)

{

int ch;

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

scanf("%d",&ch);

int arr[ch];

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

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

quickSort(arr, 0, ch-1);

printf("Sorted array: \n");

printArray(arr, ch-1);

int x;

printf("Enter element you want to search in your array\n");

scanf("%d",&x);

int result = binarySearch(arr,0,ch-1,x);

if(result == 1)

printf("Element is not present in array");

else

printf("Element is present at index %d", result);

return 0;

}

void printArray(int arr[], int size)

{

int i;

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

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

printf("\n");

}

Output:

