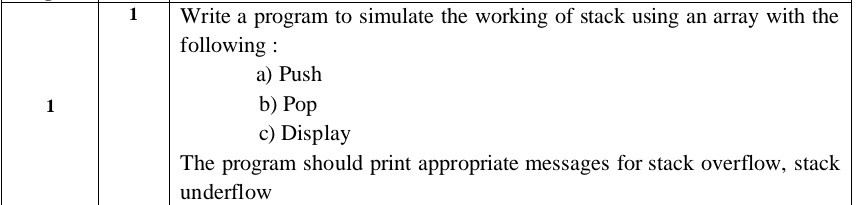
# DATA STRUCTURES LAB REPORT

## NAME: Neha B Chadaga

## USN: 1BM19CS098

## SUBJECT: Data Structures

## ACADEMIC YEAR: 2020-2021



#include<stdio.h>

#include<process.h>

#define stack\_size 3

int item,s[3],top=-1;

int item\_deleted,choice,ch=1;

void push()

{

if(top==stack\_size-1)

{

printf("Stack Overflow\n");

return 0;

}

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

scanf("%d",&item);

top=top+1;

s[top]=item;

}

int pop()

{

if(top==-1)

{

printf("Cannot delete item due to Stack Underflow\n");

return 0;

}

item\_deleted=s[top];

top=top-1;

return item\_deleted;

}

void display()

{

int i;

if(top==-1)

{

printf("Stack is Empty(Underflow condition)\n");

return 0;

}

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

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

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

}

int main()

{

while(ch==1)

{

printf("Enter your choice:\n1.Push\n2.pop\n3.Display\n4.Exit\n");

scanf("%d",&choice);

switch(choice)

{

case 1: push();

break;

case 2: item\_deleted=pop();

if(item\_deleted!=0)

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

break;

case 3: display();

break;

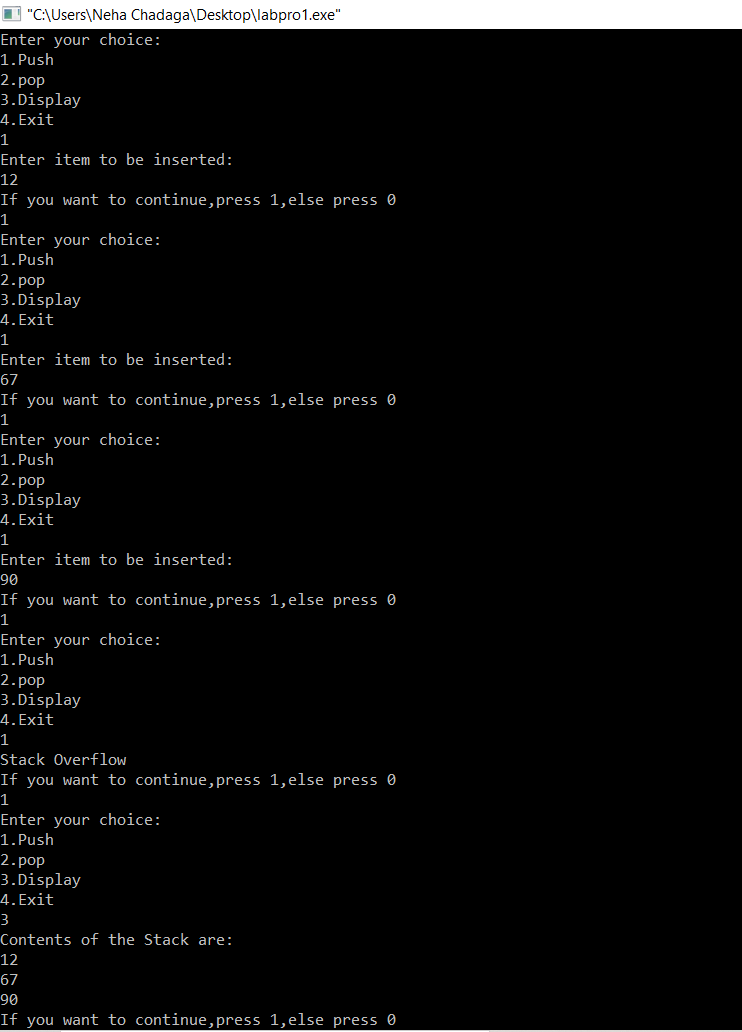
case 4: exit(0);

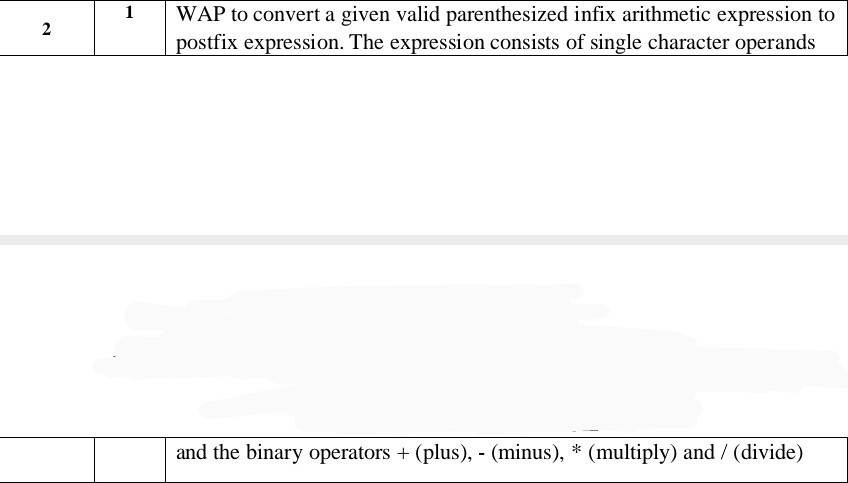
}

printf("If you want to continue,press 1,else press 0\n");

scanf("%d",&ch);

}





#include<stdio.h>

#include<string.h>

#include<process.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])>F(symbol))

    {

     postfix[j]=s[top--];

     j++;

    }

    if(F(s[top])!=F(symbol))

      s[++top]=symbol;

      else

        top--;

  }

  while(s[top]!='#')

  {

    postfix[j++]=s[top--];

  }

  postfix[j]='\0';

}

void main()

{

 char infix[20],postfix[20];

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

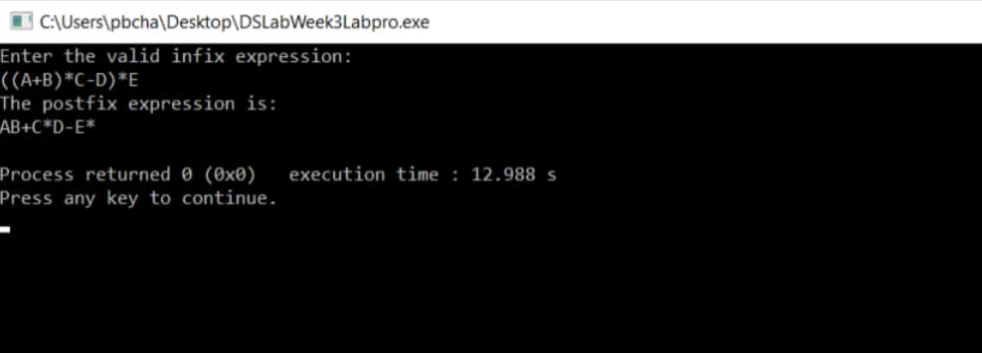
 scanf("%s",infix);

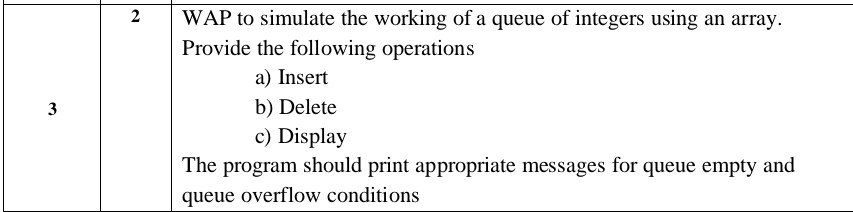
 infix\_postfix(infix,postfix);

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

 printf("%s\n",postfix);

}





#include<stdio.h>

#include<process.h>

#define QUE\_SIZE 3

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

void insertrear()

{

if(rear==QUE\_SIZE-1)

{

printf("Queue is Full\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 Queue:\n");

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

{

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

}

}

void main()

{

int choice;

for(;;)

{

printf("1.Insert at rear\n2. Delete from front\n3.Display\n4.Exit\n");

printf("Enter 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:

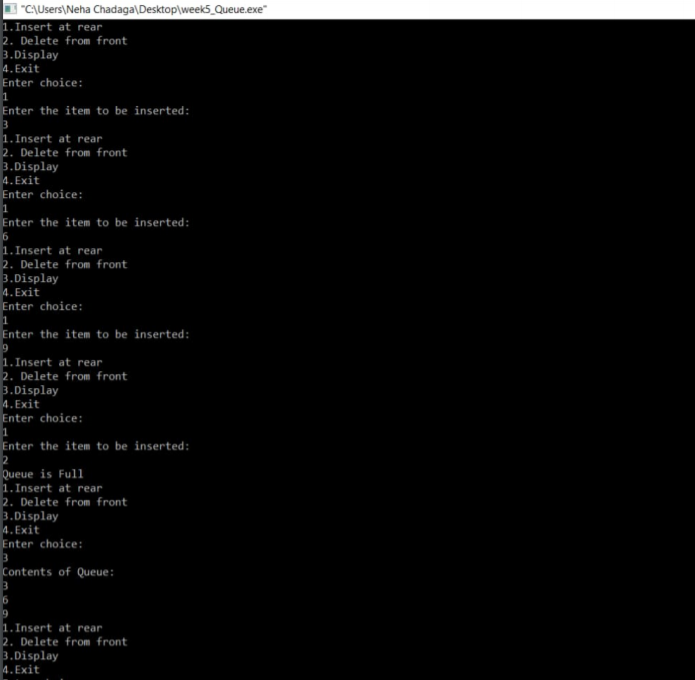
displayQ();

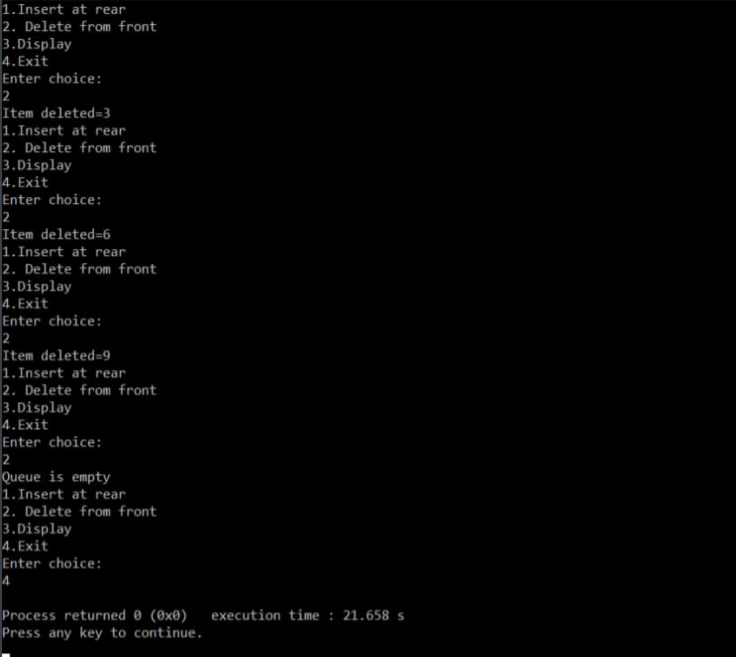
break;

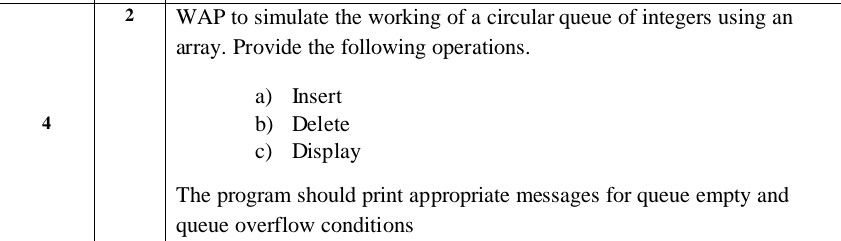
default: exit(0);

}

}







#include<stdio.h>

#include<process.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--;

return item;

}

void display()

{

int i,f,r;

if(count==0)

{

printf("Queue is empty\n");

return;

}

f=front;

r=rear;

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

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

{

/\*if(front<=rear)

{

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

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

}

else

{

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

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

}\*/

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

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

}

}

int main()

{

int choice;

for(;;)

{

printf("1.Insert at rear\n2.Delete Front\n3.Display\n4.Exit\n");

printf("Enter 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: display();

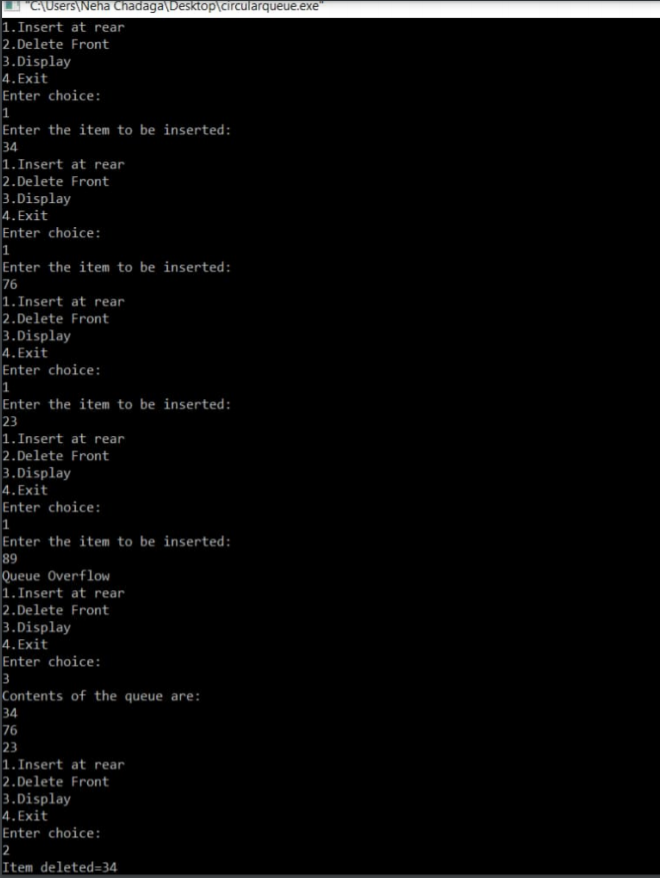
break;

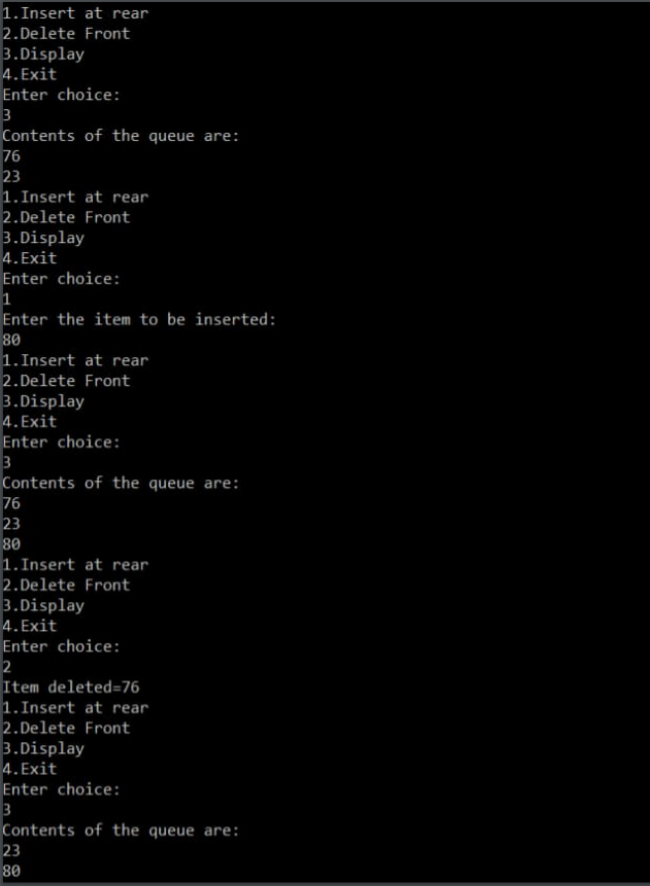
case 4: exit(0);

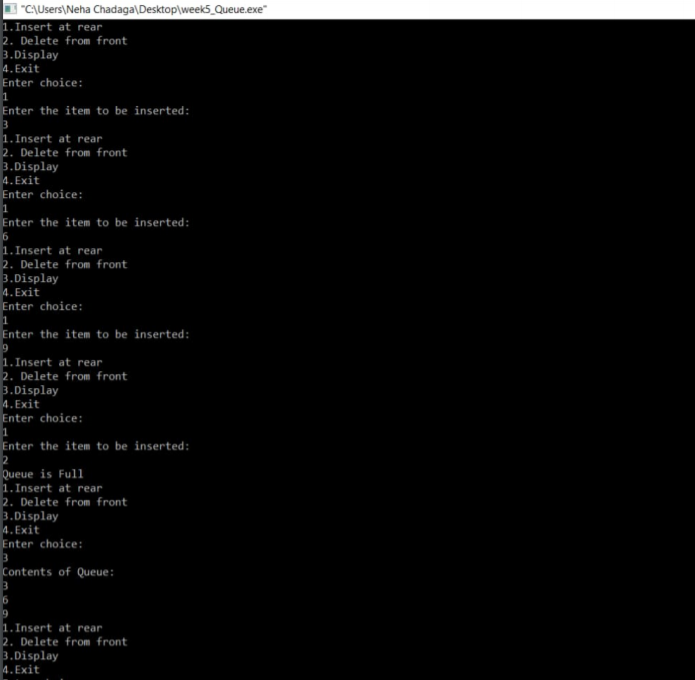
}

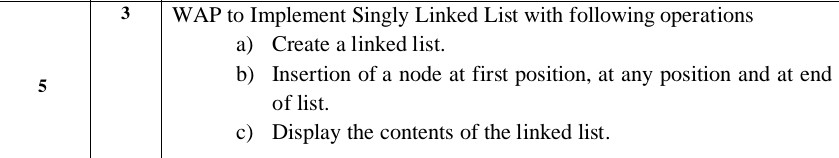
}

}









#include<stdio.h>

#include<process.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 is 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 first;

}

if(pos==1)

{

temp->link=first;

first=temp;

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;

}

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);

}

}

void main()

{

int item,choice,pos;

NODE first=NULL;

for(;;)

{

printf("1.Insert\_front\n2.Insert\_rear\n3.Insert at given Position\n4.Display\_list\n5.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);

first=insert\_front(first,item);

break;

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

scanf("%d",&item);

first=insert\_rear(first,item);

break;

case 3:printf("enter the item to be inserted at given position\n");

scanf("%d",&item);

printf("enter the position\n");

scanf("%d",&pos);

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

break;

case 4:display(first);

break;

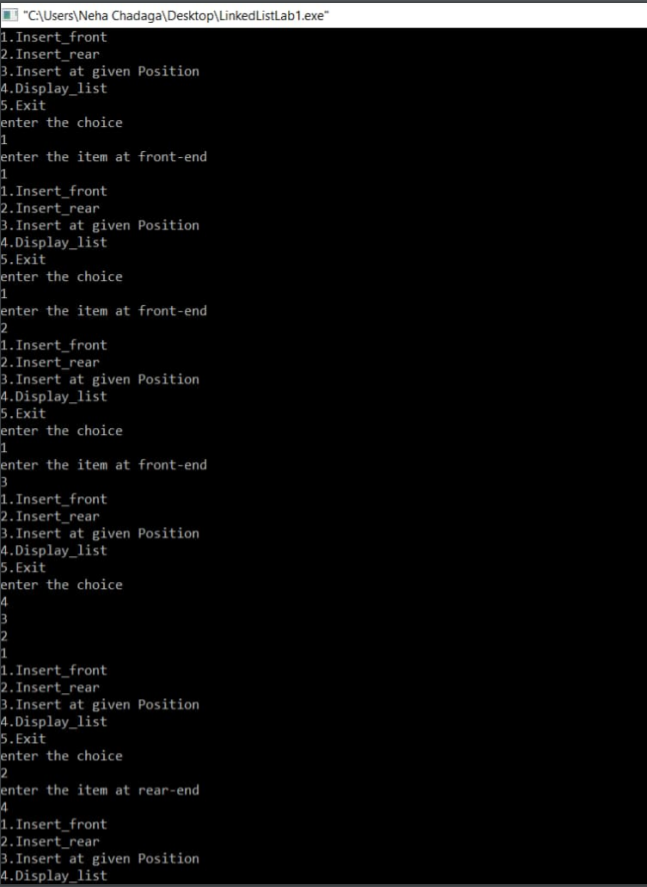
default:exit(0);

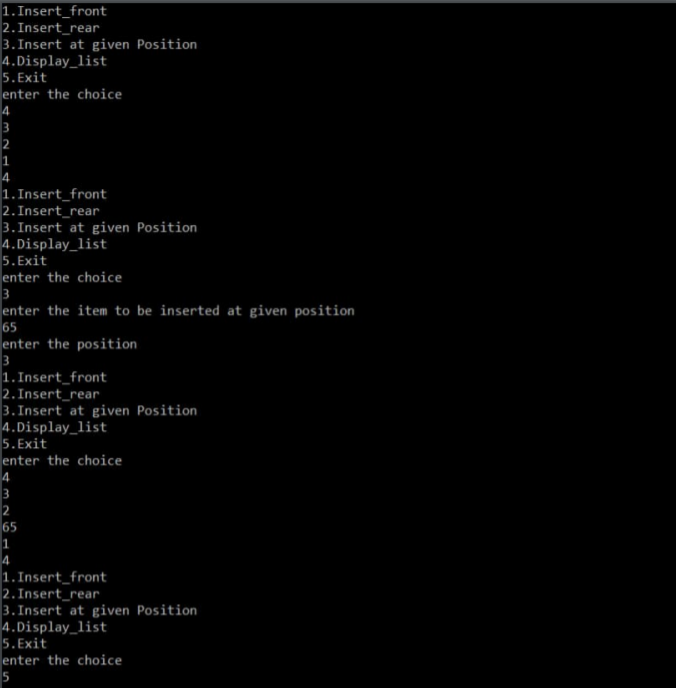
break;

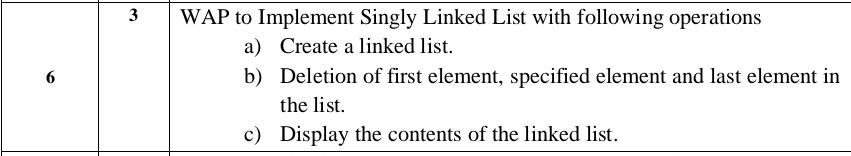
}

}

}







#include<stdio.h>

#include<process.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 is 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 first;

}

if(pos==1)

{

temp->link=first;

first=temp;

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("CANNOT DELETE AS LIST IS EMPTY\n");

return first;

}

temp=first;

temp=temp->link;

printf("ITEM DELETED AT FRONT END=%d\n",first->info);

free(first);

return temp;

}

NODE delete\_rear(NODE first)

{

NODE cur,prev;

if(first==NULL)

{

printf("CANNOT DELETE AS LIST IS EMPTY\n");

return first;

}

if(first->link==NULL)

{

printf("ITEM DELETED=%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=%d\n",cur->info);

free(cur);

prev->link=NULL;

return first;

}

NODE delete\_pos(int pos,NODE first)

{

NODE cur;

NODE prev;

int count,flag=0;

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)

{

flag=1;

break;

}

count++;

prev=cur;

cur=cur->link;

}

if(flag==0)

{

printf("invalid position\n");

return first;

}

printf("ITEM DELETED AT POSITION %d is %d\n",pos,cur->info);

prev->link=cur->link;

freenode(cur);

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);

}

}

void main()

{

int item,choice,pos;

NODE first=NULL;

for(;;)

{

printf("1.Insert\_front\n2.Insert\_rear\n3.Insert at given Position\n4.Delete Front\n5.Delete Rear\n6.Delete at a given position\n7.Display the list\n8.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);

first=insert\_front(first,item);

break;

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

scanf("%d",&item);

first=insert\_rear(first,item);

break;

case 3:printf("enter the item to be inserted at given position\n");

scanf("%d",&item);

printf("enter the position\n");

scanf("%d",&pos);

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

break;

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

break;

case 5:first=delete\_rear(first);

break;

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

scanf("%d",&pos);

first=delete\_pos(pos,first);

break;

case 7:display(first);

break;

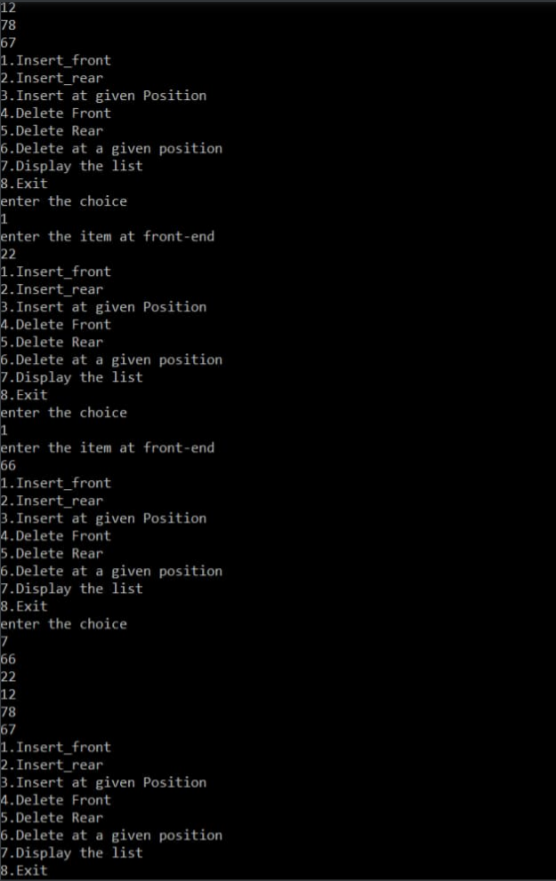
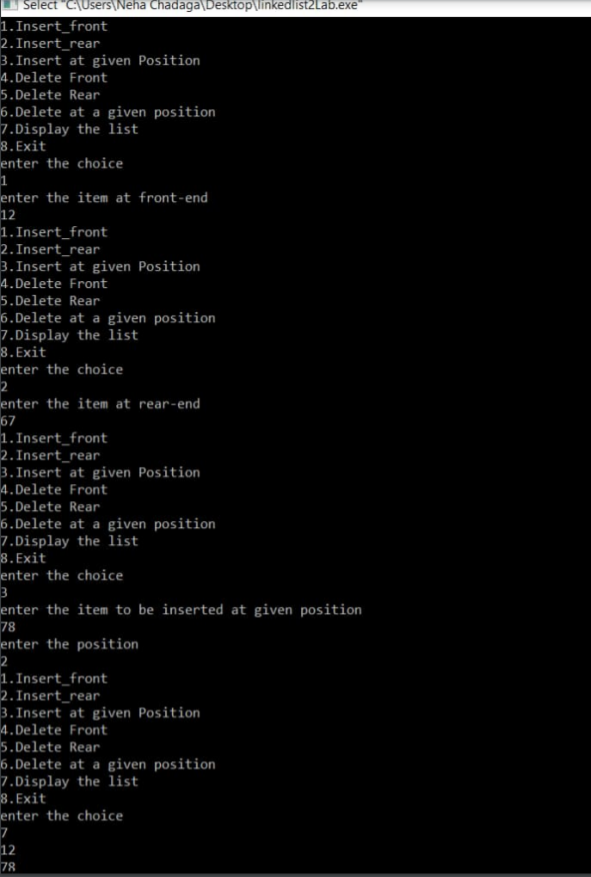
default:exit(0);

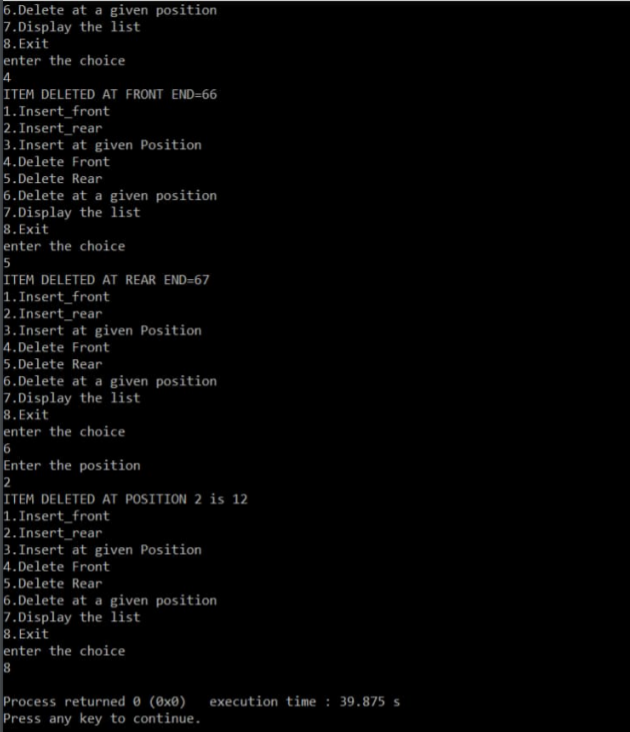
break;

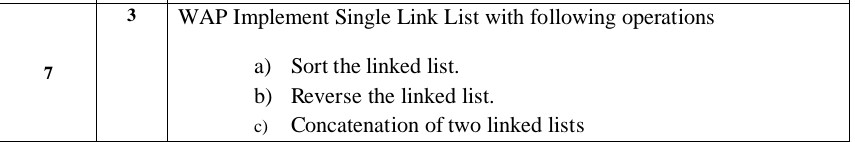
}

}

}







#include<stdio.h>

#include<process.h>

int n,i;

struct node

{

int info;

struct node \*link;

};

typedef struct node \*NODE;

NODE a,b;

NODE getnode()

{

NODE x;

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

if(x==NULL)

{

printf("Memory is 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 first;

}

if(pos==1)

{

temp->link=first;

first=temp;

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("CANNOT DELETE AS LIST IS EMPTY\n");

return first;

}

temp=first;

temp=temp->link;

printf("ITEM DELETED AT FRONT END=%d\n",first->info);

free(first);

return temp;

}

NODE delete\_rear(NODE first)

{

NODE cur,prev;

if(first==NULL)

{

printf("CANNOT DELETE AS LIST IS EMPTY\n");

return first;

}

if(first->link==NULL)

{

printf("ITEM DELETED=%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=%d\n",cur->info);

free(cur);

prev->link=NULL;

return first;

}

NODE delete\_pos(int pos,NODE first)

{

NODE cur;

NODE prev;

int count,flag=0;

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)

{

flag=1;

break;

}

count++;

prev=cur;

cur=cur->link;

}

if(flag==0)

{

printf("invalid position\n");

return first;

}

printf("ITEM DELETED AT POSITION %d is %d\n",pos,cur->info);

prev->link=cur->link;

freenode(cur);

return first;

}

NODE sort\_asc(NODE first)

{

int tmp;

NODE cur,next;

cur=first;

next=NULL;

if(first==NULL){

printf("List is empty\n");

return;

}

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;

}

NODE sort\_desc(NODE first)

{

int tmp;

NODE cur,next;

cur=first;

next=NULL;

if(first==NULL){

printf("List is empty\n");

return;

}

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;

}

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 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;

}

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);

}

}

void main()

{

int item,choice,pos;

NODE first=NULL;

for(;;)

{

printf("1.Insert\_front\n2.Insert\_rear\n3.Insert at given Position\n4.Delete Front\n5.Delete Rear\n6.Delete at a given position\n7.Display the list\n8.Sort in Ascending\n9.Sort in Descending\n10.Reverse\n11.Concat\n12.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);

first=insert\_front(first,item);

break;

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

scanf("%d",&item);

first=insert\_rear(first,item);

break;

case 3:printf("enter the item to be inserted at given position\n");

scanf("%d",&item);

printf("enter the position\n");

scanf("%d",&pos);

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

break;

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

break;

case 5:first=delete\_rear(first);

break;

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

scanf("%d",&pos);

first=delete\_pos(pos,first);

break;

case 7:display(first);

break;

case 8:first=sort\_asc(first);

break;

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

break;

case 10:first=reverse(first);

display(first);

break;

case 11:printf("Enter number of nodes in List2\n");

scanf("%d",&n);

a=NULL;

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

{

printf("Enter Item:\n");

scanf("%d",&item);

a=insert\_rear(a,item);

}

/\*printf("Enter number of nodes in List3\n");

scanf("%d",&n);

b=NULL;

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

{

printf("Enter Item:\n");

scanf("%d",&item);

b=insert\_rear(b,item);

}\*/

first=concat(first,a);

display(first);

break;

default:exit(0);

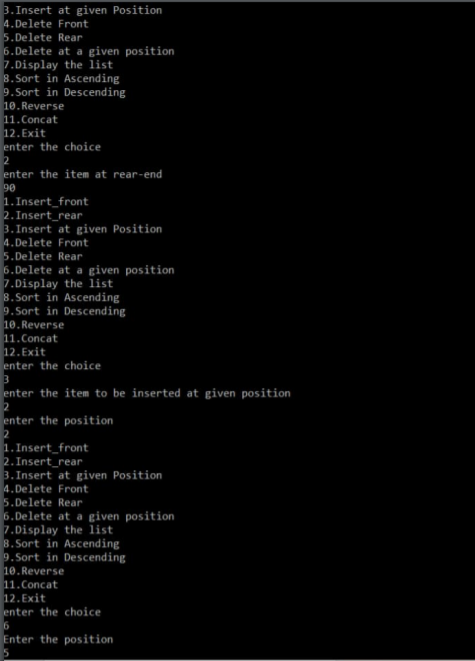
break;

}

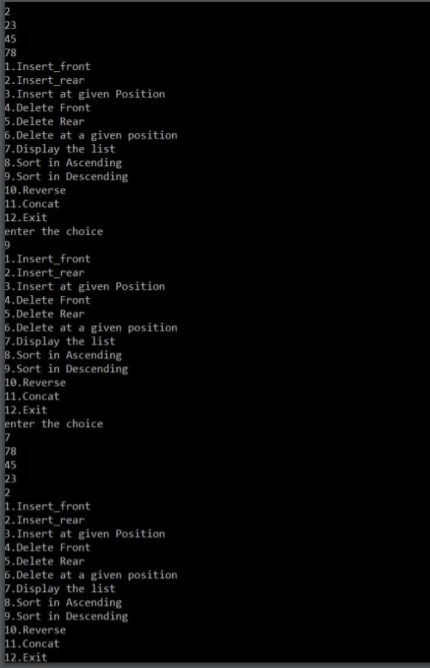
}

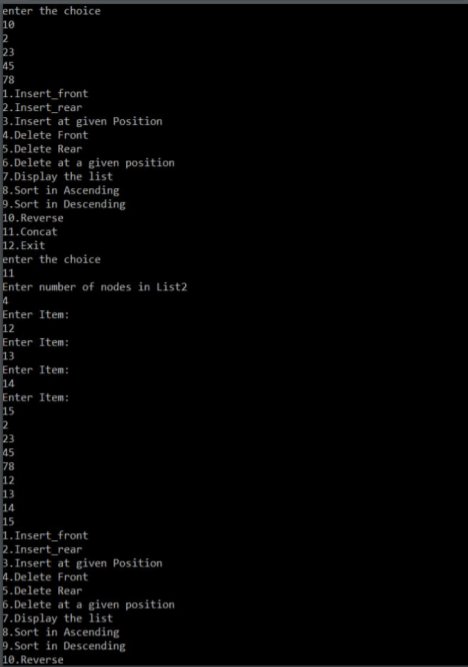
}

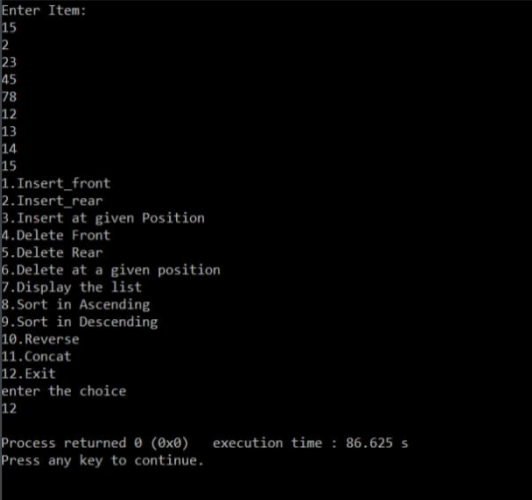


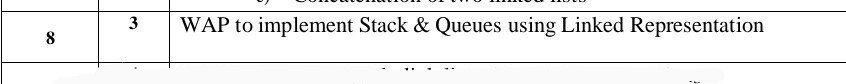












#include<stdio.h>

#include<process.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 is 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 delete\_front(NODE first)

{

NODE temp;

if(first==NULL)

{

printf("CANNOT DELETE AS LIST IS EMPTY\n");

return first;

}

temp=first;

temp=temp->link;

printf("ITEM DELETED AT FRONT END=%d\n",first->info);

free(first);

return temp;

}

NODE delete\_rear(NODE first)

{

NODE cur,prev;

if(first==NULL)

{

printf("CANNOT DELETE AS LIST IS EMPTY\n");

return first;

}

if(first->link==NULL)

{

printf("ITEM DELETED=%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=%d\n",cur->info);

free(cur);

prev->link=NULL;

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);

}

}

void main()

{

int item,choice,pos;

NODE first=NULL,first1=NULL;

for(;;)

{

printf("1.Push in Stack\n2.Insert in Queue\n3.Pop from Stack\n4.Delete from rear in Queue\n5.Display Stack\n6.Display Queue\n7.Exit\n");

printf("enter the choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("Enter item to be pushed in Stack:\n");

scanf("%d",&item);

first=insert\_front(first,item);

break;

case 2:printf("Enter item to be inserted in Queue:\n");

scanf("%d",&item);

first1=insert\_front(first1,item);

break;

case 3:first=delete\_front(first);

break;

case 4:first1=delete\_rear(first1);

break;

case 5:display(first);

break;

case 6:display(first1);

break;

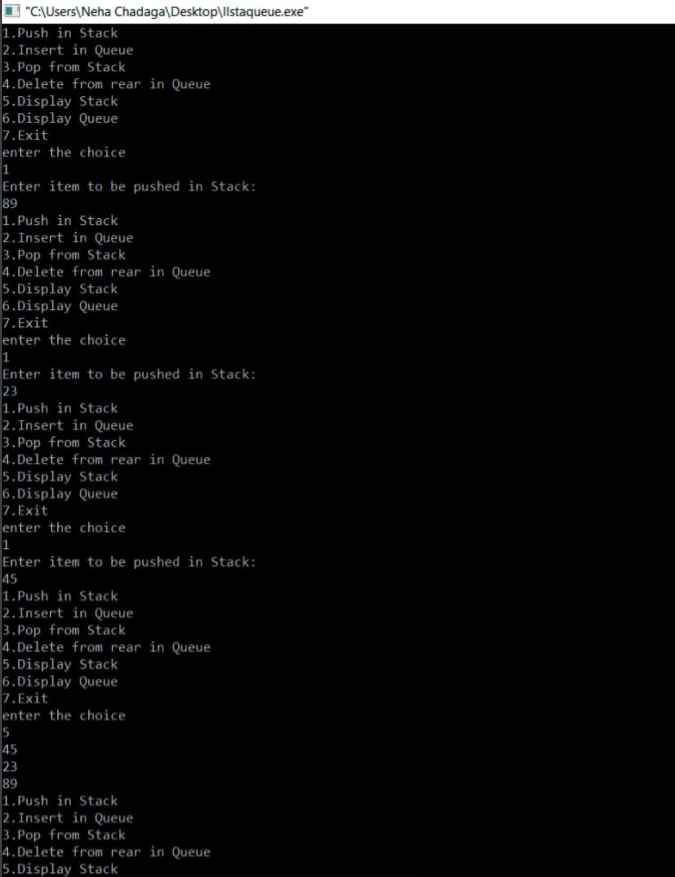
default:exit(0);

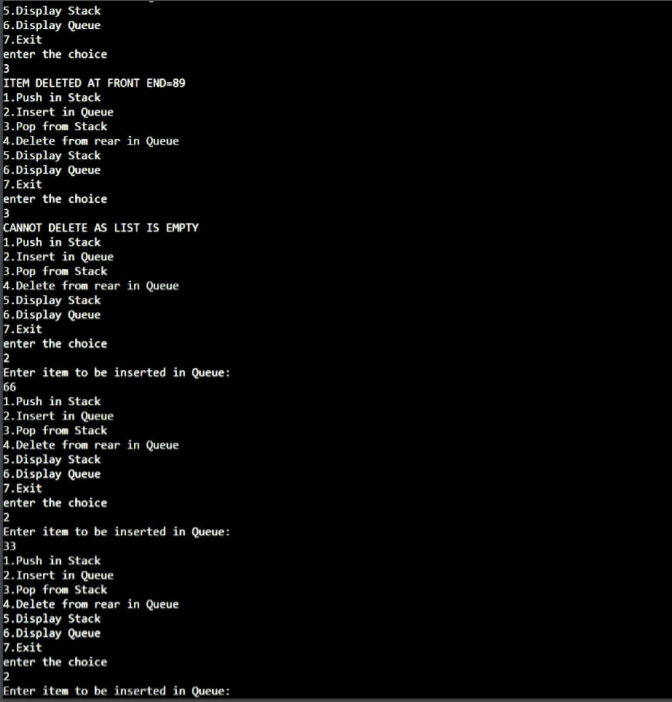
break;

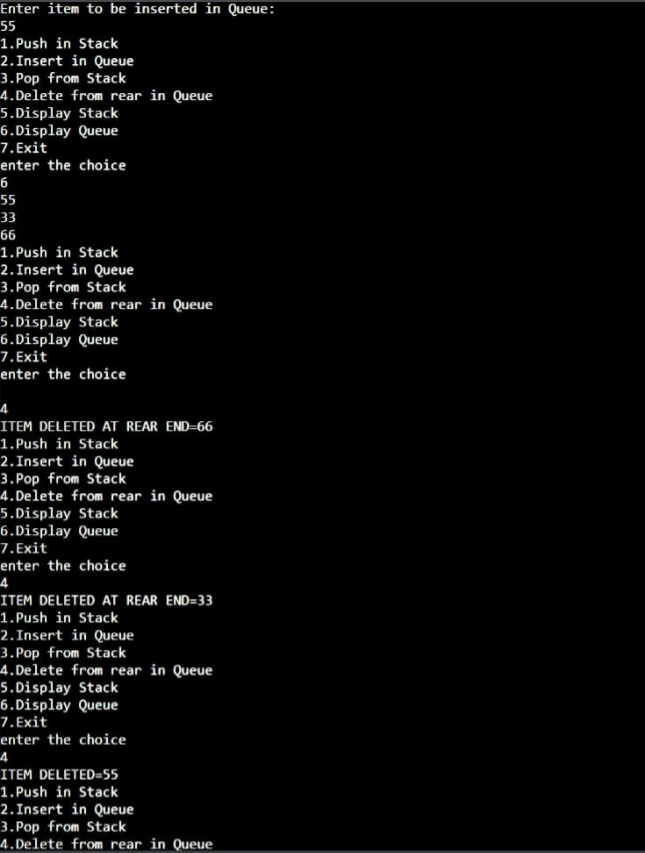
}

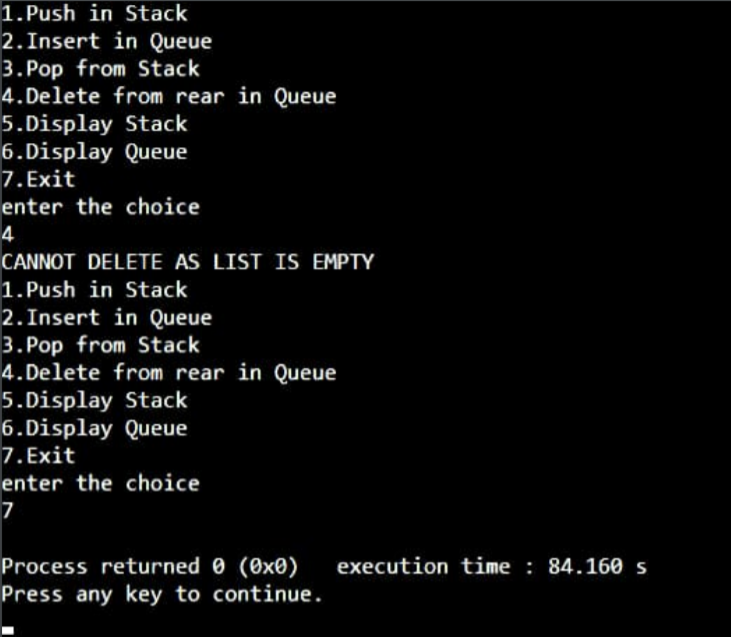
}

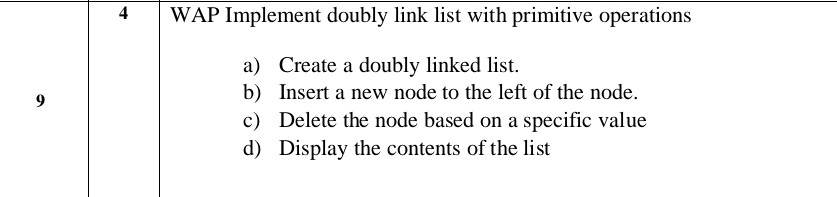
}











#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 is 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 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 ITEM FOUND IN 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;

}

NODE ddeletepos(int pos, NODE head)

{

NODE cur,prev,temp;

int count=1,flag=0;

if (head->rlink==head)

{

printf("LIST IS EMPTY.\n");

return head;

}

if(pos==1)

{

cur=head->rlink;

prev=cur->rlink;

head->rlink=prev;

prev->llink=head;

printf("THE NODE DELETED IS %d",cur->info);

free(cur);

return head;

}

prev=head;

cur=head->rlink;

while (cur!=head)

{

if (count==pos)

{

flag=1;

break;

}

count++;

cur=cur->rlink;

prev=cur->llink;

}

if(flag==0)

{

printf("Invalid Position.\n");

return head;

}

printf("ITEM DELETED AT POSITION %d is %d\n",pos,cur->info);

temp=cur->rlink;

prev->rlink=cur->rlink;

temp->llink=prev;

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;

}

}

int main()

{

NODE head;

int item, choice,key,pos;

head=getnode();

head->llink=head;

head->rlink=head;

for(;;)

{

printf("1.Insert front\n2.Insert rear\n3.Insert at Left Position\n4.Delete at specified Position\n5.Display\n6.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:printf("Enter the key element, the left of which an item is to be inserted:\n");

scanf("%d",&key);

head=dinsert\_leftpos(key,head);

break;

case 4:printf("Enter position of node to be deleted:\n");

scanf("%d",&pos);

head=ddeletepos(pos,head);

break;

case 5:ddisplay(head);

break;

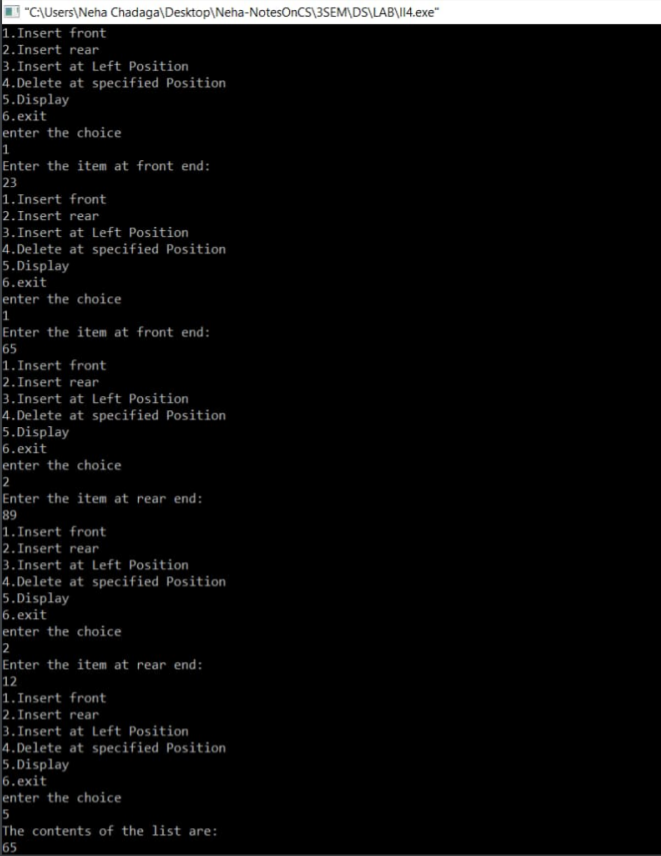
default:exit(0);

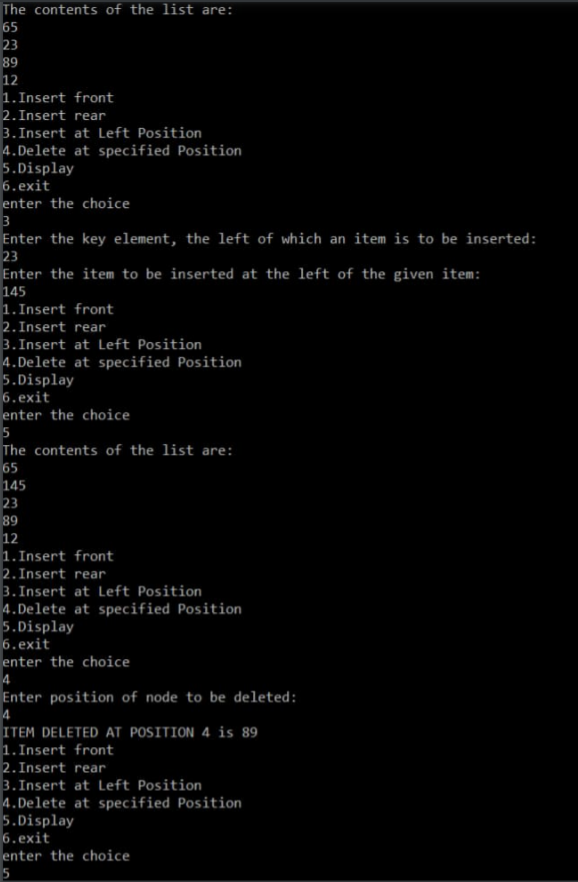
}

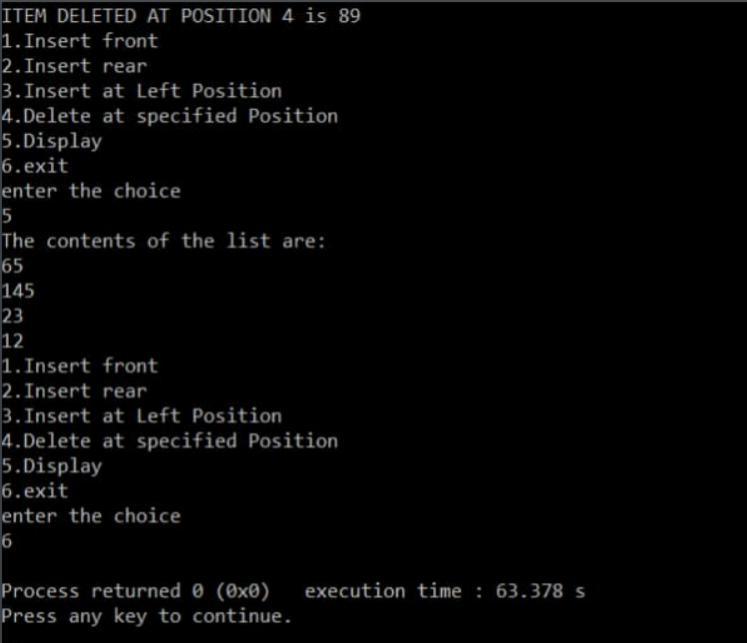
}

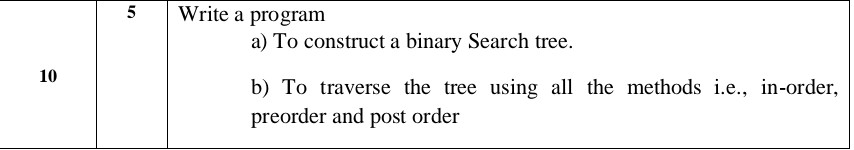
return 0;

}









#include<stdio.h>

#include<process.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 is 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);

}

}

NODE delete(NODE root,int item)

{

NODE cur,parent,q,suc;

if(root==NULL)

{

printf("Tree is Empty.\n");

return root;

}

parent=NULL;

cur=root;

while(cur!=NULL&&item!=cur->info)

{

parent=cur;

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

}

if(cur==NULL)

{

printf("Not Found.\n");

return root;

}

if(cur->llink==NULL)

q=cur->rlink;

else if(cur->rlink==NULL)

q=cur->llink;

else

{

suc=cur->rlink;

while(suc->llink!=NULL)

suc=suc->llink;

suc->llink=cur->llink;

q=cur->rlink;

}

if(parent==NULL)

return q;

if(cur==parent->llink)

parent->llink=q;

else

parent->rlink=q;

freenode(cur);

return root;

}

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);

}

}

void main()

{

int item,choice;

NODE root=NULL;

for(;;)

{

printf("1.Insert\n2.Display\n3.Preorder\n4.Postorder\n5.Inoreder\n6.Delete\n7.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:printf("PREORDER\n");

preorder(root);

break;

case 4:printf("POSTORDER\n");

postorder(root);

break;

case 5:printf("INORDER\n");

inorder(root);

break;

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

scanf("%d",&item);

root=delete(root,item);

break;

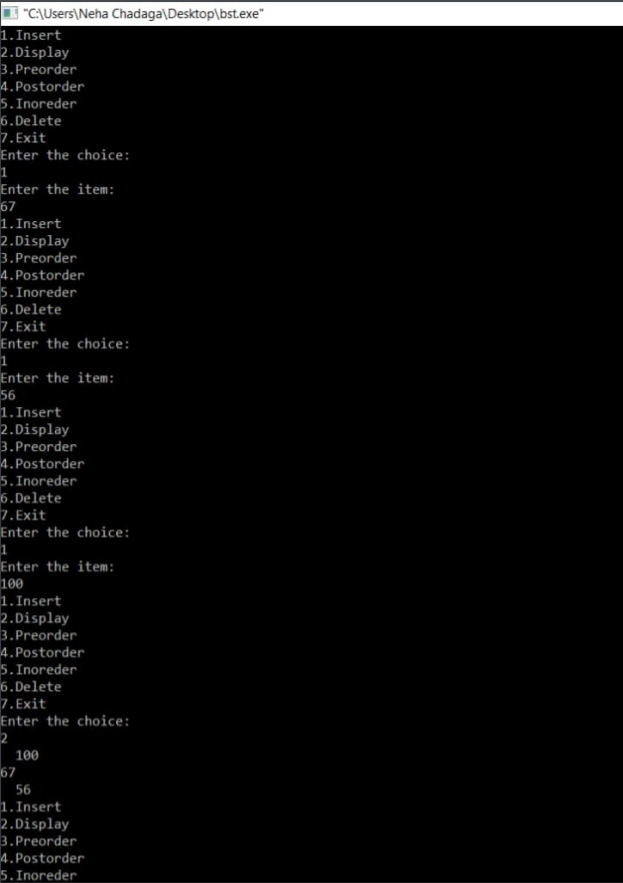
default:exit(0);

break;

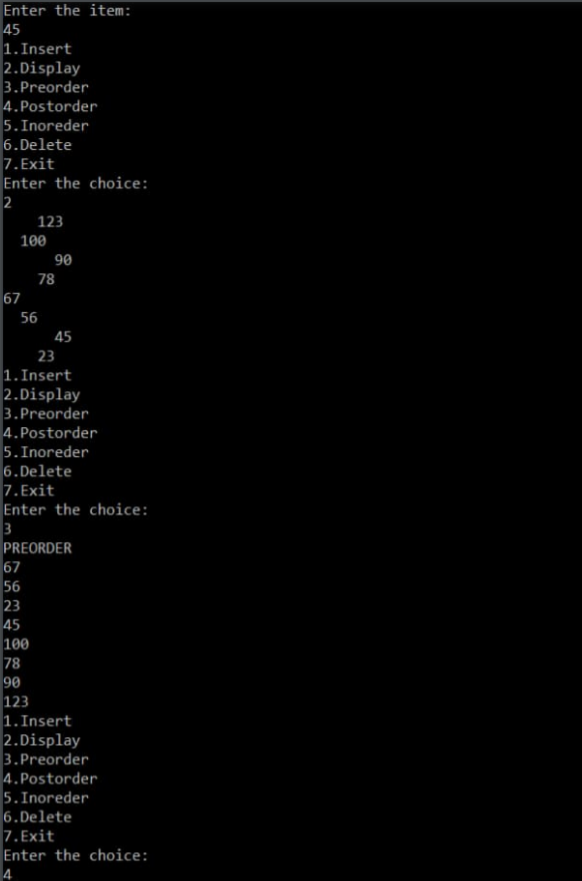
}

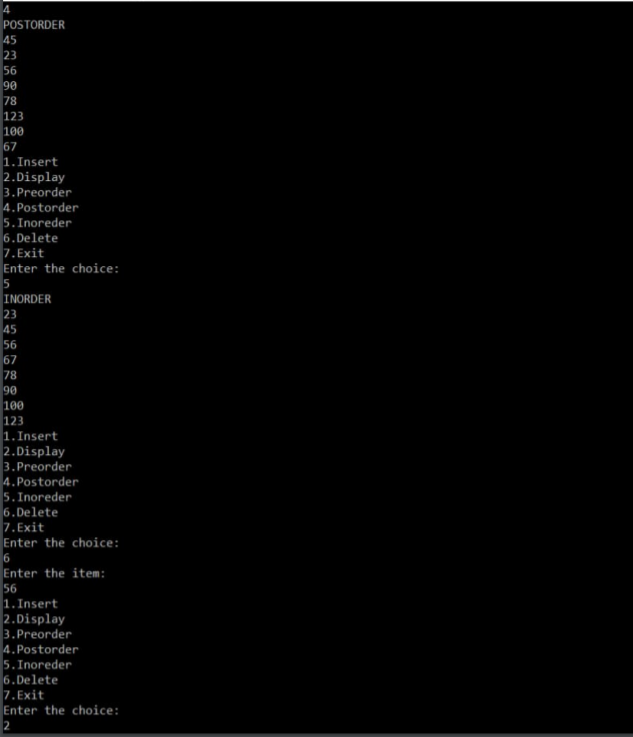
}

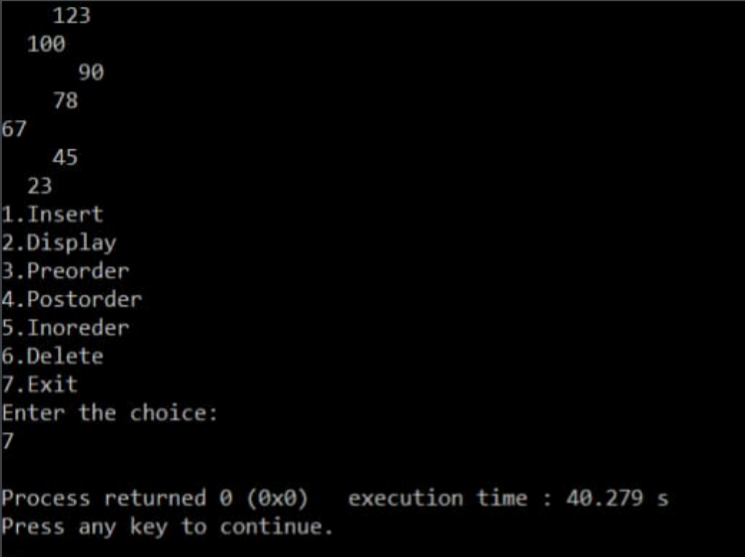
}











# END OF REPORT