VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

****

LAB REPORT

on

DATA STRUCTURES

Submitted by

NAME: NEHA BHASKAR KAMATH

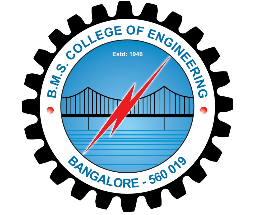
USN: 1BM21CS113

in partial fulfilment 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

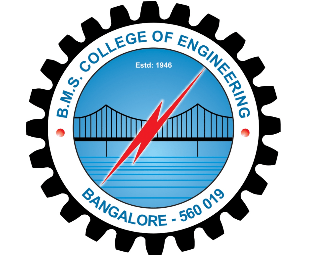
Oct 2022-Feb 2023

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 work entitled “DATA STRUCTURES” carried out by NEHA BHASKAR KAMATH(1BM21CS113) , who is bonafide student of B. M. S. College of Engineering. It is in partial fulfilment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2022-23.

The Lab report has been approved as it satisfies the academic requirements in respect of Data Structures Lab - (22CS3PCDST) work prescribed for the said degree.

Sonika Sharma D          Dr. Jyothi S Nayak

Assistant professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

INDEX

|  |  |  |
| --- | --- | --- |
| Sl.No | Experiment Title | Page No |
| 1 | Implementation of stacks | 4-6 |
| 2 | Conversion of infix to postfix expression | 7-9 |
| 3 | Implementation of Linear queues | 10-12 |
| 4 | Implementation of Circular queues | 13-15 |
| 5 | Insert operations in singly linked list | 16-20 |
| 6 | Delete operations in singly linked list | 21-26 |
| 7.1 | Sorting a singly linked list | 27-29 |
| 7.2 | Reversing a singly linked list | 30-32 |
| 7.3 | Concatenation of two singly linked lists | 32-35 |
| 8 | Implementation of stacks and queues using singly linked list | 36-41 |
| 9 | Implementation of doubly linked list | 42-47 |
| 10 | Implementation of binary search tree | 48-52 |

COURSE OUTCOME

|  |  |
| --- | --- |
| CO1 | Apply the concept of linear and nonlinear data structures. |
| CO2 | Analyse data structure operations for a given problem. |
| CO3 | Design and develop solutions using the operations of linear and non linear data structure for a given specification. |
| CO4 | Conduct practical experiments for demonstrating the operations of different data structures. |

**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 and stack underflow.

**C-CODE:**

#include <stdio.h>

#include<stdlib.h>

#define size 10

void push(int st[],int \*top,int data)

{

    if(\*top==size-1)

    printf("Stack overflow condition!\n");

    else

    {

     (\*top)++;

    st[\*top]=data;

    }

}

int pop(int st[], int \*top)

{

    int n;

    if(\*top==-1)

    printf("Stack underflow condition!\n");

    else

    {

        n=st[\*top];

        (\*top)--;

        return n;

    }

}

void display(int st[], int \*top)

{

    int i;

    if(\*top==-1)

    printf("No values to display.\n");

    else

    {

        for(i=0;i<=\*top;i++)

        {

            printf("Element %d is %d.\n",i+1,st[i]);

        }

    }

}

void main()

{

    int st[size],top=-1,data,c;

    while(1)

    {

        printf("Enter 1 for inserting values.\nEnter 2 for deleting values.\nEnter 3 for displaying values.\nEnter 4 to exit.\n");

        scanf("%d",&c);

        switch(c)

        {

            case 1:

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

            scanf("%d",&data);

            push(st, &top,data);

            break;

            case 2:

            printf("The popped element is %d.\n", pop(st, &top));

            break;

            case 3:

            display(st, &top);

            break;

            case 4:

            exit(0);

            default:

            printf("Enter a valid choice1\n");

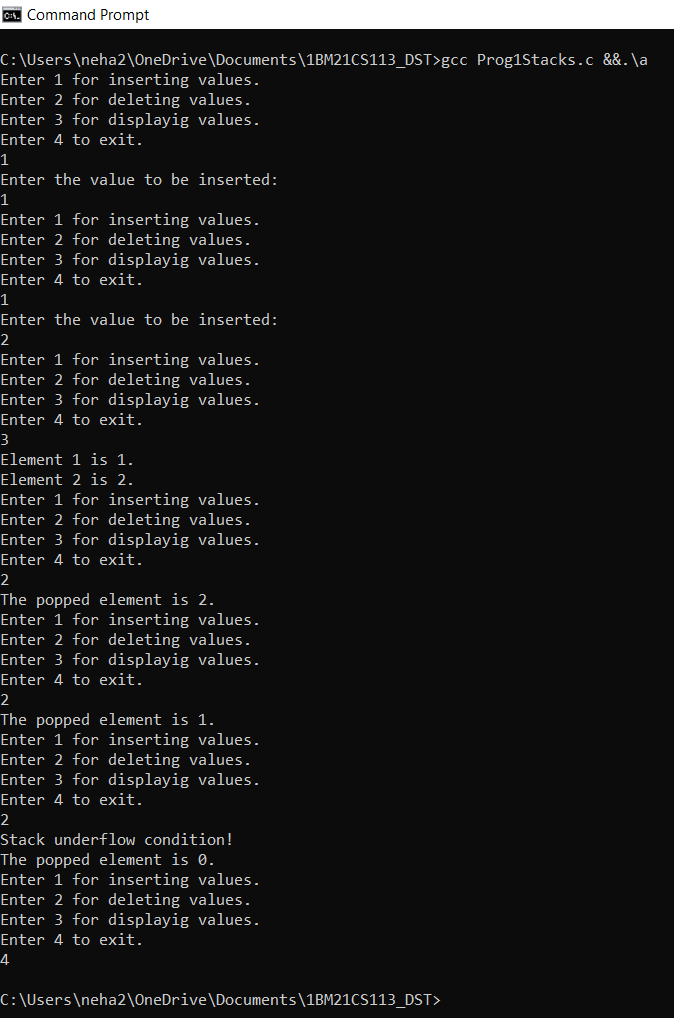
            break;

        }

    }

}

**OUTPUT:**



**LAB PROGRAM 2:**

Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression.

**C-CODE:**

#include <stdio.h>

#include<string.h>

int top=-1;

char s[20];

char infix[20];

char postfix[20];

void inf\_to\_post();

int sp(char);

int ip(char);

void push(char);

char pop();

 void main()

 {

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

     scanf("%s",infix);

     inf\_to\_post();

     printf("The postfix expression is %s.\n",postfix);

 }

void push(char item)

{

    s[++top]=item;

}

char pop()

{

    return s[top--];

}

int sp(char item)

{

    switch(item)

    {

        case '+':

        case '-': return 2;

        case '\*':

        case '/': return 4;

        case '^':

        case '$': return 5;

        case '(':return 0;

        case '#': return -1;

        default:return 8;

    }

}

int ip(char item)

{

   switch(item)

    {

        case '+':

        case '-': return 1;

        case '\*':

        case '/': return 3;

        case '^':

        case '$': return 6;

        case '(':return 9;

        case ')': return 0;

        default:return 7;

    }

}

void inf\_to\_post()

{

    int i,j=0;

    push('#');

    for(i=0;i<strlen(infix);i++)

    {

        while(sp(s[top])>ip(infix[i]))

        {

            postfix[j]=pop();

            j++;

        }

        if(sp(s[top])<ip(infix[i]))

        {

            push(infix[i]);

        }

        if(sp(s[top])==ip(infix[i]))

        {

            pop();

        }

    }

    while(s[top]!='#')

    {

        postfix[j]=pop();

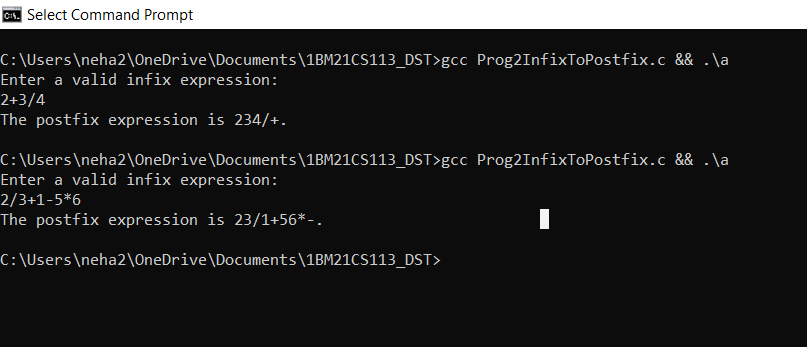
        j++;

    }

    postfix[j]='\0';

}

**OUTPUT:**



**LAB PROGRAM 3:**

Write a program to simulate the working of linear queue using an array. Perform the following operations:

(a)Insert

(b)Delete

(c)Display

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

**C-CODE:**

#include<stdio.h>

#include<stdlib.h>

#define size 10

void insert\_rear(int q[],int \*r, int item)

{

    if(\*r==size-1)

    printf("Queue overflow!\n");

    else

    {

        (\*r)++;

        q[\*r]=item;

    }

}

int delete\_front(int q[],int \*r, int \*f)

{

    int del\_item=-1;

    if(\*f>\*r)

    printf("Queue underflow!\n");

    else

    {

        del\_item=q[\*f];

        (\*f)++;

    }

    return del\_item;

}

void display(int q[],int \*r, int \*f)

{

    int i;

    if(\*f>\*r)

    printf("Queue is empty!\n");

    else

    {

        for(i=\*f;i<=\*r;i++)

        printf("Element %d is %d.\n",i+1,q[i]);

    }

}

int main()

{

    int q[size],data,c;

    int f=0;

    int r=-1;

    while(1)

    {

        printf("Enter 1 for inserting values.\nEnter 2 for deleting values.\nEnter 3 for displayig values.\nEnter 4 to exit.\n");

        scanf("%d",&c);

        switch(c)

        {

            case 1:

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

            scanf("%d",&data);

            insert\_rear(q,&r,data);

            break;

            case 2:

            printf("The deleted element is %d.\n", delete\_front(q,&r,&f));

            break;

            case 3:

            display(q,&r,&f);

            break;

            case 4:

            exit(0);

            default:

            printf("Enter a valid choice1\n");

            break;

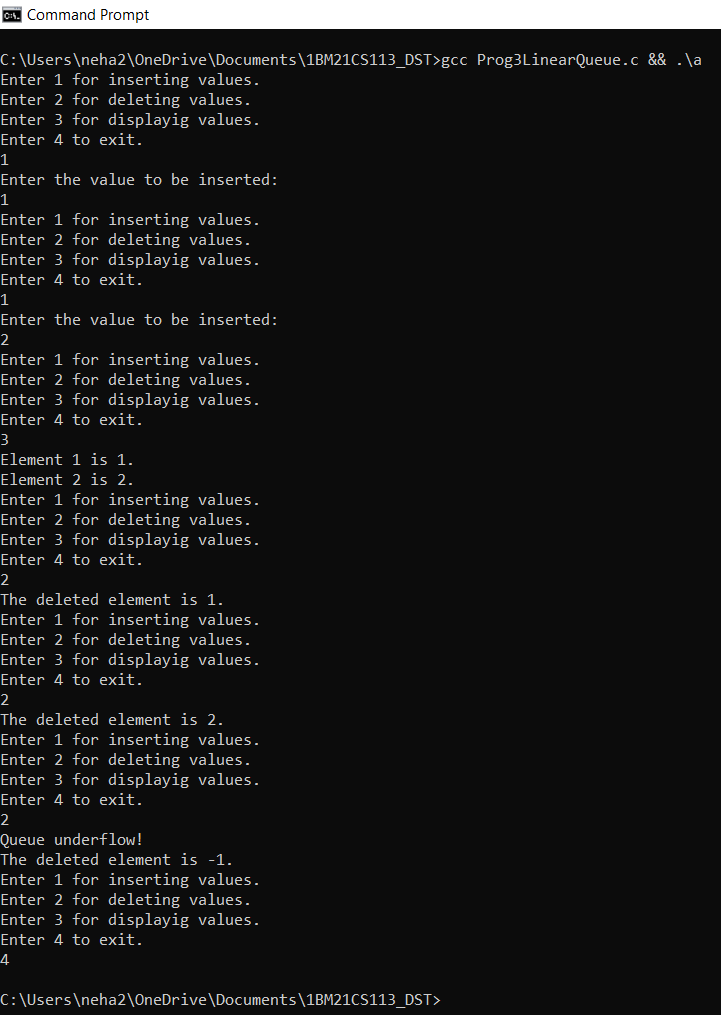
        }

    }

    return 0;

}

**OUTPUT:**



**LAB PROGRAM 4:**

Write a program to simulate the working of circular queue using an array. Perform the following operations:

(a)Insert

(b)Delete

(c)Display

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

**C-CODE:**

#include<stdio.h>

#include<stdlib.h>

#define size 3

void insert\_rear(int q[],int \*r, int item, int \*count)

{

    if(\*count==size)

    printf("Queue overflow!\n");

    else

    {

        q[\*r]=item;

        \*r=((\*r)+1)%size;

        (\*count)=(\*count)+1;

    }

}

int delete\_front(int q[],int \*r, int \*f, int \*count)

{

    int del\_item=-1;

    if(\*count==0)

    printf("Queue underflow! \n");

    else

    {

        del\_item=q[(\*f)];

        (\*f)=((\*f)+1)%size;

        (\*count)=(\*count)-1;

    }

    return del\_item;

}

void display(int q[],int \*f, int \*count)

{

    int i,temp;

    temp=\*f;

    for(i=0;i<\*count;i++)

    {

      Printf ("Element %d is %d.\n",i+1,q[temp]);

      temp=(temp+1)%size;

    }

}

int main()

{

    int q[size],data,c;

    int f=0;

    int r=0;

    int count=0;

    while(1)

    {

        printf("Enter 1 for inserting values.\nEnter 2 for deleting values.\nEnter 3 for displayig values.\nEnter 4 to exit.\n");

        scanf("%d",&c);

        switch(c)

        {

            case 1:

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

            scanf("%d",&data);

            insert\_rear(q,&r,data,&count);

            break;

            case 2:

            printf("The deleted element is %d.\n",delete\_front(q,&r,&f,&count));

            break;

            case 3:

            display(q,&f,&count);

            break;

            case 4:

            exit(0);

            default:

            printf("Enter a valid choice1\n");

            break;

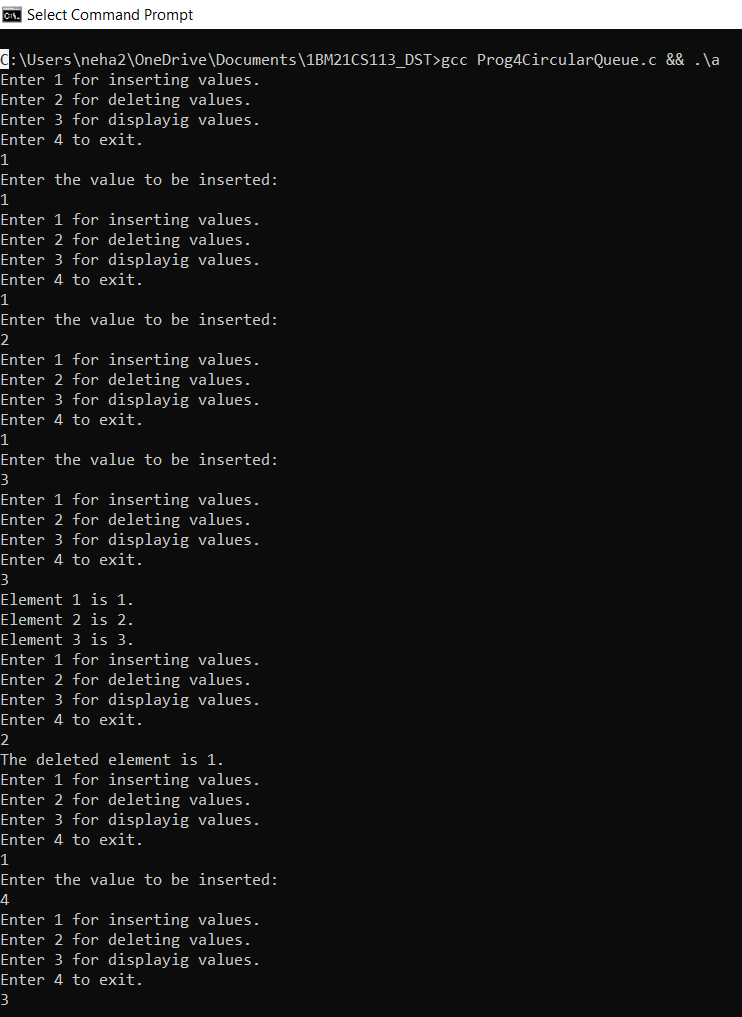
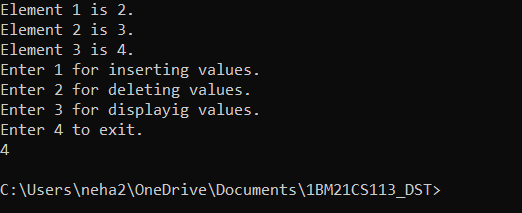
        }

    }

    return 0;

}

**OUTPUT:**



**LAB 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 the list.

(c) Display the contents of the linked list.

**C-CODE:**

#include <stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

};

struct Node \*getNode()

{

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

    if(temp==NULL)

    {

        printf("Memory full!\n");

        return NULL;

    }

    else

    return temp;

}

void display(struct Node \*head)

{

    int i;

    i=0;

    if(head==NULL)

    printf("Nothing to display!\n");

    else

    {

    struct Node \*ptr=head;

    while(ptr!=NULL)

    {

        printf("Element %d is %d\n",(i+1),(ptr->data));

       // printf("%d\n",ptr->data);

        ptr=ptr->next;

        i++;

    }

}

}

struct Node \*insertAtFirst(struct Node \*head,int data)

{

    struct Node \*ptr=getNode();

    ptr->data=data;

    ptr->next=NULL;

    if(head==NULL)

    {

        head=ptr;

        return head;

    }

    else

    {

    ptr->next=head;

    head=ptr;

    return head;

    }

}

struct Node \*insertInBet(struct Node \*head,int data,int index)

{

    int i=0;

    struct Node \*ptr=getNode();

    ptr->data=data;

    ptr->next=NULL;

    if(head==NULL)

    {

        head=ptr;

        return head;

    }

    else

    {

    struct Node \*p=head;

    if(index==0)

    return p;

    while(i!=index-1)

    {

        p=p->next;

        i++;

    }

    ptr->data=data;

    ptr->next=p->next;

    p->next=ptr;

    return head;

   }

}

struct Node \*insertAtEnd(struct Node \*head,int data)

{

    struct Node \*ptr=getNode();

    ptr->data=data;

    ptr->next=NULL;

    if(head==NULL)

    {

        head=ptr;

        return head;

    }

    else

    {

    struct Node \*p=head;

    while(p->next!=NULL)

    {

        p=p->next;

    }

    ptr->next=NULL;

    p->next=ptr;

    return head;

    }

}

int main()

{

    struct Node \*head=NULL;

    int c,data,index;

    while(1)

    {

        printf("Enter 1 to insert at the beginning.\nEnter 2 to insert in between.\nEnter 3 to insert at the end.\nEnter 4 to display.\nEnter 5 to exit!\n");

        scanf("%d",&c);

        switch(c)

        {

            case 1:

            printf("Enter the value to be inserted at the beginning:\n");

            scanf("%d",&data);

            head=insertAtFirst(head,data);

            break;

            case 2:

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

            scanf("%d",&data);

            printf("Enter the index where the value has to be inserted:\n");

            scanf("%d",&index);

            head=insertInBet(head,data,index);

            break;

            case 3:

            printf("Enter the value to be inserted at the end:\n");

            scanf("%d",&data);

            head=insertAtEnd(head,data);

            break;

            case 4:

            display(head);

            break;

            case 5:

            exit(0);

            default:

            printf("Invalid choice!\n");

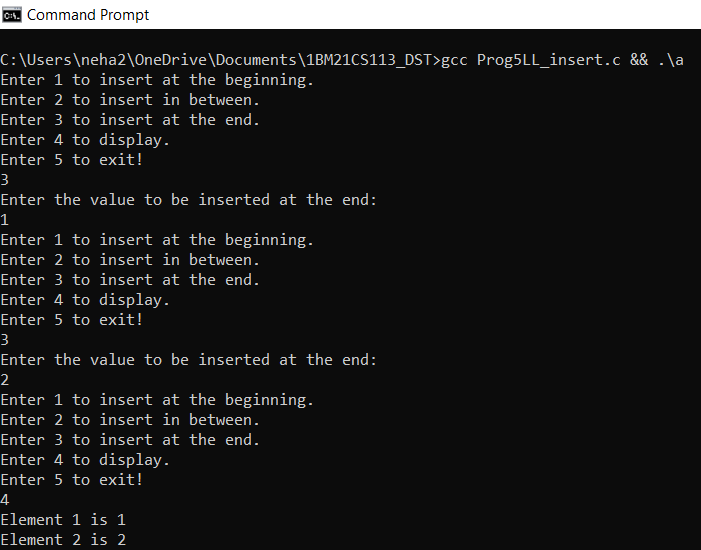
        }

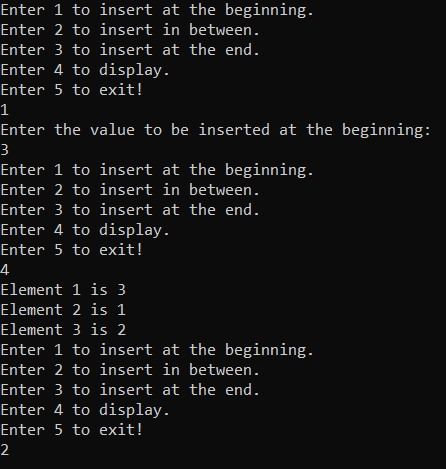
    }

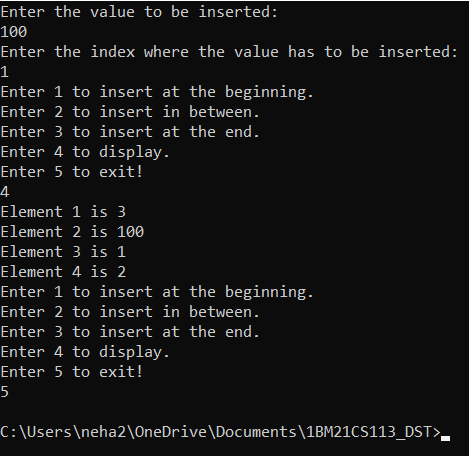
    return 0;

}

**OUTPUT:**







**LAB PROGRAM 6:**

Write a program to implement singly linked list with following operations:

(a) Create a linked list.

(b) Deletion of first element, specified element and the last element in the list.

(c) Display the contents of the linked list.

**C-CODE:**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

};

void show(struct Node \*head)

{

    int i;

    struct Node \*ptr=head;

    i=0;

    if(head==NULL)

    {

        printf("Nothing to display!\n");

    }

    else

    {

        while(ptr!=NULL)

        {

            printf("Element %d is %d\n",(i+1),(ptr->data));

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

            ptr=ptr->next;

            i++;

        }

    }

}

struct Node \*insertAtEnd(struct Node \*head,int data)

{

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

    ptr->data=data;

    ptr->next=NULL;

    if(head==NULL)

    {

        head=ptr;

        return head;

    }

    else

    {

    struct Node \*p=head;

    while(p->next!=NULL)

    {

        p=p->next;

    }

    ptr->next=NULL;

    p->next=ptr;

    return head;

    }

}

struct Node \*deleteAtFirst(struct Node \*head)

{

    if(head==NULL)

    {

        printf("No linked list!\n");

        return NULL;

    }

    else if(head->next==NULL)

    {

        head=NULL;

        return head;

    }

    else

    {

    struct Node \*p=head;

    head=head->next;

    free(p);

    return head;

    }

}

struct Node \*deleteInBet(struct Node \*head,int index)

{

    if(head==NULL)

    {

        printf("No linked list!\n");

        return NULL;

    }

    else if(head->next==NULL && index==0)

    {

        head=NULL;

        return head;

    }

    else

    {

    int i=0;

    struct Node \*p=head;

    struct Node \*q=head->next;

    while(i!=index-1)

    {

        p=p->next;

        q=q->next;

        i++;

    }

    p->next=q->next;

    free(q);

    return head;

}

}

struct Node\* deleteAtEnd(struct Node \*head)

{

    if(head==NULL)

    {

        printf("Nothing to delete!\n");

        return NULL;

    }

    else if(head->next==NULL)

    {

        head=NULL;

        return head;

    }

    else

    {

    int i=0;

    struct Node \*p=head;

    struct Node \*q=head->next;

    while(q->next!=NULL)

    {

        p=p->next;

        q=q->next;

        i++;

    }

    p->next=NULL;

    free(q);

    return head;

}

}

int main()

{

    struct Node \*head=NULL;

    int c,data,index,i;

    while(1)

    {

        printf("Enter 1 to create a linked list.\n");

        printf("Enter 2 to delete at the beginning.\nEnter 3 to delete in between.\nEnter 4 to delete at the end.\n");

        printf("Enter 5 to display.\nEnter 6 to exit!\n");

        scanf("%d",&c);

        switch(c)

        {

            case 1:

            printf("Enter the value to be inserted at the end:\n");

            scanf("%d",&data);

            head=insertAtEnd(head,data);

            break;

            case 2:

            head=deleteAtFirst(head);

            break;

            case 3:

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

            scanf("%d",&i);

            head=deleteInBet(head,i);

            break;

            case 4:

            head=deleteAtEnd(head);

            break;

            case 5:

            show(head);

            break;

            case 6:

            exit(0);

            default:

            printf("Invalid choice!\n");

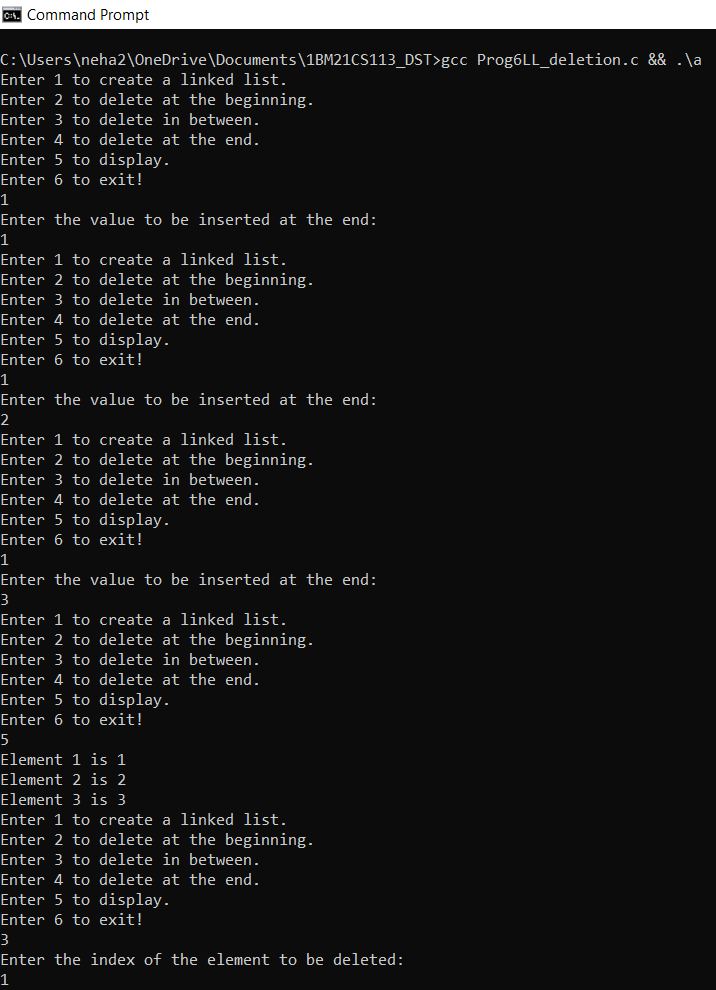
        }

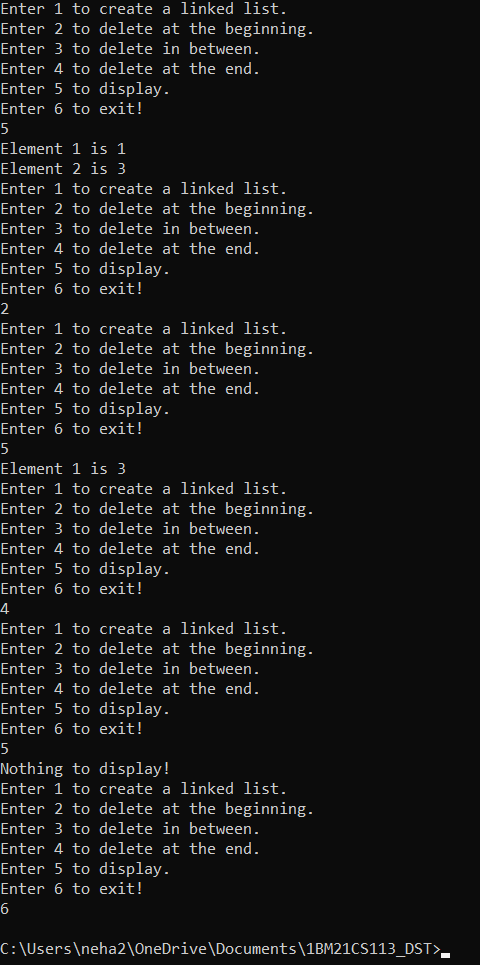
    }

    return 0;

}

**OUTPUT:**





**LAB PROGRAM 7:**

Write a program to implement a singly linked list with following operations:

(a) Sort the linked list.

(b) Reverse the linked list.

(c) Concatenation of two linked list.

(a)Sorting:

**C-CODE:**

#include <stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

};

void display(struct Node \*head)

{

       int i=0;

      if(head==NULL)

    {

        printf("Nothing to display!\n");

    }

    else

    {

        struct Node \*ptr=head;

        while(ptr!=NULL)

        {

            printf("Element %d is %d.\n",(i+1),ptr->data);

            ptr=ptr->next;

            i++;

        }

    }

}

struct Node \*insertAtEnd(struct Node \*head,int data)

{

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

    ptr->data=data;

    ptr->next=NULL;

    if(head==NULL)

    {

        head=ptr;

        return head;

    }

    else

    {

    struct Node \*p=head;

    while(p->next!=NULL)

    {

        p=p->next;

    }

    ptr->next=NULL;

    p->next=ptr;

    return head;

    }

}

struct Node \*sort(struct Node \*head)

{

    int temp;

    if(head==NULL)

    {

        printf("No linked list!\n");

        return NULL;

    }

    else

    {

        struct Node \*p=head;

        while(p->next!=NULL)

        {

            struct Node \*q=p->next;

            while(q!=NULL)

            {

                if(p->data>q->data)

                {

                    temp=p->data;

                    p->data=q->data;

                    q->data=temp;

                }

                q=q->next;

            }

            p=p->next;

        }

    }

    return head;

}

int main()

{

    int n,i,data;

    struct Node \*head=NULL;

    printf("Enter the number of terms you want to enter in the linked list:\n");

    scanf("%d",&n);

    for(i=1;i<=n;i++)

    {

        printf("Enter element %d to be inserted:\n",i);

        scanf("%d",&data);

        head=insertAtEnd(head,data);

    }

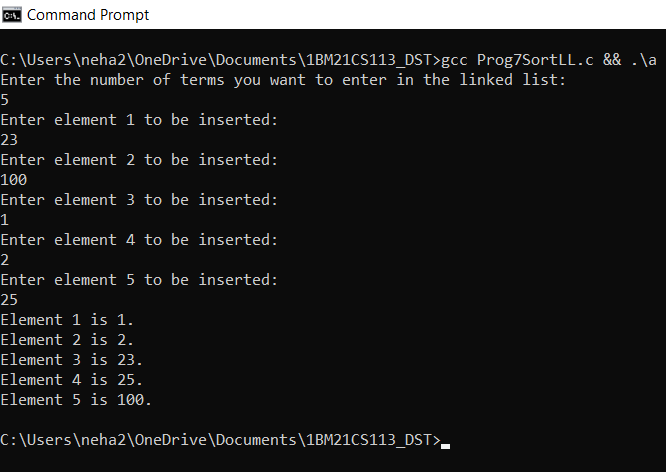
    head=sort(head);

    display(head);

    return 0;

}

**OUTPUT:**



(b) Reversing:

**C-CODE:**

#include <stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

};

void display(struct Node \*head)

{

       int i=0;

    if(head==NULL)

    {

        printf("Nothing to display!\n");

    }

    else

    {

        struct Node \*ptr=head;

        while(ptr!=NULL)

        {

            printf("Element %d is %d.\n",(i+1),ptr->data);

            ptr=ptr->next;

            i++;

        }

    }

}

struct Node \*insertAtEnd(struct Node \*head,int data)

{

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

    ptr->data=data;

    ptr->next=NULL;

    if(head==NULL)

    {

        head=ptr;

        return head;

    }

    else

    {

    struct Node \*p=head;

    while(p->next!=NULL)

    {

        p=p->next;

    }

    ptr->next=NULL;

    p->next=ptr;

    return head;

    }

}

struct Node \*reverse(struct Node \*head)

{

    //int i;

    struct Node \*dummy=NULL;

    if(head==NULL)

    {

        printf("No linked list!\n");

        return NULL;

    }

    else

    {

        while(head!=NULL)

        {

            struct Node \*Next=head->next;

            head->next=dummy;

            dummy=head;

            head=Next;

        }

        return dummy;

    }

}

int main()

{

    int n,i,data,c;

    struct Node \*head=NULL;

    struct Node \*head2;

    printf("Enter the number of terms you want to enter in the linked list:\n");

    scanf("%d",&n);

    for(i=1;i<=n;i++)

    {

        printf("Enter element %d to be inserted:\n",i);

        scanf("%d",&data);

        head=insertAtEnd(head,data);

    }

    printf("Before reversing:\n");

    display(head);

    printf("\n");

    printf("After reversing:\n");

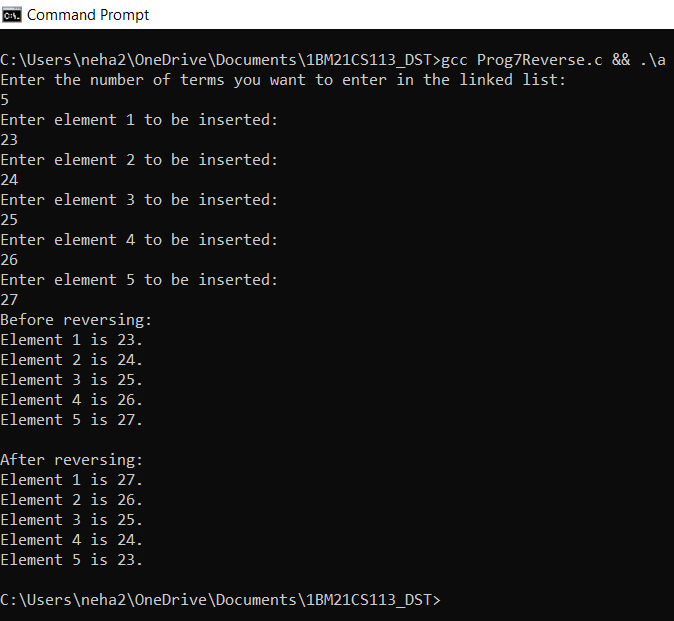
    head2=reverse(head);

    display(head2);

    return 0;

}

**OUTPUT:**



(c) Concatenation:

**C-CODE:**

#include <stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

};

void display(struct Node \*head)

{

       int i=0;

      if(head==NULL)

    {

        printf("Nothing to display!\n");

    }

    else

    {

        struct Node \*ptr=head;

        while(ptr!=NULL)

        {

            printf("Element %d is %d.\n",(i+1),ptr->data);

            ptr=ptr->next;

            i++;

        }

    }

}

struct Node \*insertAtEnd(struct Node \*head,int data)

{

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

    ptr->data=data;

    ptr->next=NULL;

    if(head==NULL)

    {

        head=ptr;

        return head;

    }

    else

    {

    struct Node \*p=head;

    while(p->next!=NULL)

    {

        p=p->next;

    }

    ptr->next=NULL;

    p->next=ptr;

    return head;

    }

}

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

{

    struct Node \*p=head1;

    if(head1==NULL && head2!=NULL)

    return head2;

    else if(head2==NULL && head1!=NULL)

    return head1;

    else if(head1==NULL && head2==NULL)

    {

        printf("Both the linked lists are empty to concatenate!\n");

        return NULL;

    }

    else

    {

        while(p->next!=NULL)

        {

            p=p->next;

        }

        p->next=head2;

        return head1;

    }

}

int main()

{

    int n,i,data;

    struct Node \*head1=NULL;

    struct Node \*head2=NULL;

    printf("Enter the number of terms you want to enter in the first linked list:\n");

    scanf("%d",&n);

    for(i=1;i<=n;i++)

    {

        printf("Enter element %d to be inserted:\n",i);

        scanf("%d",&data);

        head1=insertAtEnd(head1,data);

    }

    printf("Enter the number of terms you want to enter in the second linked list:\n");

    scanf("%d",&n);

    for(i=1;i<=n;i++)

    {

        printf("Enter element %d to be inserted:\n",i);

        scanf("%d",&data);

        head2=insertAtEnd(head2,data);

    }

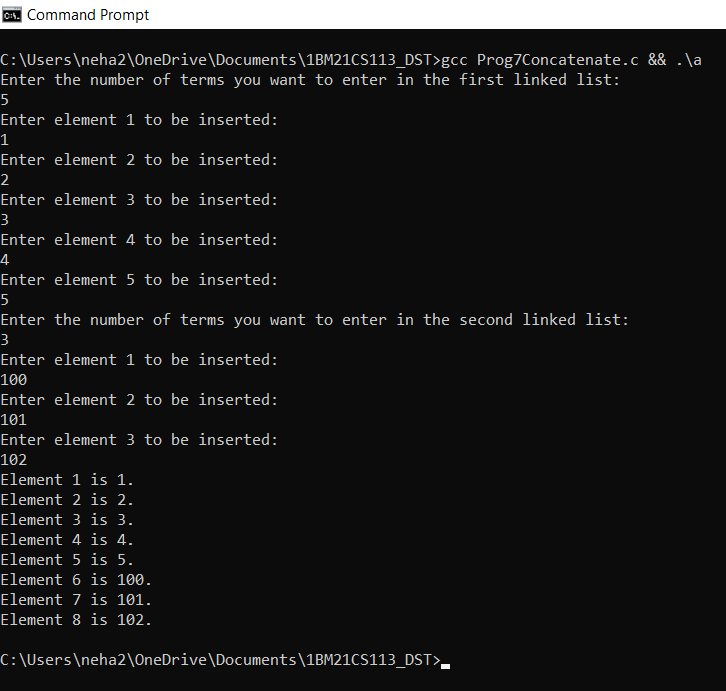
    head1=concatenate(head1,head2);

    display(head1);

    return 0;

}

**OUTPUT:**



**LAB PROGRAM 8:**

Write a program to implement stacks and queues using linked representation.

(a) Stacks using linked lists:

**C-CODE:**

#include <stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

};

void display(struct Node \*top)

{

    int i;

    struct Node \*ptr=top;

    i=0;

    if(top==NULL)

    {

        printf("Nothing to display!\n");

    }

    else

    {

        while(ptr!=NULL)

        {

            printf("Element %d is %d\n",(i+1),(ptr->data));

            ptr=ptr->next;

            i++;

        }

    }

}

struct Node \*push(struct Node \*top,int data)

{

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

    ptr->data=data;

    ptr->next=NULL;

    if(top==NULL)

    {

        top=ptr;

        return top;

    }

    else

    {

    ptr->next=top;

    top=ptr;

    return top;

    }

}

struct Node \*pop(struct Node \*top)

{

    if(top==NULL)

    {

        printf("Nothing to delete, the stack is empty!\n");

        return NULL;

    }

    else

    {

    int del\_item=-1;

    struct Node \*p=top;

    printf("The popped element is %d.\n",top->data);

    top=top->next;

    free(p);

    return top;

    }

}

void main()

{

    struct Node \*top=NULL;

    int c,data;

    while(1)

    {

        printf("Enter 1 to push values into the stack.\nEnter 2 to pop values from the stack.\nEnter 3 to display.\nEnter 4 to exit.\n");

        scanf("%d",&c);

        switch(c)

        {

            case 1:

            printf("Enter the value to be pushed:\n");

            scanf("%d",&data);

            top=push(top,data);

            break;

            case 2:

            top=pop(top);

            break;

            case 3:

            display(top);

            break;

            case 4:

            exit(0);

            default:

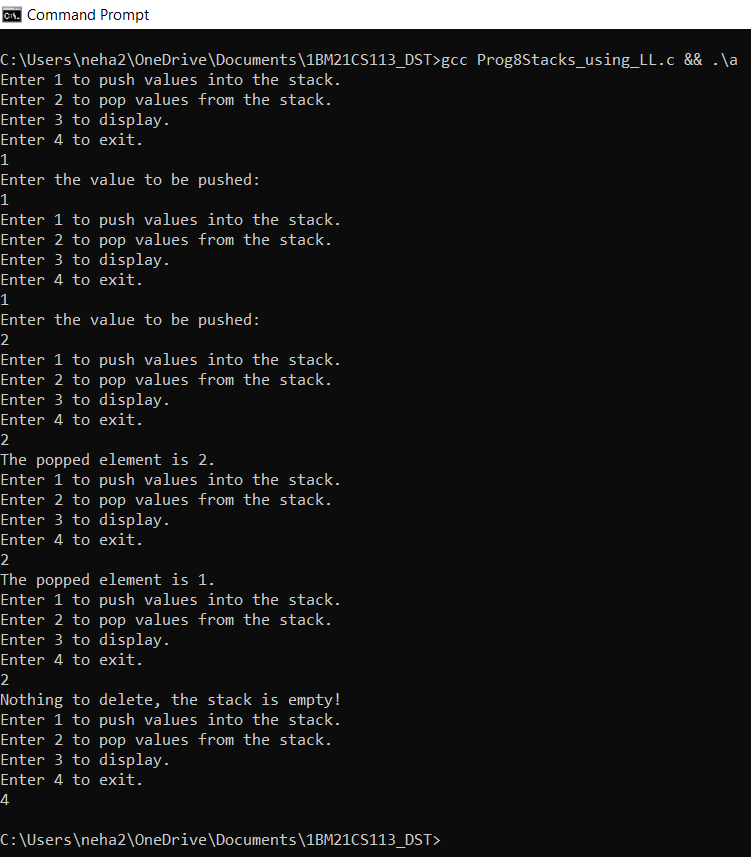
            printf("Invalid choice!\n");

        }

    }

}

**OUTPUT:**



(b) Queues using linked lists:

**C-CODE:**

#include <stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

};

void display(struct Node \*head)

{

    int i;

    struct Node \*ptr=head;

    i=0;

    if(head==NULL)

    {

        printf("Nothing to display!\n");

    }

    else

    {

        while(ptr!=NULL)

        {

            printf("Element %d is %d\n",(i+1),(ptr->data));

            ptr=ptr->next;

            i++;

        }

    }

}

struct Node \*insert\_rear(struct Node \*head,int data)

{

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

    ptr->data=data;

    ptr->next=NULL;

    if(head==NULL)

    {

        head=ptr;

        return head;

    }

    else

    {

    struct Node \*p=head;

    while(p->next!=NULL)

    {

        p=p->next;

    }

    ptr->next=NULL;

    p->next=ptr;

    return head;

    }

}

struct Node \*delete\_front(struct Node \*head)

{

    if(head==NULL)

    {

        printf("Nothing to delete, the queue is empty!\n");

        return NULL;

    }

    else

    {

    struct Node \*p=head;

    printf("The deleted element is %d\n",p->data);

    head=head->next;

    free(p);

    return head;

    }

}

void main()

{

    struct Node \*head=NULL;

    int c,data;

    while(1)

    {

        printf("Enter 1 to insert values into the queue.\nEnter 2 to delete values from the queue.\nEnter 3 to display.\nEnter 4 to exit.\n");

        scanf("%d",&c);

        switch(c)

        {

            case 1:

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

            scanf("%d",&data);

            head=insert\_rear(head,data);

            break;

            case 2:

            head=delete\_front(head);

            break;

            case 3:

            display(head);

            break;

            case 4:

            exit(0);

            default:

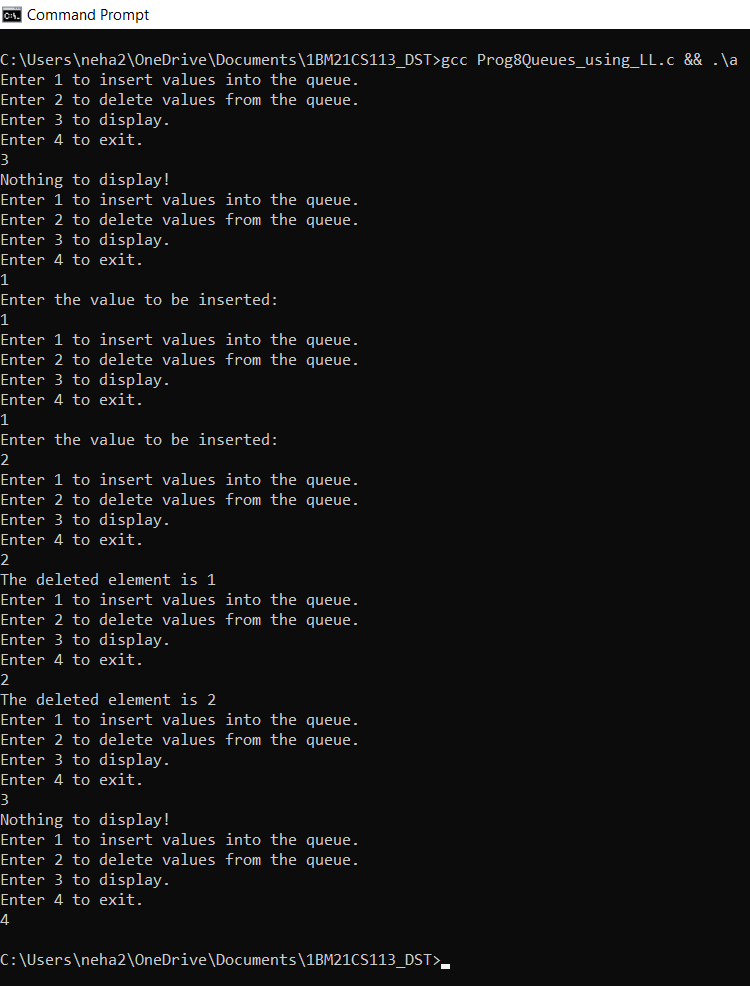
            printf("Invalid choice!\n");

        }

    }

}

**OUTPUT:**



**LAB PROGRAM 9:**

Write a program to implement doubly linked list with primitive operations:

(a) Create a doubly linked list.

(b) Insert a new node to the left of the node.

(c) Delete the node based on a specific value.

(d) Display the contents of the list.

**C-CODE:**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*next;

    struct Node \*prev;

};

struct Node \*getNode()

{

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

    if(ptr==NULL)

    {

        printf("Memory full!\n");

        return NULL;

    }

    else

    return ptr;

}

void forwardTraversal(struct Node \*head)

{

    int i=0;

    if(head==NULL)

    {

        printf("Nothing to display!\n");

    }

    else

    {

        struct Node \*p=head;

        while(p!=NULL)

        {

            printf("Element %d is %d\n",(i+1),p->data);

            p=p->next;

            i++;

        }

    }

}

struct Node \*insert\_at\_end(struct Node \*head,int data, int \*c)

{

    struct Node \*ptr=getNode();

    ptr->prev=NULL;

    ptr->next=NULL;

    ptr->data=data;

    if(head==NULL)

    {

        return ptr;

        \*c=\*c+1;

    }

    else

    {

        struct Node \*p=head;

        while(p->next!=NULL)

        {

            p=p->next;

        }

        ptr->next=NULL;

        ptr->prev=p;

        p->next=ptr;

        \*c=\*c+1;

        return head;

    }

}

struct Node \*insert\_to\_left(struct Node \*head, int data, int index, int \*c)

{

     int i=0;

    struct Node \*ptr=getNode();

    ptr->prev=NULL;

    ptr->next=NULL;

    ptr->data=data;

    if(index>\*c)

    {

        printf("Invalid index!\n");

        return head;

    }

    if(head==NULL)

    {

        return ptr;

        \*c=\*c+1;

    }

    else

    {

        struct Node \*p=head;

        while(i!=index)

        {

            p=p->next;

            i++;

        }

        p->prev->next=ptr;

        ptr->prev=p->prev;

        p->prev=ptr;

        ptr->next=p;

        \*c=\*c+1;

        return head;

    }

}

struct Node \*delete\_at\_value(struct Node \*head, int val)

{

    while(head->data==val)

    {

        struct Node \*p=head;

        head=head->next;

        head->prev=NULL;

        free(p);

    }

    if(head!=NULL)

    {

        struct Node \*p=head;

        struct Node \*q=head->next;

        while(q->next!=NULL)

        {

            if(q->data==val)

            {

                struct Node \*r=q;

                q=q->next;

                p->next=r->next;

                r->next->prev=p;

                free(r);

            }

            else

            {

                p=p->next;

                q=q->next;

            }

        }

        if(q->next==NULL && q->data==val)

        {

            p->next=NULL;

            free(q);

            return head;

        }

        else

        return head;

    }

    return head;

}

int main()

{

    int count=0;

    struct Node \*head=NULL;

    int n,i;

    int c,data,index;

    while(1)

    {

        printf("Enter 1 to create a doubly linked list.\nEnter 2 to insert to the left of the node.\nEnter 3 to delete the node having specified value.\nEnter 4 to display.\n");

        printf("Enter 5 exit.\n");

       scanf("%d",&c);

        switch(c)

        {

            case 1:

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

            scanf("%d",&data);

            head=insert\_at\_end(head, data, &count);

            break;

            case 2:

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

            scanf("%d",&data);

            printf("Enter the index to the left of which the node has to be inserted:\n");

            scanf("%d",&index);

            head=insert\_to\_left(head,data,index, &count);

            break;

            case 3:

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

            scanf("%d",&data);

            head=delete\_at\_value(head,data);

            break;

            case 4:

            forwardTraversal(head);

            break;

            case 5:

            exit(0);

            default:

            printf("Invalid choice!\n");

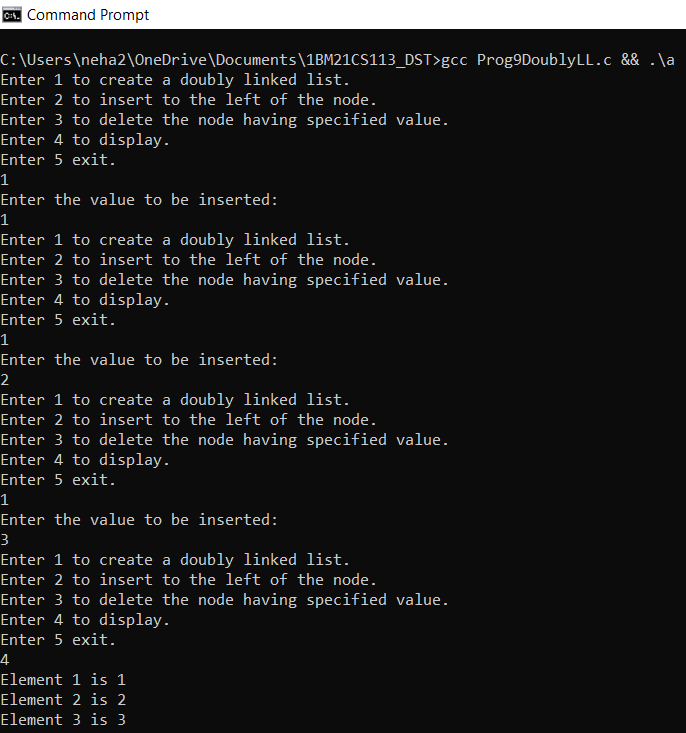
        }

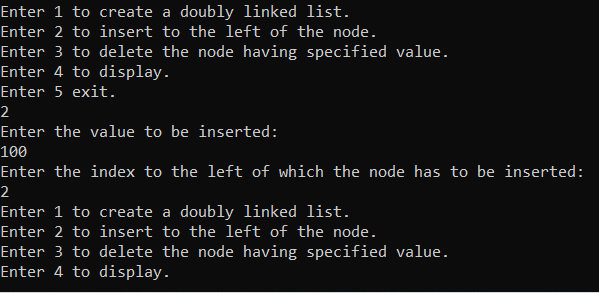
    }

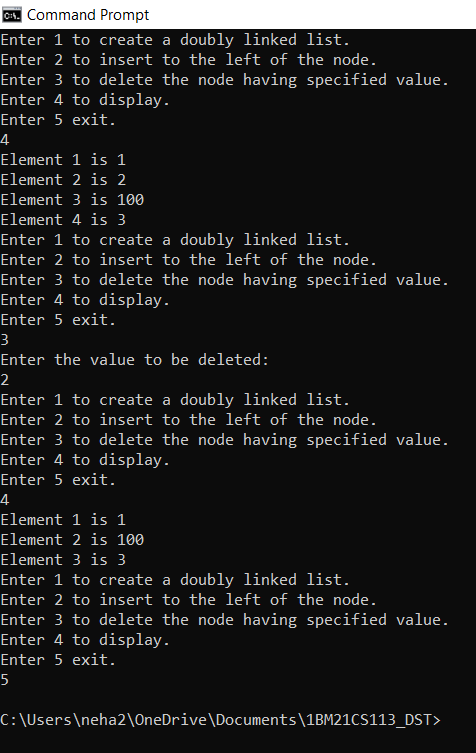
    return 0;

}

**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, pre-order and post-order.

(c) To display the elements in the tree.

**C-CODE:**

#include <stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node \*left;

    struct Node \*right;

};

struct Node \*getNode()

{

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

    if(temp==NULL)

    {

        printf("Memory full!\n");

        return NULL;

    }

    else

    {

        return temp;

    }

}

void inorder(struct Node \*root)

{

    if(root!=NULL)

    {

        inorder(root->left);

        printf("%d ",root->data);

        inorder(root->right);

    }

}

void preorder(struct Node \*root)

{

    if(root!=NULL)

    {

        printf("%d ",root->data);

        preorder(root->left);

        preorder(root->right);

    }

}

void postorder(struct Node \*root)

{

    if(root!=NULL)

    {

        postorder(root->left);

        postorder(root->right);

        printf("%d ",root->data);

    }

}

struct Node \*createBST(struct Node \*root, int data)

{

    struct Node \*new=getNode();

    new->data=data;

    new->right=NULL;

    new->left=NULL;

    if(root==NULL)

    {

        root=new;

        return root;

    }

    else

    {

        if(data<root->data)

        {

            if(root->left==NULL)

            {

               root->left=new;

            }

            else

            {

                root->left=createBST(root->left,data);

            }

        }

        else if(data>root->data)

        {

            if(root->right==NULL)

            {

                root->right=new;

            }

            else

            {

                root->right=createBST(root->right,data);

            }

        }

        return root;

    }

}

void main()

{

    struct Node \*root=NULL;

    int n,data;

    while(1){

    printf("Enter 1 to create a binary search tree.\nEnter 2 to get preorder traversal.\nEnter 3 to get inorder traversal.\n");

    printf("Enter 4 to get postorder traversal.\nEnter 5 to exit!\n");

    scanf("%d",&n);

    switch(n)

    {

        case 1:

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

        scanf("%d",&data);

        root=createBST(root,data);

        break;

        case 2:

        preorder(root);

        printf("\n");

        break;

        case 3:

        inorder(root);

        printf("\n");

        break;

        case 4:

        postorder(root);

        printf("\n");

        break;

        case 5:

        exit(0);

        default:

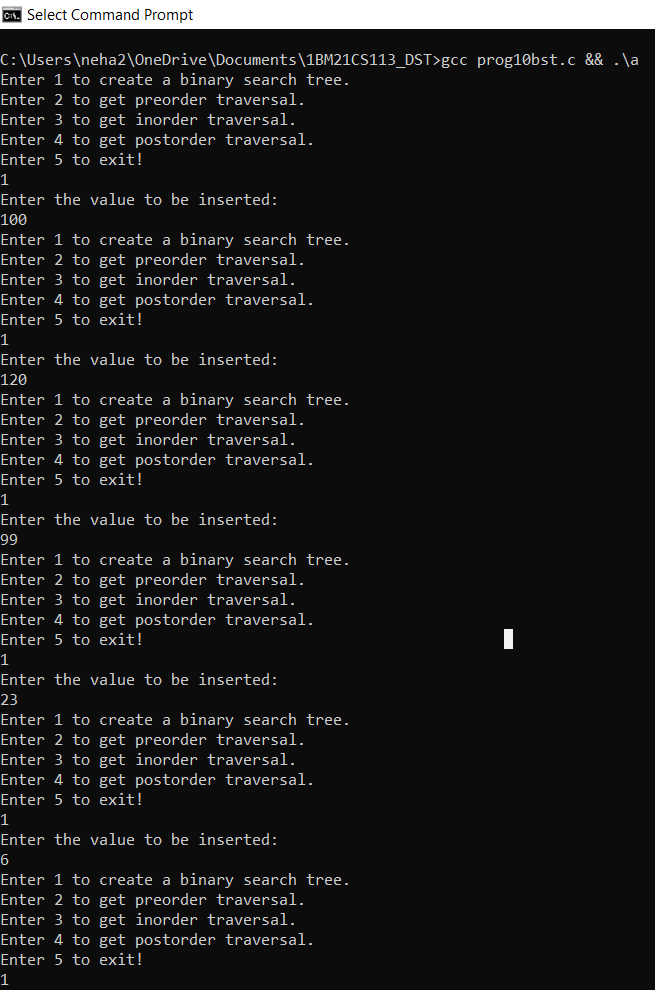
        printf("Invalid choice!\n");

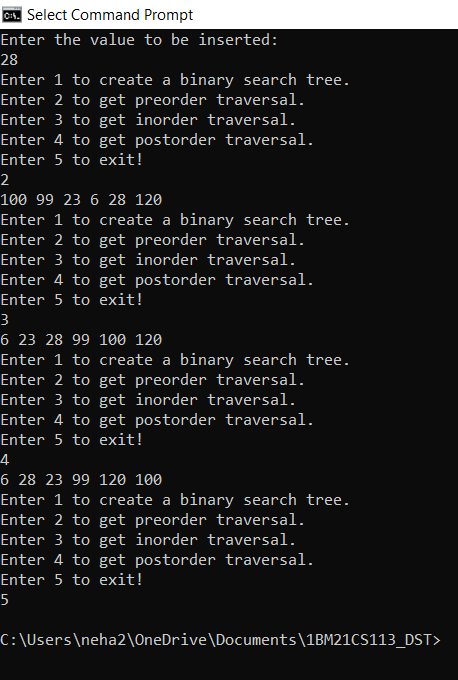
    }

    }

}

**OUTPUT:**

****

****