VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“JnanaSangama”, Belgaum -590014, Karnataka.**



**DATA STRUCTURE LAB RECORD**

***Submitted by***

**HEMANG SINGH (1BM19CS061)**

***Under the Guidance of***

**Prof. SHEETAL VA**

**Assistant Professor, BMSCE**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**

**B.M.S. COLLEGE OF ENGINEERING**

### (Autonomous Institution under VTU)

**BENGALURU-560019**

**Sep-2020 to Jan-2021**

1

# B. M. S. College of Engineering,

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

## Department of Computer Science and Engineering



**CERTIFICATE**

This is to certify that the LAB RECORD carried out by **HEMANG SINGH (1BM19CS061)** who is the bonafide students of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visveswaraiah Technological University, Belgaum during the year 2020-2021. The lab report has been approved as it satisfies the academic requirements in respect of **DATA STRUCTURE LAB RECORD (19CS3PCDST)** work prescribed for the said degree.

Signature of the Guide Signature of the HOD

Prof. Prof. Sheelal VA Dr. Umadevi V

Assistant Professor Associate Prof.& Head, Dept. of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

External Viva

Name of the Examiner Signature with date

1.

2.

2

# LAB-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<conio.h> #define STACK\_SIZE 3 int top=-1; int s[10];

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){

printf("stack is empty\n");

}

else

return s[top--];

}

void display(){ int i;

if(top==-1){

printf("stack is empty\n"); return;

}

printf("items in the stack are: "); for(i=top;i>=0;i--){

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

}

}

main(){

int choice;

int item\_deleted; for(;;){

printf("\n1:push 2:pop 3:display 4:exit\n"); printf("Enter your choice: "); scanf("%d",&choice);

switch(choice){

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

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);

}

}

}



**LAB-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()

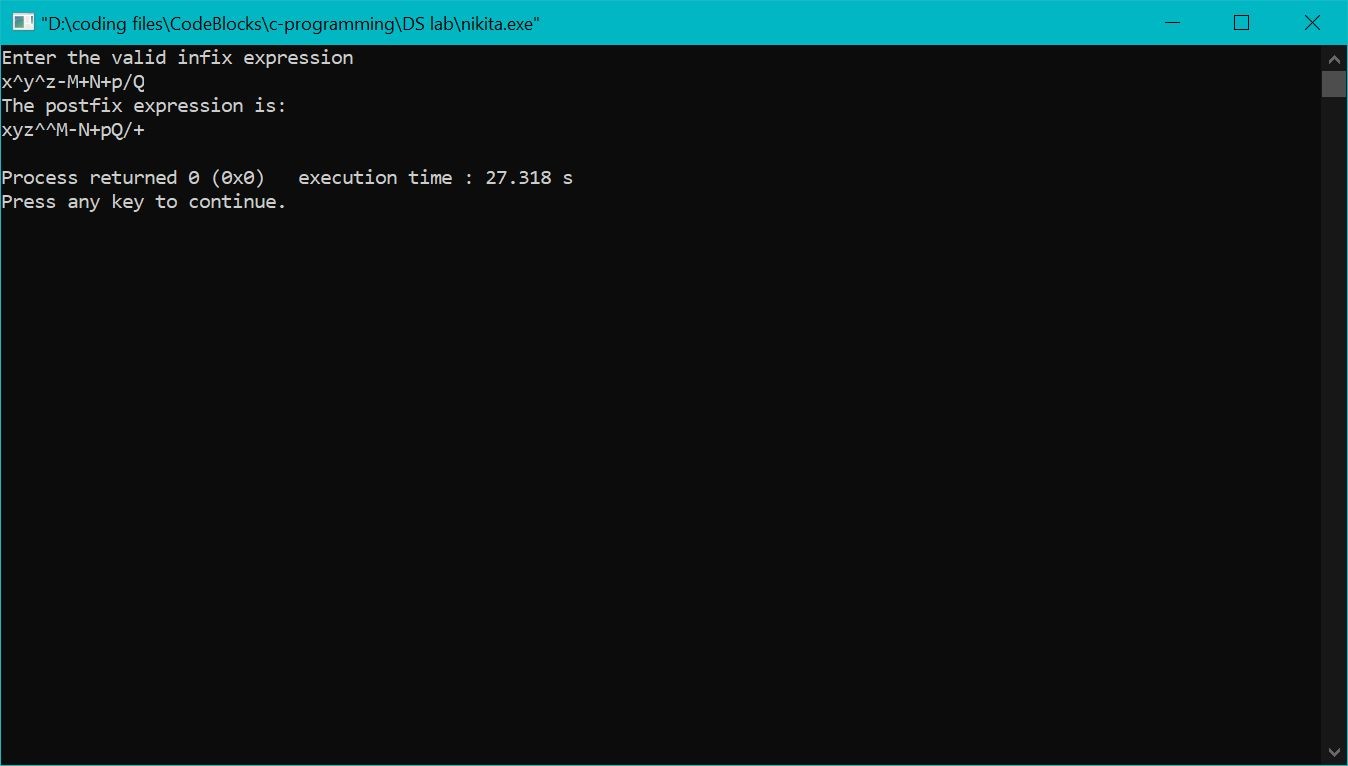
{

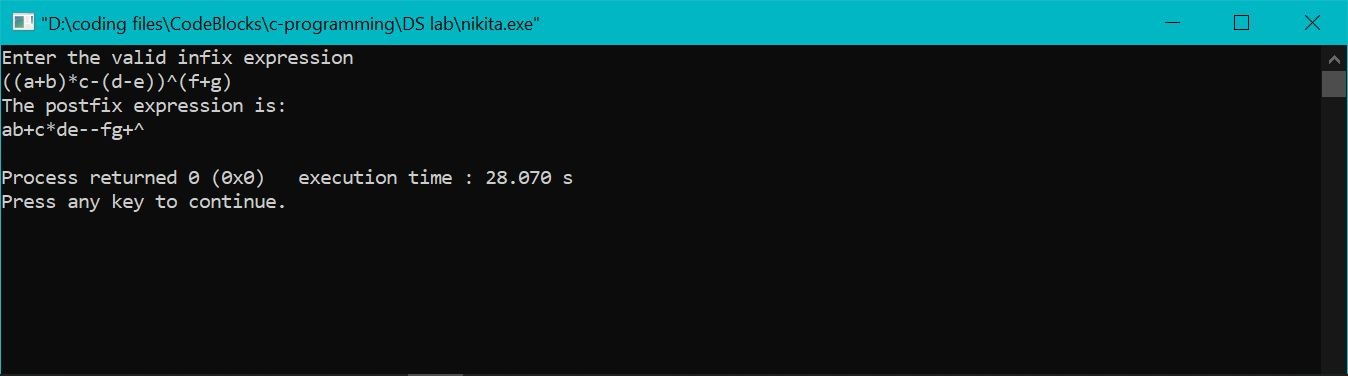
char infix[20]; char postfix[20];

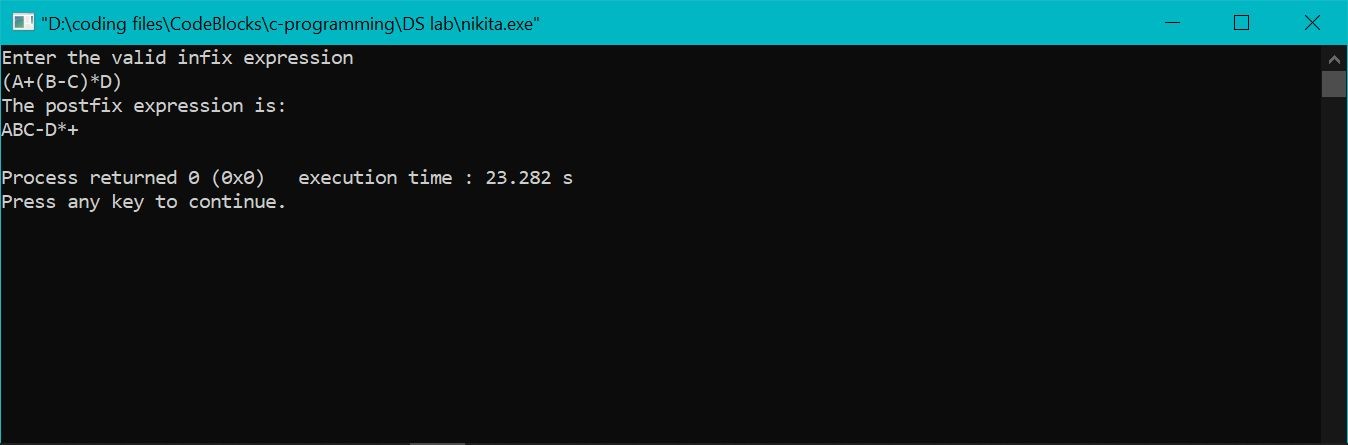
printf("Enter the valid infix expression\n");

scanf("%s",infix); infix\_postfix(infix,postfix); printf("The postfix expression is:\n"); printf("%s\n",postfix);

}







## LAB - 3

/\*Write a Program to simulate the working of queue of integers using an array. Provide the following

operations.

1. Insert Rear
2. Delete Front
3. Display the contents of queue

The program should print the appropriate messages for a queue empty and queue full condition.

\*/

#include<stdio.h> #include<conio.h> #include<process.h> #define QUE\_SIZE 3

int item,front=0,rear=-1,q[10]; 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 Queue\*\* \n"); for(i=front;i<=rear;i++){

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

}

}

void main(){

int choice; for(;;){

printf("\n1.Insert Rear \n2.Delete front \n3.Display \n4.Exit\n");

printf("Enter the choice\n"); scanf("%d",&choice); switch(choice){

case 1: printf("Enter the items 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;

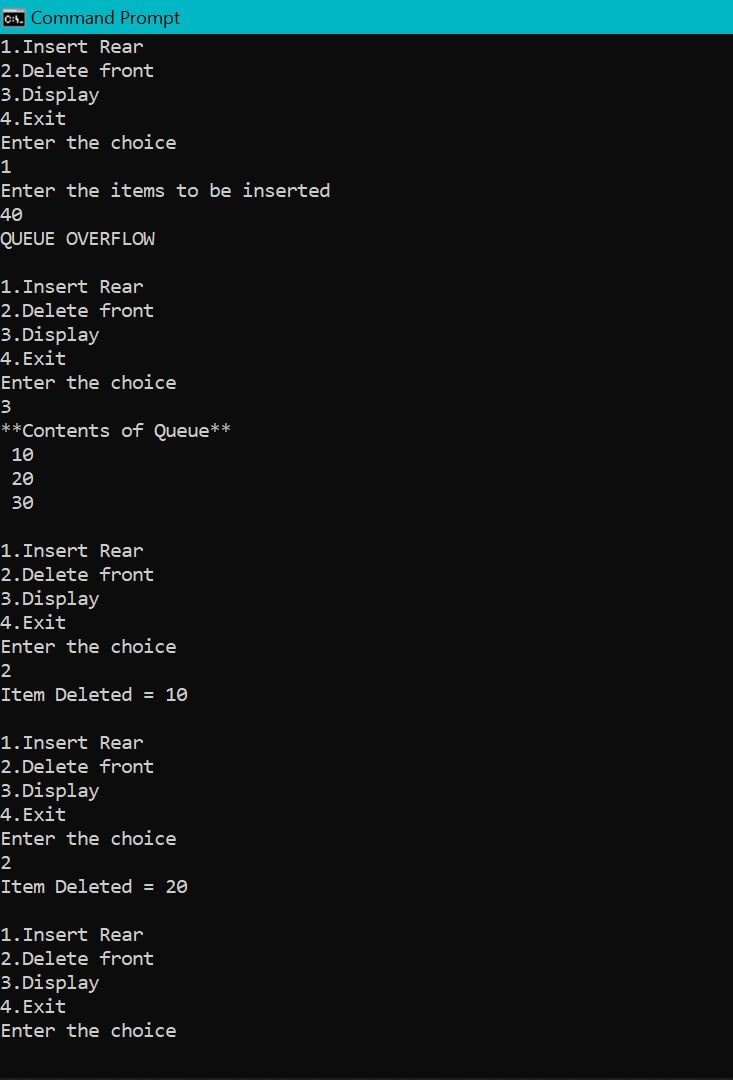
default: exit(0);

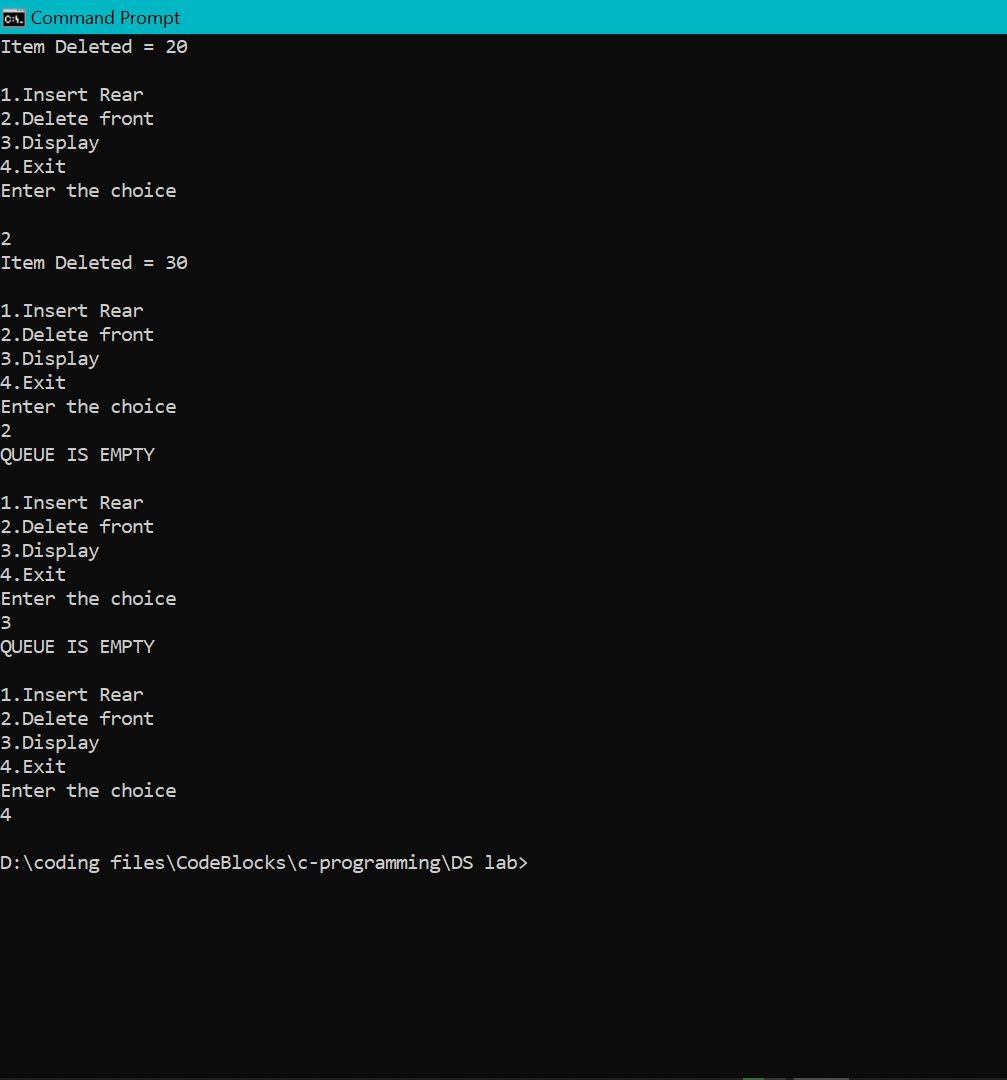
}

}

}







# LAB-4

### Double Ended Queue

#include<stdio.h> #include<stdlib.h> #define qsize 5

int f=0,r=-1,ch; int item,q[10];

int isfull()

{

return(r==qsize-1)?1:0;

}

int isempty()

{

return(f>r)?1:0;

}

void insert\_rear()

{

if(isfull())

{

printf("queue overflow\n"); return;

}

r=r+1; q[r]=item;

}

void delete\_front()

{

if(isempty())

{

printf("queue empty\n"); return;

}

printf("item deleted is %d\n",q[(f)++]); if(f>r)

{

f=0;

r=-1;

}

}

void insert\_front()

{

if(f!=0)

{

f=f-1; q[f]=item; return;

}

else if((f==0)&&(r==-1))

{

q[++(r)]=item; return;

}

else

printf("insertion not possible\n");

}

void delete\_rear()

{

if(isempty())

{

printf("queue is empty\n"); return;

}

printf("item deleted is %d\n",q[(r)--]); if(f>r)

{

f=0; r=-1;

}

}

void display()

{

int i; if(isempty())

{

printf("queue empty\n"); return;

}

for(i=f;i<=r;i++)

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

}

void main()

{

for(;;)

{

printf("1.insert\_rear\n2.insert\_front\n3.delete\_rear\n4.delete\_front\n5.display\n6.exit\n"); printf("enter choice\n");

scanf("%d",&ch); switch(ch)

{

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

scanf("%d",&item); insert\_rear(); break;

case 2:printf("enter the item\n");

scanf("%d",&item); insert\_front(); break;

case 3:delete\_rear();

break; case 4:delete\_front();

break; case 5:display();

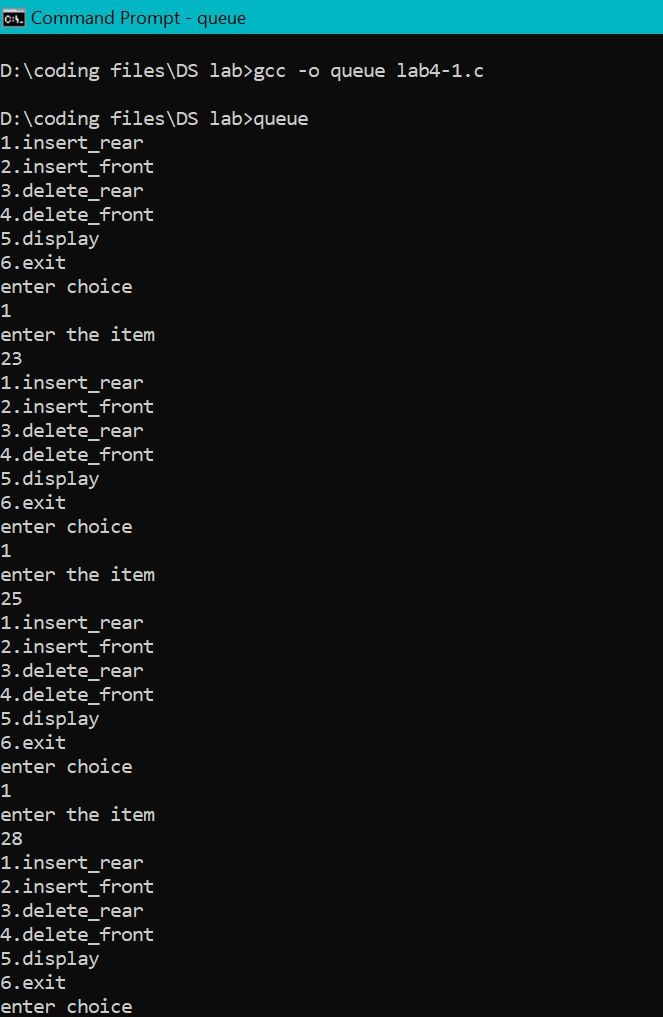
break;

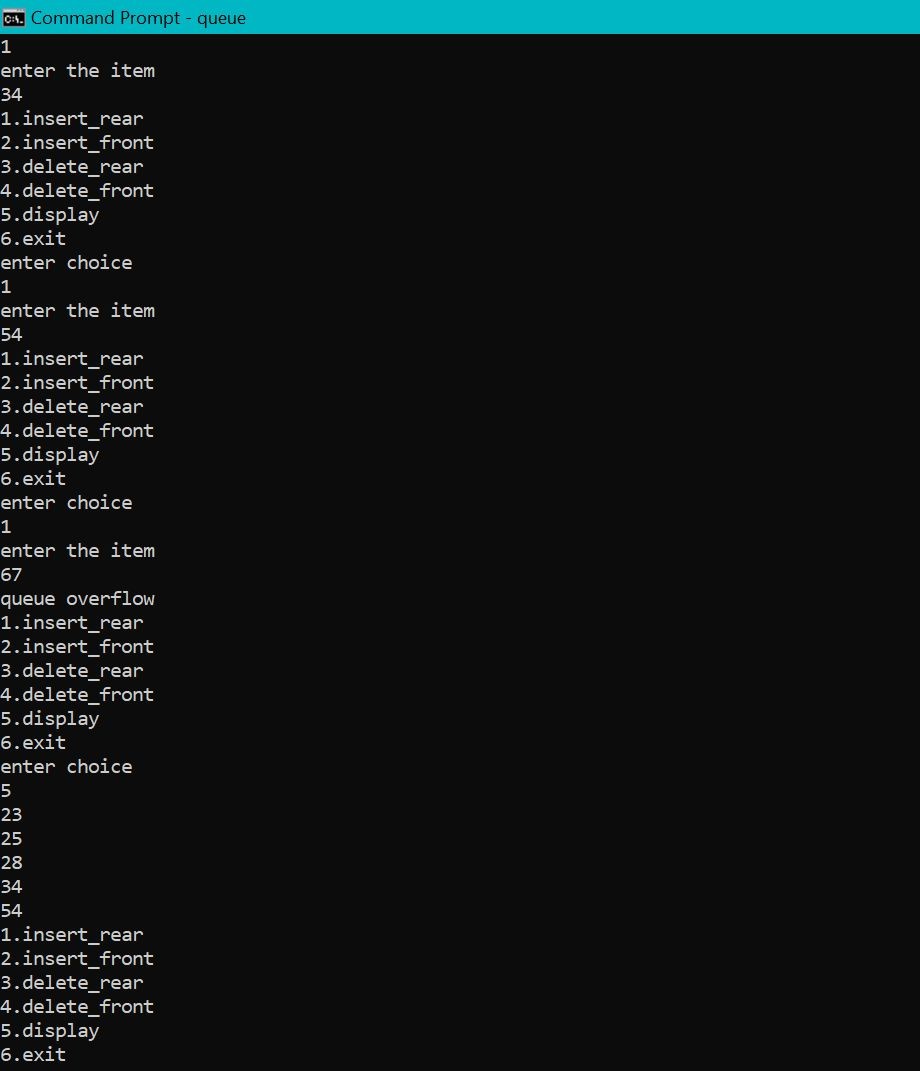
default:exit(0);

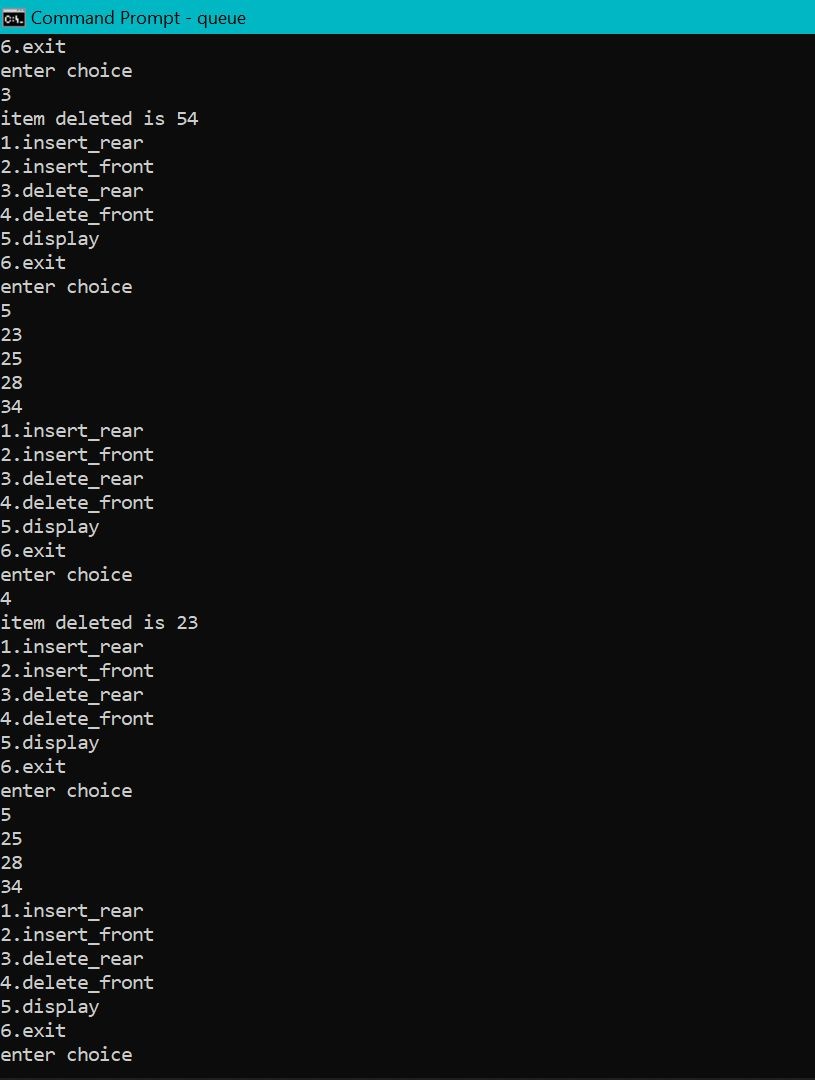
}

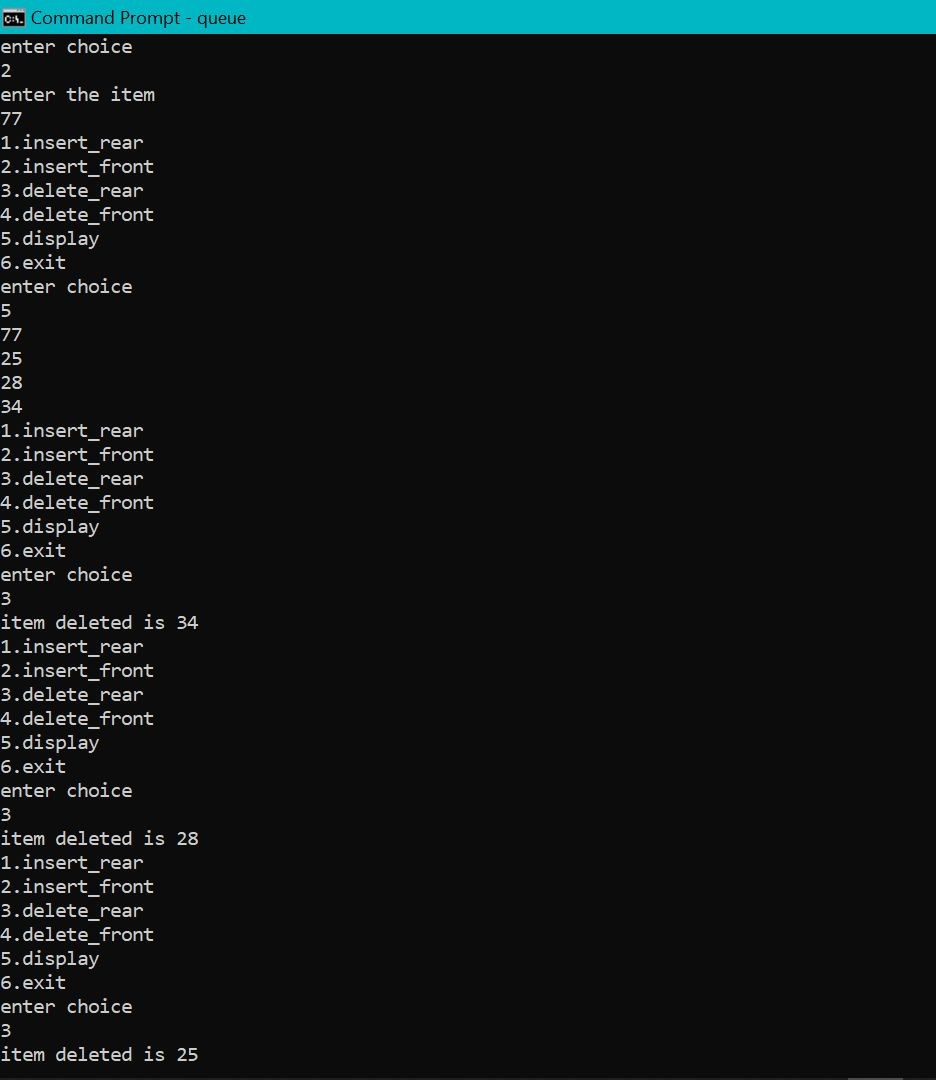
}

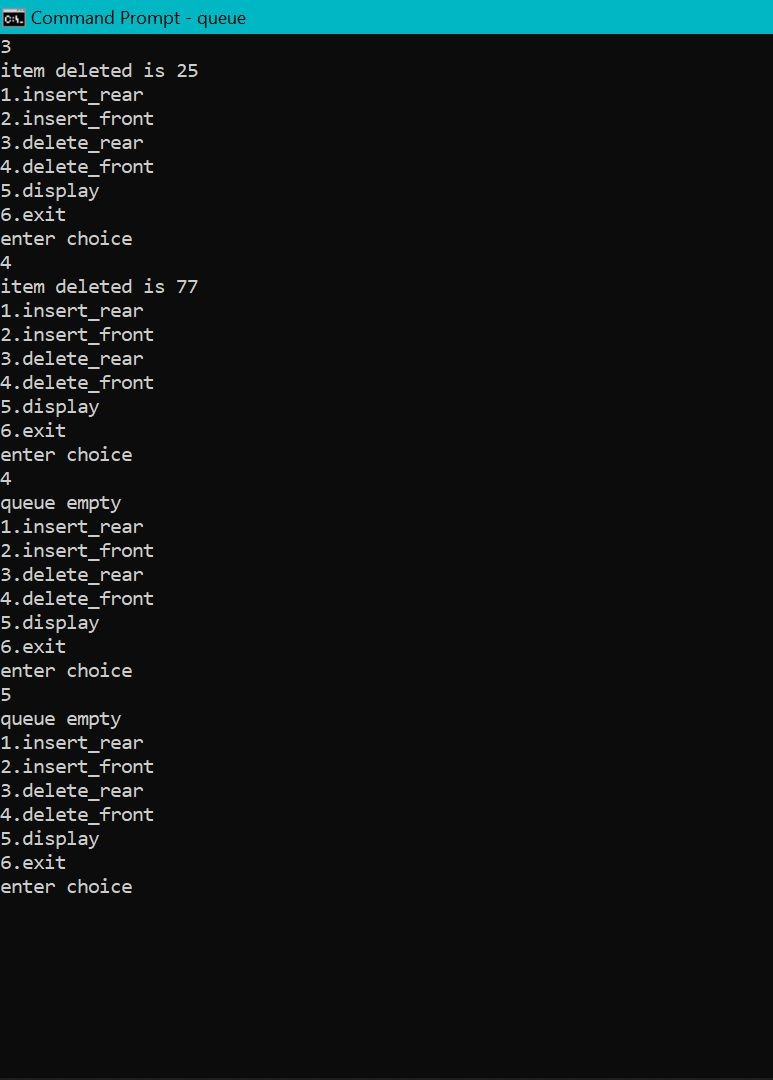
}











### Input and Output restricted queue

# include<stdio.h> # define Size 5

int deque\_arr[Size];

int front = -1,rear = -1;

void insert\_rear()

{

int added\_item;

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

{ printf("\*\*\*Queue Overflow\*\*\*\n"); return;

}

if (front == -1)

{

front = 0;

rear = 0;

}

else

if(rear == Size-1) rear = 0;

else

rear = rear+1;

printf("Enter the element for adding in queue : "); scanf("%d",&added\_item);

deque\_arr[rear] = added\_item ;

}

void insert\_front()

{ int added\_item;

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

{ printf("Queue Overflow \n"); return;

}

if (front == -1)

{ front = 0; rear = 0;

}

else

if(front== 0) front=Size-1;

else

front=front-1;

printf("Enter the element for adding in queue : "); scanf("%d", &added\_item);

deque\_arr[front] = added\_item ;

}

void delete\_front()

{ if (front == -1)

{ printf("Queue Underflow\n"); return ;

}

printf("Element deleted from queue is : %d\n",deque\_arr[front]); if(front == rear)

{ front = -1; rear=-1;

}

else

if(front == Size-1) front = 0;

else

front = front+1;

}

void delete\_rear()

{

if (front == -1)

{

printf("Queue Underflow\n"); return ;

}

printf("Element deleted from queue is : %d\n",deque\_arr[rear]); if(front == rear)

{

front = -1; rear=-1;

}

else

if(rear == 0) rear=Size-1;

else

rear=rear-1;

}

void display\_queue()

{

int front\_pos = front,rear\_pos = rear; if(front == -1)

{ printf("Queue is empty\n");

return;

}

printf("Queue elements :\n"); if( front\_pos <= rear\_pos )

{

while(front\_pos <= rear\_pos)

{

printf("%d \n",deque\_arr[front\_pos]); front\_pos++;

}

}

else

{

while(front\_pos <= Size-1)

{ printf("%d \n",deque\_arr[front\_pos]); front\_pos++;

}

front\_pos = 0; while(front\_pos <= rear\_pos)

{

printf("%d \n",deque\_arr[front\_pos]); front\_pos++;

}

}

printf("\n");

}

/\*Input Queue\*/

void input\_que()

{ int choice; do

{ printf("1.Insert at rear\n2.Delete from front\n3.Delete from rear\n4.Display\n5.Quit\n"); printf("Enter your choice :");

scanf("%d",&choice); switch(choice)

{ case 1:

insert\_rear(); break;

case 2: delete\_front(); break;

case 3: delete\_rear(); break;

case 4: display\_queue(); break;

case 5:

break; default:

printf("Wrong choice\n");

}

}

while(choice!=5);

}

/\*Output Queue\*/ void output\_que()

{ int choice; do

{ printf("1.Insert at rear\n2.Insert at front\n3.Delete from front\n4.Display\n5.Quit\n"); printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1: insert\_rear(); break;

case 2: insert\_front(); break;

case 3: delete\_front(); break;

case 4: display\_queue(); break;

case 5:

break; default:

printf("Wrong choice\n");

}

}while(choice!=5);

}

main()

{ int choice;

printf("1.Input restricted dequeue\n2.Output restricted dequeue\n"); printf("Enter your choice : ");

scanf("%d",&choice); switch(choice)

{

case 1 : input\_que(); break;

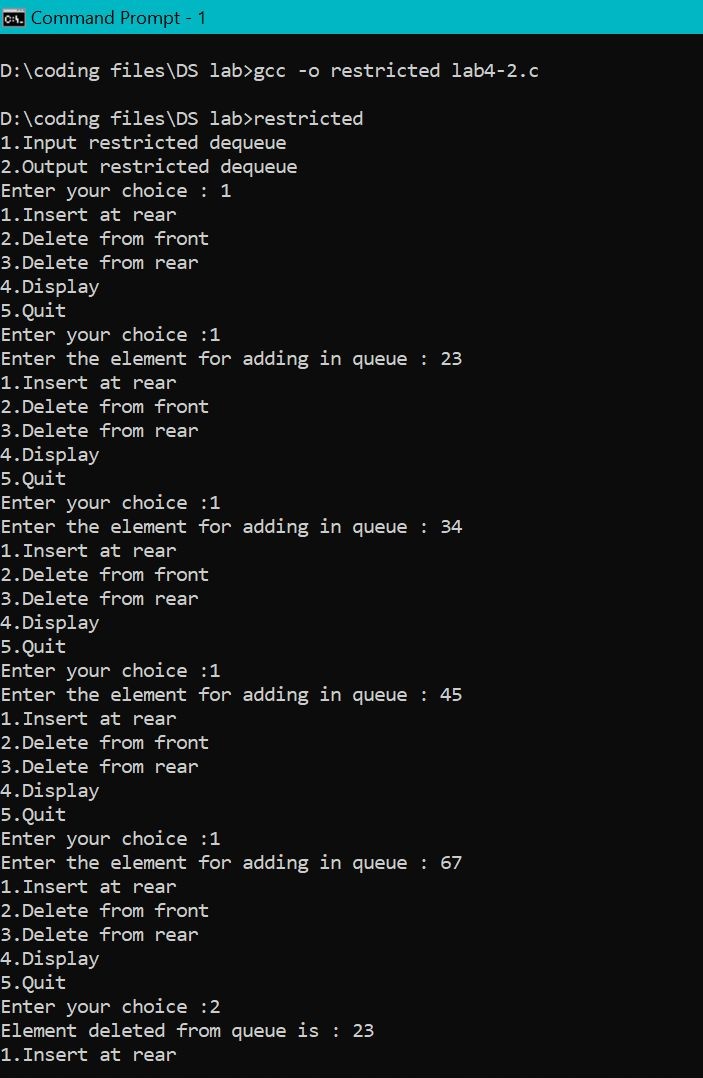
case 2: output\_que(); break;

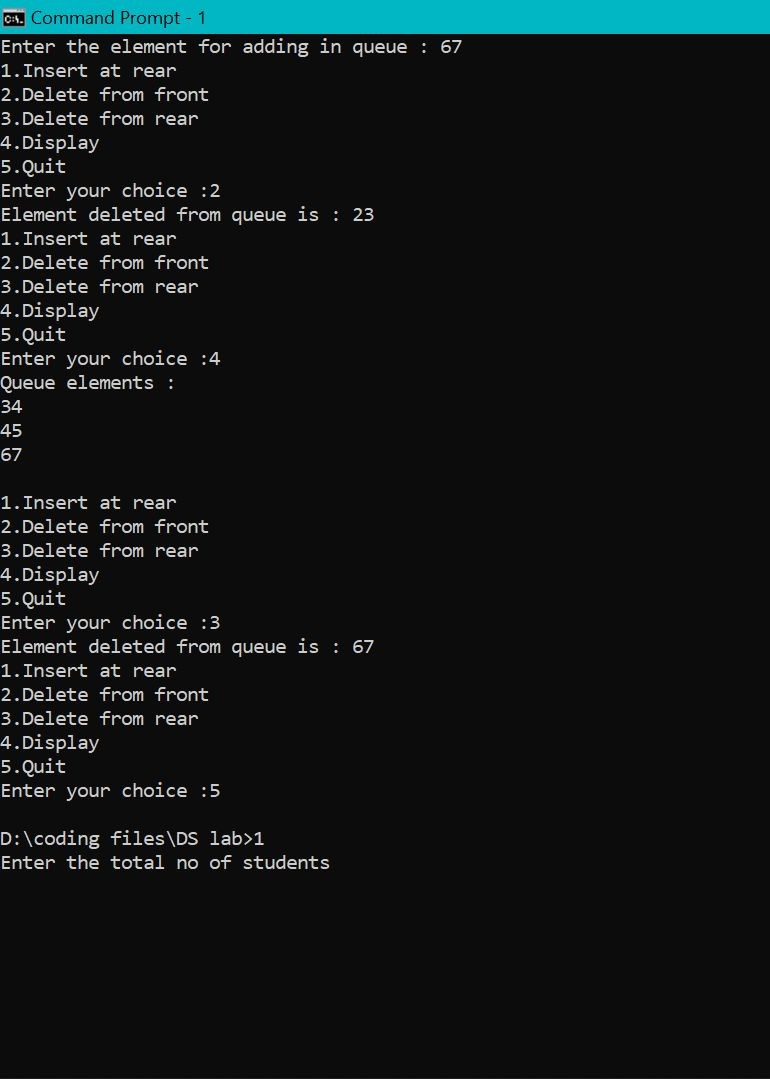
default:

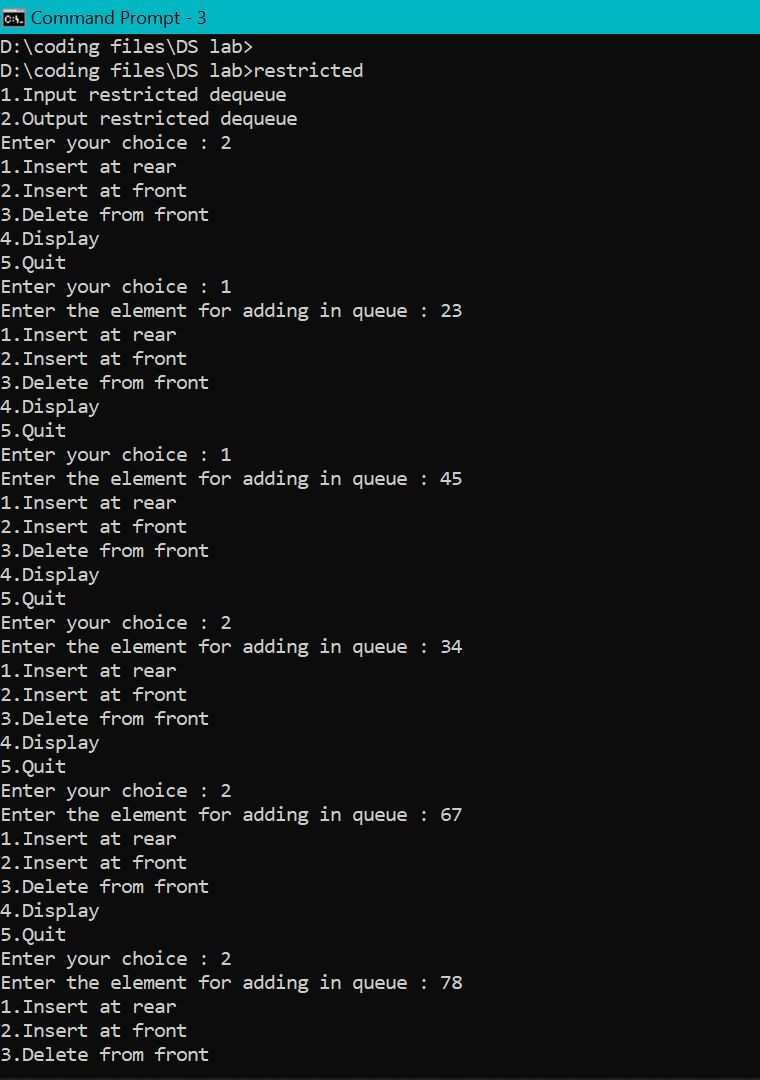
printf("Wrong choice\n");

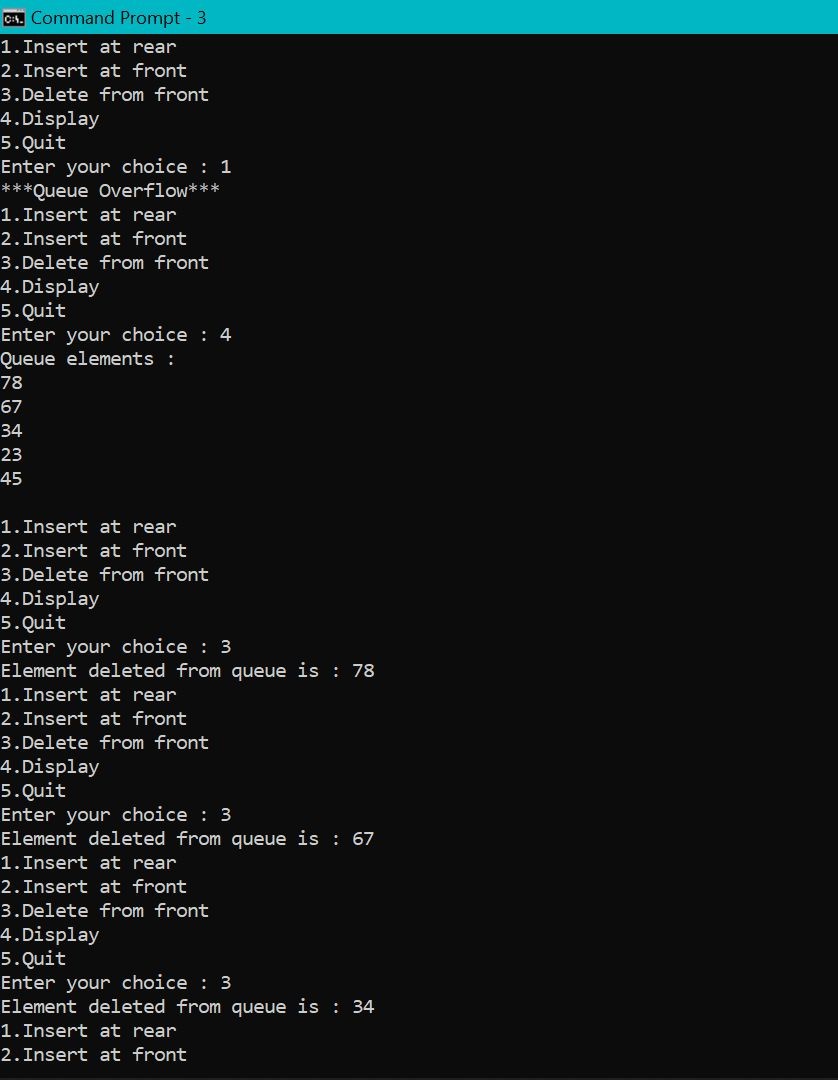
}

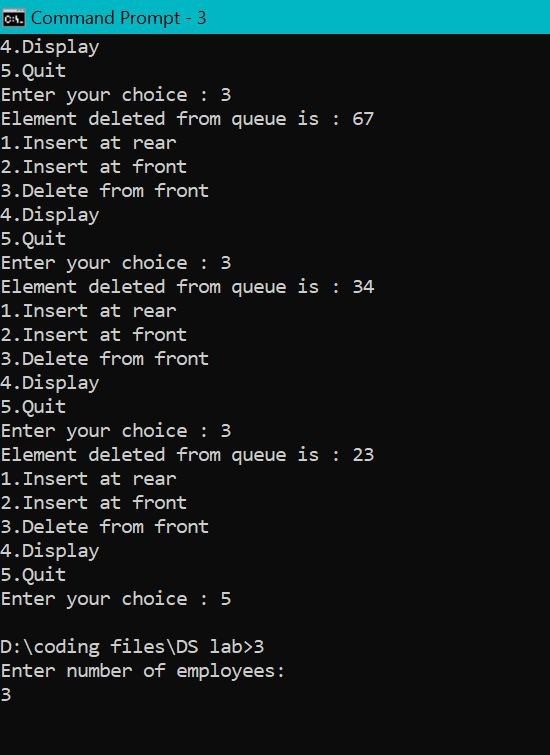
}











# LAB - 5 & 6

/\*C-Program to implement LINKED LIST with functions insertion & deletion with specified position

LAB 5 & 6

\*/

#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("mem full\n"); exit(0);

}

return x;

}

void freenode(NODE x){ free(x);

}

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 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 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\_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 given position is %d\n",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,key,pos; int count=0;

NODE first=NULL; for(;;){

printf("\n1:Insert\_rear\n2:Delete\_rear\n3:insert\_info\_position\n4:Delete\_info\_position\n5:Display

\_list\n6:Exit\n");

printf("enter the choice\n"); scanf("%d",&choice); switch(choice){

case 1:printf("enter the item at rear-end\n"); scanf("%d",&item); first=insert\_rear(first,item);

break;

case 2:first=delete\_rear(first); 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:printf("enter the position\n"); scanf("%d",&pos); first=delete\_pos(pos,first); break;

case 5:display(first);

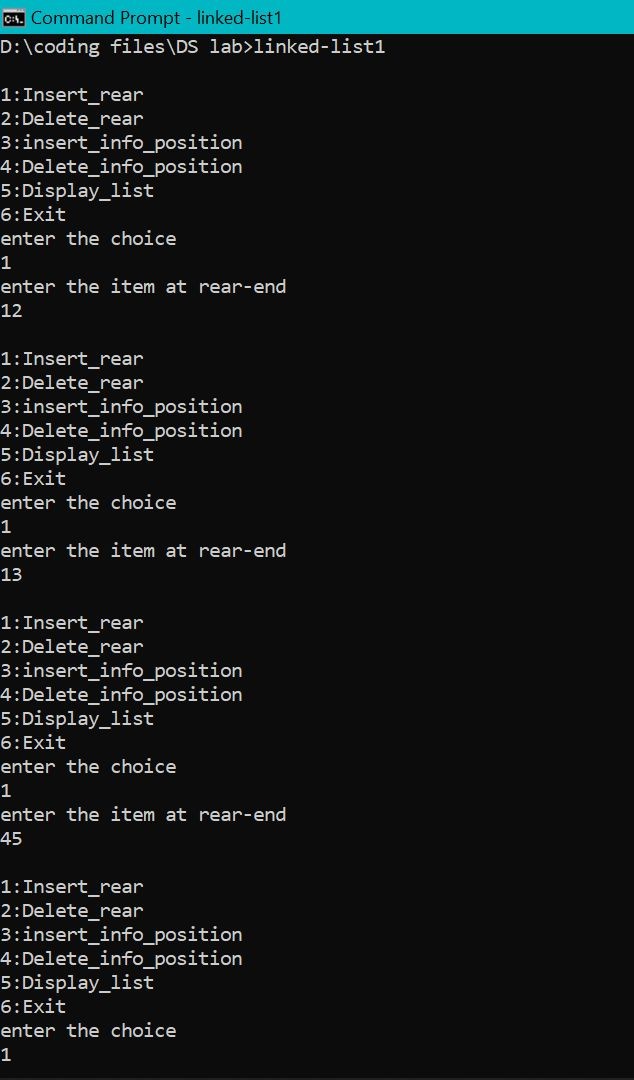
break; default:exit(0);

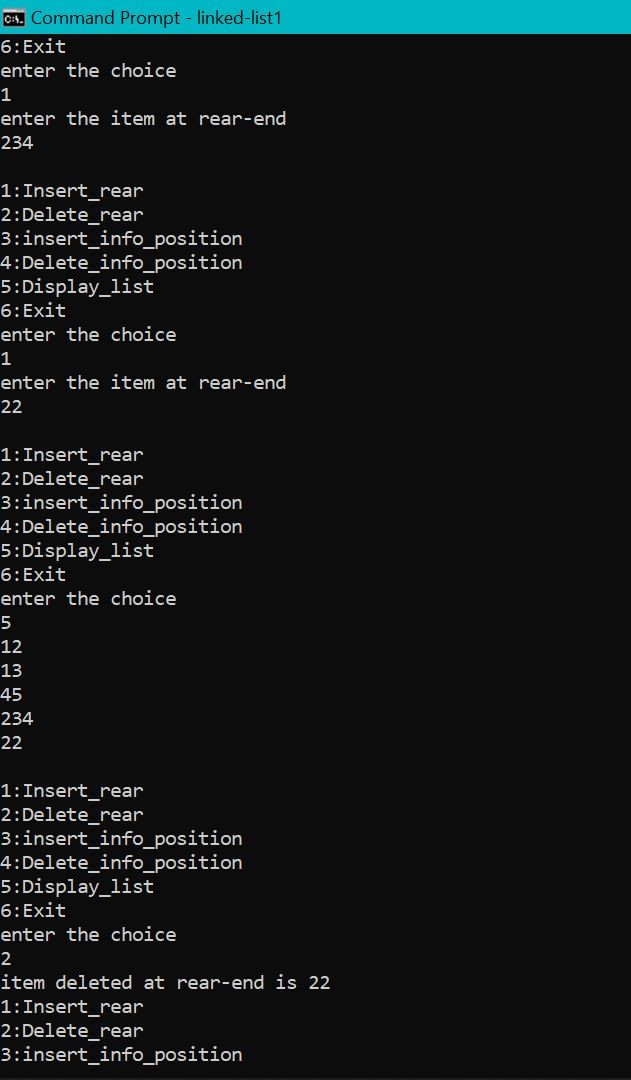
break;

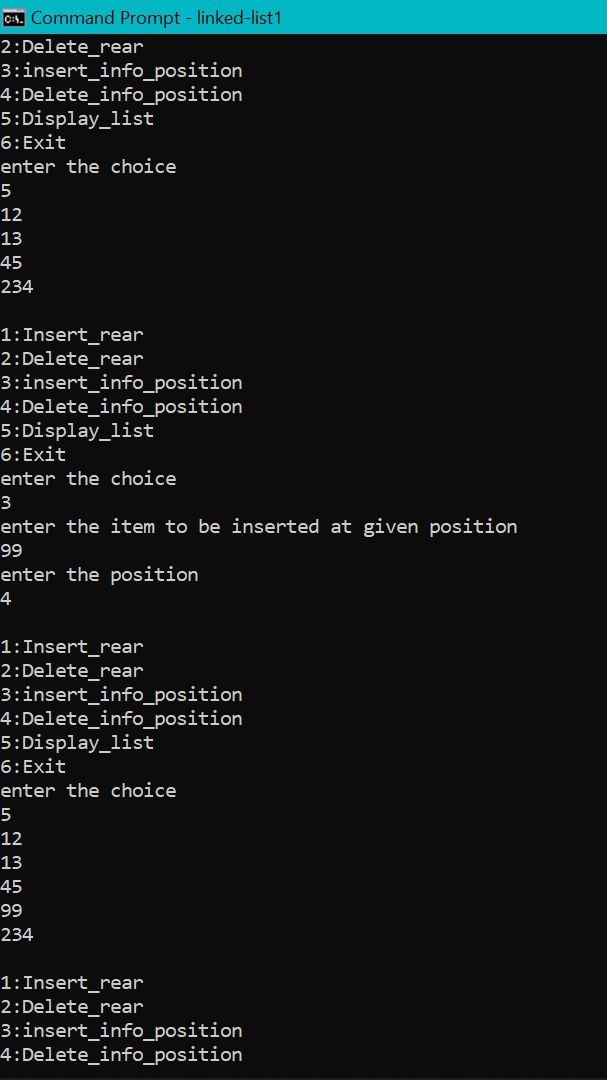
}

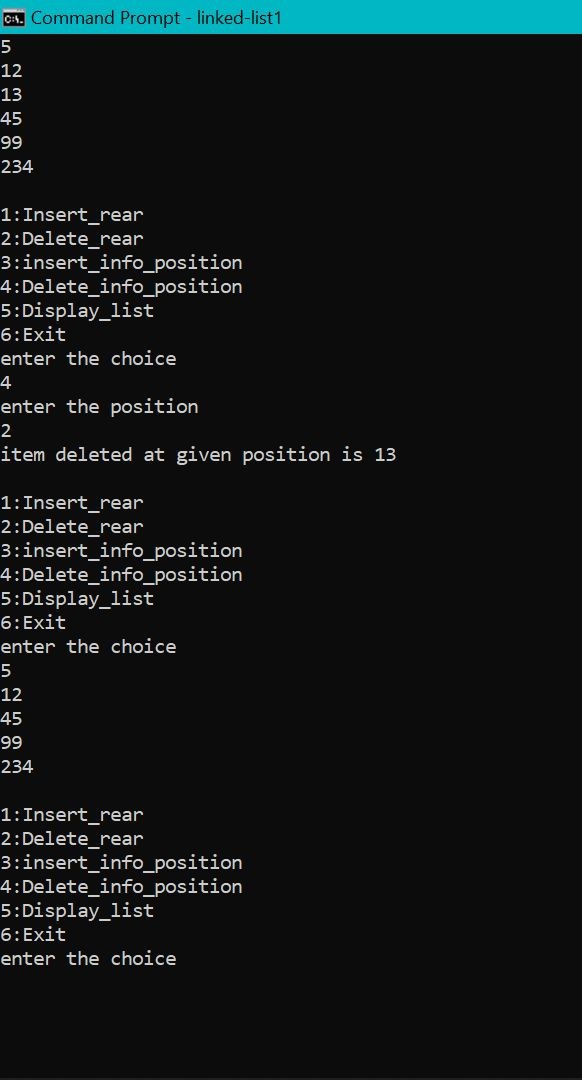
}

}









# Lab-7,8

/\*

WAP Implement Single Link List with following operations

1. Sort the linked list.
2. Reverse the linked list.
3. Concatenation of two linked lists

WAP to implement Stack & Queues using Linked Representation

\*/

#include <stdio.h> #include <stdlib.h>

struct node

{

int data;

struct node\* next;

};

struct node \*rear=NULL, \*front =NULL, \*top=NULL;

struct node\* getnode(int item)

{

struct node\* newn = (struct node\*)malloc(sizeof(struct node)); newn->data = item;

newn->next = NULL; return newn;

}

void display(struct node\* head)

{

if(head == NULL)

{

printf("List is empty.\n"); return;

}

struct node\* ptr = head; while(ptr)

{

printf("%d->", ptr->data); ptr = ptr->next;

}

printf("\b \b\b \n");

}

struct node\* insertfront(struct node\* head, int item)

{

struct node\* newn = getnode(item); newn->next = head;

head = newn; return head;

}

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

{

int temp; temp = \*a;

\*a = \*b;

\*b = temp;

}

struct node\* sort (struct node\* head)

{

int sorted;

if(head == NULL) return head; struct node\* ptr = head;

do

{

ptr = head; sorted = 0; while(ptr->next)

{

if(ptr->data > ptr->next->data)

{

swap(&ptr->data, &ptr->next->data); sorted = 1;

}

ptr = ptr->next;

}

} while(sorted == 1); return head;

}

void reverse(struct node\*\* head)

{

struct node\* prev = NULL; struct node\* current = \*head; struct node\* next = NULL; while (current != NULL) {

next = current->next; current->next = prev; prev = current; current = next;

}

\*head = prev;

}

struct node\* concatenate(struct node\* head1, struct node\* head2)

{

struct node\* ptr = head1; while(ptr->next)

{

ptr = ptr->next;

}

ptr->next = head2; return head1;

}

void qinsert()

{

struct node \*newnode;

newnode=(struct node \*) malloc(sizeof(struct node)); printf("Enter the element:\n");

scanf("%d",&newnode->data); newnode->next=NULL;

if(rear==NULL)

{

rear=newnode; front=newnode;

}

else

{

rear->next=newnode; rear=newnode;

}

}

void qdel()

{

if(front==NULL)

{

printf("Queue is empty\n");return;

}

else

{

printf("Deleted ele is %d",front->data); if(front==rear)

{

printf("Queue is empty\n"); front=NULL; rear=NULL;

}

else

front=front->next;

}

}

void qdisplay()

{

struct node \*temp; if(front ==NULL)

{

printf("Queue is empty"); return;

}

temp=front;

while (temp !=NULL)

{

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

}

}

void spush()

{

int item;

struct node \*newnode; printf("Enter the element\n");

scanf("%d",&item);

newnode=(struct node\*)malloc(sizeof(struct node)); newnode->data=item;

newnode->next=NULL; if(top==NULL)

top=newnode; else

newnode->next=top; top=newnode;

}

void spop()

{

if(top==NULL) printf("stack is empty");

else

{

printf("element removed is %d:", top->data); top=top->next;

}

}

void sdisplay()

{

struct node \*temp; temp=top; if(top==NULL)

printf("Stack is empty"); while(temp!=NULL)

{

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

temp=temp->next;

}

}

int main()

{

printf("Linked list program containing sort, reverse, and concatenate functions.\n");

int n1, n2, n, ch, flag = 0; int choice;

struct node\* head1 = NULL; struct node\* head2 = NULL; do

{

printf("Enter the choice\n1.Stack\n2.Queue\n3: Linked list 1\n4: Linked list 2\n5: Exit\n");

scanf("%d", &n1); switch(n1)

{

case 1:

{

do

{ printf("\n1. Push \n2. Display \n3. Pop\n"); printf("\nEnter your choice : "); scanf("%d",&choice);

switch(choice)

{

case 1: spush(); break; case 2: sdisplay();break; case 3: spop(); break;

;

}

}while(choice!=10);

}

\n");

case 2:

{

do

{ printf("\nQueue implementation using linked list\n"); printf("\n1. Create \n2. Display \n3. Delete \n4. Exit

printf("\nEnter your choice : "); scanf("%d",&choice); switch(choice)

{ case 1: qinsert(); break; case 2: qdisplay();break; case 3: qdel(); break;

}

}while(choice!=10);

}

case 3:

{

do

{

printf("3: Insert\n4: Sort\n5: Reverse\n6:

Concatenate with list 1\n7: Display list\n8: Go back to main menu\n9: Exit\n");

scanf("%d", &n2); switch(n2)

{

inserted: ");

insertfront(head1, n);

concatenate(head1, head2);

case 3: {

}

case 4: {

}

case 5: {

}

case 6: {

}

case 7: {

}

case 8: {

}

case 9: {

}

printf("Enter item to be

scanf("%d", &n); head1 =

break;

head1 = sort(head1); break;

reverse(&head1); break;

head1 = break;

display(head1); break;

flag = 1; break;

exit(0);

default: printf("Invalid input.\n");

}

if(flag == 1)

{

}

case 4: {

}

}while(1); break;

flag = 0; do

{

break;

printf("3: Insert\n4: Sort\n5: Reverse\n6: Concatenate with list 1\n7: Display list\n8: Go back to main menu\n9: Exit\n");

scanf("%d", &n2); switch(n2)

{

inserted: ");

insertfront(head2, n);

case 3: {

}

case 4: {

}

case 5: {

}

printf("Enter item to be

scanf("%d", &n); head2 =

break;

head2 = sort(head2); break;

reverse(&head2); break;

concatenate(head2, head1);

case 6: {

}

case 7: {

}

case 8: {

}

case 9: {

}

head2 = break;

display(head2); break;

flag = 1; break;

exit(0);

default: printf("Invalid input.\n");

}

if(flag == 1)

{

flag = 0; break;

}

}while(1); break;

}

case 9: exit(0);

default: printf("Invalid input.\n");

}

}while(1); return 0;

}

**LAB-9**

### WAP Implement doubly link list with primitive operations

1. **Create a doubly linked list.**
2. **Insert a new node to the left of the node.**
3. **Delete the node based on a specific value**
4. **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;

}

}

void dsearch(int key,NODE head)

{

NODE cur; int count;

if (head->rlink==head)

{

printf("List is empty\n");

}

cur=head->rlink; count=1;

while (cur!=head && cur->info!=key)

{

cur=cur->rlink; count++;

}

if (cur==head)

{

printf("Search unsuccessfull\n");

}

else

{

printf("Key element found at the position %d\n",count);

}

}

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;

}

NODE dinsert\_rightpos(int item,NODE head)

{

NODE temp,cur,next; 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;

}

next=cur->rlink; temp=getnode(); temp->llink=NULL; temp->rlink=NULL;

printf("Enter the item to be inserted at the right of the given item:\n"); scanf("%d",&temp->info);

cur->rlink=temp;

temp->llink=cur; next->llink=temp; temp->rlink=next; return head;

}

NODE ddelete\_duplicates(int item,NODE head)

{

NODE prev,cur,next; int count=0;

if (head->rlink==head)

{

printf("List is empty\n"); return head;

}

cur=head->rlink; while (cur!=head)

{

if (cur->info!=item)

{

cur=cur->rlink;

}

else

{

count++;

if (count==1)

{

cur=cur->rlink; continue;

}

else

{

prev=cur->llink; next=cur->rlink; prev->rlink=next; next->llink=prev; free(cur); cur=next;

}

}

}

if (count==0)

{

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

}

else

{

printf("Removed all the duplicate elements of the given item successfully\n");

}

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:dsearch\n7:dinsert lestpos\n8:dinsert rightpos\n9:ddelete duplicates\n10: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 to be searched:\n"); scanf("%d",&key);

dsearch(key,head);

break;

case 7:printf("Enter the key element:\n"); scanf("%d",&key); head=dinsert\_leftpos(key,head);

break;

case 8:printf("Enter the key element:\n"); scanf("%d",&key); head=dinsert\_rightpos(key,head); break;

case 9:printf("Enter the key element whose duplicates should be removed:\n"); scanf("%d",&key);

head=ddelete\_duplicates(key,head); break;

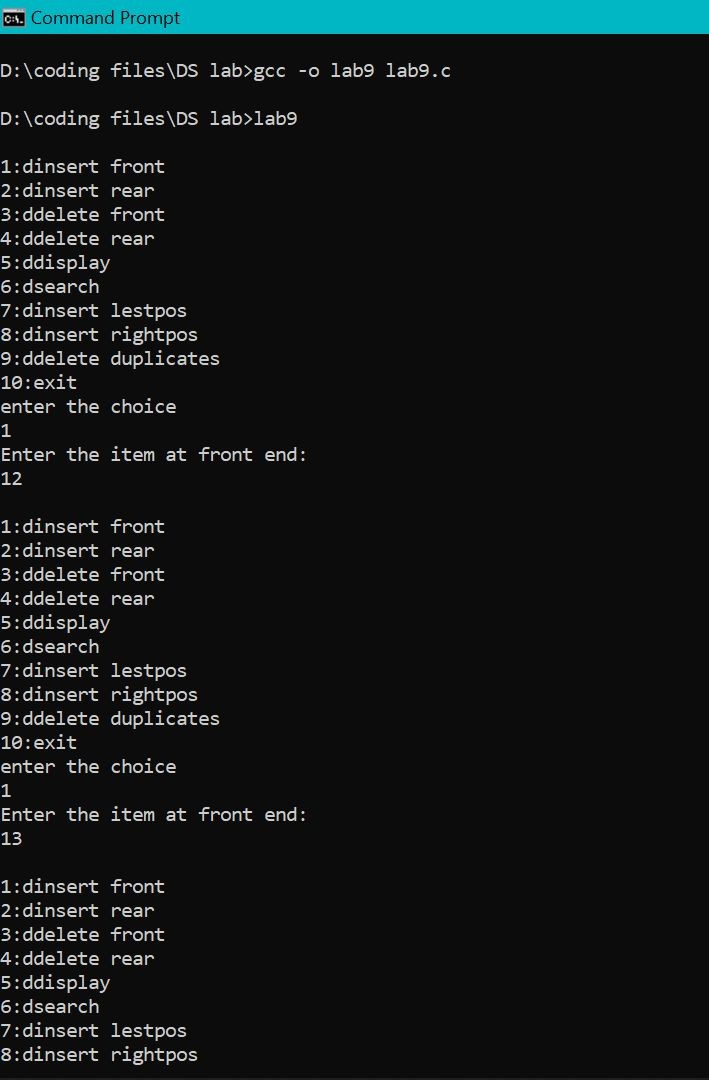
default:exit(0);

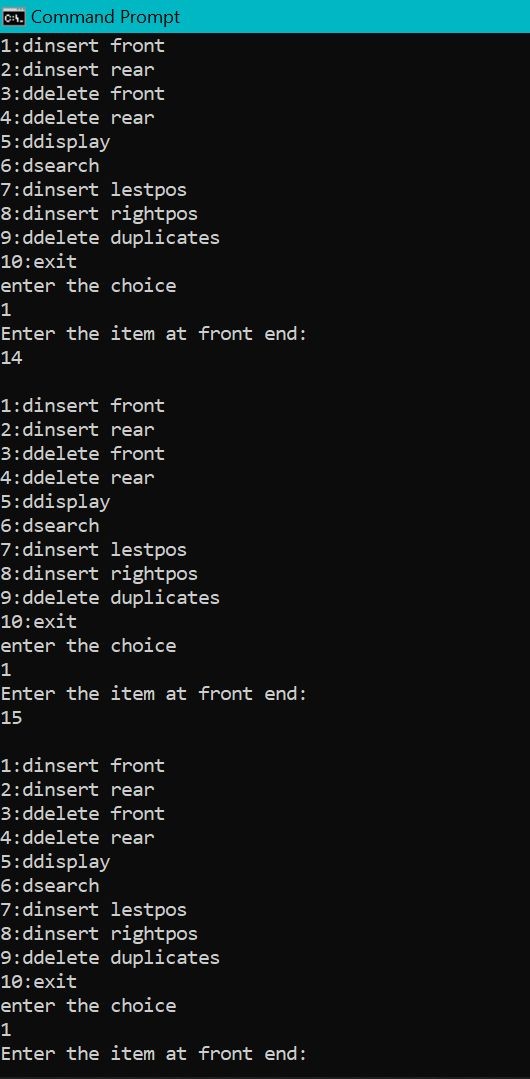
}

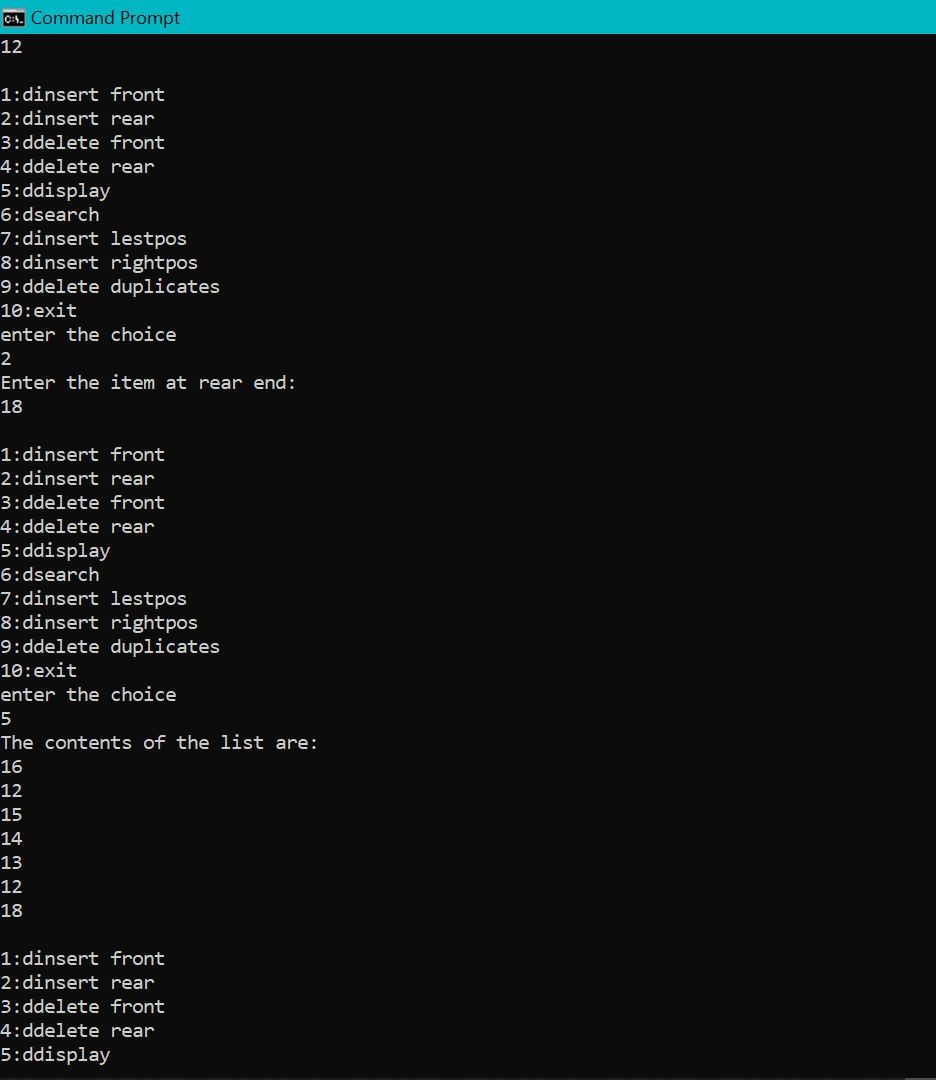
}

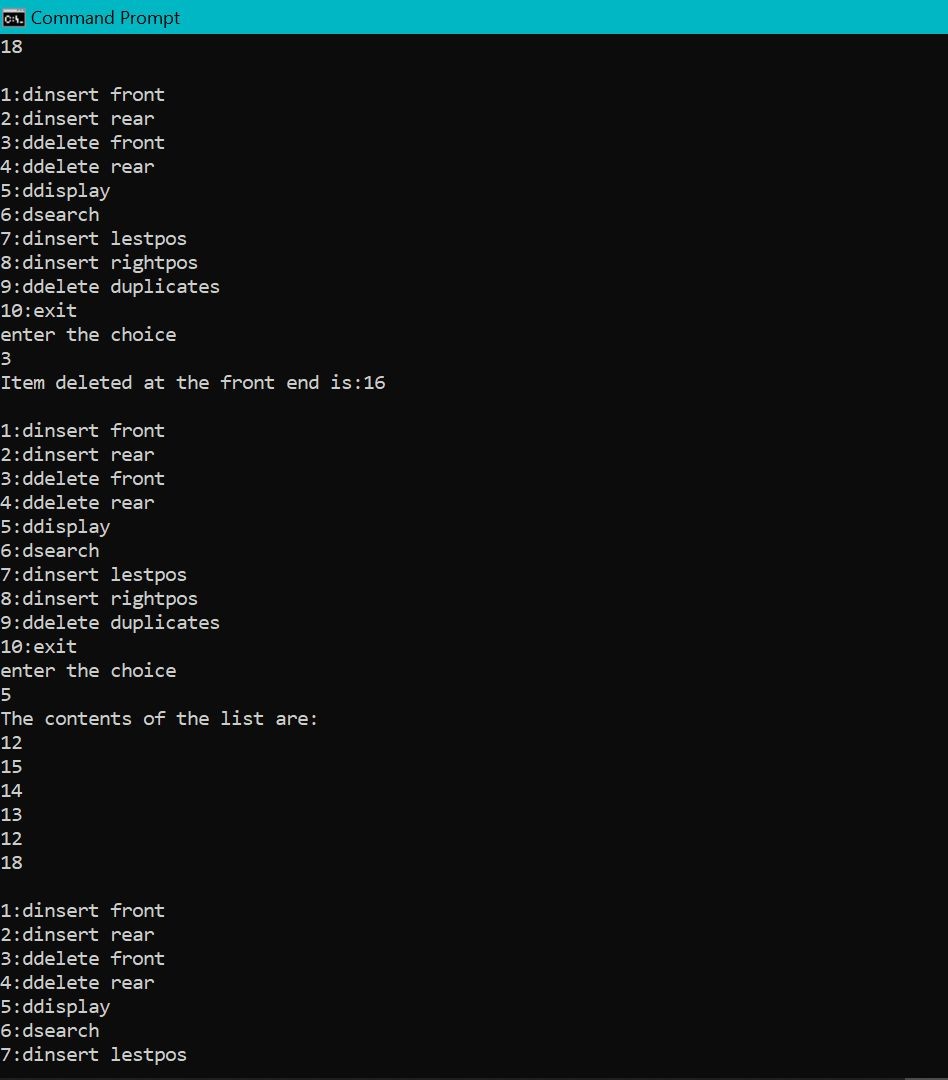
return 0;

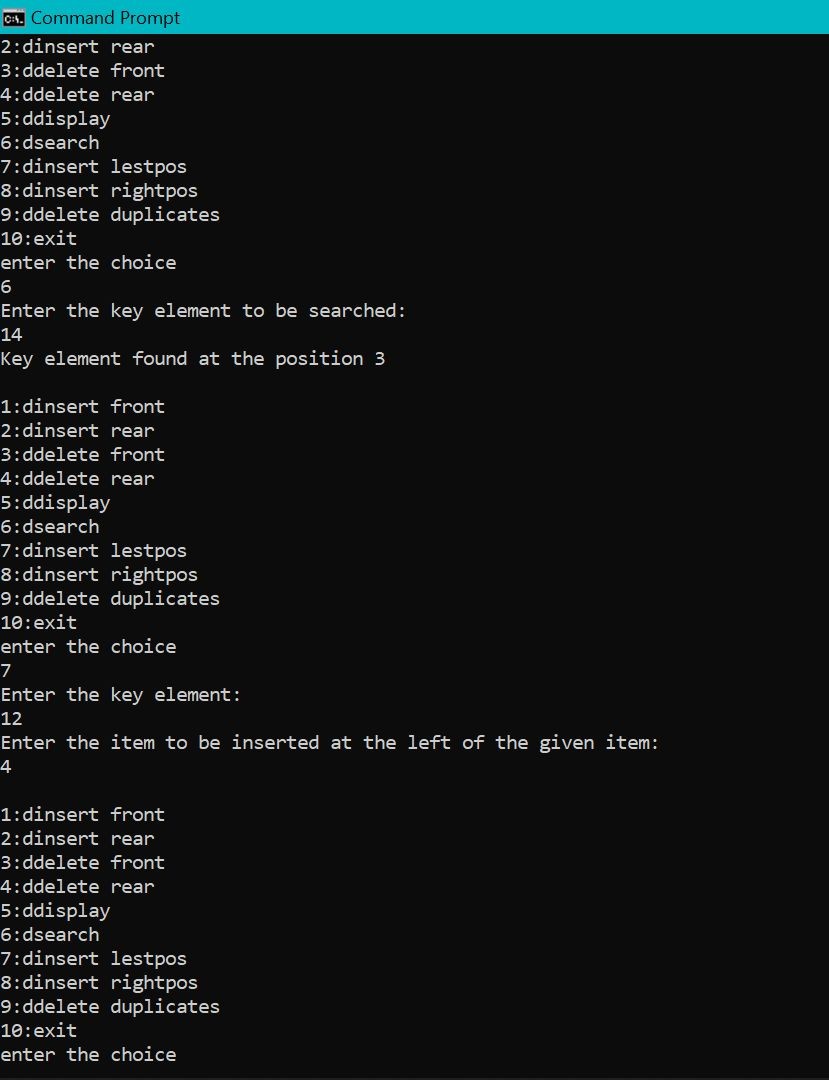
}

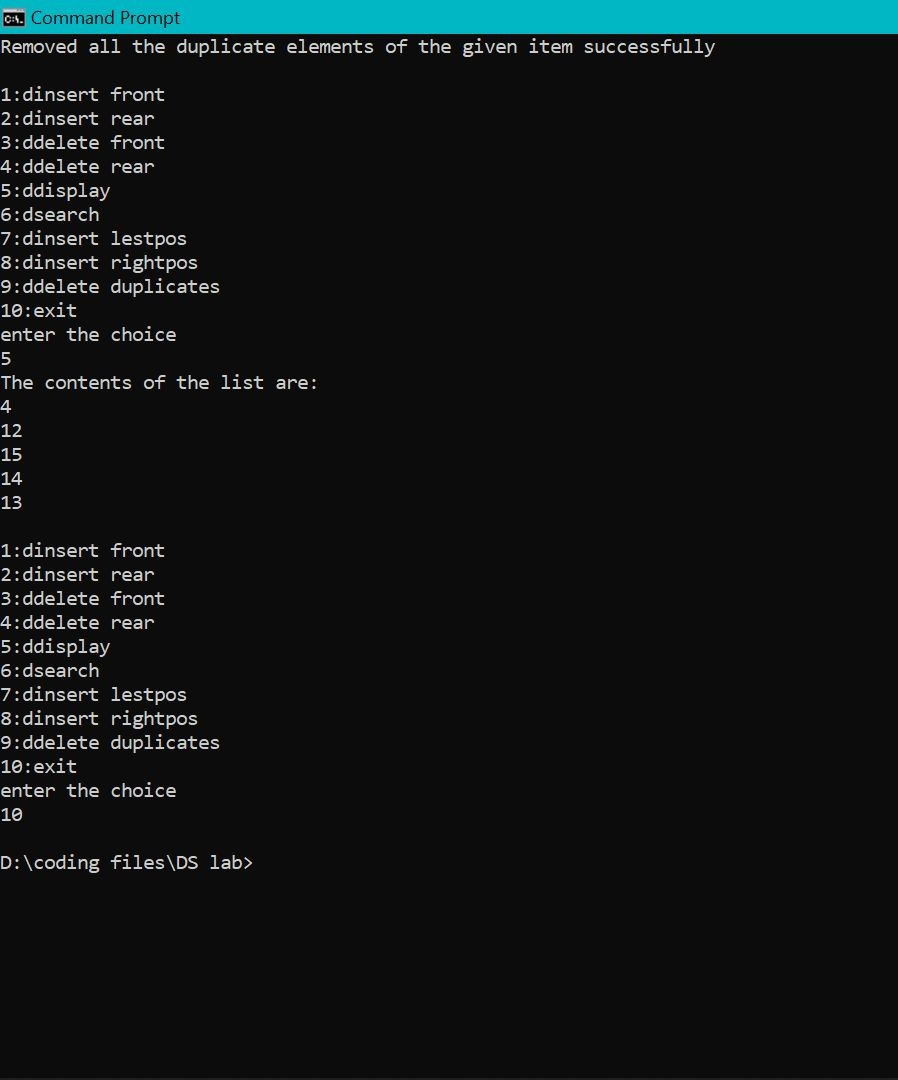












# LAB-10

### Write a program

1. **To construct a binary Search tree.**
2. **To traverse the tree using all the methods i.e., in-order, preorder and post order**
3. **To display the elements in the tree**

#include<stdio.h> #include<stdlib.h> struct node{

int info;

struct node \*rlink,\*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->rlink); preorder(root->llink);

}

}

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;

}

}

}

