Program 1:

Write a program to stimulate the working of stack using an array with the following: a. Push, b. Pop, c. Display. Program should print appropriate messages for stack overflow, stack underflow.

```
#include<stdio.h>

#include<process.h>

#include<conio.h>

#define STACK_SIZE 5

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)return -1;
return s[top--];
}
void display()
{ int i;
if(top==-1)
{printf(" Stack is empty \n");
return;
}
printf("contents of the stack\n");
for(i=top;i>=0;i--)
{ printf("%d\n",s[i]);
}
}
int main()
{
        int item_deleted;
```

```
        int choice;
        system("cls");
for(;;)
        {
                printf("\n 1:Push\n 2:Pop\n 3:Display\n 4:Exit\n");
                printf(" Enter the choice\n");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1: printf("enter the item to be inserted \n");
                                scanf("%d\n",&item);
                                push();
                                break;(-1);

                        case 2: item_deleted=pop(); (-1);
                                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();
return 0;
}
```

Program 2:

Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and binary operators +,-,*and /.

```
#include<stdio.h>
#include<conio.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])>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()
{int t;
char infix[20];
char postfix[20];
system("cls");
printf("Enter the valid infix expression\n");
scanf("%s",infix);
for(t=0;t<strlen(infix);t++)
{
if(infix[t]=='+'||infix[t]=='-'||infix[t]=='*'||infix[t]=='/'||infix[t]=='^'||infix[t]=='('
{
if(infix[t+1]=='+'||infix[t+1]=='-'||infix[t+1]=='*'||infix[t+1]=='/'||infix[t+1]=='^'||infix[t+1]==')')
```

```
{ printf("Invalid");exit(0);

}

}

}

infix_postfix(infix,postfix);

printf("the postfix exp is\n");

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

getch();

return 0;

}
```

Program 3:

Write a program to stimulate the working of queue of integers using an array with the following: a. Insert, b. Delete, c. Display. Program should print appropriate messages for queue empty, queue overflow conditions.

```
#include<stdio.h>

#include<stdlib.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]);
}}
int main()
{
        int choice;
        for(;;)
        {
                printf("1:insertrear 2:deletefront 3:display 4: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);

        }

}
```

```
}
```

Program 4:

Write a program to stimulate the working of circular queue of integers using an array with the following: a. Insert, b. Delete, c. Display. Program should print appropriate messages for queue empty, queue overflow conditions.

```
#include<stdio.h>
#include<stdlib.h>
#include<process.h>
#include<conio.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");
                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");
```

```
            return;
        }
        f=front;
        printf("contents of queue \n");
        for(i=0;i<=count;i++)
        {
            printf("%d\n",q[f]);
            f=(f+1)%que_size;
        }
}
int main()
{
        int choice;
        for(;;)
        {
            printf("\n1.Insert rear \n2.Delete front \n3.Display \n4.exit \n ");
            printf("Enter the choice : ");
            scanf("%d",&choice);
```

```
        switch(choice)
        {
                case 1:printf("Enter the item to be inserted :");
                    scanf("%d",&item);
                    insertrear();
                    break;
                case 2:item=deletefront();
                        if(item==-1)
                        printf("queue is empty\n");
                        else
                        printf("item deleted is %d \n",item);
                        break;
           case 3:displayq();
                        break;
           default:exit(0);
        }
}
getch();
```

```
    return 0;
}
```

Program 5:

Write a program to implement singly linked list with the following operations: a. Create a linked list, b. Insertion of a node at first position, at any position and at the end of list, c. Display the contents of the list.

```
#include<stdio.h>
#include<conio.h>
#include<malloc.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("mem full\n");
exit(0);
}
return x;
}
int freenode(NODE x)
{
free(x);
return 0;
}
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;
}

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 insert_pos( int item, int pos, NODE first)
{
```

```
NODE temp;
NODE prev,cur;
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;
```

```
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;
}
int main()
{
int item,choice,pos;
NODE first=NULL;
system("cls");
for(;;)
{
printf("\n 1:Insert_front\n 2:Insert at specified position \n 3:Insert_rear\n 4:Display_list\n 6: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 to be inserted:\n");
      scanf("%d",&item);
      printf("enter the position at which item to be inserted:\n");
      scanf("%d",&pos);
      first=insert_pos(item,pos,first);
break;
case 3:printf("enter the item at rear-end\n");
scanf("%d",&item);
first=insert_rear(first,item);
break;
case 4:display(first);
break;
default:exit(0);
break;
}
}
getch();
return 0;
```

}

Program 6:

Write a program to implement singly linked list with the following operations: a. Create a linked list, b. Deletion of a node at first position, at any position and at the end of list, c. Display the contents of the list.

```
#include<stdio.h>
#include<conio.h>
#include<malloc.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("mem full\n");
exit(0);
}
return x;
}
int freenode(NODE x)
{
free(x);
return 0;
}
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("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 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("iten deleted at rear-end is %d",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);
}
}
```

```
NODE delete_pos(int pos, NODE first)
{
NODE cur;
NODE prev;
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;
}
if(count!=pos)
{
printf("invalid position specified \n");
return first;
```

```
}
prev->link=cur->link;
freenode(cur);
return first;
}
int main()
{
int item,choice,pos;
NODE first=NULL;
system("cls");
for(;;)
{
printf("\n 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n 4:Delete_rear\n 5.Delete at specified position \n 6:Display_list\n
7: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: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 position of the item to be deleted: \n");
      scanf("%d",&pos);
      first=delete_pos(pos,first);
   break;

case 6:display(first);
```

```
break;
default:exit(0);
break;
}
}
getch();
return 0;
}
```

Program 7:

Write a program to implement singly linked list with the following operations: a. Sort a linked list, b. Reverse a linked list, c. Concatenation of two linked lists.

```
#include<stdio.h>
#include<conio.h>
#include<malloc.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_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 IF(NODE second,int item){
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(second==NULL)
return temp;
temp->link==second;
second=temp;
```

```
return second;
}
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 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 IR(NODE second,int item){
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(second==NULL)
return temp;
cur=second;
while(cur->link!=NULL)
```

```
cur=cur->link;
cur->link=temp;
return second;
}
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 delted 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;
NODE prev,cur;
int count;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL&&pos==1)
return temp;
```

```
if(first==NULL){
printf("invalid pos \n");
return first;
}
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_pos(int pos,NODE first){
NODE cur;
NODE prev;
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;
}
if(count!=pos){
printf("invalid position specified \n");
```

```
return first;
}
prev->link=cur->link;
freenode(cur);
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 asc(NODE first){
```

```
NODE prev=first;
NODE cur=NULL;
int temp;
if(first==NULL){
return 0;
}
else{
while(prev!=NULL){
cur=prev->link;
while(cur!=NULL){
if(prev->info > cur->info){
temp=prev->info;
prev->info=cur->info;
cur->info=temp;
}
cur=cur->link;
}
prev=prev->link;
```

```
}
}
return first;
}
NODE des(NODE first){
NODE prev=first;
NODE cur=NULL;
int temp;
if(first==NULL){
return 0;
}
else{
while(prev!=NULL){
cur=prev->link;
while(cur!=NULL){
if(prev->info < cur->info){
temp=prev->info;
prev->info=cur->info;
```

```
cur->info=temp;
}
cur=cur->link;
}
prev=prev->link;
}}
return first;
}

NODE concate(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);
}
}
int main(){
int item,choice,pos,element,option,choice2,item1,num;
system("cls");
NODE first=NULL;
NODE second=NULL;
for(;;){
```

```
printf("\n 1:insert_front \n 2: delete_front \n 3: insert_rear \n 4: delete_rear \n 5: random_position \n 6: reverse \n 7: sort \n 8: concate\n 9: display_list \n 10: 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: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("press 1 to insert or 2 to delete at any desired position \n");
```

```
scanf("%d",&element);
if(element==1){
printf("enter the position to inset \n");
scanf("%d",&pos);
printf("enter the item to inset \n");
scanf("%d",&item);
first=insert_pos(item,pos,first);}
if(element==2){
printf("enter the position to delete \n");
scanf("%d",&pos);
first=delete_pos(pos,first);
}
break;
case 6: first=reverse(first);
      break;
case 7: printf("press 1 for ascending sort and 2 for descending sort: \n");
scanf("%d",&option);
if(option==1)
```

```
first=asc(first);
if(option==2)
first=des(first);
break;
case 8: printf("create a second list \n");
printf("enter the number of elements in second list \n");
scanf("%d",&num);
for(int i=1;i<=num;i++){
printf("\n press 1 to insert front and 2 to insert rear \n");
scanf("%d",&choice2);
if(choice2==1){
printf("enter the item at front-end \n");
scanf("%d",&item1);
}
if(choice2==2){
printf("enter the item at rear-end \n");
scanf("%d",&item1);
second=IR(second,item1);
```

```
}
}
first=concate(first,second);
break;
case 9:display(first);
break;
default:exit(0);
break;
}}
getch();
return 0;
}
```

Program 8:

Write a program to Implement Stack and Queues using Linked Representation.

```
#include<stdio.h>
#include<conio.h>
#include<malloc.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("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_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;
}
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 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_s(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("stack 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;
}
void display_s(NODE first)
{
NODE temp;
if(first==NULL)
```

```
printf("stack empty cannot display items\n");
for(temp=first;temp!=NULL;temp=temp->link)
{
printf("%d\n",temp->info);
}
}
int main()
{
int item,choice,pos;
NODE first=NULL;
system("cls");
for(;;)
{
printf("\n Queue operations :\n 1:Insert_rear\n 2:Delete_front\n 3:Display_list(Queue)\n \n Stack operations \n 4:Insert_front\n 5: Delete_front \n 6:Dislay_list(Stack)\n 7:Exit \n \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_front(first);
break;
case 3:display(first);
break;
case 4:printf("enter the item at front-end\n");
scanf("%d",&item);
first=insert_front(first,item);
break;
case 5:first=delete_front_s(first);
break;
case 6:display_s(first);
break;
default:exit(0);
```

```
break;
}
}
getch();
return 0;
}
```

Program 9:

Write a program to implement a doubly linked list with the following operations: a. Create a doubly linked list, b. Insert a new node to the left of the node, c. Delete a node based on specific value, c. Display the contents of the list.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<stdlib.h>
#include <malloc.h>
struct node
{
        int info;
        struct node *llink;
```

```
        struct node *rlink;
        };
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 dinsert_front(int item,NODE head)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
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;
cur=head->llink;
head->llink=temp;
temp->rlink=head;
temp->llink=cur;
cur->rlink=temp;
return head;
}


NODE ddelete_front(NODE head)
{
```

```
NODE cur,next;
if(head->rlink==head)
{
printf("dq empty\n");
return head;
}
cur=head->rlink;
next=cur->rlink;
head->rlink=next;
next->llink=head;
printf("the node deleted is %d",cur->info);
freenode(cur);
return head;
}


NODE ddelete_rear(NODE head)
{
```

```
NODE cur,prev;
if(head->rlink==head)
{
printf("dq empty\n");
return head;
}
cur=head->llink;
prev=cur->llink;
head->llink=prev;
prev->rlink=head;
printf("the node deleted is %d",cur->info);
freenode(cur);
return head;
}


void display(NODE head)
{
```

```
NODE temp;
if(head->rlink==head)
{
printf("dq empty\n");
return;
}
printf("contents of dq\n");
temp=head->rlink;
while(temp!=head)
{
printf("%d  ",temp->info);
temp=temp->rlink;
}
printf("\n");
}



NODE delete_all_key(int item,NODE head)
```

```
{
NODE prev,cur,next;
int count;
  if(head->rlink==head)
   {
    printf("List Empty");
    return head;
    }
count=0;
cur=head->rlink;
cur=cur->rlink;
while(cur!=head)
{
 if(item!=cur->info)
 cur=cur->rlink;
 else
 {
 count++;
```

```
  prev=cur->llink;
  next=cur->rlink;
  prev->rlink=next;
  next->llink=prev;
  freenode(cur);
  cur=next;
 }
}
if(count==0)
  printf("key not found");
  else
 printf("key found at %d positions and are deleted\n", count);

return head;



}
NODE insert_leftpos(int item,NODE head){
```

```
NODE temp,cur,prev;
if(head->rlink==head){
printf("Empty \n");
return head;
}
cur=head->rlink;
while(cur!=head){
if(item==cur->info)break;
cur=cur->rlink;
}
if(cur==head){
printf("key nt found \n");
return head;
}
prev=cur->llink;
printf("Enter towards left of %d:",item);
temp=getnode();
scanf("%d",&temp->info);
```

```
prev->rlink=temp;
temp->llink=prev;
cur->llink=temp;
temp->rlink=cur;
return head;}
NODE insert_rightpos(int item,NODE head){
NODE temp,cur,prev;
if(head->llink==head){
printf("Empty \n");
return head;
}
cur=head->llink;
while(cur!=head){
if(item==cur->info)break;
cur=cur->llink;
}
if(cur==head){
printf("key nt found \n");
```

```
return head;
}
prev=cur->rlink;
printf("Enter towards right of %d:",item);
temp=getnode();
scanf("%d",&temp->info);
prev->llink=temp;
temp->rlink=prev;
cur->rlink=temp;
temp->llink=cur;
return head;}



NODE search(NODE head)
{
  NODE temp=head->rlink;
  int count=0,key,flag=0;
  printf("Enter the key :");
```

```
  scanf("%d",&key);
  while(temp!=head)
  {       count++;
     if(temp->info==key)
     {
        flag=1;
        printf("key %d found in position %d",key,count);
     }temp=temp->rlink;
  }
  if(flag==0)
  {
     printf("Key is not found in list");
  }
  return head;

}
```

```
int main()
{
NODE head,last;
int item, choice,option;
head=getnode();
head->rlink=head;
head->llink=head;
system("cls");
for(;;)
{
        printf("\n1:insert front\n2:insert rear\n3:delete front\n4:delete rear\n5:display\n6:Delete repeating occurences\n 7:search\n
8:Inserting node before/after key node \n9: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);
```

```
                last=dinsert_front(item,head);
                break;
        case 2: printf("enter the item at rear end\n");
                scanf("%d",&item);
                last=dinsert_rear(item,head);
                break;
        case 3:last=ddelete_front(head);
                break;
        case 4: last=ddelete_rear(head);
                break;
        case 5: display(head);
  break;
case 6: printf("enter the item \n");
     scanf("%d",&item);
     last=delete_all_key(item,head);
     break;
case 8: printf("press 1:for insert behind    2: for insert after");
      scanf("%d",&option);
```

```
            if(option==1){
            printf("enter key node\n");
            scanf("%d",&item);
            last=insert_leftpos(item,head);}
            else if(option==2){
            printf("enter key node\n");
            scanf("%d",&item);
            last=insert_rightpos(item,head);
            break;
            }break;
      case 7: search(head);
                        break;
                default:exit(0);
                }
         }
getch();
return(0);
}
```

Program 10:

Write a program to: a. Construct a binary search tree, b. Traverse the tree using all the methods i.e., in-order, pre-order and post-order, c. Display the elements in the tree.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node
{
int info;
struct node*llink;
struct node*rlink;
};
typedef struct node*NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
{
printf("Memory not available!");
exit(0);
}
return x;
}
void freenode(NODE x)
```

```
{
free(x);
}
NODE insert(int item,NODE root)
{
NODE temp,cur,prev;
char direction[10];
int i;
temp=getnode();
temp->info=item;
temp->llink=NULL;
temp->rlink=NULL;
if(root==NULL)
 return temp;
printf("Give direction to insert..\n");
scanf("%s",direction);
prev=NULL;
cur=root;
for(i=0;i<strlen(direction)&&cur!=NULL;i++)
{
prev=cur;
if(direction[i]=='l')
cur=cur->llink;
else
cur=cur->rlink;
}
```

```
if(cur!=NULL||i!=strlen(direction))
{
printf("Insertion not possible\n");
freenode(temp);
return(root);
}
if(cur==NULL)
{
if(direction[i-1]=='l')
prev->llink=temp;
else
prev->rlink=temp;
}
return(root);
}
void preorder(NODE root)
{
if(root!=NULL)
{
printf("the item is %d\n",root->info);
preorder(root->llink);
preorder(root->rlink);
}
}
void inorder(NODE root)
{
```

```
if(root!=NULL)
{
inorder(root->llink);
printf("The item is%d\n",root->info);
inorder(root->rlink);
}
}
void postorder(NODE root)
{
if (root!=NULL)
{
postorder(root->llink);
postorder(root->rlink);
printf("The item is%d\n",root->info);
}
}
void display(NODE root,int i)
{
int j;
if(root!=NULL)
{
display(root->rlink,i+1);
for (j=1;j<=i;j++)
printf("  ");
printf("%d\n",root->info);
display(root->llink,i+1);
```

```
}
}

int main()
{
NODE root=NULL;
int choice,i,item;

for(;;)
{
printf("1.Insert\n2.Preorder\n3.Inorder\n4.Postorder\n5.Display\n");
printf("Enter the choice:\n");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("Enter the item:\n");
                scanf("%d",&item);
                root=insert(item,root);
                break;
case 2: if(root==NULL)
                {
                 printf("Tree is empty!");
                }
                else
                {
                 printf("Given tree is..");
```

```
                display(root,1);
                printf("The preorder traversal is:\n");
                preorder(root);
               }
               break;
case 3:if(root==NULL)
        {
               printf("Tree is empty");
        }
        else
        {
               printf("Given tree is..");
               display(root,1);
               printf("The inorder traversal is \n");
               inorder(root);
               }
        break;
case 4:if (root==NULL)
               {
               printf("Tree is empty");
                }
        else
        {
               printf("Given tree is..");
               display(root,1);
               printf("The postorder traversal is \n");
```

```
                postorder(root);
            }
            break;
case 5:display(root,1);
            break;
default:printf("Invalid choice entered.\n");
      exit(0);
}
}
return 0;
}
```