DATA STRUCTURE LAB MANUAL

Name:P.Revathi

Id number:S170286

Class:CSE-1E

1.Single linkedlist operations

#include"stdio.h"

#include"stdlib.h"

int length();

struct node

{

int data;

struct node\*link;

};

struct node\*root=NULL;

void append()

{

struct node\*root,\*temp,\*q;

q=root;

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

printf("enter the node data:");

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

temp->link=NULL;

if(root==NULL)

{

root=temp;

}

while(q->link!=NULL)

{

q=q->link;

}

q->link=temp;

}

int inatbe()

{

struct node\*temp;

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

printf("enter your node data:");

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

temp->link=root;

root=temp;

}

int insert()

{

struct node\*temp;

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

printf("enter your node data:");

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

temp->link=NULL;

int loc,len,i=1;

printf("enter insert location:");

scanf("%d",&loc);

len=length();

if(loc>len)

{

printf("invalid location\npresent list have %d nodes\n",len);

}

else

{ struct node \*p;

p=root;

while(i<loc)

{

p=p->link;

i++;

}

temp->link=p->link;

p->link=temp;

}

}

int display()

{

struct node\*temp;

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

temp=root;

if(temp==NULL)

{

printf("\nno nodes are avilable\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->link;

}

}

}

int length()

{

int len;

struct node \*temp;

temp=root;

while(temp!=NULL)

{

len++;

temp=temp->link;

}

return len;

}

int Delete()

{

int loc,len=length(),x=1;

struct node \*k=root,\*y;

printf("enter location:");

scanf("%d",&loc);

if(loc>len)

{

printf("invalid location\n");

}

else if(loc==1)

{

root=k->link;

k->link=NULL;

free(k);

}

else

{

while(x<loc-1)

{

k=k->link;

x++;

}

y=k->link;

k->link=y->link;

y->link=NULL;

free(y);

}

}

int Reverse()

{

struct node\*root,\*curr,\*prev,\*next,\*temp;

curr=next=root;

prev=0;

while(next!=0)

{

next=curr->link;

curr->link=prev;

prev=curr;

curr=next;

}

root=prev;

temp=root;

if(temp==NULL)

{

printf("\nno nodes are avilable\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->link;

}

}

}

int main()

{ int ch,k;

printf("single linked list operations\n\n");

printf("1.append\n2.inatbe\n3.insert\n4.length\n5.display\n6.delete\n7.reverse\n8.quit\n");

while(1)

{

printf("\nchoose your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

append();

break;

case 2:

inatbe();

break;

case 3:

insert();

break;

case 4:

k=length();

printf("%d",k);

break;

case 5:

display();

break;

case 6:

Delete();

break;

case 7:

Reverse();

break;

case 8:

exit(1);

break;

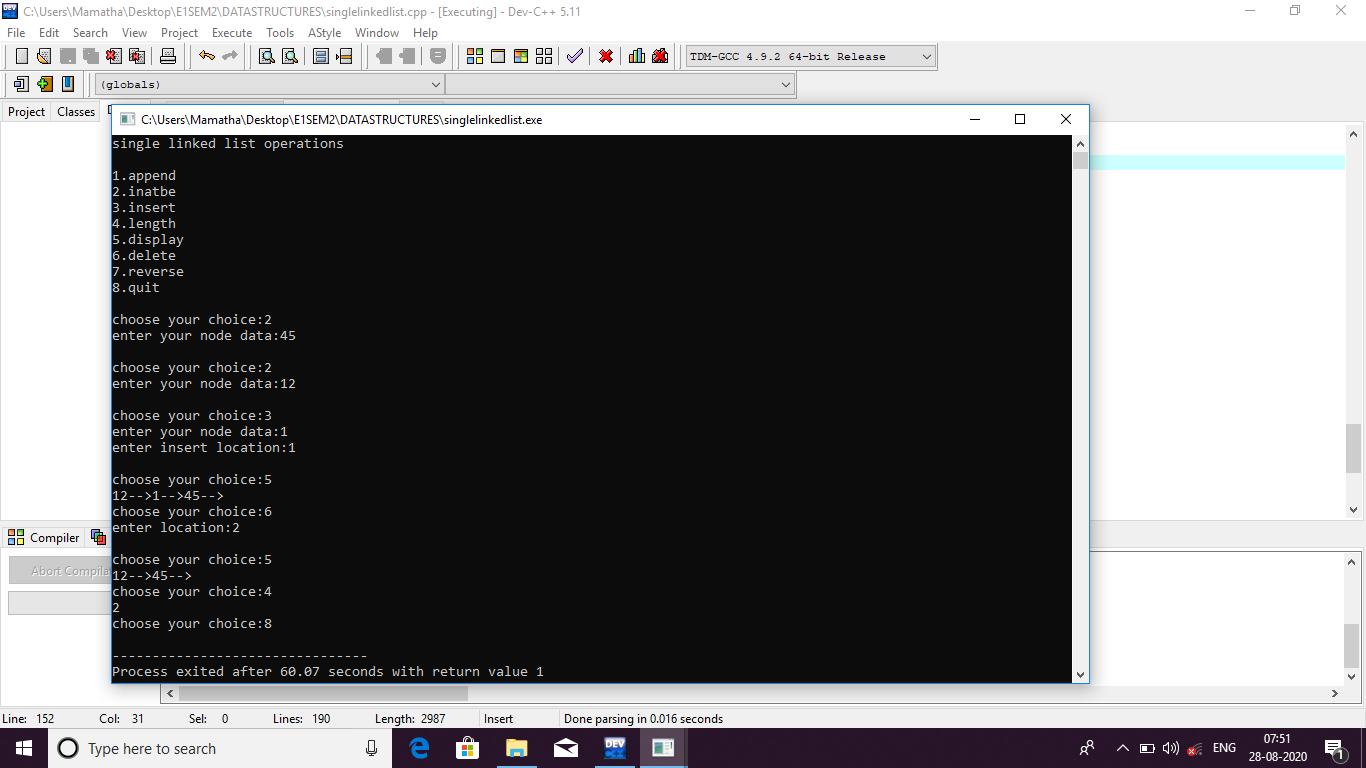
default :

printf("\nyour choice is invalid\n");

}

}

}



2.Double linked list operations

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*left;

struct node\*right;

};

struct node \*root=NULL;

void append()

{

struct node\*temp,\*p;

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

printf("enter node data:");

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

temp->right=NULL;

temp->left=NULL;

if(root==NULL)

{

root=temp;

}

else

while(p->right!=NULL)

{

p=p->right;

}

p->right=temp;

temp->left=p;

}

int length()

{ int count=0;

struct node \*p;

p=root;

while(p!=NULL)

{

count++;

p=p->right;

}

return count;

}

void addatbegin()

{

struct node \*temp;

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

printf("enter the node data:");

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

temp->right=temp->left=NULL;

if(root==NULL)

{

root=temp;

}

else

temp->right=root;

root->left=temp;

root=temp;

}

int addafter()

{

int loc,i=1;

struct node \*temp,\*p=root;

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

printf("enter the node data:");

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

temp->right=temp->left=NULL;

printf("enter the location:");

scanf("%d",&loc);

if(loc>length())

{

printf("INVALID location\n");

}

else

{

while(i<loc)

{

p=p->right;

i++;

}

temp->right=p->right;

p->right->left=temp;

temp->left=p;

p->right=temp;

}

}

void Delete()

{

int loc,i=1;

struct node\*p=root,\*q;

printf("enter location:");

scanf("%d",&loc);

if(loc>length())

{

printf("INVALID location\n");

}

else if(loc==1)

{

root->left=NULL;

root=p->right;

free(p);

}

else if(loc==length())

{

while(p->right!=NULL)

{

p=p->right;

}

p->left->right=NULL;

p->left=NULL;

free(p);

}

else

while(i<loc-1)

{

p=p->right;

i++;

}

q=p->right;

p->right=q->right;

q->right->left=p;

free(q);

}

void display()

{

struct node\*p=root;

if(root==NULL)

{

printf("no nodes to display..\n");

}

else

while(p!=NULL)

{

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

p=p->right;

}

}

int main()

{

int ch,k;

printf("double linked list operatations...");

printf("\n1.append\n2.add at begin\n3.add after\n4.display\n5.delete\n6.reverse\n7.length\n8.quit\n");

while(1)

{

printf("enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

append();

break;

case 2:

addatbegin();

break;

case 3:

addafter();

break;

case 4:

display();

break;

case 5:

Delete();

break;

case 7:

k=length();

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

break;

case 8:

exit(1);

break;

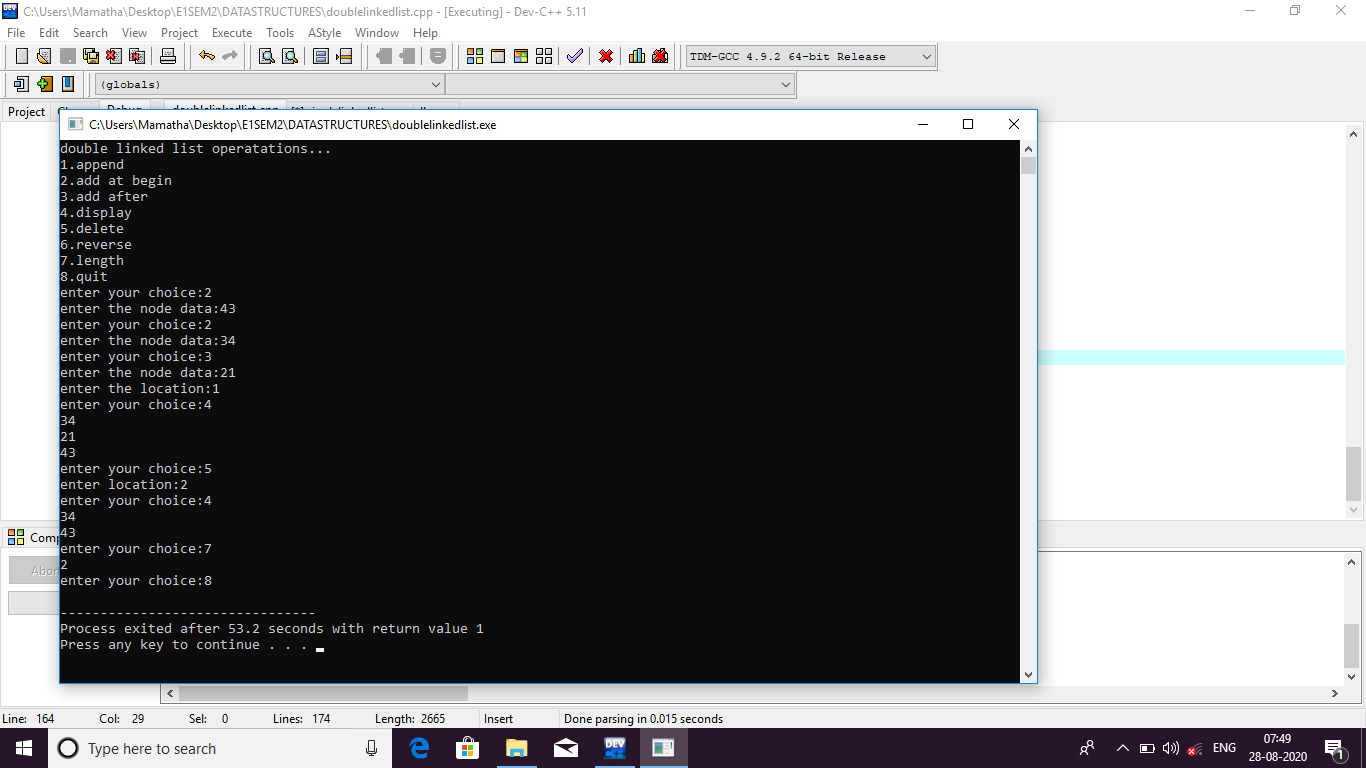
default:

printf("enter another choice..");

}

}

}



3.a)Stack ADT using array:

#include<stdio.h>

#include<stdlib.h>

#define CAPACITY 5

int stack[CAPACITY],top=-1;

void push(int ele)

{

if(top==CAPACITY-1)

printf("stack is overflow..\n");

else

top++;

stack[top]=ele;

}

void pop()

{

if(top==-1)

printf("stact is undeflow..\n");

else

top--;

}

void peek()

{

if(top==-1)

printf("stack have no elements..\n");

else

printf("stack[%d]=%d\n",top,stack[top]);

}

void traverse()

{

int i;

if(top==-1)

printf("stack have no elements..\n");

else

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

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

}

void size()

{

int i=0,count=0;

while(i!=top)

{

i++;

count=count+1;

}

printf("size of the stack is %d\n",count+1);

}

int main()

{

int ch,ele;

printf("\t\t======STACK OPERATIONS======");

printf("\n1.Push\n2.Pop\n3.Peek\n4.Traverse\n5.Size\n6.Quit\n");

while(1)

{

printf("enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("enter element:");

scanf("%d",&ele);

push(ele);

break;

case 2:

pop();

break;

case 3:

peek();

break;

case 4:

traverse();

break;

case 5:

size();

break;

case 6:

exit(0);

break;

default:

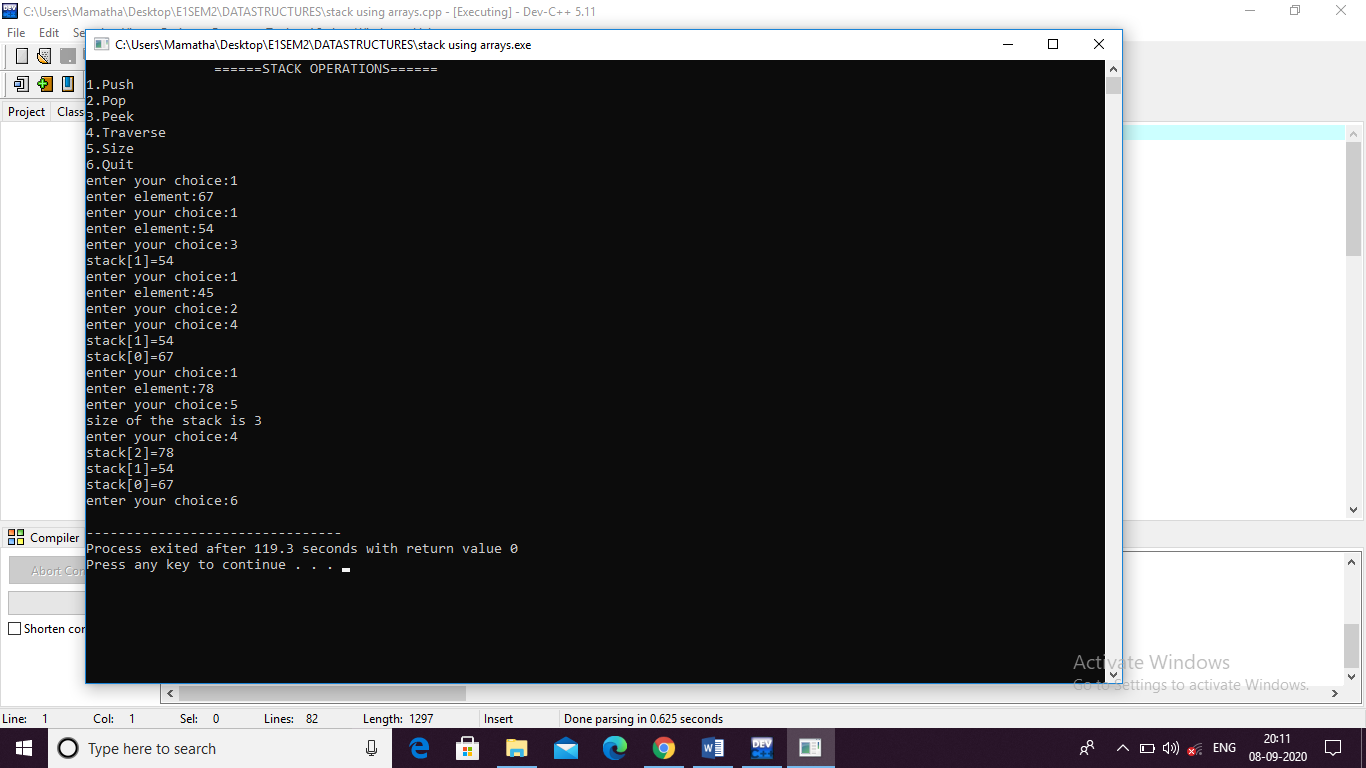
printf("invalid response....\n");

}

}

return 0;

}



3.b)Stack ADT Using Linkedlist

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*link;

};

struct node\*root=NULL;

struct node\*base=NULL;

int push()

{

struct node\*temp;

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

printf("enter your element:");

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

temp->link=NULL;

if(root==NULL)

{

base=root=temp;

}

else

{

root->link=temp;

root=temp;

}

}

int pop()

{

struct node \*temp,\*p;

temp=root;

p=base;

if(root==NULL)

{

printf("stack is empty");

}

else

{

while(p->link!=root)

{

p=p->link;

}

root=p;

p->link=NULL;

free(temp);

}

}

int peek()

{

struct node \*temp;

temp=root;

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

}

int traverse()

{

struct node \*temp;

temp=base;

while(temp->link!=NULL)

{

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

temp=temp->link;

}

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

}

int size()

{

struct node \*temp;

temp=base;

int i;

while(temp->link!=NULL)

{

temp=temp->link;

i++;

}

printf("%d",i+1);

}

int main()

{

int ch,ele;

printf("\t\t======STACK OPERATIONS======");

printf("\n1.Push\n2.Pop\n3.Peek\n4.Traverse\n5.Size\n6.Quit\n");

while(1)

{

printf("\nenter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

peek();

break;

case 4:

traverse();

break;

case 5:

size();

break;

case 6:

exit(0);

break;

default:

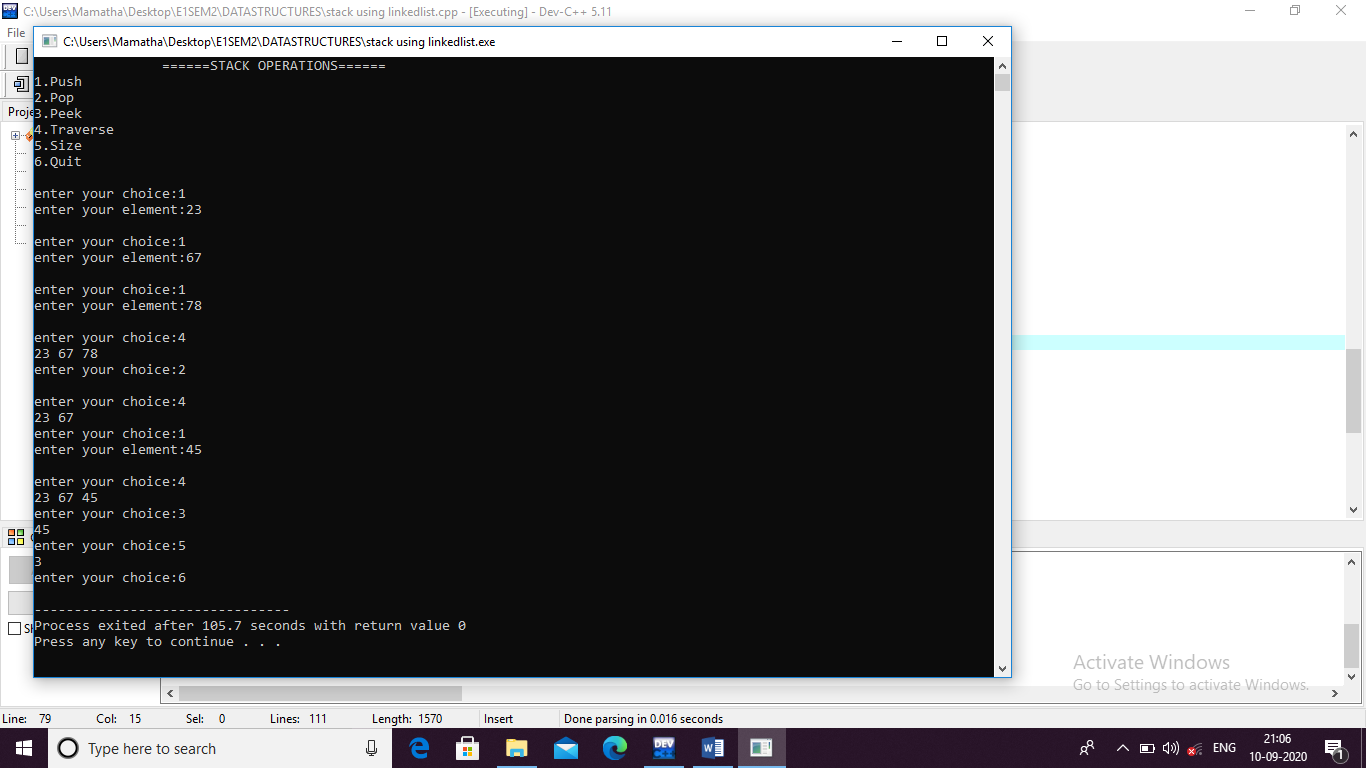
printf("invalid response....\n");

}

}

return 0;

}



4.Expression Convertion of infix to postfix using stack

#include<stdio.h>

#include<string.h>

#define MAX 50

int top=-1;

char stack[MAX];

void push(char item)

{

if(top>MAX-1)

{

printf("stack is overflow..");

}

else

{

stack[++top]=item;

}

}

char pop()

{

char item;

item=stack[top];

top--;

return item;

}

int is\_operator(char item)

{

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

{

return 1;

}

else

{

return 0;

}

}

int precedence(char item)

{

if(item=='^')

return 3;

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

return 2;

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

return 1;

}

int main()

{

char infix[MAX],postfix[MAX],item,temp;

int i=0,j=0;

printf("enter your infix expression:");

gets(infix);

while((item=infix[i])!='\0')

{

if(item=='(')

{

push(item);

}

else if((item>='A'&&item<='Z')||(item>='a'&&item<='z'))

{

postfix[j]=item;

j++;

}

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

{

temp=pop();

while(is\_operator(item)==1&&precedence(item)<=precedence(temp))

{

postfix[j]=temp;

j++;

temp=pop();

}

push(temp);

push(item);

}

else if(item==')')

{

temp=pop();

while(temp!='(')

{

postfix[j]=temp;

j++;

temp=pop();

}

}

else

{

printf("Invalid Expression\n");

printf("postfix expression is not possible...and the operators is");

}

i++;

}

while(top>-1)

{

postfix[j]=pop();

j++;

}

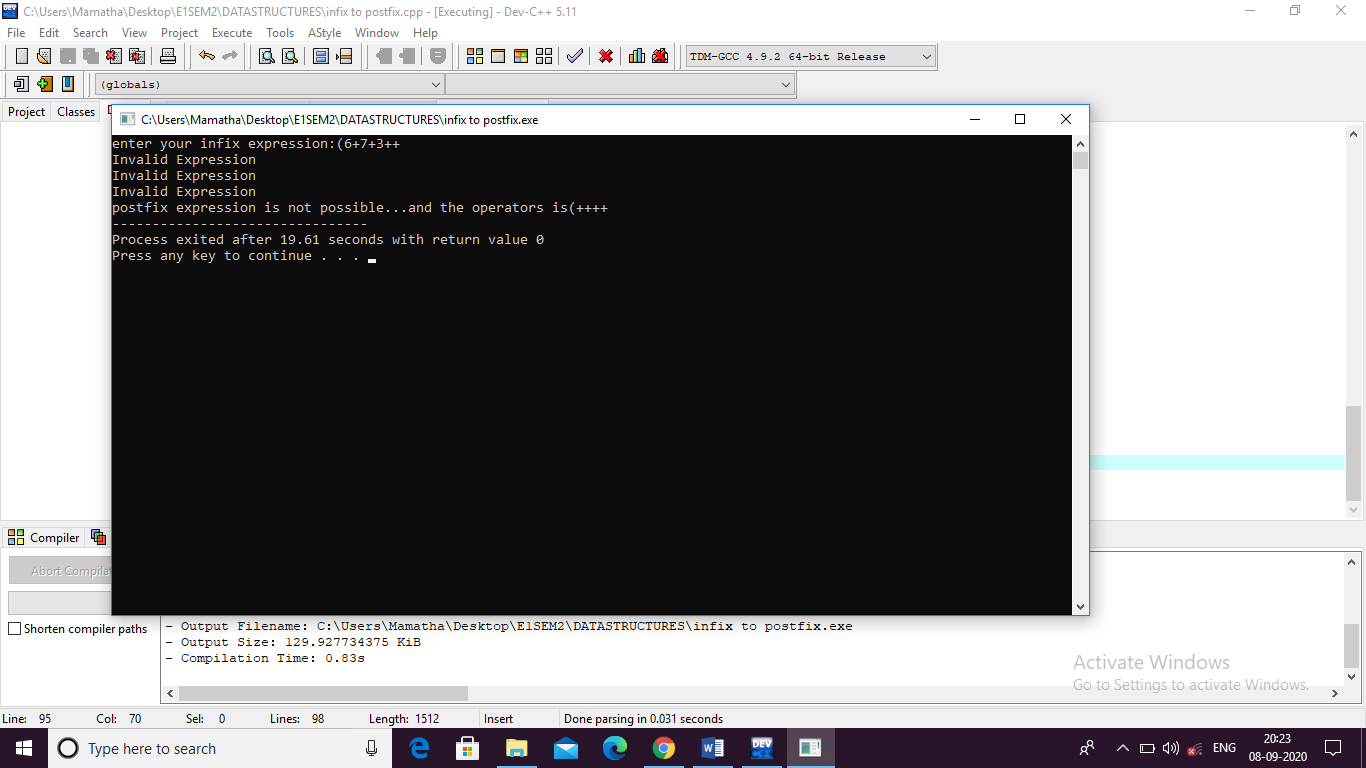
postfix[j]='\0';

printf("postfix expression:");

printf("%s",postfix);

return 0;

}



5.Evalution of Postfix expression

#include<stdio.h>

#include<ctype.h>

#define SIZE 50

int stack[SIZE];

int top=-1;

void push(int ele)

{

stack[++top]=ele;

}

int pop()

{

return stack[top--];

}

int result(char ch,int op1,int op2)

{

switch(ch)

{

case '+':

push(op1+op2);

break;

case '-':

push(op1-op2);

break;

case '\*':

push(op1\*op2);

break;

case '/':

push(op1/op2);

break;

case '%':

push(op1%op2);

break;

}

}

int main()

{

char postfix[SIZE],ch;

int i=0,op1,op2,r;

printf("enter the postfix expression:");

gets(postfix);

while((ch=postfix[i])!='\0')

{

i++;

if(isdigit(ch))

{

push(ch);

}

else if(ch=='+'||ch=='-'||ch=='\*'||ch=='/'||ch=='^'||ch=='%')

{

op2=pop();

op1=pop();

r=result(ch,op1,op2);

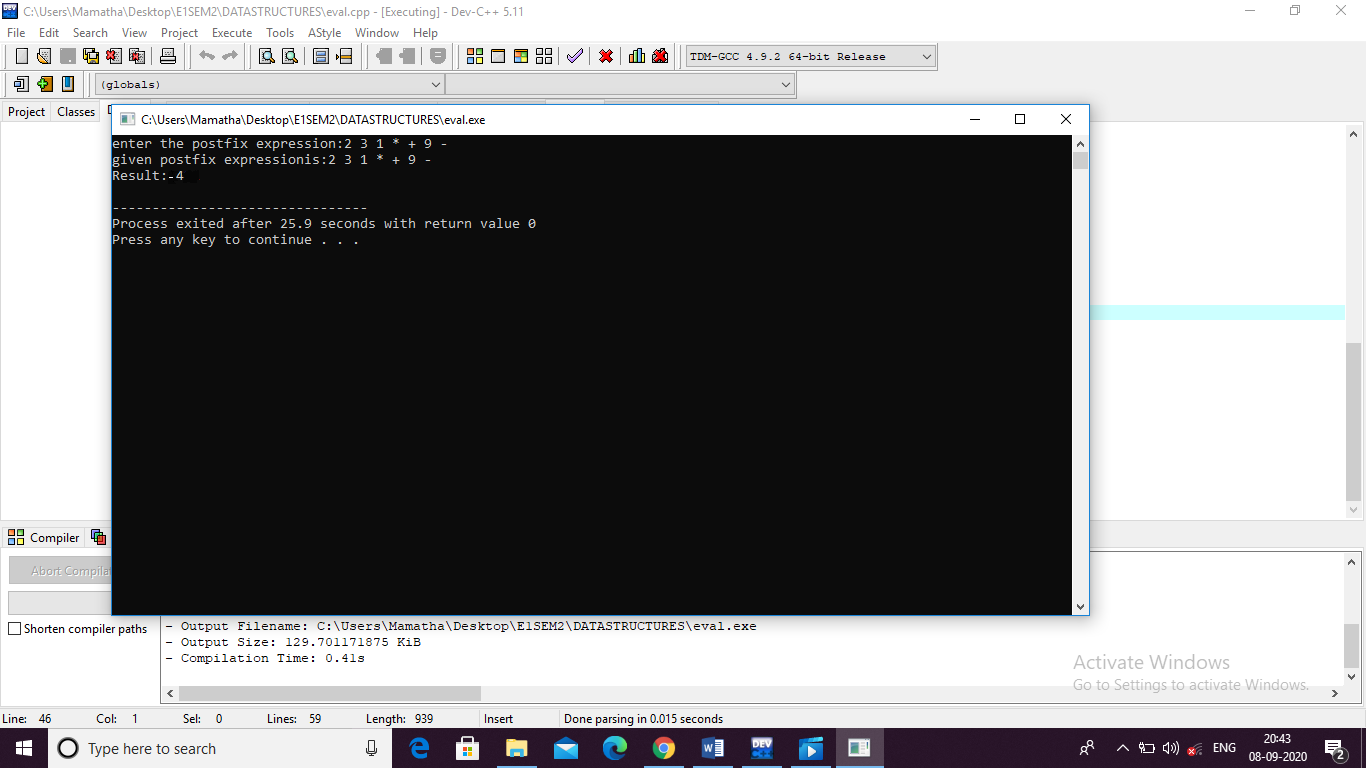
}

}

printf("given postfix expressionis:%s\n",postfix);

printf("Result:%d\n",stack[top]);

}



6.a)Queue ADT using array

#include<stdio.h>

#include<stdlib.h>

#define L 5

int queue[L];

int front=-1;

int rear=-1;

void enqueue(int ele)

{

if(rear==L-1)

{

printf("queue is full\n");

}

else if(front==-1&&rear==-1)

{

front=rear=0;

queue[rear]=ele;

}

else

{

rear++;

queue[rear]=ele;

}

}

void dequeue()

{

if(front==-1&&rear==-1)

{

printf("No elements to delete\n");

}

else if(front==rear)

{

front=rear=-1;

}

else

{

front++;

}

}

void display()

{

int i;

if(front==-1&&rear==-1)

{

printf("No elements to display\n");

}

else

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

{

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

}

}

int main()

{

int ch,ele;

printf("=========QUEUE operations=========");

printf("\n1.ENQUEUE\n2.DEQUEUE\n3.DISPLAY\n4.exit\n");

while(1)

{

printf("\nenter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("enter the element:");

scanf("%d",&ele);

enqueue(ele);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(1);

break;

default:

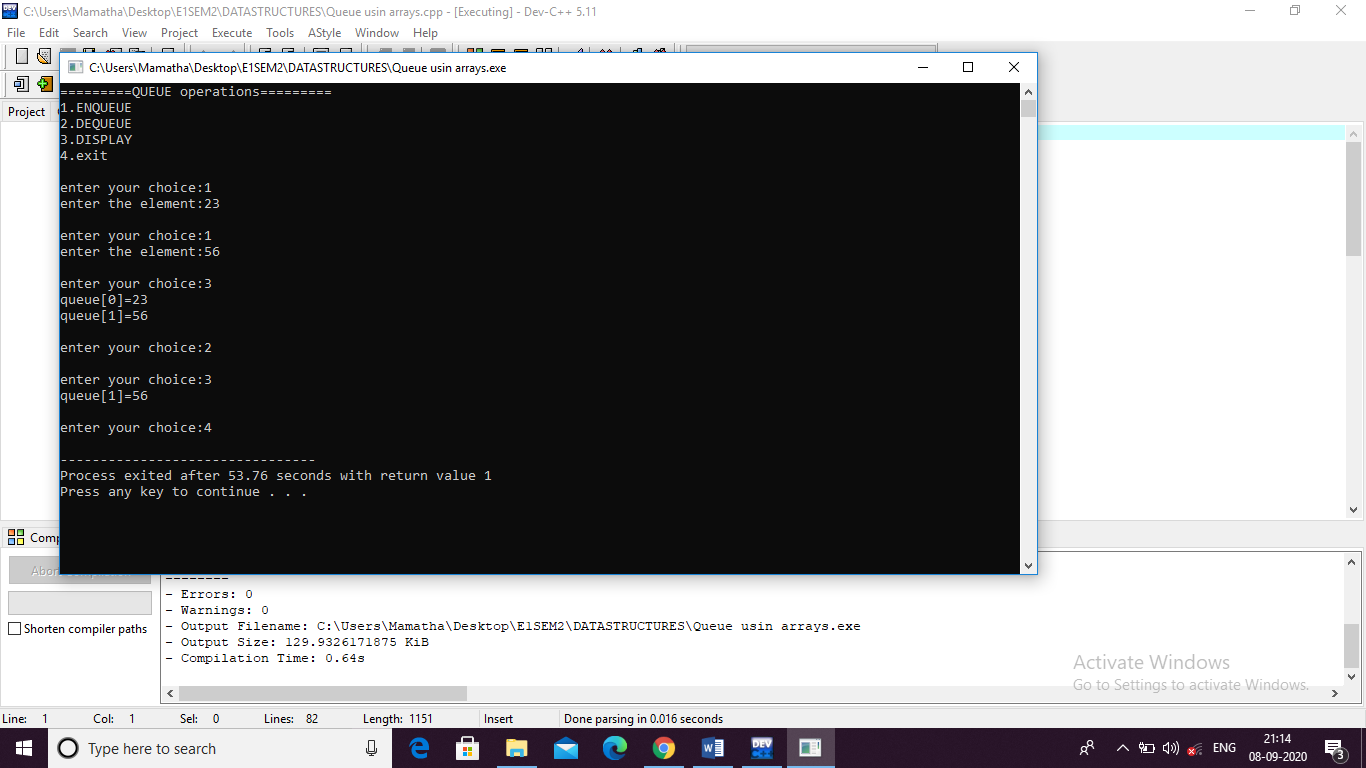
printf("INVALID RESPONSE..");

}

}

return 0;

}



6.b)Queue ADT usiing linkedlist

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*link;

};

struct node \*front=NULL;

struct node \*rear=NULL;

int enqueue()

{

struct node\*temp;

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

printf("enter the node data:");

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

temp->link=NULL;

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

{

front=rear=temp;

}

else

{

rear->link=temp;

rear=temp;

}

}

int dequeue()

{

struct node\*temp;

temp=front;

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

{

printf("NO ELEMENTS TO DELETE\n");

}

else

front=front->link;

temp->link=NULL;

free(temp);

}

void display()

{

struct node\*temp;

temp=front;

if(front==NULL)

{

printf("NO ELEMENTS TO DISPLAY\n");

}

else

{

while(temp!=NULL)

{

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

temp=temp->link;

}

}

}

int main()

{

int ch,ele;

printf("=========QUEUE operations=========");

printf("\n1.ENQUEUE\n2.DEQUEUE\n3.DISPLAY\n4.exit\n");

while(1)

{

printf("\nenter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(1);

break;

default:

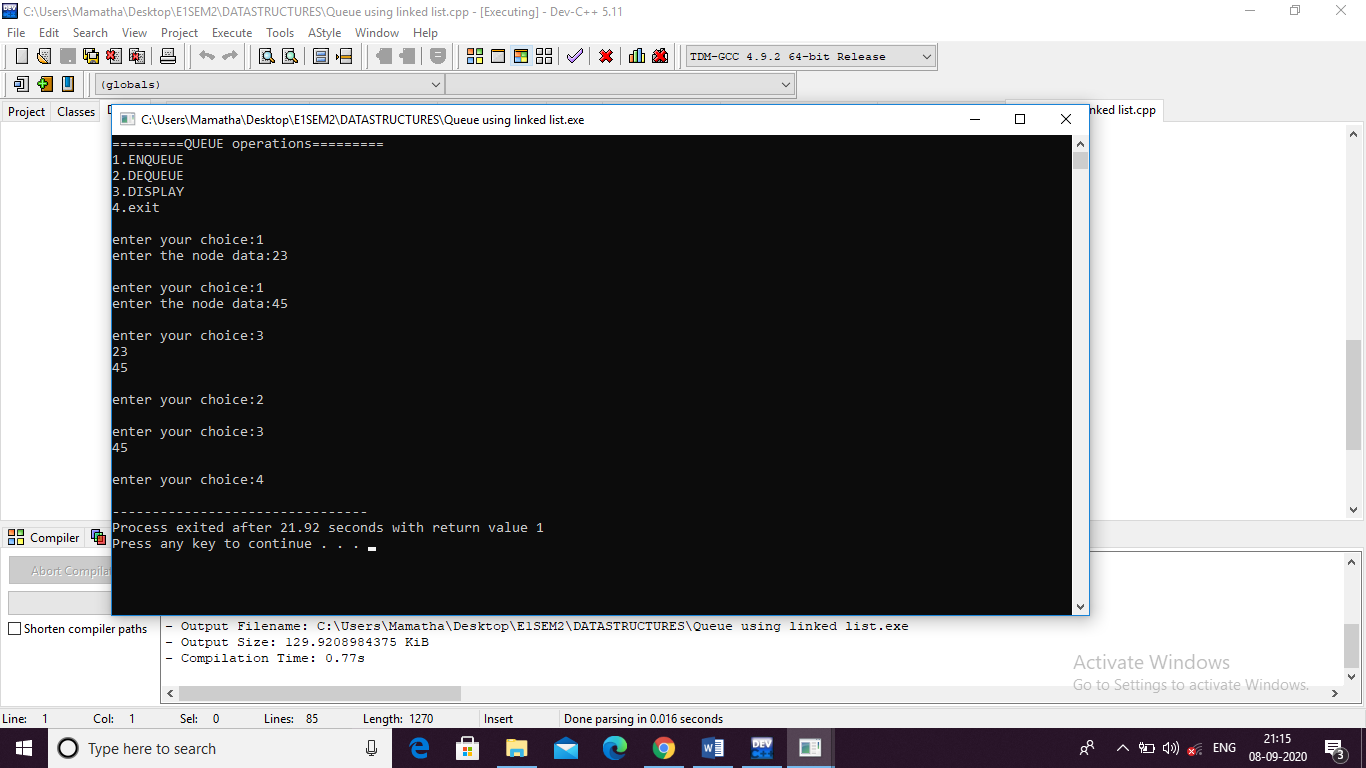
printf("INVALID RESPONSE..");

}

}

return 0;

}



7.a)Priority Queue ADT using array

#include<stdio.h>

#include<stdlib.h>

#define MAX 5

int front, rear;

int pri\_que[MAX];

void create()

{

front = rear = -1;

}

void check(int data)

{

int i,j;

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

{

if (data >= pri\_que[i])

{

for (j = rear + 1; j > i; j--)

{

pri\_que[j] = pri\_que[j - 1];

}

pri\_que[i] = data;

return;

}

}

pri\_que[i] = data;

}

void insert\_by\_priority(int data)

{

if (rear >= MAX - 1)

{

printf("\nQueue overflow no more elements can be inserted");

return;

}

if ((front == -1) && (rear == -1))

{

front++;

rear++;

pri\_que[rear] = data;

return;

}

else

check(data);

rear++;

}

void delete\_by\_priority(int data)

{

int i;

if ((front==-1) && (rear==-1))

{

printf("\nQueue is empty no elements to delete");

return;

}

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

{

if (data == pri\_que[i])

{

for (; i < rear; i++)

{

pri\_que[i] = pri\_que[i + 1];

}

pri\_que[i] = -99;

rear--;

if (rear == -1)

front = -1;

return;

}

}

printf("\n%d not found in queue to delete", data);

}

void display\_pqueue()

{

if ((front == -1) && (rear == -1))

{

printf("\nQueue is empty");

return;

}

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

{

printf(" %d ", pri\_que[front]);

}

front = 0;

}

int main()

{

int n, ch;

printf("\t\t\*\*\*\*\*PRIORITY QUEUE OPERATIONS\*\*\*\*\*");

printf("\n1 - Insert");

printf("\n2 - Delete");

printf("\n3 - Display");

printf("\n4 - Exit");

create();

while (1)

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("\nEnter value to be inserted : ");

scanf("%d",&n);

insert\_by\_priority(n);

break;

case 2:

printf("\nEnter value to delete : ");

scanf("%d",&n);

delete\_by\_priority(n);

break;

case 3:

display\_pqueue();

break;

case 4:

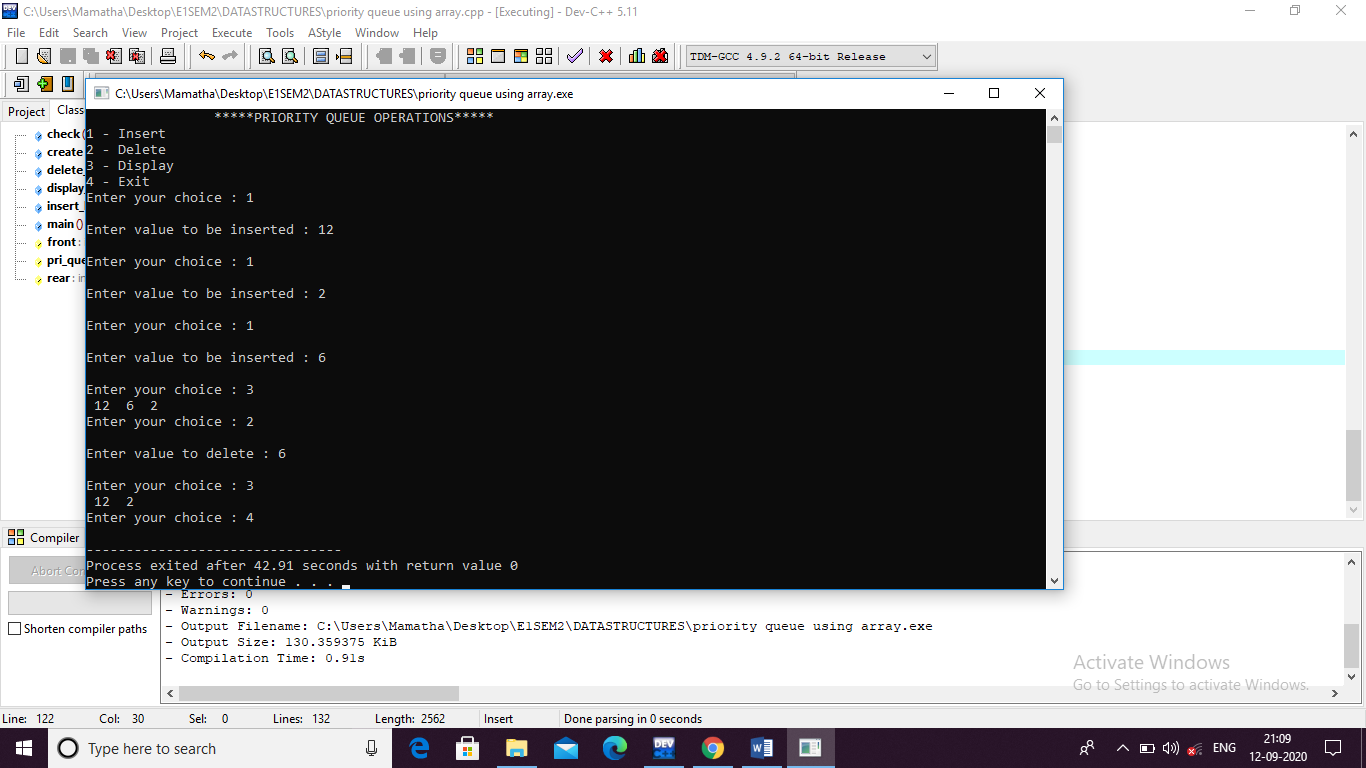
exit(0);

default:

printf("\nChoice is incorrect, Enter a correct choice");

}

}

}

7.b)Circular Queue ADT using array

#include<stdio.h>

#include<stdlib.h>

#define S 5

int queue[S];

int front=-1;

int rear=-1;

void enqueue(int ele)

{

if((front==0&&rear==S-1)||front==rear+1)

{

printf("circular queue is overflow..\n");

}

else if(front==-1&&rear==-1)

{

front++; rear++;

queue[rear]=ele;

}

else if(front!=0&&rear==S-1)

{

rear=0;

queue[rear]=ele;

}

else

{

rear++;

queue[rear]=ele;

}

}

void dequeue()

{

if(front==-1&&rear==-1)

{

printf("circular queue is underflow\n");

}

else if(front==rear)

{

front=rear=-1;

}

else if(front==S-1)

{

front=0;

}

else

{

front++;

}

}

int display()

{

int i;

if((front==-1&&rear==-1))

{

printf("circular queue is underflow\n");

}

else

{

if(front<rear)

{

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

{

printf("cqueue[%d]=%d\n",i,queue[i]);

}

}

else

{

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

printf("cqueue[%d]=%d\n",i,queue[i]);

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

printf("cqueue[%d]=%d\n",i,queue[i]);

}

}

}

int length()

{

int count;

if(front==-1&&rear==-1)

{

printf("circular queue is underflow\n");

}

else

{

while(display()!=S-1)

{

count++;

}

}

return count;

}

int main()

{

int ch,ele;

printf("CIRCULAR QUEUE OPERATIONS \n1.ENQUEUE\n2.DEQUEUE\n3.DISPLAY\n4.LENGTH\n5.EXIT\n");

while(1)

{

printf("enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("enter the element:");

scanf("%d",&ele);

enqueue(ele);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

printf("circular queue length",length());

case 5:

exit(1);

break;

default:

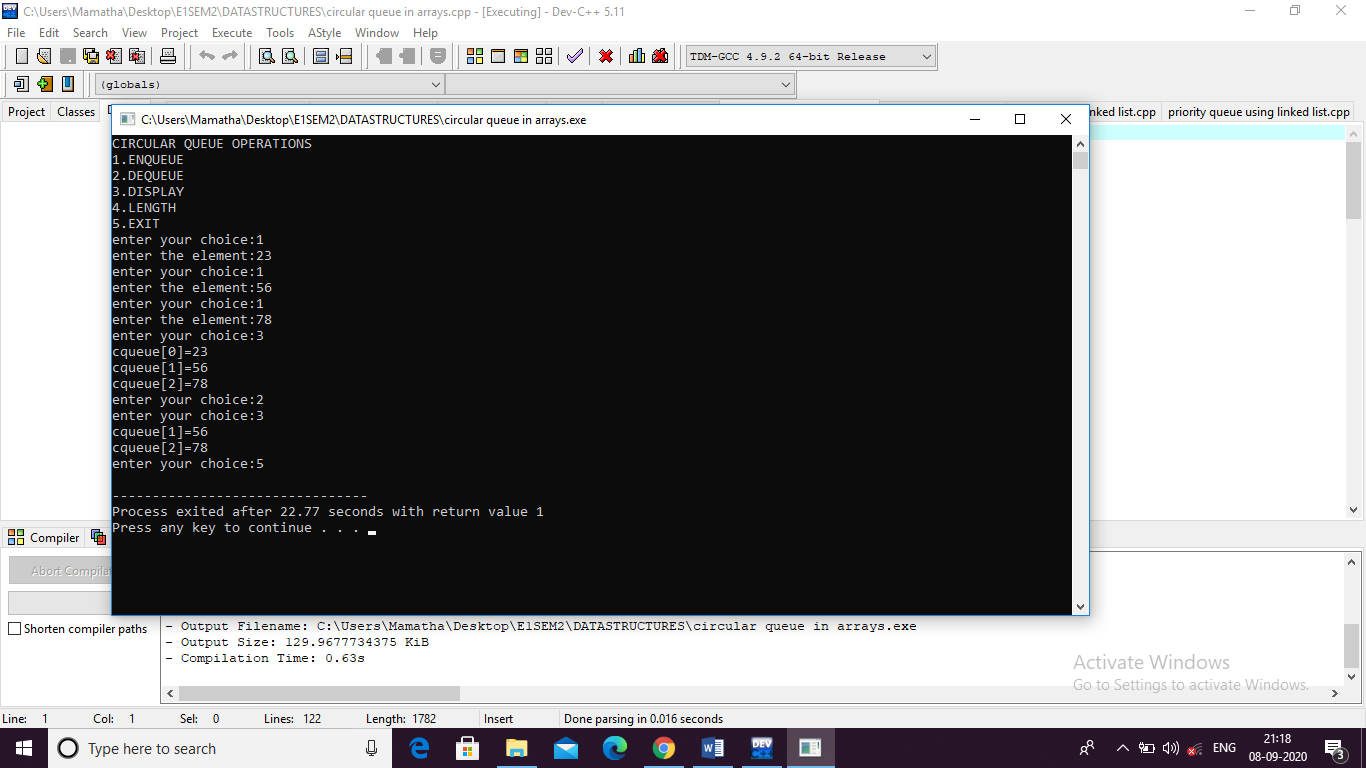
printf("INVALID RESPONSE\n");

}

}

return 0;

}



8.a)Insertion sort

#include<stdio.h>

int main()

{

int arr[20],i,j,n,temp,k;

printf("define the length of the array:");

scanf("%d",&n);

printf("enter the elements into the array:\n");

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

{

printf("array[%d]=",i);

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

}

//insertion sorting

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

{

temp=arr[i];

j=i-1;

while(arr[j]>temp&j>=0)

{

arr[j+1]=arr[j];

j--;

}

arr[j+1]=temp;

}

//displaying sorted array

printf("sorted array is\n");

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

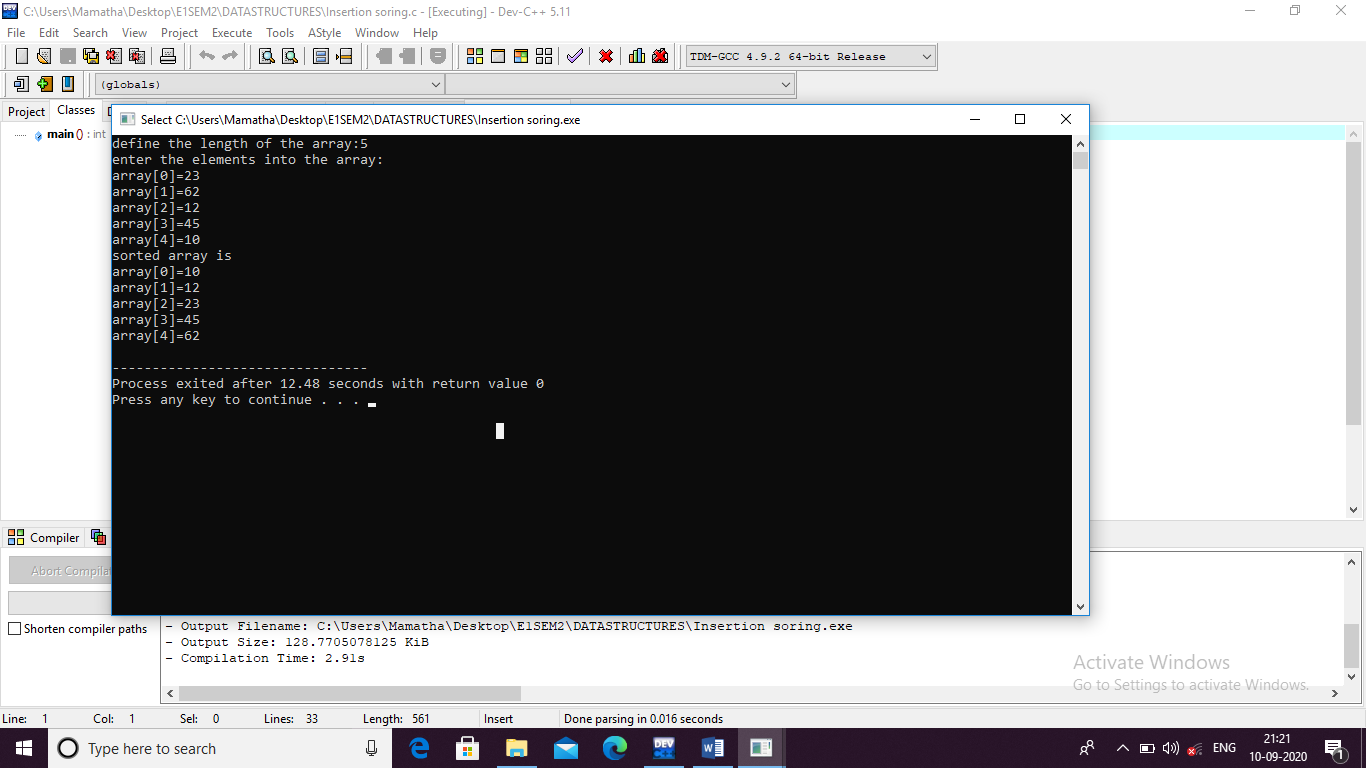
{

printf("array[%d]=%d\n",i,arr[i]);

}

return 0;

}



8.b)Merge sort

#include<stdio.h>

#define MAX 50

int MERGE(int a[],int lb,int ub,int mid)

{

int b[MAX],i,j,k=lb,m;

i=lb;j=mid+1;

while(i<=mid&&j<=ub)

{

if(a[i]<a[j])

{

b[k]=a[i];

i++;

}

else

{

b[k]=a[j];

j++;

}

k++;

}

if(i>mid)

{

while(j<=ub)

{

b[k]=a[j];

j++; k++;

}

}

else

{

while(i<=mid)

{

b[k]=a[i];

i++;

k++;

}

}

for(k=0;k<=ub;k++)

{

a[k]=b[k];

}

}

int merge(int a[],int lb,int ub)

{

int mid;

if(lb<ub)

{

mid=(lb+ub)/2;

}

merge(a,lb,mid);

merge(a,mid+1,ub);

MERGE(a,lb,ub,mid);

}

int main()

{

int a[MAX],i,n,lb,ub,k;

printf("enter the length of array:");

scanf("%d",&n);

printf("enter elements into the array:");

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

{

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

}

lb=0;

ub=n-1;

merge(a,lb,ub);

printf("sorted array is ");

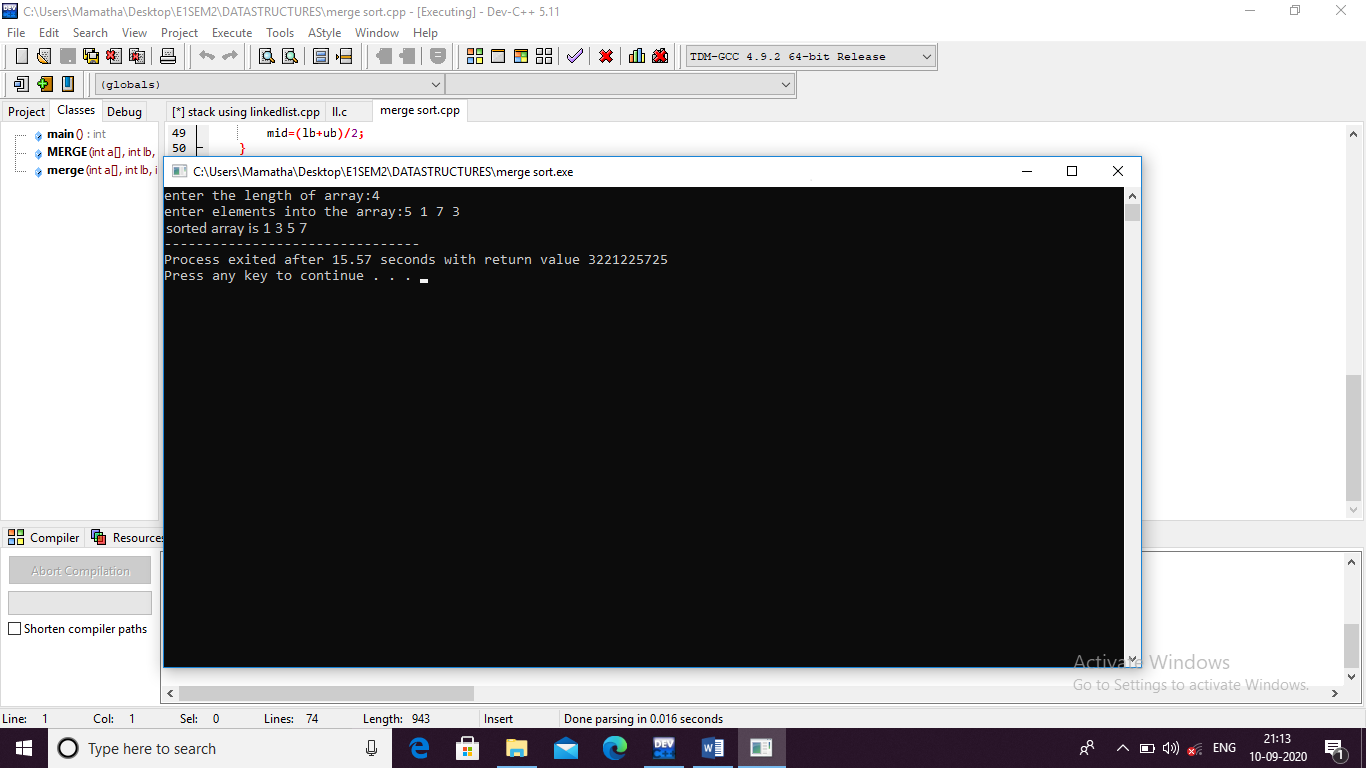
for(i=0;i<=ub;i++)

{

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

}

return 0;

}

9.a)Quick sort

#include<stdio.h>

int quicksort(int a[],int lb,int ub)

{

int pivot=lb;

int start=lb;

int end=ub;

int temp,i;

while(start<end)

{

while(a[start]<=a[pivot])

{

start++;

}

while(a[end]>a[pivot])

{

end--;

}

if(start<end)

{

temp=a[start];

a[start]=a[end];

a[end]=temp;

}

}

temp=a[end];

a[end]=a[pivot];

a[pivot]=temp;

quicksort(a,lb,ub);

quicksort(a,lb,end-1);

quicksort(a,end+1,ub);

}

int main()

{

int a[50],i,n,lb,ub;

printf("enter the length of array:");

scanf("%d",&n);

printf("enter elements into the array:");

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

{

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

}

lb=0;

ub=n-1;

quicksort(a,lb,ub);

printf("sorted array is ");

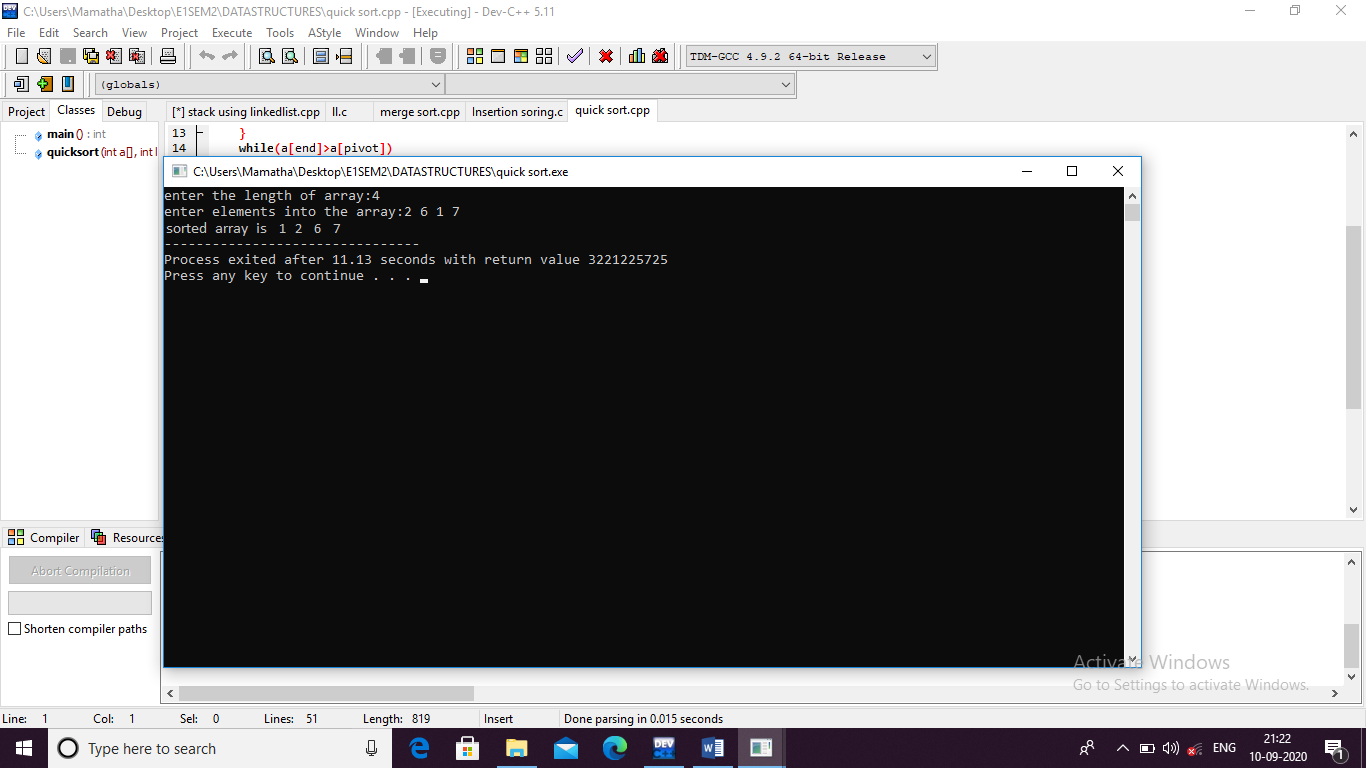
for(i=lb;i<=ub;i++)

{

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

}

return 0;

}

9.b)Selection sort

#include<stdio.h>

int main()

{

int arr[20],i,j,n,temp;

printf("define the length of the array:");

scanf("%d",&n);

printf("enter the elements into the array:\n");

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

{

printf("array[%d]=",i);

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

}

//selection sorting

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

{

int min=i;

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

{

if(arr[j]<arr[min])

{

min=j;

}

}

temp=arr[i];

arr[i]=arr[min];

arr[min]=temp;

}

//displaying sorted array

printf("sorted array is\n");

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

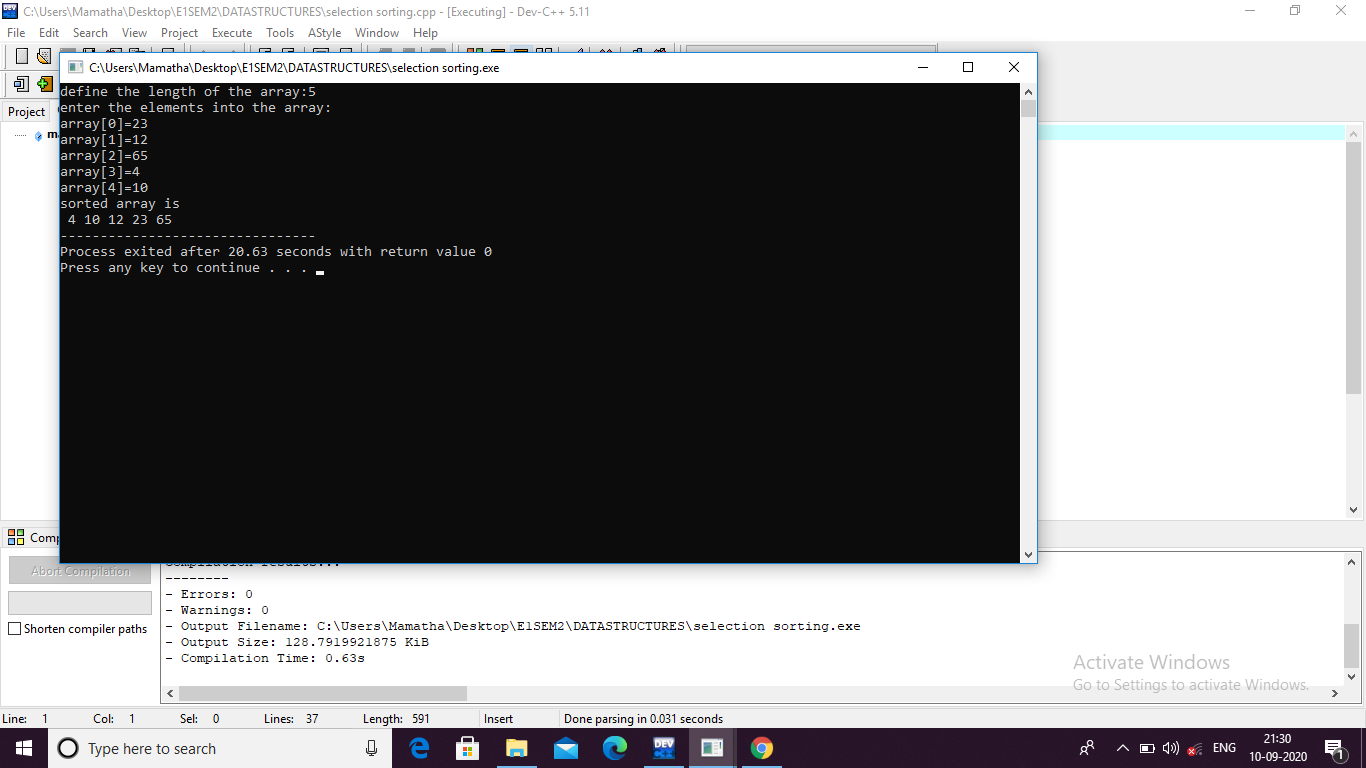
{

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

}

return 0;

}



10.Heap sort algorithm

#include<stdio.h>

void create(int []);

void down\_adjust(int [],int);

void main()

{

int heap[30],n,i,last,temp;

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

scanf("%d",&n);

printf("\nEnter elements:");

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

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

heap[0]=n;

create(heap);

while(heap[0] > 1)

{

last=heap[0];

temp=heap[1];

heap[1]=heap[last];

heap[last]=temp;

heap[0]--;

down\_adjust(heap,1);

}

printf("\nArray after sorting:\n");

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

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

}

void create(int heap[])

{

int i,n;

n=heap[0]; //no. of elements

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

down\_adjust(heap,i);

}

void down\_adjust(int heap[],int i)

{

int j,temp,n,flag=1;

n=heap[0];

while(2\*i<=n && flag==1)

{

j=2\*i; //j points to left child

if(j+1<=n && heap[j+1] > heap[j])

j=j+1;

if(heap[i] > heap[j])

flag=0;

else

{

temp=heap[i];

heap[i]=heap[j];

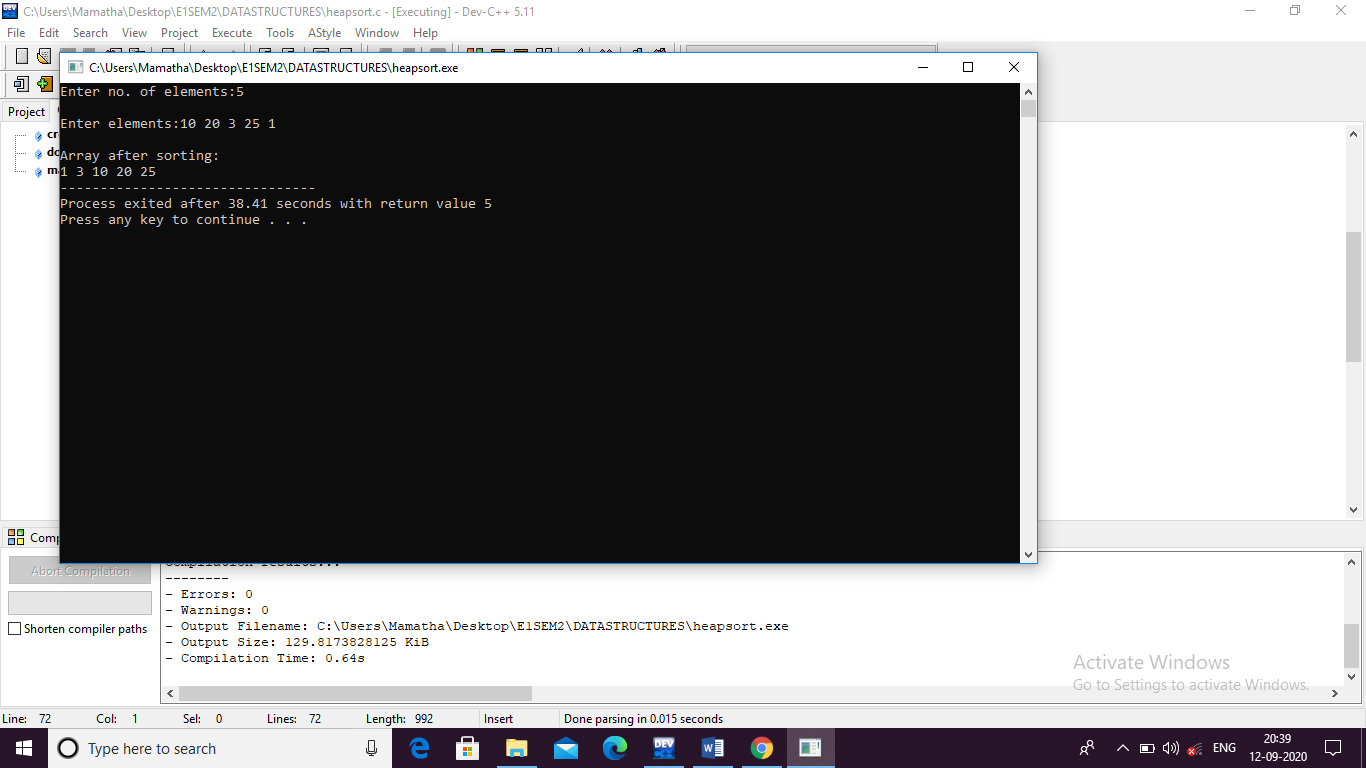
heap[j]=temp;

i=j;

}

}

}



11,12.Binary Search Tree operations

1.Create bst

2.Insert the element

3.Delete the element

4.Inorder

5.Preorder

6.Postorder

PROGRAM CODE:

#include <stdio.h>

#include <conio.h>

#include <malloc.h>

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*tree;

void create\_tree(struct node \*);

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

void preorderTraversal(struct node \*);

void inorderTraversal(struct node \*);

void postorderTraversal(struct node \*);

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

int main()

{

int option, val;

struct node \*ptr;

create\_tree(tree);

printf("\n \*\*\*\*\*\*MAIN MENU\*\*\*\*\*\*\* \n");

printf("\n 1. Insert Element");

printf("\n 2. Preorder Traversal");

printf("\n 3. Inorder Traversal");

printf("\n 4. Postorder Traversal");

printf("\n 5. Delete an element");

printf("\n 6. Exit");

do

{

printf("\n Enter your option : ");

scanf("%d", &option);

switch(option)

{

case 1:

printf("Enter the value of the new node : ");

scanf("%d", &val);

tree = insertElement(tree, val);

break;

case 2:

printf(" The elements of the tree are : ");

preorderTraversal(tree);

break;

case 3:

printf(" The elements of the tree are : ");

inorderTraversal(tree);

break;

case 4:

printf(" The elements of the tree are : ");

postorderTraversal(tree);

break;

case 5:

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

scanf("%d", &val);

tree = deleteElement(tree, val);

break;

}

}while(option!=6);

getch();

return 0;

}

void create\_tree(struct node \*tree)

{

tree = NULL;

}

struct node \*insertElement(struct node \*tree, int val)

{

struct node \*ptr, \*nodeptr, \*parentptr;

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

ptr->data = val;

ptr->left = NULL;

ptr->right = NULL;

if(tree==NULL)

{

tree=ptr;

tree->left=NULL;

tree->right=NULL;

}

else

{

parentptr=NULL;

nodeptr=tree;

while(nodeptr!=NULL)

{

parentptr=nodeptr;

if(val<nodeptr->data)

nodeptr=nodeptr->left;

else

nodeptr = nodeptr->right;

}

if(val<parentptr->data)

parentptr->left = ptr;

else

parentptr->right = ptr;

}

return tree;

}

void preorderTraversal(struct node \*tree)

{

if(tree != NULL)

{

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

preorderTraversal(tree->left);

preorderTraversal(tree->right);

}

}

void inorderTraversal(struct node \*tree)

{

if(tree != NULL)

{

inorderTraversal(tree->left);

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

inorderTraversal(tree->right);

}

}

void postorderTraversal(struct node \*tree)

{

if(tree != NULL)

{

postorderTraversal(tree->left);

postorderTraversal(tree->right);

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

}

}

struct node \*deleteElement(struct node \*tree, int val)

{

struct node \*cur, \*parent, \*suc, \*psuc, \*ptr;

if(tree->left==NULL)

{

printf("\n The tree is empty ");

return(tree);

}

parent = tree;

cur = tree->left;

while(cur!=NULL && val!= cur->data)

{

parent = cur;

cur = (val<cur->data)? cur->left:cur->right;

}

if(cur == NULL)

{

printf("\n The value to be deleted is not present in the tree");

return(tree);

}

if(cur->left == NULL)

ptr = cur->right;

else if(cur->right == NULL)

ptr = cur->left;

else

{

// Find the in–order successor and its parent

psuc = cur;

cur = cur->left;

while(suc->left!=NULL)

{

psuc = suc;

suc = suc->left;

}

if(cur==psuc)

{

// Situation 1

suc->left = cur->right;

}

else

{

// Situation 2

suc->left = cur->left;

psuc->left = suc->right;

suc->right = cur->right;

}

ptr = suc;

}

// Attach ptr to the parent node

if(parent->left == cur)

parent->left=ptr;

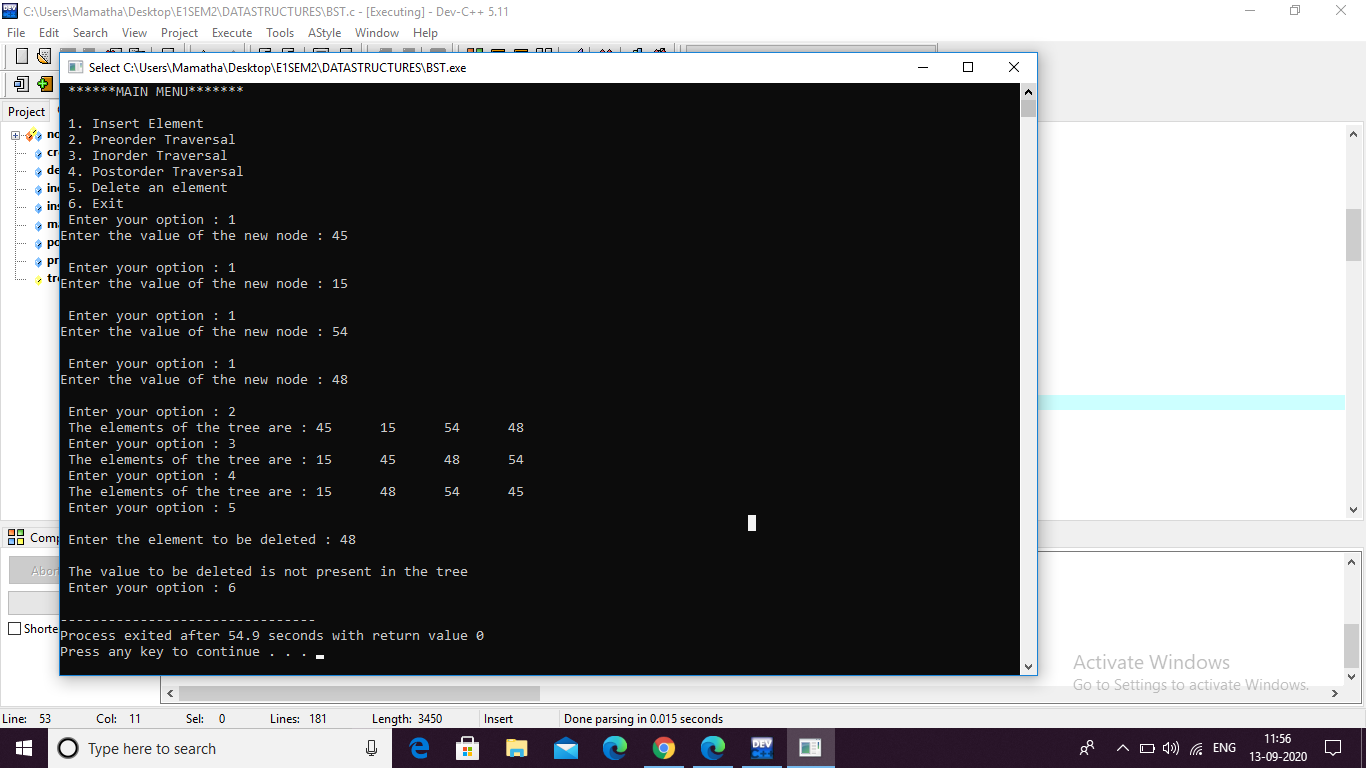
else

parent->right=ptr;

free(cur);

return tree;

}



14.a)Depth First Traversal

#include <stdio.h>

#define MAX 5

void depth\_first\_search(int adj[][MAX],int visited[],int start)

{

int stack[MAX];

int top =-1, i;

printf("%c–",start + 65);

visited[start] = 1;

stack[++top] = start;

while(top!=-1)

{

start = stack[top];

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

{

if(adj[start][i] && visited[i] == 0)

{

stack[++top] = i;

printf("%c–", i + 65);

visited[i] = 1;

break;

}

}

if(i == MAX)

top--;

}

}

int main()

{

int adj[MAX][MAX];

int visited[MAX] = {0}, i, j;

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

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

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

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

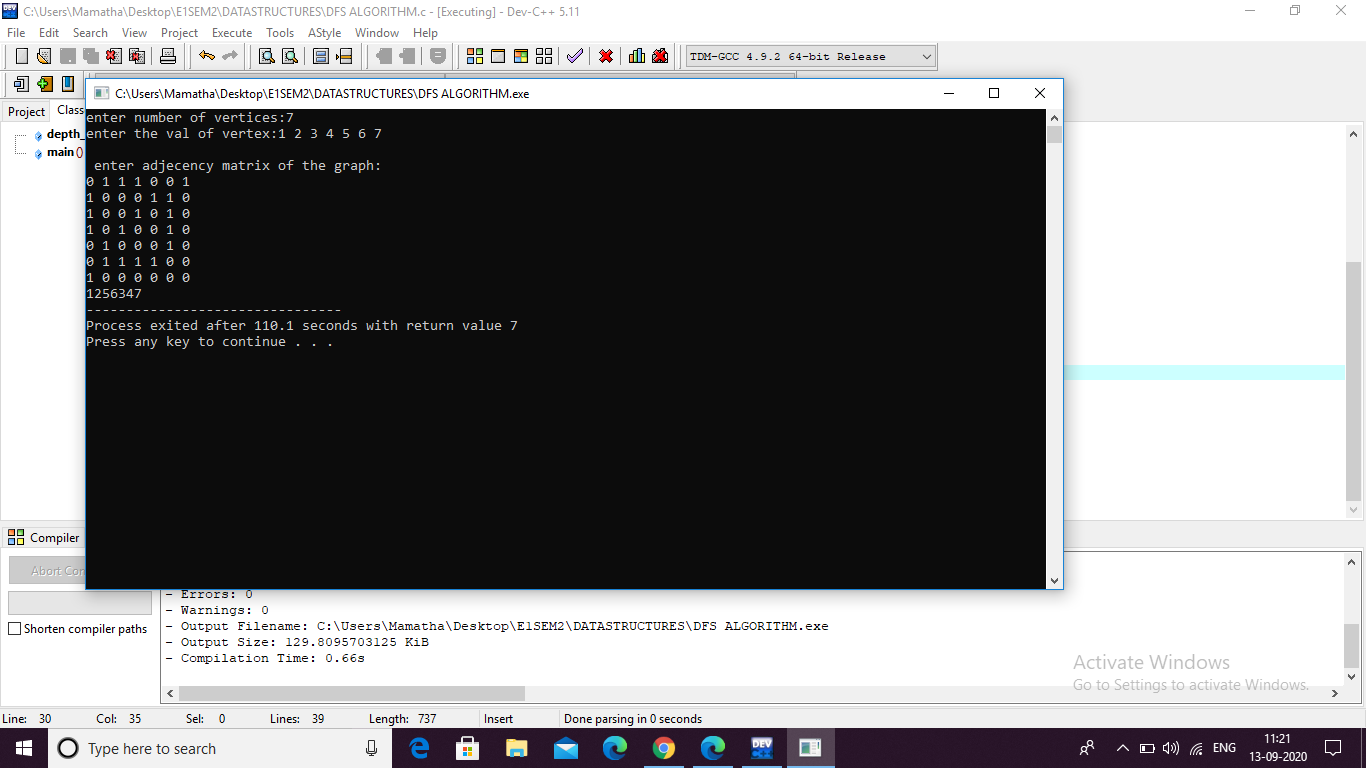
printf("DFS Traversal: ");

depth\_first\_search(adj,visited,0);

printf("\n");

return 0;

}



14.b)Breadth First Traversal

#include <stdio.h>

#define MAX 10

void breadth\_first\_search(int adj[][MAX],int visited[],int start)

{

int queue[MAX],rear = -1,front =-1, i;

queue[++rear] = start;

visited[start] = 1;

while(rear != front)

{

start = queue[++front];

if(start == 4)

printf("5\t");

else

printf("%c \t",start + 65);

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

{

if(adj[start][i] == 1 && visited[i] == 0)

{

queue[++rear] = i;

visited[i] = 1;

}

}

}

}

int main()

{

int visited[MAX] = {0};

int adj[MAX][MAX], i, j;

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

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

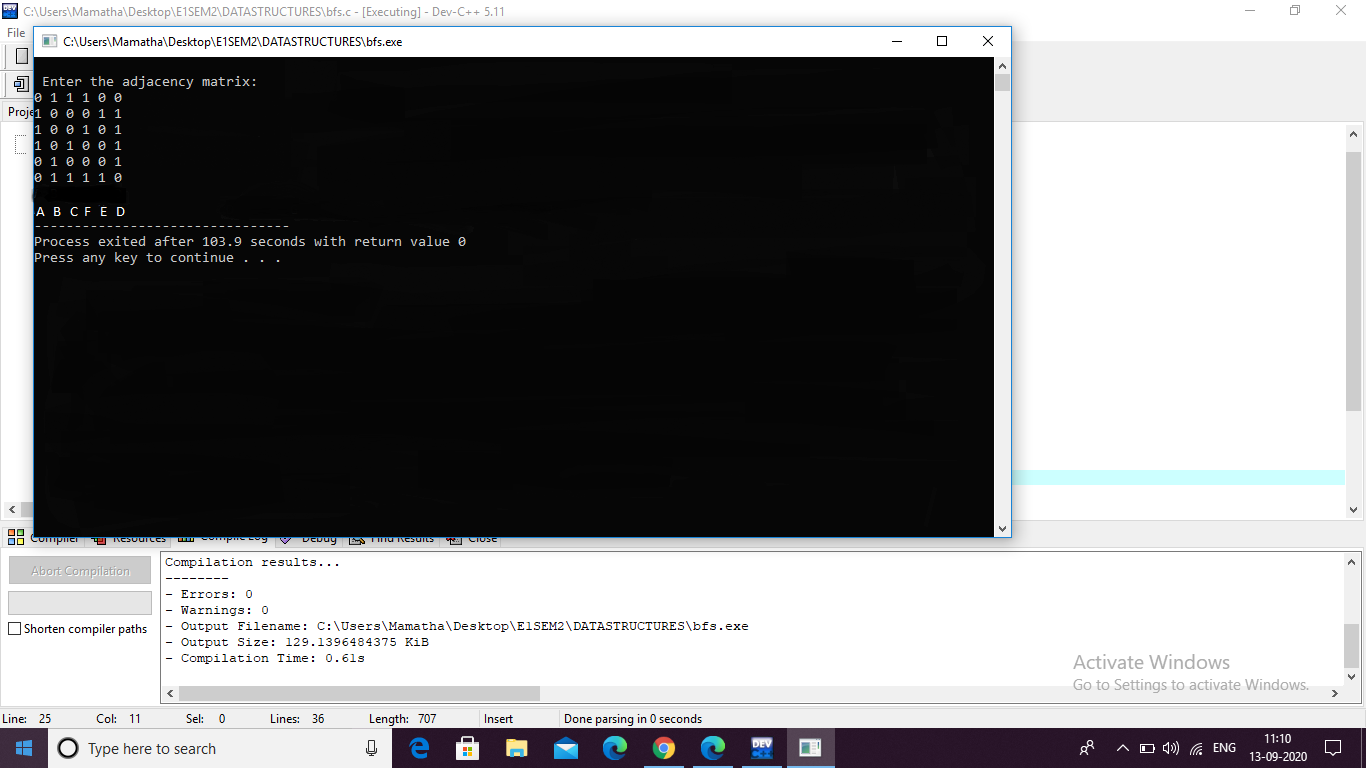
for(j = 0; j < MAX; j++)

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

breadth\_first\_search(adj,visited,0);

return 0;

}



**------ Thank you sir \*\*\***