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

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**Sep-2020 to Jan-2021**

**Data Structures Lab Record: -**

**Name: -** Khushil M Sindhwad.

**USN:** - 1BM19CS072

**Sec:** - 3rd Sem CSE B

**Batch:** - B1

**Lab-Program 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

**Code: -**

#include <stdio.h>

#include <stdlib.h>

#define STACK\_SIZE 5

int top=-1;

int s[10];

int item;

void push()

{

    if(top==STACK\_SIZE-1)

    {

        printf("\nSTACK OVERFLOW\n");

        return;

    }

    top+=1;

    s[top]=item;

}

int pop()

{

    if(top==-1)

        return -1;

    return s[top--];

}

void display()

{

    int i;

    if(top==-1)

    {

        printf("\nSTACK UNDERFLOW\n");

        return;

    }

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

    for (i=0;i<=top;i++)

    {

        printf("%d\n",s[i]);

    }

}

void main()

{

    int item\_deleted,choice;

    while(1)

    {

        printf("\n1: Push\n2: Pop\n3: Display\n4: EXIT\n");

        printf("Enter your choice:");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:

            printf("\nEnter the item to be inserted:");

            scanf("%d",&item);

            push();

            break;

            case 2:

            item\_deleted=pop();

            if(item\_deleted==-1)

            {

                printf("\nSTACK UNDERFLOW\n");

            }

            else

            {

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

            }

            break;

            case 3:

            display();

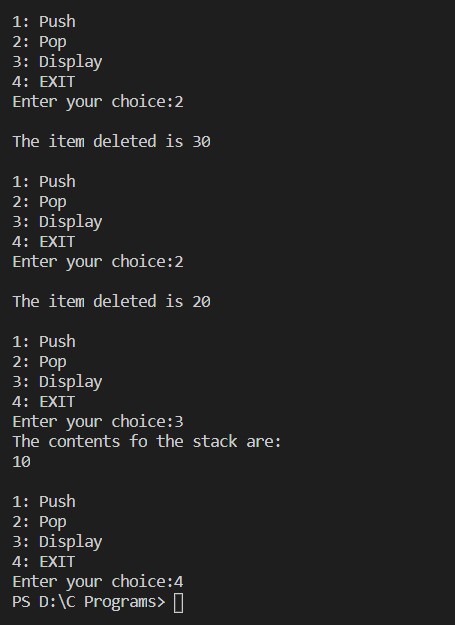
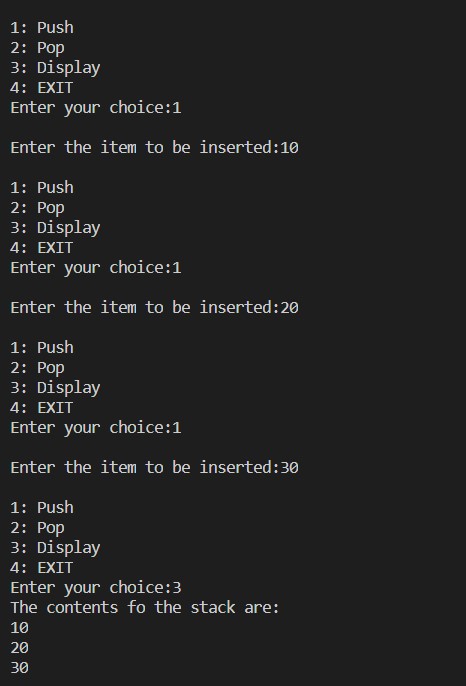
            break;

            default:exit(0);

        }

    }

}

**Output: -**

**Lab-Program 2: -**

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)

**Code: -**

#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';

}

void main()

{

    char infix[20];

    char postfix[20];

    printf("\nEnter the valid infix Expression:");

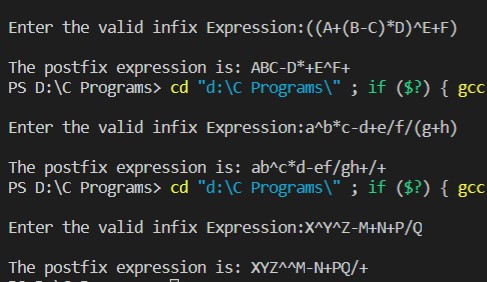
    scanf("%s",&infix);

    infix\_postfix(infix,postfix);

    printf("\nThe postfix expression is: %s\n",postfix);

}

**Output: -**



**Lab-Program 3: -**

WAP to simulate the working of a queue of integers using an array. Provide the following

operations

a) Insert b) Delete c) Display

The program should print appropriate messages for queue empty and queue overflow

Conditions

**Code: -**

#include<stdio.h>

#include<process.h>

#define QUE\_SIZE 3

int item,front=0,rear=-1,q[10];

void insertrear()

{

    if(rear==QUE\_SIZE-1)

    {

        printf("\n-------------------------------\nQUEUE OVERFLOW\n-------------------------------\n");

        return;

    }

    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("\n-----------------------------------\nQUEUE IS EMPTY\n-----------------------------------\n");

        return;

    }

    printf("Contents of the 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 Queue\n4:EXIT\n");

        printf("Enter your choice:  ");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:printf("\nEnter the value to be inserted:  ");

                   scanf("%d",&item);

                   insertrear();

                   break;

            case 2:item=deletefront();

                   if(item==-1)

                       printf("\n--------------------------------------\nQUEUE IS EMPTY\n--------------------------------------\n");

                   else

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

                   break;

            case 3:displayQ();

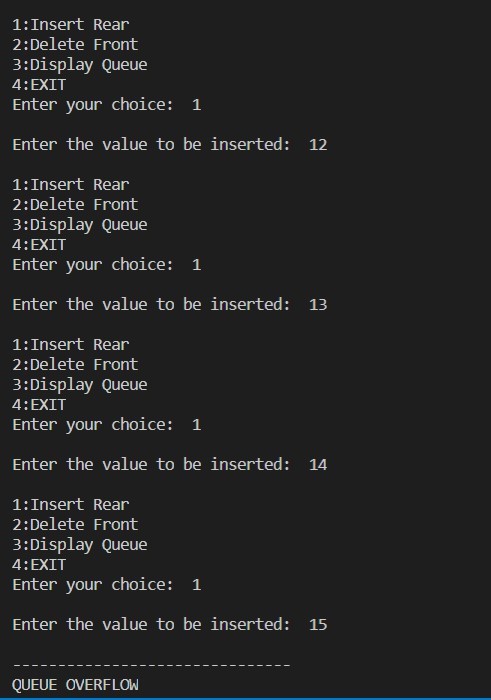
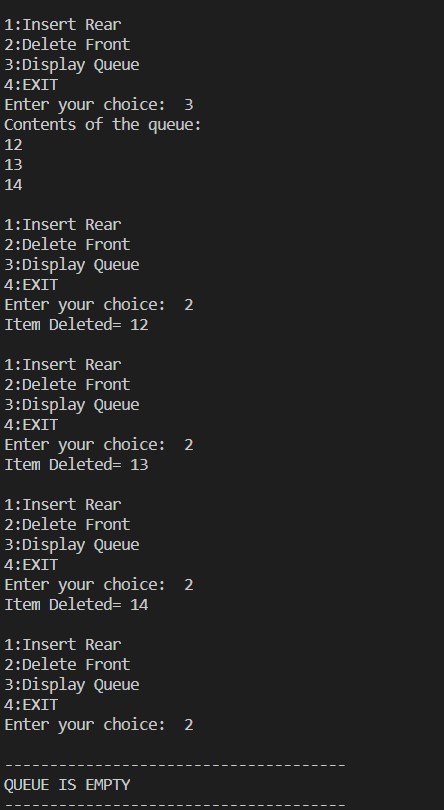
                   break;

            default:return;

        }

    }

}

**Output: -**

**Lab-Program 4: -**

WAP to simulate the working of a circular queue of integers using an array. Provide the

following operations.

a) Insert b) Delete c) Display

The program should print appropriate messages for queue empty and queue overflow

Conditions

**Code: -**

#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=count-1;

    return item;

}

void displayQ()

{

    int i,f;

    if(count==0)

    {

        printf("queue is empty\n");

        return;

    }

    f=front;

    printf("Contents of queue \n");

    for(i=1;i<=count;i++)

    {

        printf("%d\n",q[f]);

        f=(f+1)%QUE\_SIZE;

    }

}

void main()

{

    int choice;

    for(;;)

    {

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

        printf("enter the choice\n");

        scanf("%d",&choice);

        switch(choice)

        {

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

            scanf("%d",&item);

            insertrear();

            break;

        case 2:item=deletefront();

            if(item==-1)

            printf("queue is empty\n");

            else

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

            break;

        case 3:displayQ();

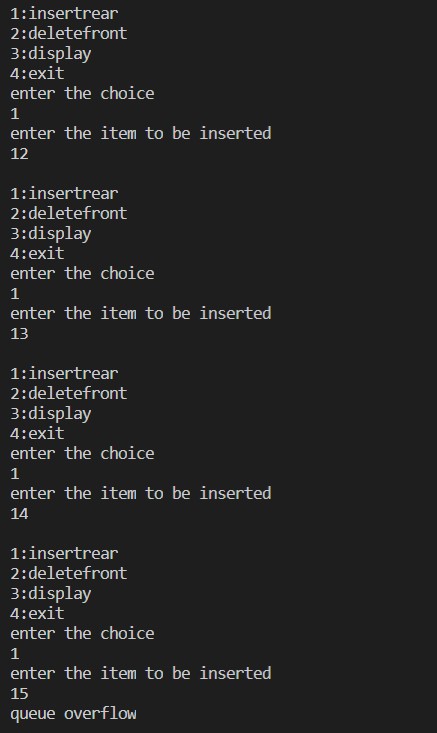
            break;

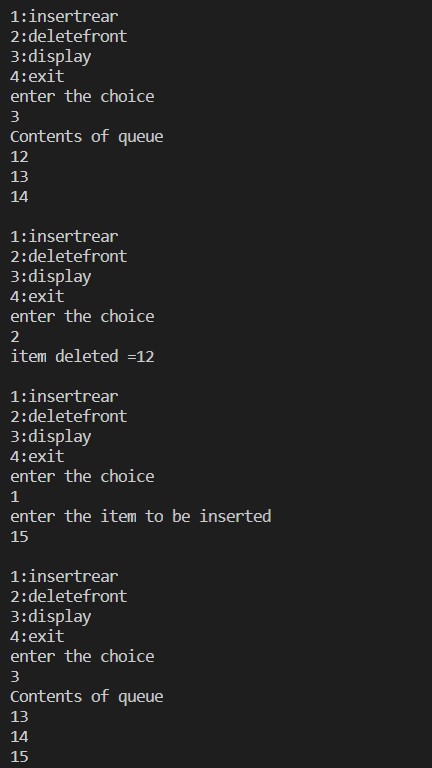
        default:exit(0);

        }

    }

}

**Output: -**



**Lab-Program 5: -**

WAP to Implement Singly Linked List with following operations

a) a) Create a linked list. b) Insertion of a node at first position, at any position and at end of

list. c) Display the contents of the linked list.

**Code: -**

#include <stdio.h>

#include <stdlib.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("List is empty, Cannot Delete item\n");

        return first;

    }

    temp=first;

    temp=temp->link;

    printf("Item Deleted at the 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("The List is Empty, Cannot Delete Item\n");

        return first;

    }

    if(first->link==NULL)

    {

        printf("Item Deleted is: %d",first->info);

        free(first);

        return NULL;

    }

    prev=NULL;

    cur=first;

    while(cur->link!=NULL)

    {

        prev=cur;

        cur=cur->link;

    }

    printf("Item Deleted at the 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 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("IP\n");

    return first;

}

void display(NODE first)

{

    NODE temp;

    if(first==NULL)

    {

        printf("List is EMPTY , Cannot Display Items\n");

        return;

    }

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

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

    {

        printf("%d\n",temp->info);

    }

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

}

void main()

{

    int item,choice,pos;

    NODE first=NULL;

    for(;;)

    {

        printf("\n1:Insert\_front\n2:Delete\_front\n3:Insert\_rear\n4:Delete\_rear\n5:insert\_pos\n6:display\_list\n7:Exit\n");

        printf("Enter the choice: ");

        scanf("%d",&choice);

        switch(choice)

        {

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

                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: ");

                scanf("%d",&item);

                first=insert\_rear(first,item);

                break;

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

                break;

            case 5:printf("Enter the position: ");

                scanf("%d",&pos);

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

                break;

            case 6:display(first);

                break;

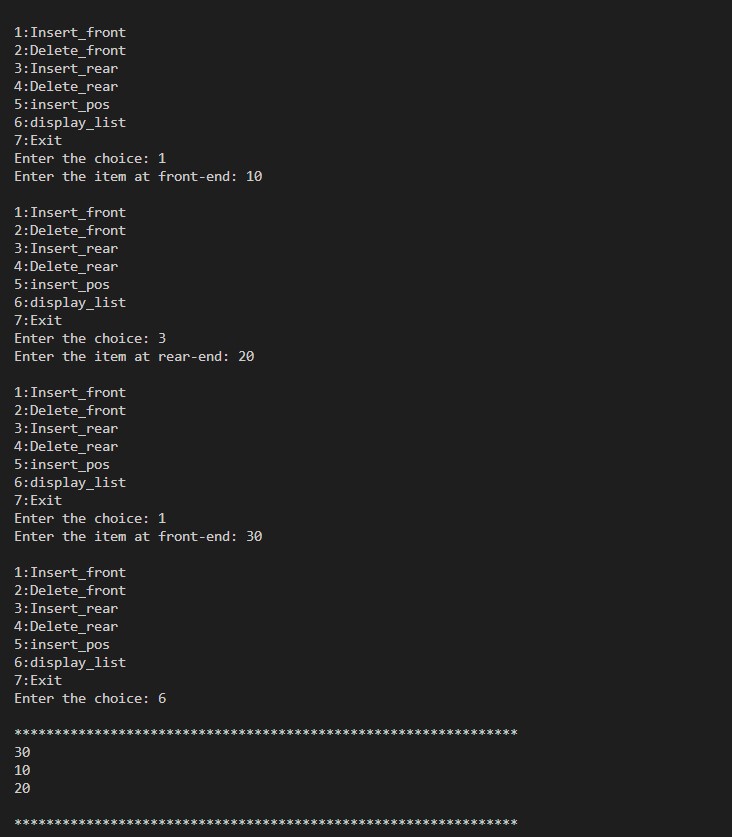
            default:exit(0);

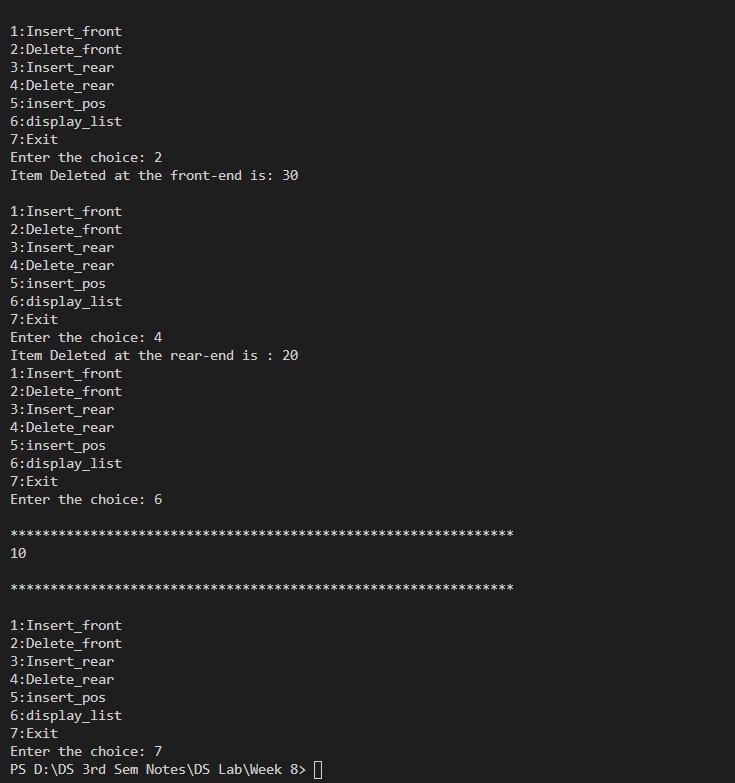
                break;

        }

    }

}

**Output: -**



**Lab-Program 6: -**

WAP to Implement Singly Linked List with following operations

a) a) Create a linked list. b) Deletion of first element, specified element and last element in

the list. c) Display the contents of the linked list.

**Code: -**

#include <stdio.h>

#include <stdlib.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("List is empty, Cannot Delete item\n");

        return first;

    }

    temp=first;

    temp=temp->link;

    printf("Item Deleted at the 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("The List is Empty, Cannot Delete Item\n");

        return first;

    }

    if(first->link==NULL)

    {

        printf("Item Deleted is: %d",first->info);

        free(first);

        return NULL;

    }

    prev=NULL;

    cur=first;

    while(cur->link!=NULL)

    {

        prev=cur;

        cur=cur->link;

    }

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

    free(cur);

    prev->link=NULL;

    return first;

}

NODE delete\_pos(int pos,NODE first)

{

    NODE prev,cur;

    int count;

    if(first==NULL || pos<=0)

    {

        printf("Invalid position\n");

        return NULL;

    }

    if(pos==1)

    {

        cur=first;

        first=first->link;

        freenode(cur);

        return first;

    }

    prev=NULL;

    cur=first;

    count=1;

    while(cur!=NULL)

    {

        if(count==pos)

        {

            break;

        }

        prev=cur;

        cur=cur->link;

        count++;

    }

    if(count!=pos)

    {

        printf("Invalid Position\n");

        return first;

    }

    prev->link=cur->link;

    freenode(cur);

    return first;

}

void display(NODE first)

{

    NODE temp;

    if(first==NULL)

    {

        printf("List is EMPTY , Cannot Display Items\n");

        return;

    }

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

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

    {

        printf("%d\n",temp->info);

    }

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

}

void main()

{

    int item,choice,pos;

    NODE first=NULL;

    for(;;)

    {

        printf("\n1:Insert\_front\n2:Delete\_front\n3:Insert\_rear\n4:Delete\_rear\n5:delete\_pos\n6:display\_list\n7:Exit\n");

        printf("Enter the choice: ");

        scanf("%d",&choice);

        switch(choice)

        {

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

                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: ");

                scanf("%d",&item);

                first=insert\_rear(first,item);

                break;

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

                break;

            case 5:printf("Enter the position: ");

                scanf("%d",&pos);

                first=delete\_pos(pos,first);

                break;

            case 6:display(first);

                break;

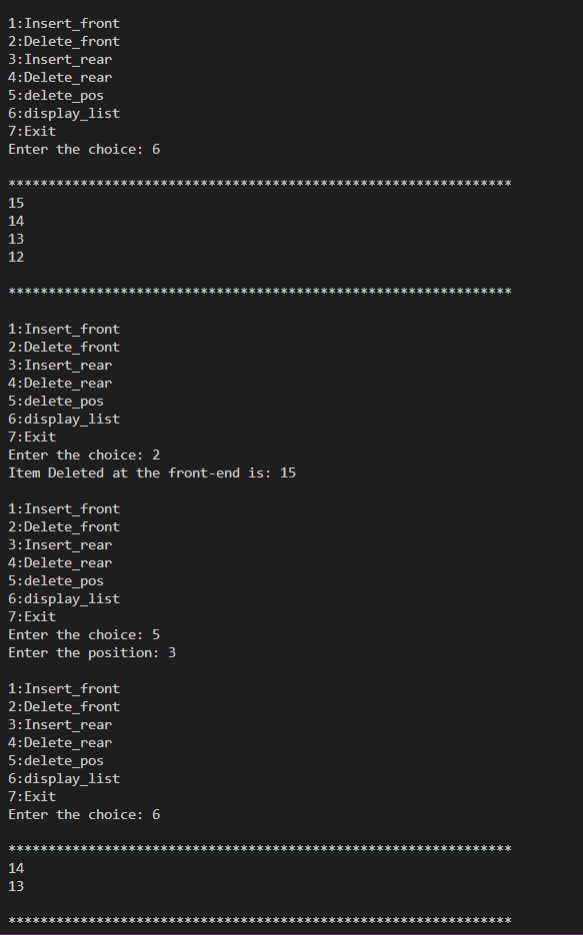
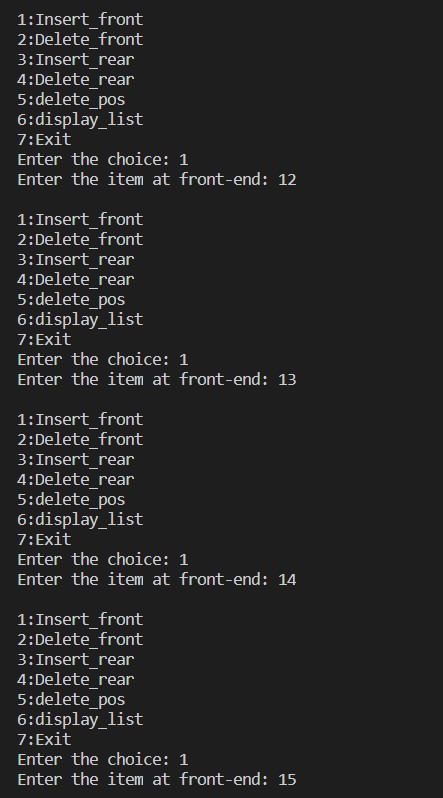
            default:exit(0);

                break;

        }

    }

}

**Output: -**

**Lab-Program 7: -**

WAP Implement Single Link List with following operations

1. Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists

**Code: -**

#include<stdio.h>

#include<stdlib.h>

struct node

{

    int info;

    struct node \*link;

};

typedef struct node \*NODE;

NODE getnode()

{

    NODE x;

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

    if(x==NULL)

    {

        printf("Memory full\n");

        exit(0);

    }

    return x;

}

void freenode(NODE x)

{

    free(x);

}

void display (NODE first)

{

    NODE temp;

    if(first==NULL)

    {

        printf("Linked List is empty ,Cannot Display items");

        return;

    }

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

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

    {

        printf("%d\n",temp->info);

    }

}

NODE order\_list(int item,NODE first)

{

    NODE temp,prev,cur;

    temp=getnode();

    temp->info=item;

    temp->link=NULL;

    if(first==NULL) return temp;

    if(item<first->info)

    {

        temp->link=first;

        return temp;

    }

    prev=NULL;

    cur=first;

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

    {

        prev=cur;

        cur=cur->link;

    }

    prev->link=temp;

    temp->link=cur;

    return first;

}

NODE concat(NODE first,NODE second)

{

    NODE cur;

    if(first==NULL)

        return second;

    if(second==NULL)

        return first;

    cur=first;

    while(cur->link!=NULL)

        cur=cur->link;

    cur->link=second;

    return first;

}

NODE reverse(NODE first)

{

    NODE cur,temp;

    cur=NULL;

    while(first!=NULL)

    {

        temp=first;

        first=first->link;

        temp->link=cur;

        cur=temp;

    }

    return cur;

}

int main()

{

    NODE first=NULL;

    NODE second=NULL;

    int item, choice,llno;

    for(;;)

    {

        printf("1:insert in order list\n2:Concatenate the two linked lists\n3:reverse the linked list\n4:Display\n5:EXIT\n");

        printf("Enter your choice: ");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1: printf("Enter the LL number (1 or 2) :");

                    scanf("%d",&llno);

                    printf("Enter the item at the rear end: ");

                    scanf("%d",&item);

                    if(llno==1)

                        first=order\_list(item,first);

                    else

                        second=order\_list(item,second);

                    break;

            case 2: first=concat(first,second);

                    second=NULL;

                    break;

            case 3: first=reverse(first);

                    break;

            case 4: printf("The contents of the first LL:\n");

                    display(first);

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

                    printf("\nThe contents of the second LL:\n");

                    display(second);

                    break;

            case 5: exit(0);

                    break;

            default: printf("Enter a valid option\n");

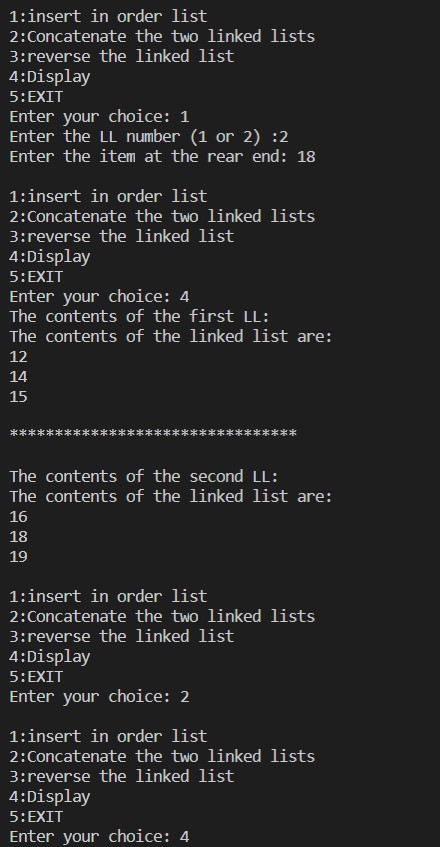
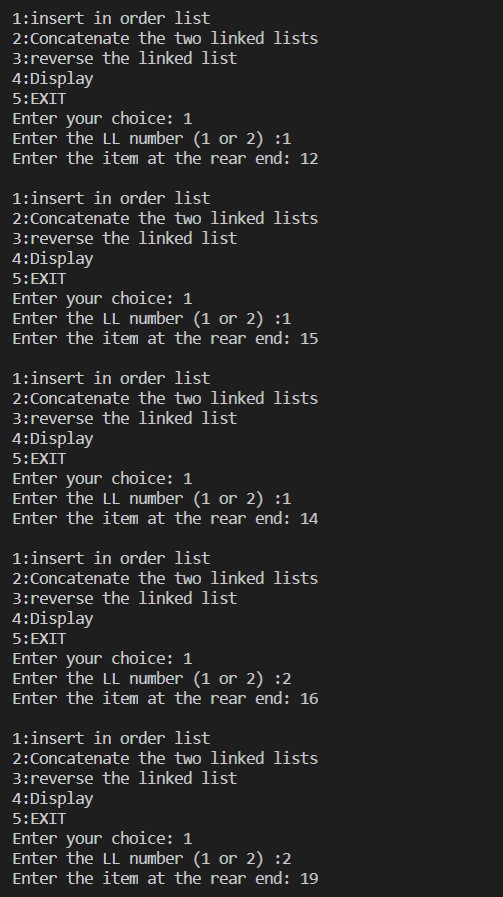
        }

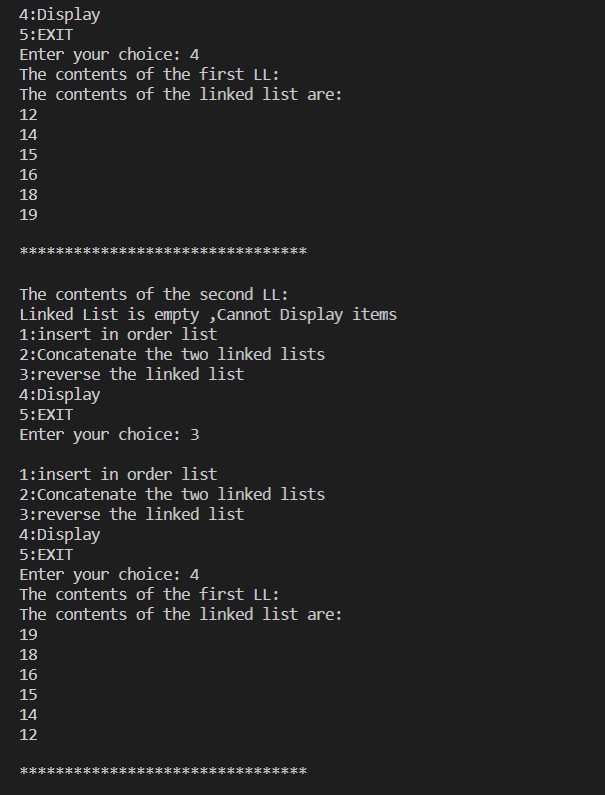
        printf("\n");

    }

    return 0;

}

**Output: -**



**Lab-Program 8: -**

WAP to implement Stack & Queues using Linked Representation  
**Code: -**

**Stacks: -**

#include <stdio.h>

#include <stdlib.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("List is empty, Cannot Delete item\n");

        return first;

    }

    temp=first;

    temp=temp->link;

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

    free(first);

    return temp;

}

void display(NODE first)

{

    NODE temp;

    if(first==NULL)

    {

        printf("List is EMPTY , Cannot Display Items\n");

        return;

    }

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

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

    {

        printf("%d\n",temp->info);

    }

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

}

void main()

{

    int item,choice,pos;

    NODE first=NULL;

    for(;;)

    {

        printf("\n1:PUSH\n2:POP\n3:insert\_pos\n4:display\_list\n5:Exit\n");

        printf("Enter the choice: ");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:printf("Enter the item : ");

                scanf("%d",&item);

                first=insert\_front(first,item);

                break;

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

                break;

            case 3:printf("Enter the position: ");

                scanf("%d",&pos);

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

                break;

            case 4:display(first);

                break;

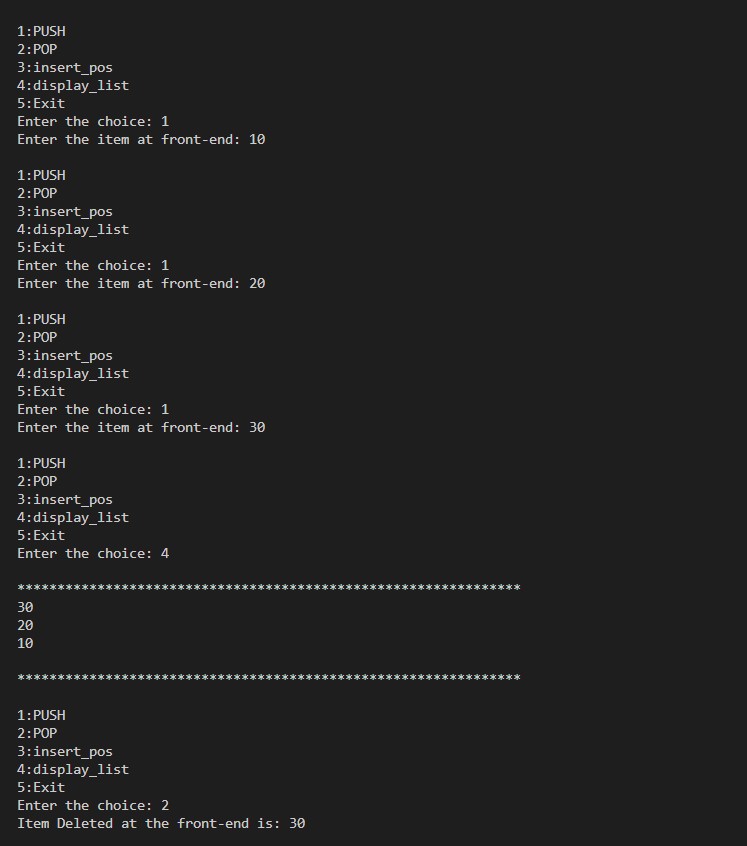
            default:exit(0);

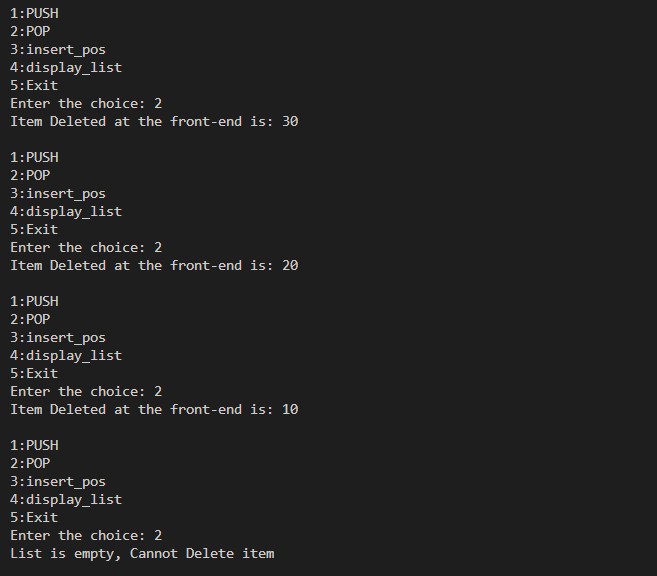
                break;

        }

    }

}

**Output for Stacks: -**



**Code for Queues: -**

#include <stdio.h>

#include <stdlib.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 delete\_front(NODE first)

{

    NODE temp;

    if(first==NULL)

    {

        printf("List is empty, Cannot Delete item\n");

        return first;

    }

    temp=first;

    temp=temp->link;

    printf("Item Deleted at the 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;

}

void display(NODE first)

{

    NODE temp;

    if(first==NULL)

    {

        printf("List is EMPTY , Cannot Display Items\n");

        return;

    }

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

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

    {

        printf("%d\n",temp->info);

    }

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

}

void main()

{

    int item,choice,pos;

    NODE first=NULL;

    for(;;)

    {

        printf("\n1:Insert\_rear\n2:Delete\_front\n3:display\_Queue\n4:Exit\n");

        printf("Enter the choice: ");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:printf("Enter the item at rear-end: ");

                scanf("%d",&item);

                first=insert\_rear(first,item);

                break;

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

                break;

            case 3:display(first);

                break;

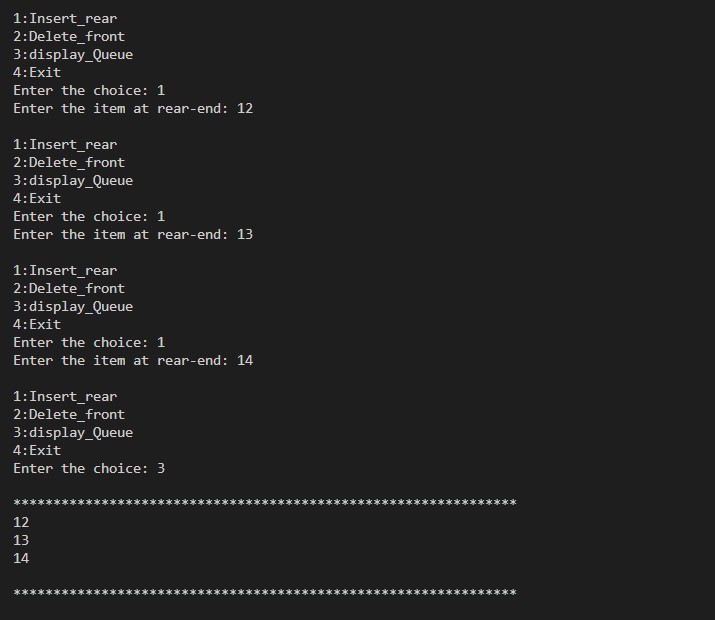
            default:exit(0);

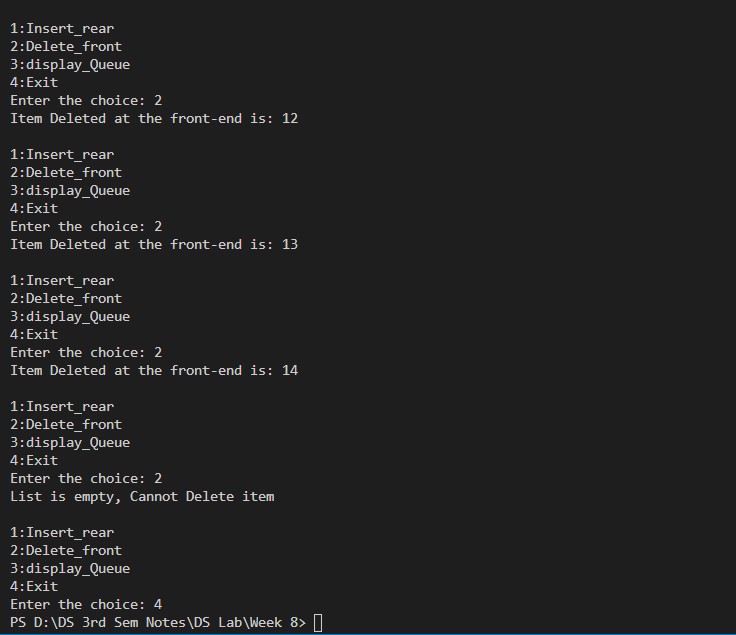
                break;

        }

    }

}

**Output for Queues: -**



**Lab-Program 9: -**

WAP Implement doubly link list with primitive operations

a) a) Create a doubly linked list. b) Insert a new node to the left of the node.

b) c) Delete the node based on a specific value. c) Display the contents of the list

**Code: -**

#include <stdio.h>

#include <stdlib.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 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("The DLL is 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("The DLL is 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;

}

NODE insert\_leftpos(int item,NODE head)

{

    NODE temp,cur,prev;

    int item2;

    if(head->rlink==head)

    {

        printf("List empty\n");

        return head;

    }

    cur=head->rlink;

    while(cur!=head)

    {

        if(item==cur->info)break;

        cur=cur->rlink;

    }

    if(cur==head)

    {

        printf("key not found\n");

        return head;

    }

    prev=cur->llink;

    temp=getnode();

    printf("Enter towards left of %d : ",item);

    scanf("%d",&item2);

    temp->info=item2;

    prev->rlink=temp;

    temp->llink=prev;

    cur->llink=temp;

    temp->rlink=cur;

    return head;

}

NODE insert\_rightpos(int item,NODE head)

{

    NODE temp,cur,next;

    int item2;

    if(head->rlink==head)

    {

        printf("List empty\n");

        return head;

    }

    cur=head->rlink;

    while(cur!=head)

    {

        if(item==cur->info)break;

        cur=cur->rlink;

    }

    if(cur==head)

    {

        printf("key not found\n");

        return head;

    }

    next=cur->rlink;

    temp=getnode();

    printf("Enter towards right of %d : ",item);

    scanf("%d",&item2);

    temp->info=item2;

    next->llink=temp;

    temp->rlink=next;

    cur->rlink=temp;

    temp->llink=cur;

    return head;

}

NODE delete\_all\_key(int item, NODE head)

{

    NODE prev,cur,next;

    int count;

    if(head->rlink==head)

    {

        printf("List empty\n");

        return head;

    }

    count=0;

    cur=head->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\n");

    else

        printf("Key found at %d positions !! and are deleted",count);

    return head;

}

void searching(int key,NODE head)

{

    NODE temp,cur;

    if(head->rlink==head)

    {

        printf("list empty\n");

        return;

    }

    cur=head->rlink;

    while(cur!=head)

    {

        if(cur->info==key)

        {

            printf("Search Successful\n");

            return;

        }

        cur=cur->rlink;

    }

    printf("Search is not successfull\n");

    return;

}

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;

}

void display(NODE head)

{

    NODE temp;

    if(head->rlink==head)

    {

        printf("The DLL is empty");

        return;

    }

    printf("the contents of the DLL\n");

    temp=head->rlink;

    while(temp!=head)

    {

        printf("%d\n",temp->info);

        temp=temp->rlink;

    }

    printf("\n");

}

void main()

{

    NODE head,last;

    int item,choice;

    head=getnode();

    head->rlink=head;

    head->llink=head;

    for(;;)

    {

        printf("\n1:Insert front\n2:Insert rear\n3:Delete front\n4:Delete rear\n5:Insert\_key\_Left\n6:Insert\_key\_Right\n7:Delete all keys\n8:Search item\n9:Delete Only duplicates\n10:Display\n11:Exit\n");

        printf("Enter the choice: ");

        scanf("%d",&choice);

        switch(choice)

        {

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

                    scanf("%d",&item);

                    last=dinsert\_front(item,head);

                    break;

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

                    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:printf("Enter the key item: ");

                    scanf("%d",&item);

                    head=insert\_leftpos(item,head);

                    break;

            case 6:printf("Enter the key item: ");

                    scanf("%d",&item);

                    head=insert\_rightpos(item,head);

                    break;

            case 7:printf("Enter the key item: ");

                    scanf("%d",&item);

                    head=delete\_all\_key(item,head);

                    break;

            case 8:printf("Enter the key item: ");

                    scanf("%d",&item);

                    searching(item,head);

                    break;

            case 9:printf("Enter the key item: ");

                    scanf("%d",&item);

                    head=ddelete\_duplicates(item,head);

                    break;

            case 10: display(head);

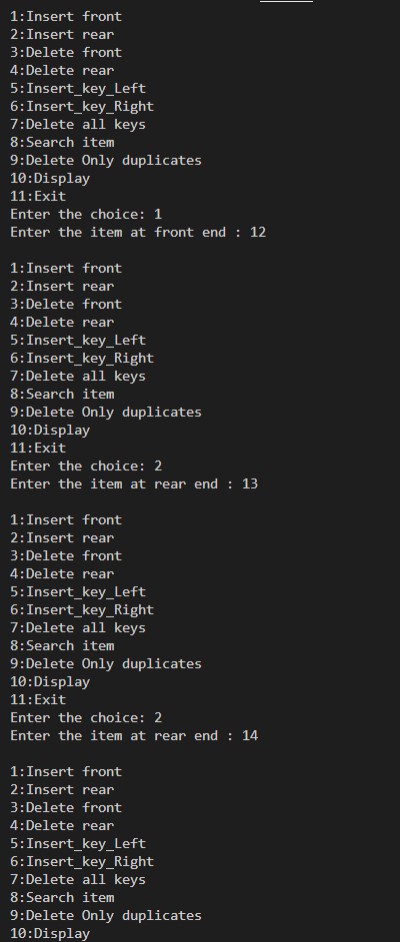
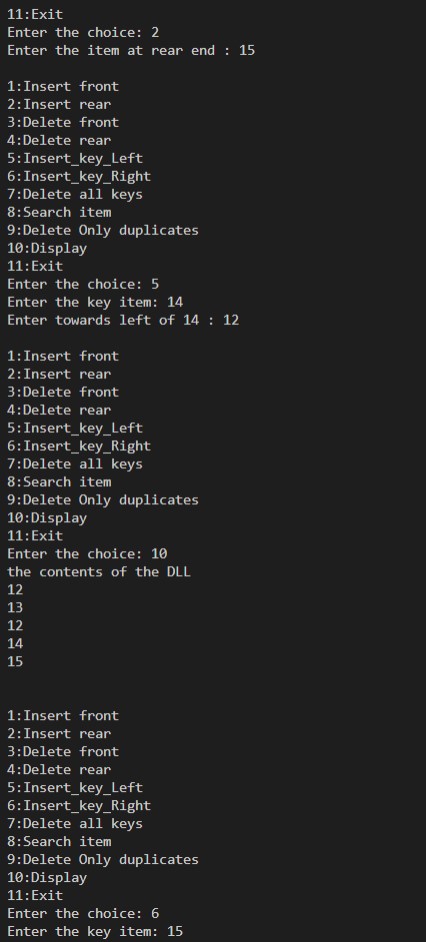
                    break;

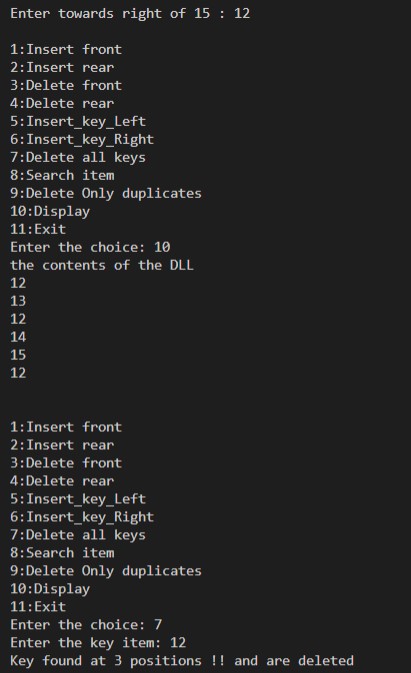
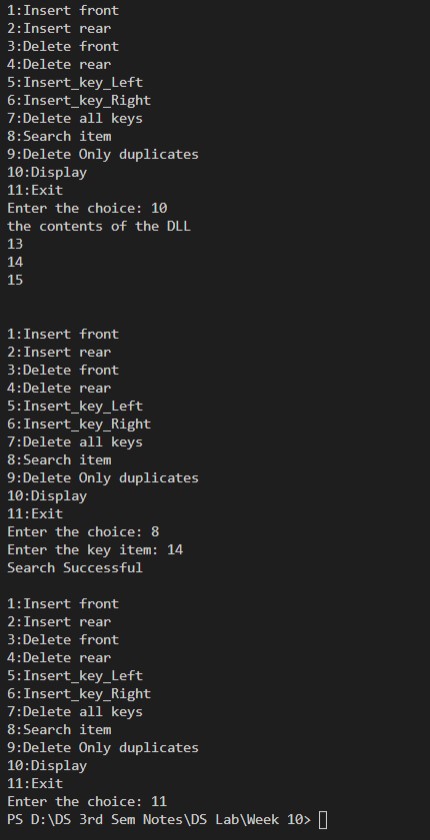
            default: return;

        }

    }

}

**Output: -**



**Lab-Program 10: -**

Write a program

a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., in-order, preorder and post order

c) To display the elements in the tree.

**Code: -**

#include<stdio.h>

#include<stdlib.h>

#include<string.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("mem 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("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("\n1.insert\n2.display\n3.pre\n4.post\n5.in\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:preorder(root);

                break;

              case 4:postorder(root);

                break;

              case 5:inorder(root);

                break;

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

                scanf("%d",&item);

                root=delete(root,item);

                break;

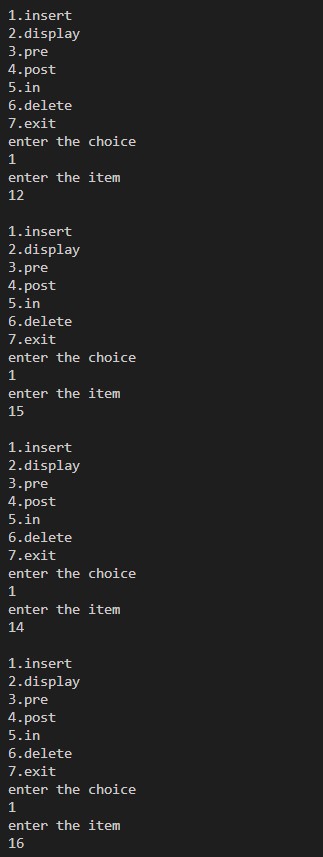
              default:exit(0);

                  break;

        }

    }

}

**Output: -**

