**DATA STRUCTURES LAB REPORT**

**NAME: Nikita Paras Toliya**

**USN: 1BM19CS103**

**SEMESTER: 3**

**SUBJECT: DATA STRUCTURES**

**YEAR:2020-2021**

**1 1 Write a program to simulate the working of stack using an array with the following:**

**a) Push**

**b) Pop**

**c) Display**

**The program should print appropriate messages for stack overflow, stack underflow.**

#include<stdio.h>

#include<process.h>

#include<conio.h>

#define STACK\_SIZE 3

int top=-1;

int s[3];

int item;

void push()

{

if (top==STACK\_SIZE-1)

{

printf("stack overflow\n");

return;

}

top=top+1;

s[top]=item;

}

int pop()

{

if(top==-1)

return-1;

return s[top--];

}

void display()

{

int i;

if (top==-1)

{

printf("stack is empty\n");

return;

}

printf("contents of the stack are:\n");

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

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

}

}

int main()

{

int item\_deleted;

int choice;

for(;;)

{

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

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

printf("enter the choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("enter the item to be inserted\n");

scanf("%d",&item);

push();

break;

case 2:

item\_deleted=pop();

if(item\_deleted==-1)

printf("stack is empty\n");

else

printf("item deleted is %d\n", item\_deleted);

break;

case 3:

display();

break;

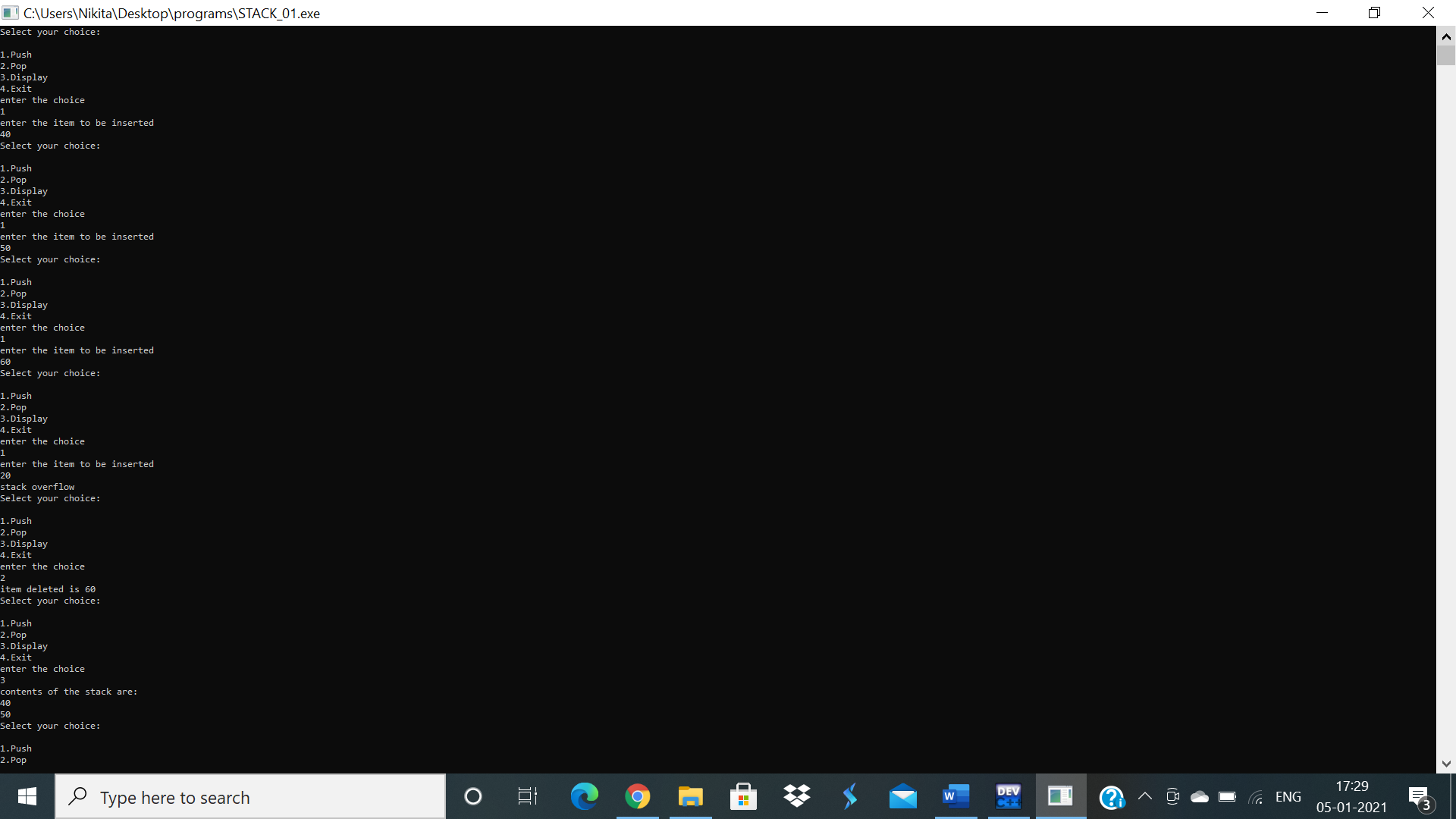
default : exit(0);

}

}

getch();

}



**2 1 WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), \* (multiply) and / (divide)**

#include<stdio.h>

#include<string.h>

int F(char symbol)

{

switch(symbol)

{

case '+':

case '-': return 2;

case '\*':

case '/':return 4;

case '^':

case '$':return 5;

case '(':return 0;

case'#':return -1;

default:return 8;

}

}

int G(char symbol)

{

switch(symbol)

{

case '+':

case '-': return 1;

case '\*':

case '/':return 3;

case '^':

case '$':return 6;

case '(':return 9;

case')':return 0;

default:return 7;

}

}

void infix\_postfix(char infix[],char postfix[])

{

int top,i,j;

char s[30],symbol;

top=-1;

s[++top]='#';

j=0;

for(i=0;i<strlen(infix);i++)

{

symbol = infix[i];

while(F(s[top])>G(symbol))

{

postfix[j]=s[top--];

j++;

}

if(F(s[top])!=G(symbol))

s[++top]=symbol;

else

top--;

}

while(s[top]!='#')

{

postfix[j++]=s[top--];

}

postfix[j]='\0';

}

int main()

{

char infix[20];

char postfix[20];

printf("Enter the valid infix expression\n");

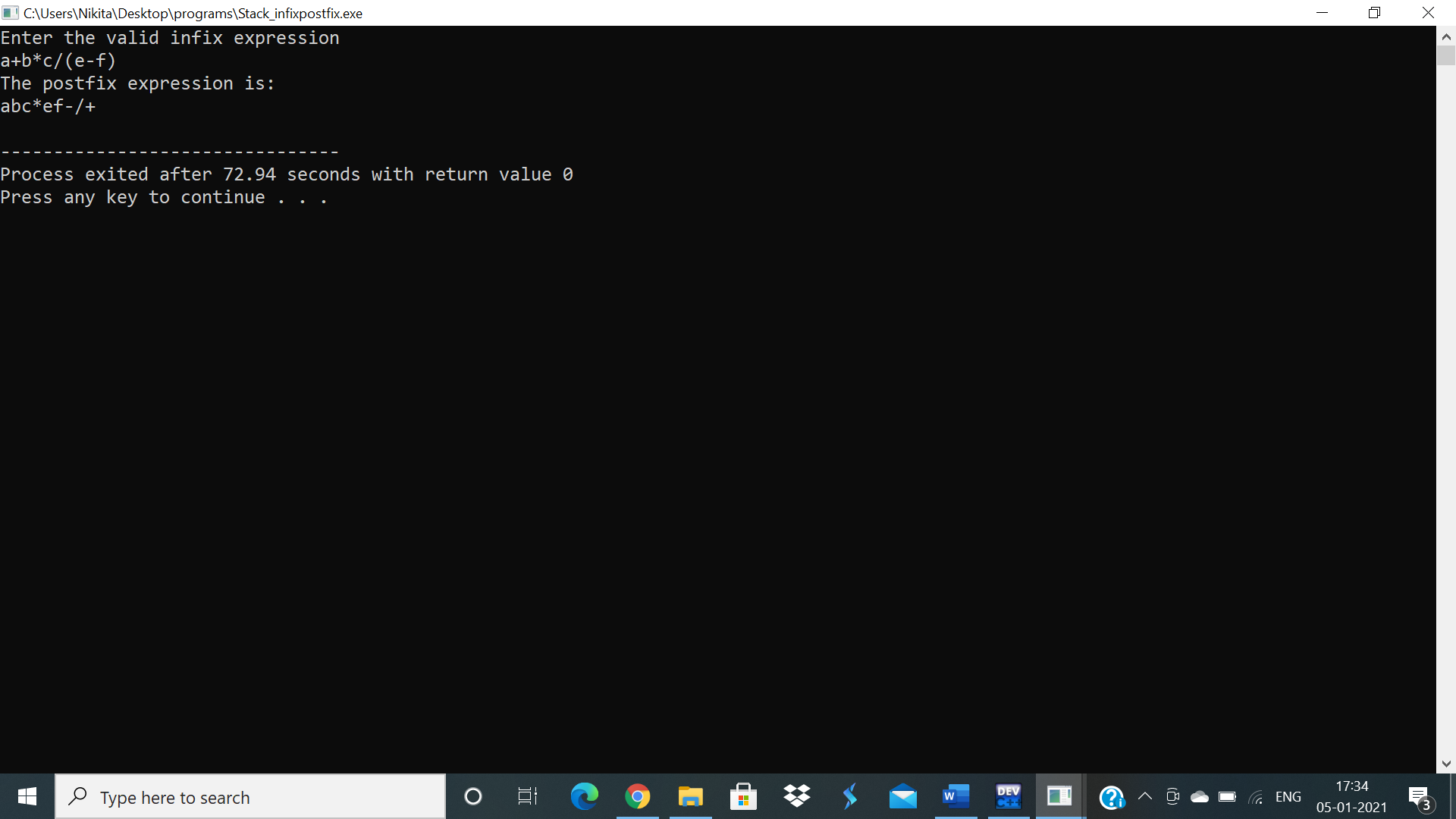
scanf("%s",infix);

infix\_postfix(infix,postfix);

printf("The postfix expression is:\n");

printf("%s\n",postfix);

}



**3 2 WAP to simulate the working of a queue of integers using an array. Provide the following operations a) Insert b) Delete c) Display The program should print appropriate messages for queue empty and queue overflow conditions.**

#include<stdio.h>

#include<stdlib.h>

#define QUE\_SIZE 5

int item,front=0,rear=-1,q[5];

void insertrear()

{

if(rear==QUE\_SIZE-1)

{

printf("Queue Overflow\n");

return;

}

rear=rear+1;

q[rear]=item;

}

int deletefront()

{

if(front>rear)

{

front=0;

rear=-1;

return -1;

}

return q[front++];

}

void displayQ()

{

int i;

if(front>rear)

{

printf("Queue is empty\n");

return;

}

printf("Contents of the queue are:\n");

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

{

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

}

}

int main()

{

int choice;

for(;;)

{

printf("\n1:Insertrear\n2:Deletefront\n3:Display\n4:Exit\n");

printf("Enter the choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1: printf("Enter the item to be inserted\n");

scanf("%d",&item);

insertrear();

break;

case 2: item=deletefront();

if(item==-1)

printf("Queue is empty\n");

else

printf("Item deleted:%d\n",item);

break;

case 3: displayQ();

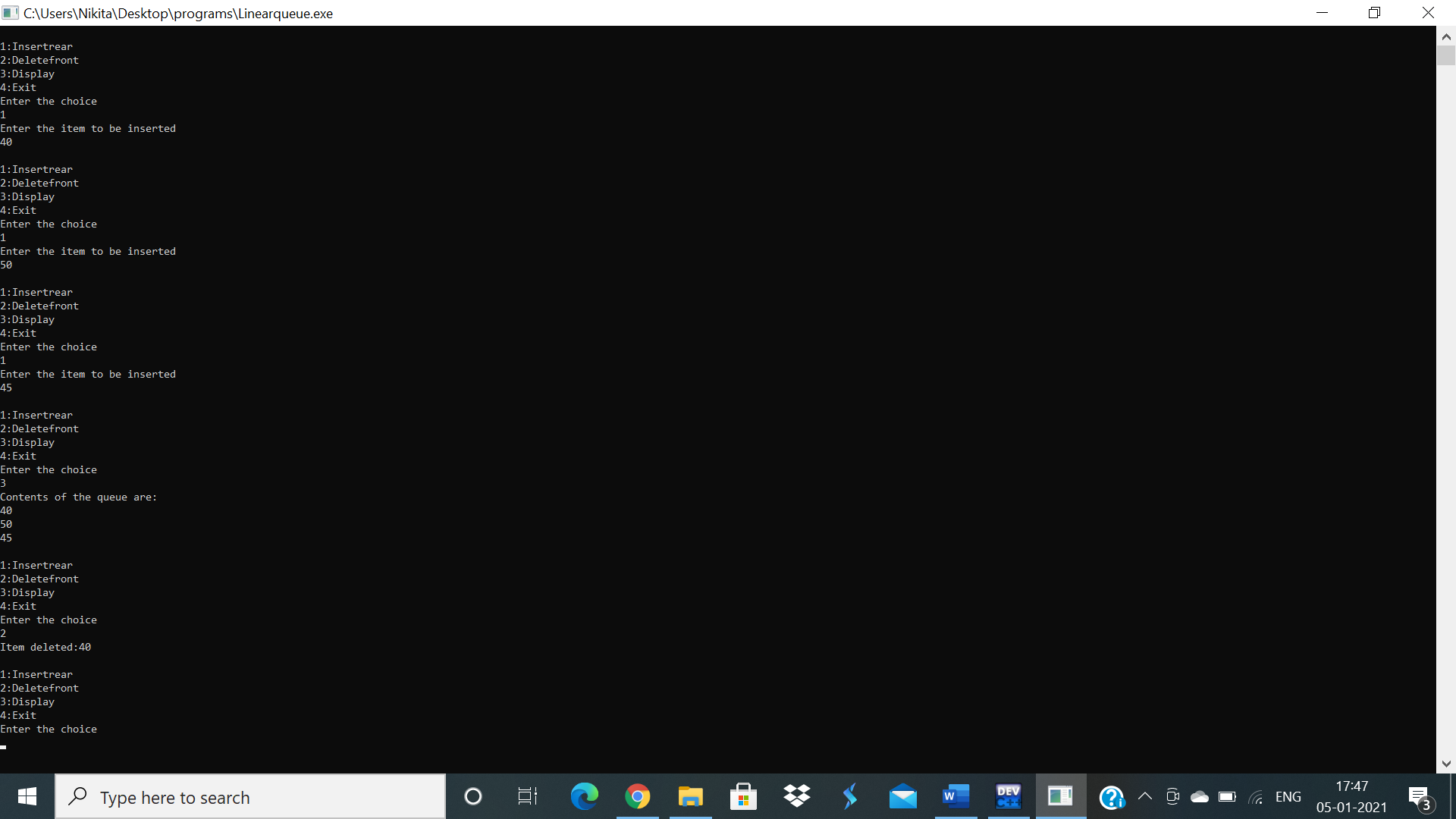
break;

default:exit(0);

}

}

}



**4 2 WAP to simulate the working of a circular queue of integers using an array. Provide the following operations. a) Insert b) Delete c) Display The program should print appropriate messages for queue empty and queue overflow conditions.**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define QUE\_SIZE 3

int item,front=0,rear=-1,q[QUE\_SIZE],count=0;

void insertrear()

{

if(count==QUE\_SIZE)

{

printf("queue overflow\n");

return;

}

rear=(rear+1)%QUE\_SIZE;

q[rear]=item;

count++;

}

int deletefront()

{

if(count==0) return -1;

item=q[front];

front=(front+1)%QUE\_SIZE;

count=count-1;

return item;

}

void displayQ()

{

int i,f;

if(count==0)

{

printf("queue is empty\n");

return;

}

f=front;

printf("Contents of queue \n");

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

{

printf("%d\n",q[f]);

f=(f+1)%QUE\_SIZE;

}

}

int main()

{

int choice;

for(;;)

{

printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");

printf("enter the choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("enter the item to be inserted\n");

scanf("%d",&item);

insertrear();

break;

case 2:item=deletefront();

if(item==-1)

printf("queue is empty\n");

else

printf("item deleted =%d\n",item);

break;

case 3:displayQ();

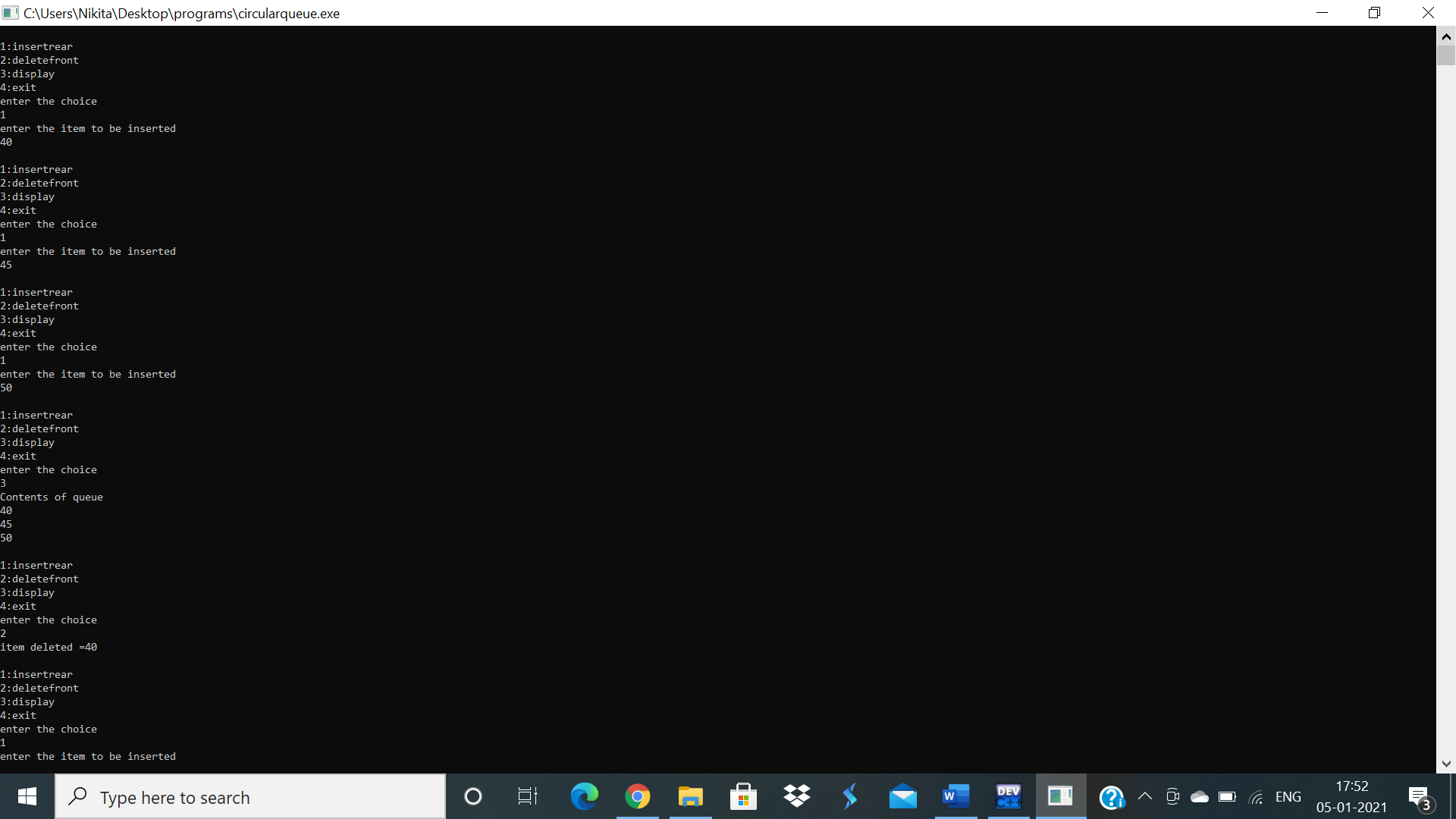
break;

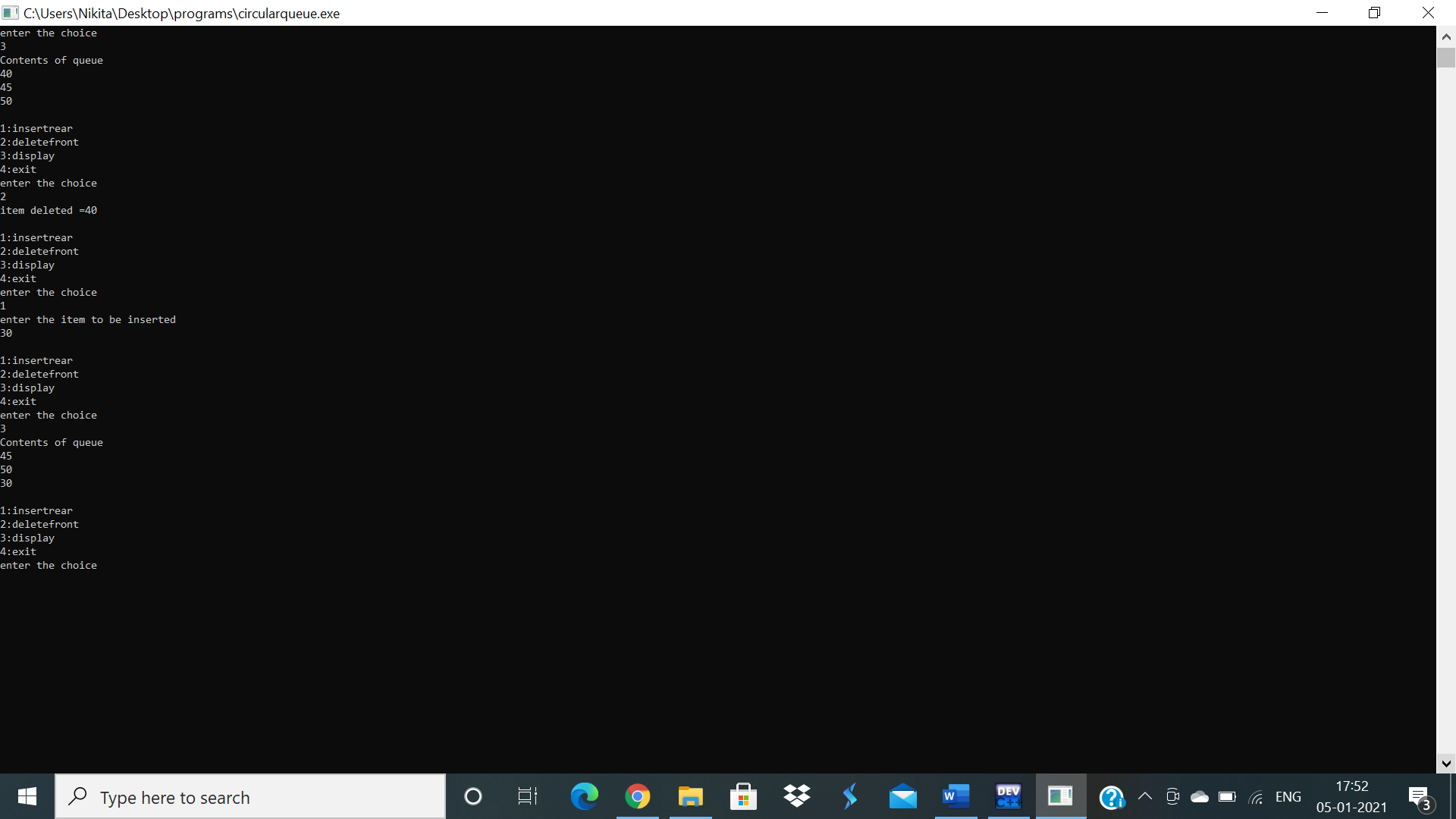
default:exit(0);

}

}

}





**5 3 WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. c) Display the contents of the linked list.**

**6 3 WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.**

**7 3 WAP Implement Single Link List with following operations a) Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists**

**8 3 WAP to implement Stack & Queues using Linked Representation**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*link;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE x;

x=(NODE)malloc(sizeof(struct node));

if(x==NULL)

{

printf("memory full\n");

exit(0);

}

return x;

}

void freenode(NODE x)

{

free(x);

}

NODE insert\_front(NODE first,int item)

{

NODE temp;

temp=getnode();

temp->info=item;

temp->link=NULL;

if(first==NULL)

return temp;

temp->link=first;

first=temp;

return first;

}

NODE insert\_rear(NODE first,int item)

{

NODE temp,cur;

temp=getnode();

temp->info=item;

temp->link=NULL;

if(first==NULL)

return temp;

cur=first;

while(cur->link!=NULL)

cur=cur->link;

cur->link=temp;

return first;

}

NODE insert\_pos(int item,int pos,NODE first)

{

NODE temp,cur,prev;

int count;

temp=getnode();

temp->info=item;

temp->link=NULL;

if (first==NULL && pos==1)

{

return temp;

}

if (first==NULL)

{

printf("Invalid position\n");

return NULL;

}

if (pos==1)

{

temp->link=first;

return temp;

}

count=1;

prev=NULL;

cur=first;

while (cur!=NULL && count!=pos)

{

prev=cur;

cur=cur->link;

count++;

}

if (count==pos)

{

prev->link=temp;

temp->link=cur;

return first;

}

printf("Invalid position\n");

return first;

}

NODE delete\_front(NODE first)

{

NODE temp;

if(first==NULL)

{

printf("list is empty cannot delete\n");

return first;

}

temp=first;

temp=temp->link;

printf("item deleted at front-end is=%d\n",first->info);

free(first);

return temp;

}

NODE delete\_rear(NODE first)

{

NODE cur,prev;

if(first==NULL)

{

printf("list is empty cannot delete\n");

return first;

}

if(first->link==NULL)

{

printf("item deleted is %d\n",first->info);

free(first);

return NULL;

}

prev=NULL;

cur=first;

while(cur->link!=NULL)

{

prev=cur;

cur=cur->link;

}

printf("item deleted at rear-end is %d",cur->info);

free(cur);

prev->link=NULL;

return first;

}

NODE delete\_pos(int pos,NODE first)

{

NODE prev,cur;

int count;

if (first==NULL || pos<=0)

{

printf("Invalid position\n");

return NULL;

}

if (pos==1)

{

cur=first;

first=first->link;

freenode(cur);

return first;

}

prev=NULL;

cur=first;

count=1;

while (cur!=NULL)

{

if (count==pos)

{

break;

}

prev=cur;

cur=cur->link;count++;

}

if (count!=pos)

{

printf("Invalid position\n");

return first;

}

prev->link=cur->link;

freenode(cur);

return first;

}

NODE concat(NODE first,NODE second)

{

NODE cur;

if(first==NULL)

return second;

if(second==NULL)

return first;

cur=first;

while(cur->link!=NULL)

cur=cur->link;

cur->link=second;

return first;

}

NODE reverse(NODE first)

{

NODE cur,temp;

cur=NULL;

while(first!=NULL)

{

temp=first;

first=first->link;

temp->link=cur;

cur=temp;

}

return cur;

}

NODE sort\_asc(NODE first)

{

int tmp;

NODE cur,next;

cur=first;

next=NULL;

if(first==NULL){

printf("List is empty\n");

}

while(cur!=NULL)

{

next=cur->link;

while(next!=NULL)

{

if(cur->info>next->info)

{

tmp=cur->info;

cur->info=next->info;

next->info=tmp;

}

next=next->link;

}

cur=cur->link;

}

return first;

}

void display(NODE first)

{

NODE temp;

if(first==NULL)

printf("list empty cannot display items\n");

for(temp=first;temp!=NULL;temp=temp->link)

{

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

}

}

NODE push(NODE first,int item)

{

NODE temp;

temp=getnode();

temp->info=item;

temp->link=NULL;

if(first==NULL)

return temp;

temp->link=first;

first=temp;

return first;

}

NODE pop(NODE first)

{

NODE temp;

if(first==NULL)

{

printf("list is empty cannot delete\n");

return first;

}

temp=first;

temp=temp->link;

printf("item deleted at front-end is=%d\n",first->info);

free(first);

return temp;

}

NODE enqueue(NODE first,int item)

{

NODE temp;

temp=getnode();

temp->info=item;

temp->link=NULL;

if(first==NULL)

return temp;

temp->link=first;

first=temp;

return first;

}

NODE dequeue(NODE first)

{

NODE temp;

if(first==NULL)

{

printf("list is empty cannot delete\n");

return first;

}

temp=first;

temp=temp->link;

printf("item deleted at front-end is=%d\n",first->info);

free(first);

return temp;

}

int main()

{

int item,choice,pos,n1,i,n2,n,flag=0;

NODE first=NULL,a,b;

for(;;)

{

printf("\n 1:Insert\_front\n 2:Delete\_front\n 3:Insert\_rear\n 4:Delete\_rear\n5:Insert Position\n6:Delete Position\n7:Concat\n8:Reverse\n9:Sort\n10:Display\n11:Stack operations\n12:Queue Operations\n");

printf("enter the choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("enter the item at front-end\n");

scanf("%d",&item);

first=insert\_front(first,item);

break;

case 2:first=delete\_front(first);

break;

case 3:printf("enter the item at rear-end\n");

scanf("%d",&item);

first=insert\_rear(first,item);

break;

case 4:first=delete\_rear(first);

break;

case 5:printf("Enter the item and the position:\n");

scanf("%d%d",&item,&pos);

first=insert\_pos(item,pos,first);

break;

case 6:printf("Enter the position:\n");

scanf("%d",&pos);

first=delete\_pos(pos,first);

break;

case 7:printf("enter the no of nodes in 1\n");

scanf("%d",&n);

a=NULL;

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

{

printf("enter the item\n");

scanf("%d",&item);

a=insert\_rear(a,item);

}

printf("enter the no of nodes in 2\n");

scanf("%d",&n);

b=NULL;

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

{

printf("enter the item\n");

scanf("%d",&item);

b=insert\_rear(b,item);

}

a=concat(a,b);

display(a);

break;

case 8:first=reverse(first);

display(first);

break;

case 9:first=sort\_asc(first);

break;

case 10:display(first);

break;

case 11:

do

{

printf("\n1:Push\n2:Pop\n3:Display\n");

printf("Enter your choice\n");

scanf("%d",&n);

switch(n)

{

case 1: printf("enter the item to be inserted in the stack\n");

scanf("%d",&item);

first=insert\_front(first,item);

break;

case 2: first=delete\_front(first);

break;

case 3: display(first);

}

}while(choice==11);

break;

case 12:

do

{

printf("\n1.Insert\n2:Delete\n3:Display\n");

printf("Enter your choice\n");

scanf("%d",&n1);

switch(n1)

{

case 1: printf("enter the item to be inserted in the queue\n");

scanf("%d",&item);

first=insert\_rear(first,item);

break;

case 2: first=delete\_front(first);

break;

case 3: display(first);

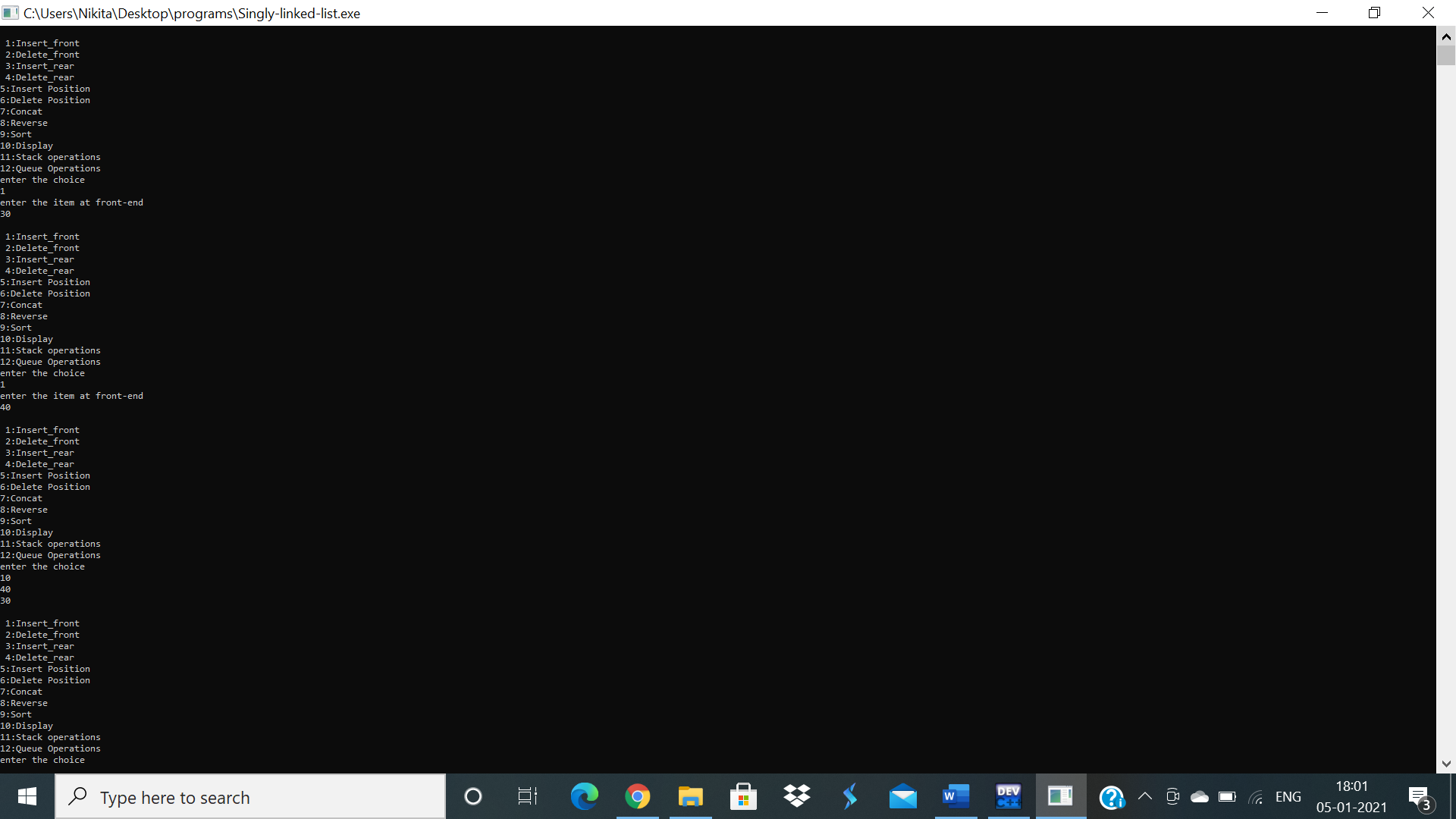
}

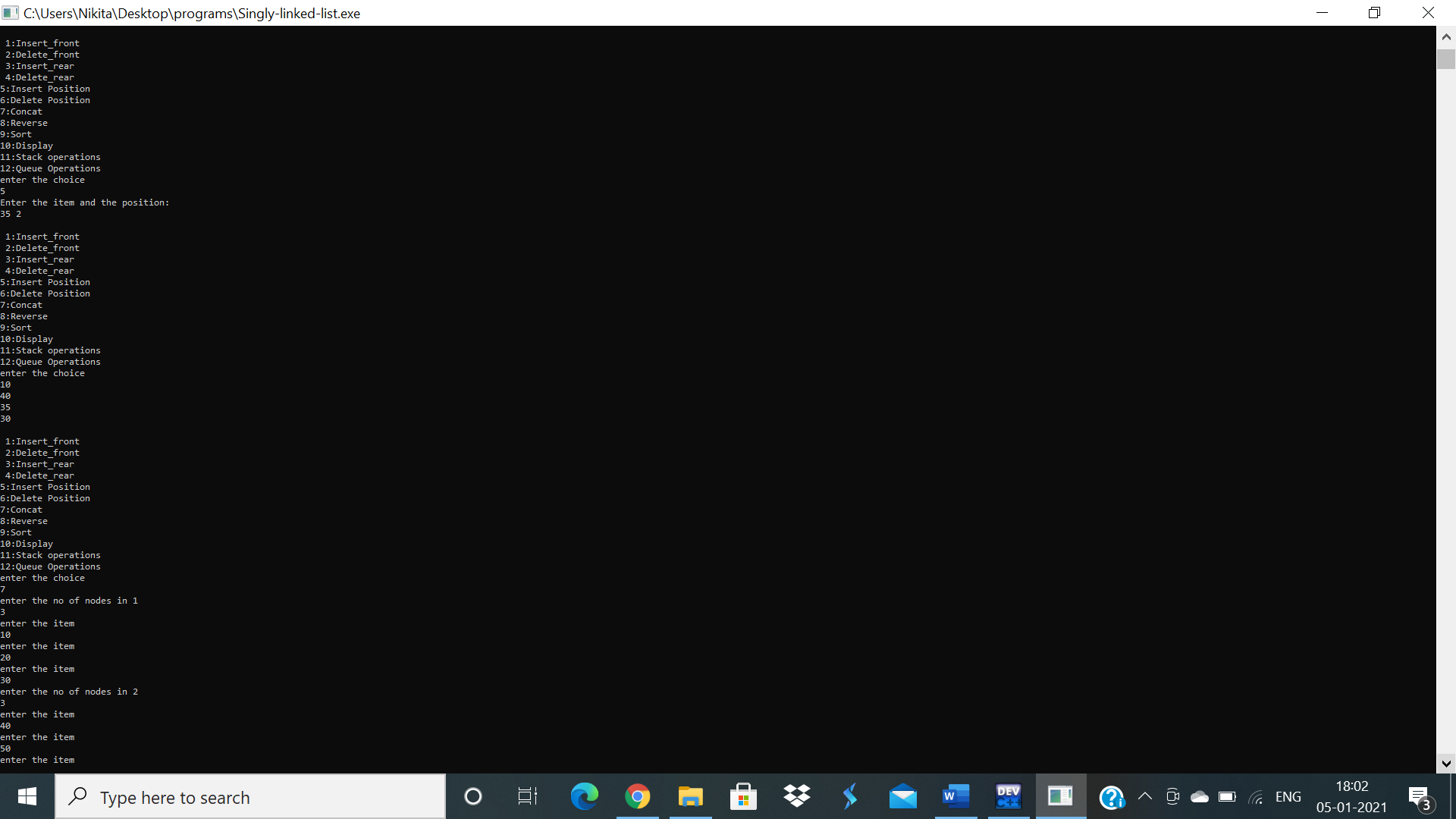
}while(choice==12);

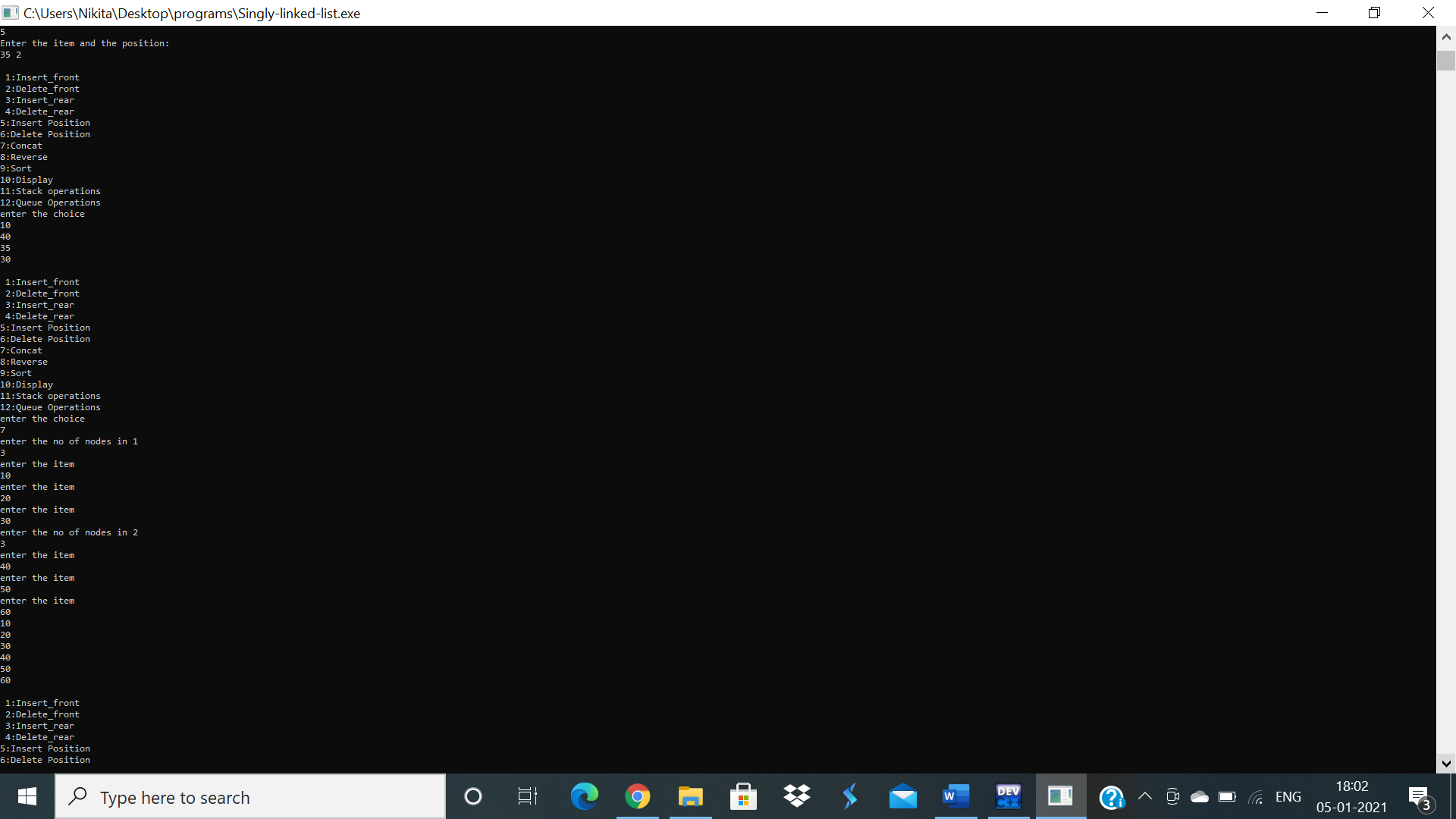
break;

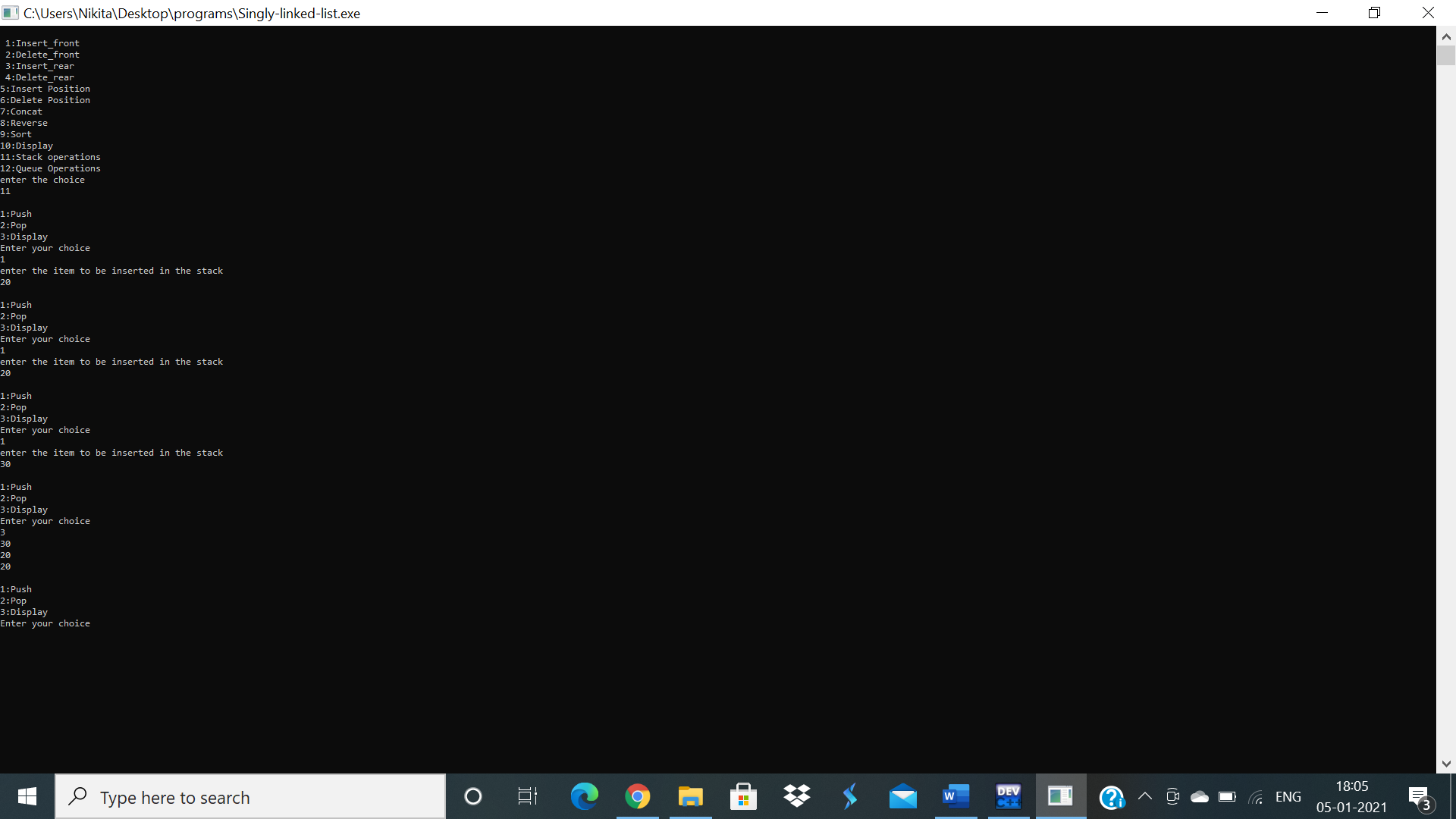
}}

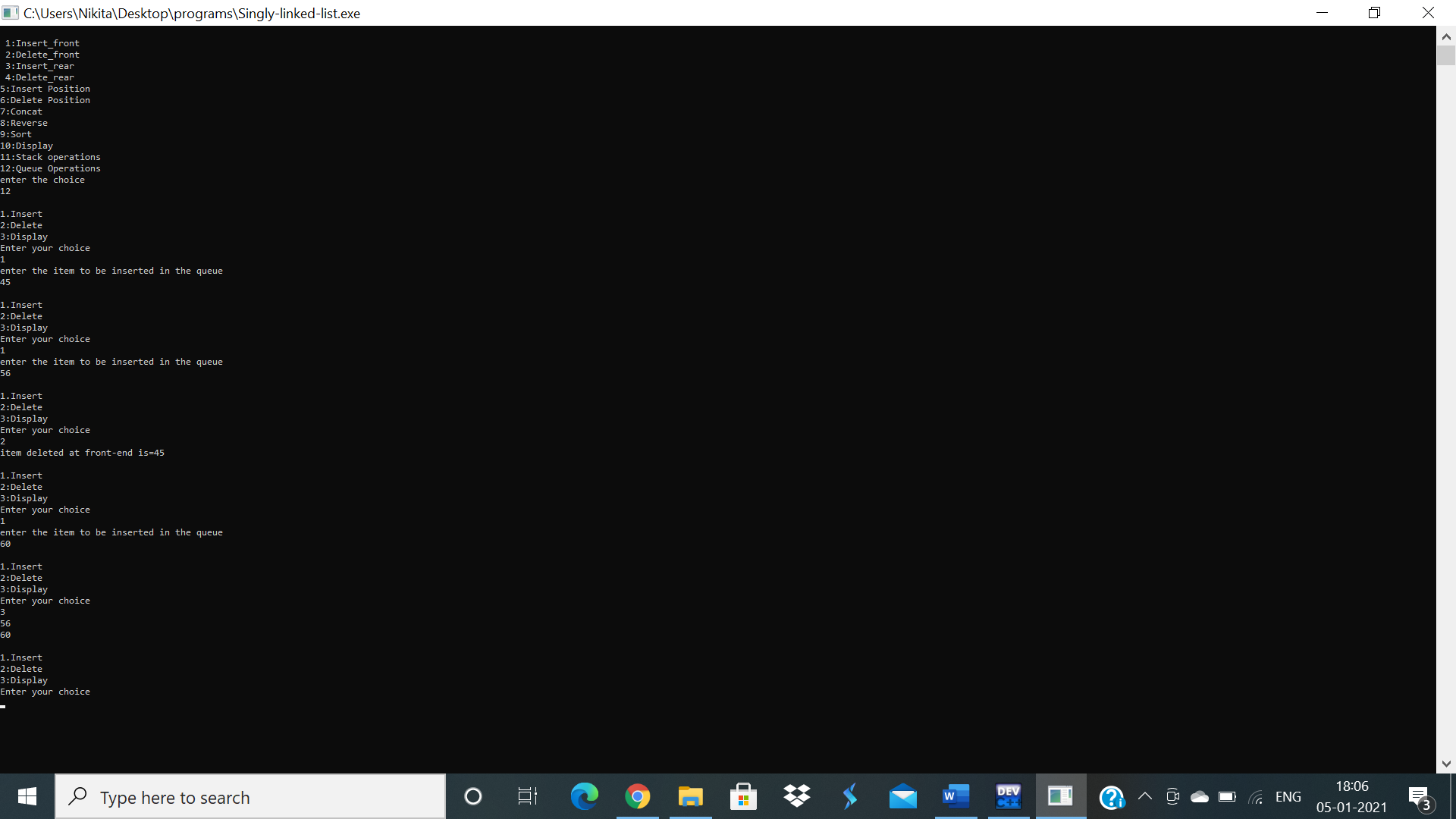
}











**9 4 WAP Implement doubly link list with primitive operations a) Create a doubly linked list. b) Insert a new node to the left of the node. c) Delete the node based on a specific value d) Display the contents of the list.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*rlink;

struct node \*llink;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE x;

x=(NODE)malloc(sizeof(struct node));

if (x==NULL)

{

printf("Memory full\n");

exit(0);

}

return x;

}

NODE dinsert\_front(int item,NODE head)

{

NODE temp,cur;

temp=getnode();

temp->info=item;

temp->llink=NULL;

temp->rlink=NULL;

cur=head->rlink;

head->rlink=temp;

temp->llink=head;

temp->rlink=cur;

cur->llink=temp;

return head;

}

NODE dinsert\_rear(int item,NODE head)

{

NODE temp,cur;

temp=getnode();

temp->info=item;

temp->llink=NULL;

temp->rlink=NULL;

cur=head->llink;

head->llink=temp;

temp->rlink=head;

cur->rlink=temp;

temp->llink=cur;

return head;

}

NODE ddelete\_front(NODE head)

{

NODE cur,next;

if (head->rlink==head)

{

printf("List is empty\n");

return head;

}

cur=head->rlink;

next=cur->rlink;

head->rlink=next;

next->llink=head;

printf("Item deleted at the front end is:%d\n",cur->info);

free(cur);

return head;

}

NODE ddelete\_rear(NODE head)

{

NODE cur,prev;

if (head->rlink==head)

{

printf("List is empty\n");

return head;

}

cur=head->llink;

prev=cur->llink;

prev->rlink=head;

head->llink=prev;

printf("Item deleted at the rear end is:%d\n",cur->info);

free(cur);

return head;

}

void ddisplay(NODE head)

{

NODE temp;

if (head->rlink==head)

{

printf("List is empty\n");

}

printf("The contents of the list are:\n");

temp=head->rlink;

while (temp!=head)

{

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

temp=temp->rlink;

}

}

NODE dinsert\_leftpos(int item,NODE head)

{

NODE cur,prev,temp;

if (head->rlink==head)

{

printf("List is empty\n");

return head;

}

cur=head->rlink;

while (cur!=head)

{

if (cur->info==item)

{

break;

}

cur=cur->rlink;

}

if (cur==head)

{

printf("No such item found in the list\n");

return head;

}

prev=cur->llink;

temp=getnode();

temp->llink=NULL;

temp->rlink=NULL;

printf("Enter the item to be inserted at the left of the given item:\n");

scanf("%d",&temp->info);

prev->rlink=temp;

temp->llink=prev;

temp->rlink=cur;

cur->llink=temp;

return head;

}

int main()

{

NODE head;

int item, choice,key;

head=getnode();

head->llink=head;

head->rlink=head;

for(;;)

{

printf("\n1:dinsert front\n2:dinsert rear\n3:ddelete front\n4:ddelete rear\n5:ddisplay\n6:dinsert lestpos\n7:exit\n");

printf("enter the choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1: printf("Enter the item at front end:\n");

scanf("%d",&item);

head=dinsert\_front(item,head);

break;

case 2: printf("Enter the item at rear end:\n");

scanf("%d",&item);

head=dinsert\_rear(item,head);

break;

case 3:head=ddelete\_front(head);

break;

case 4:head=ddelete\_rear(head);

break;

case 5:ddisplay(head);

break;

case 6:printf("Enter the key element:\n");

scanf("%d",&key);

head=dinsert\_leftpos(key,head);

break;

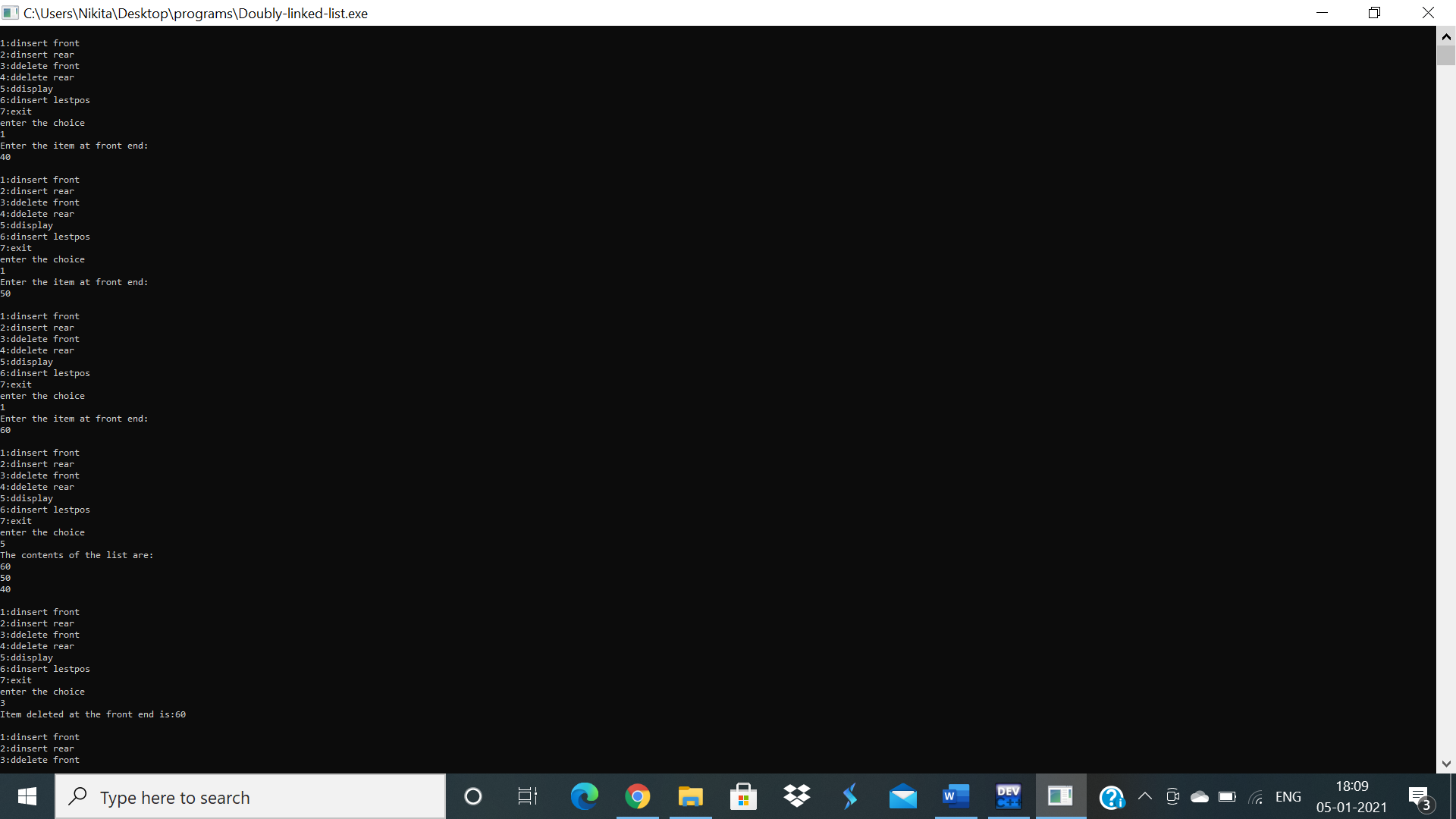
default:exit(0);

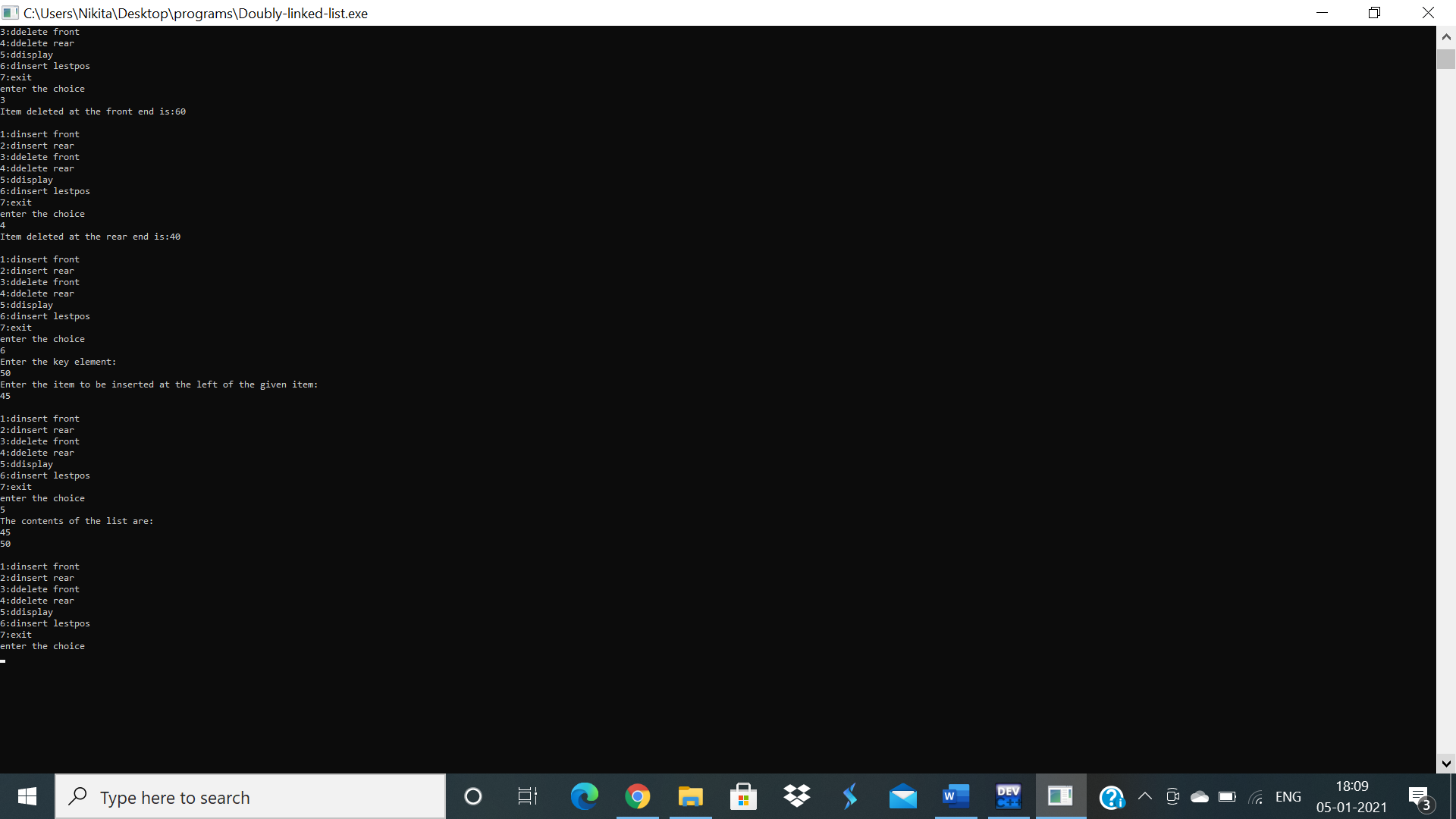
}

}

return 0;

}





**10 5 Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in-order, preorder and post order c) To display the elements in the tree.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node \*rlink;

struct node \*llink;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE x;

x=(NODE)malloc(sizeof(struct node));

if(x==NULL)

{

printf("memory full\n");

exit(0);

}

return x;

}

void freenode(NODE x)

{

free(x);

}

NODE insert(NODE root,int item)

{

NODE temp,cur,prev;

temp=getnode();

temp->rlink=NULL;

temp->llink=NULL;

temp->info=item;

if(root==NULL)

return temp;

prev=NULL;

cur=root;

while(cur!=NULL)

{

prev=cur;

cur=(item<cur->info)?cur->llink:cur->rlink;

}

if(item<prev->info)

prev->llink=temp;

else

prev->rlink=temp;

return root;

}

void display(NODE root,int i)

{

int j;

if(root!=NULL)

{

display(root->rlink,i+1);

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

printf(" ");

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

display(root->llink,i+1);

}

}

void preorder(NODE root)

{

if(root!=NULL)

{

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

preorder(root->llink);

preorder(root->rlink);

}

}

void postorder(NODE root)

{

if(root!=NULL)

{

postorder(root->llink);

postorder(root->rlink);

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

}

}

void inorder(NODE root)

{

if(root!=NULL)

{

inorder(root->llink);

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

inorder(root->rlink);

}

}

int main()

{

int item,choice;

NODE root=NULL;

for(;;)

{

printf("\n1.insert\n2.display\n3.Preorder\n4.Postorder\n5.Inorder\n6.exit\n");

printf("enter the choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("enter the item\n");

scanf("%d",&item);

root=insert(root,item);

break;

case 2:display(root,0);

break;

case 3:preorder(root);

break;

case 4:postorder(root);

break;

case 5:inorder(root);

break;

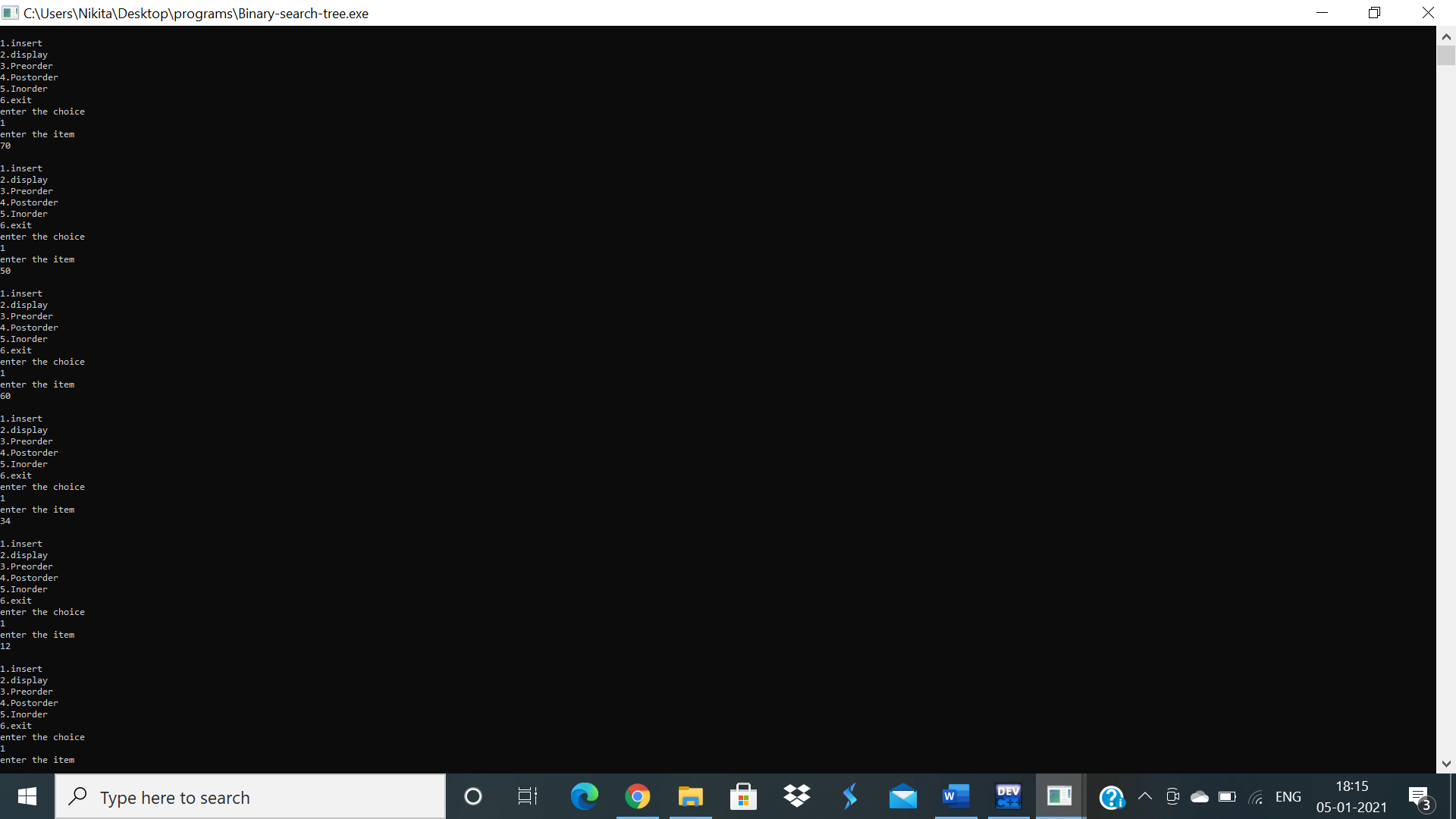
default:exit(0);

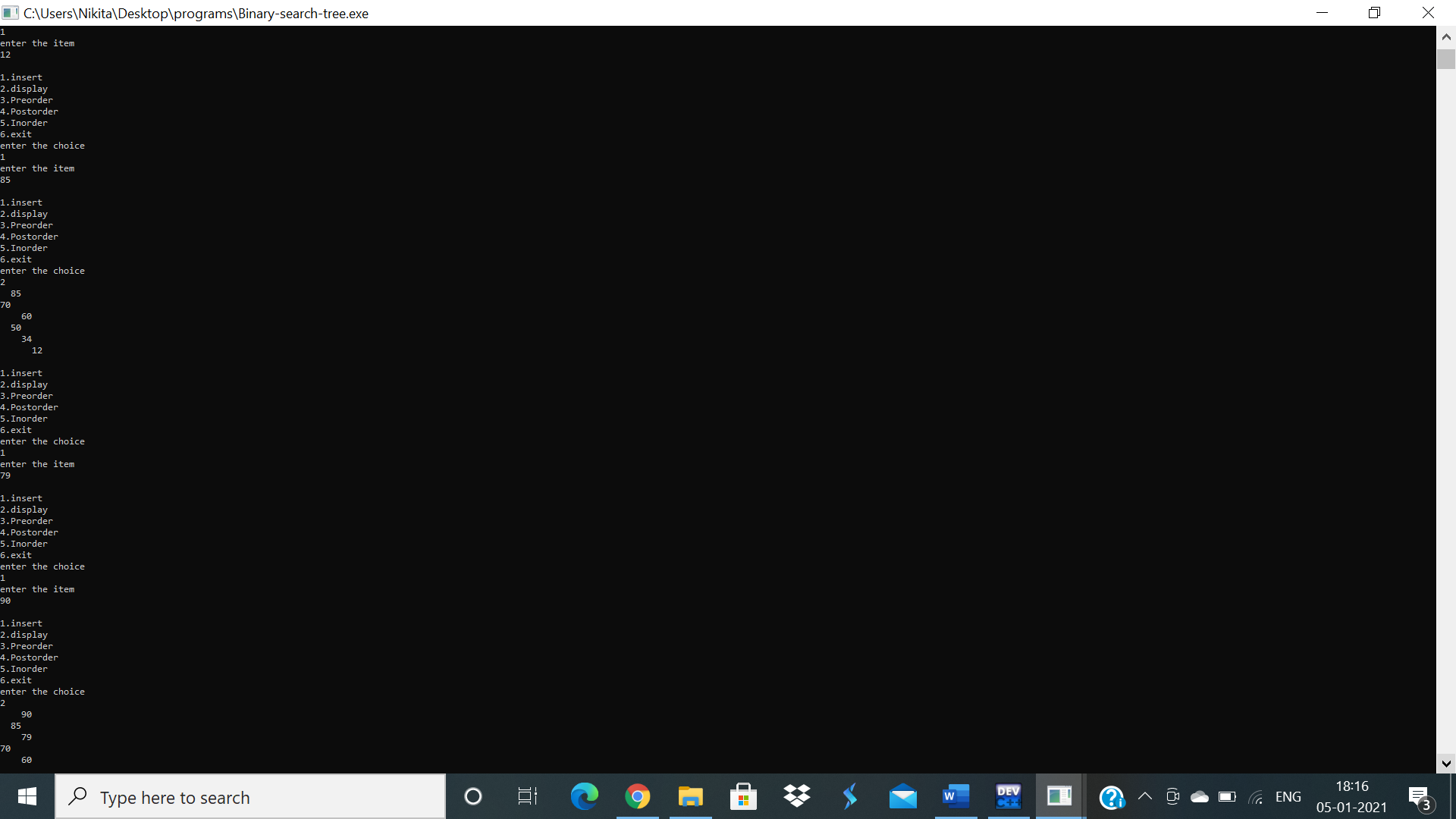
break;

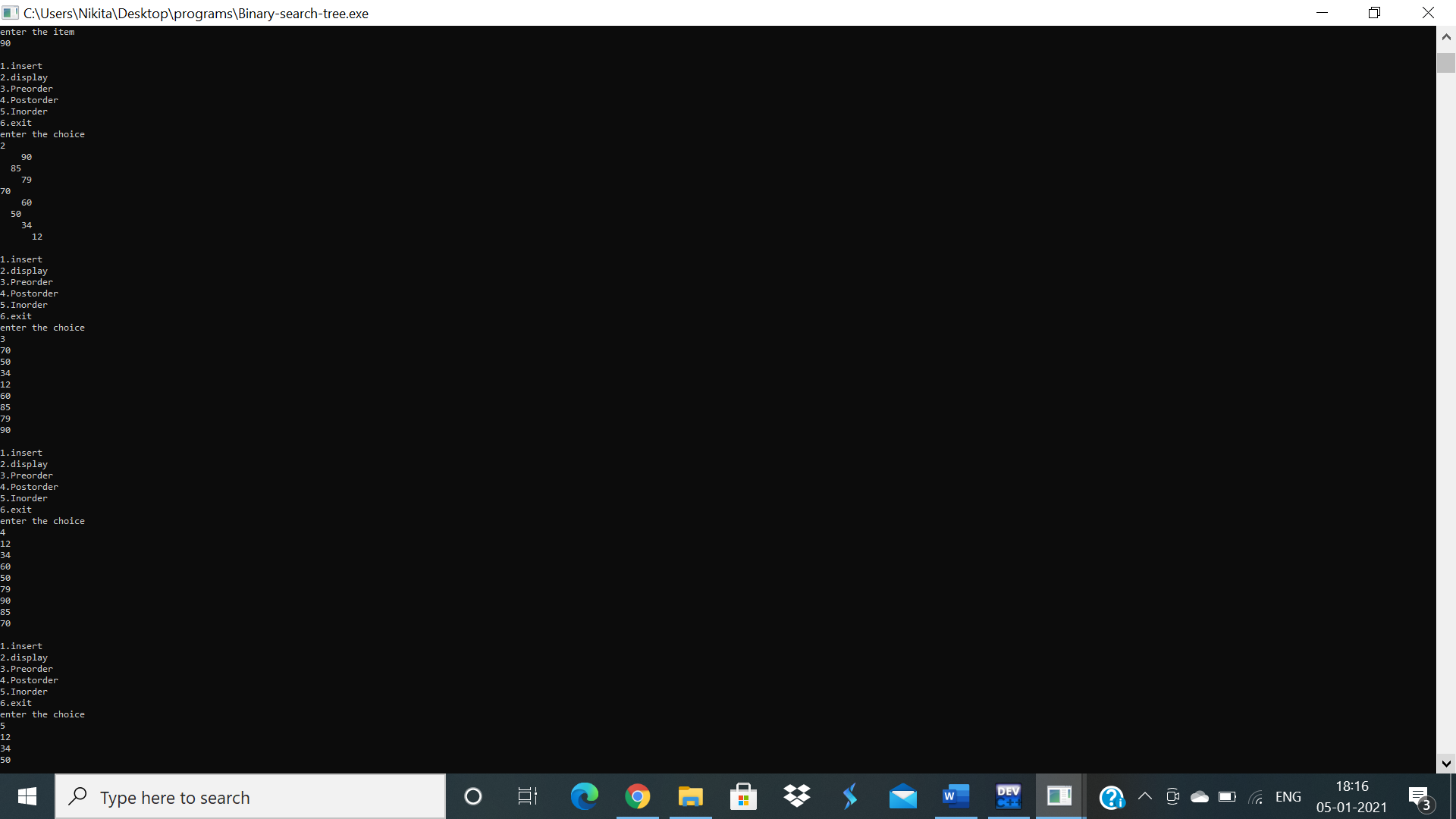
}

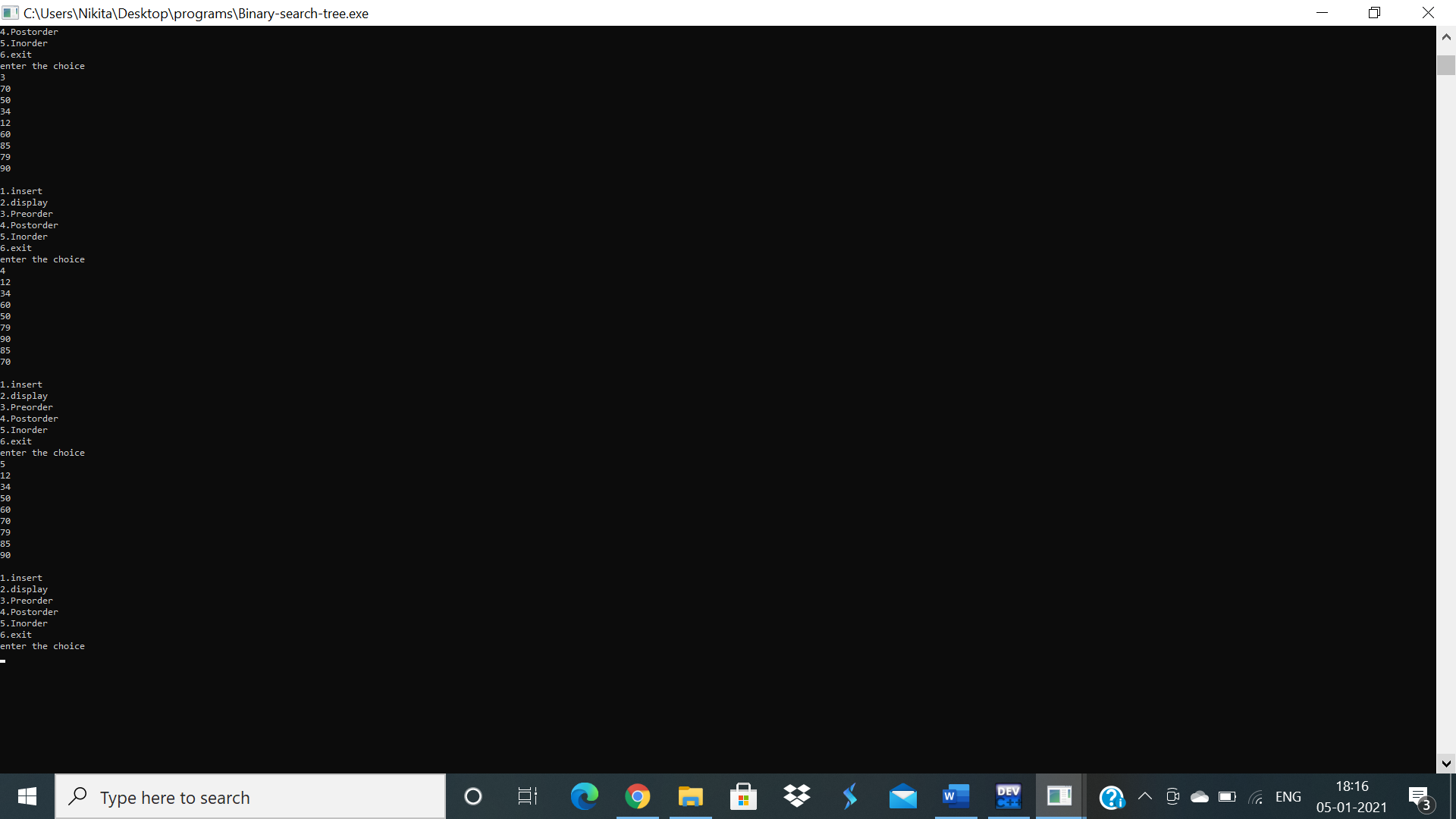
}

}









**THANK YOU**

**DONE BY:**

**Nikita Paras Toliya**

**1BM19CS103**