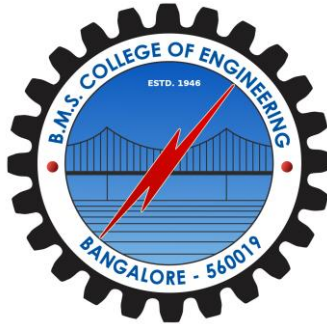


B.M.S. COLLEGE OF ENGINEERING

(AUTONOMOUS COLLEGE UNDER VTU)
BENGALURU-19



DATA STRUCTURE **LAB REPORT**

ACADEMIC YEAR : SEPT-DEC 2020

NAME : NEAL KAUL
USN : 1BM19CS096
SEM : 3

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

```
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t-----");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
                printf("\n\t EXIT POINT ");
                break;
            }
        }
    }
}
```

```

    }
    default:
    {
        printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
    }

}
}
while(choice!=4);
return 0;
}
void push()
{
    if(top>=n-1)
    {
        printf("\n\tStack is over flow");

    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n");
        for(i=top; i>=0; i--)
            printf("\n%d",stack[i]);
        printf("\n Press Next Choice");
    }
    else
    {
        printf("\n The STACK is empty");
    }
}

```

```
}  
  
}
```

OUTPUT:

```
          STACK OPERATIONS USING ARRAY  
-----  
          1.PUSH  
          2.POP  
          3.DISPLAY  
          4.EXIT  
Enter the Choice:1  
Enter a value to be pushed:1  
  
Enter the Choice:1  
Enter a value to be pushed:2  
  
Enter the Choice:1  
Enter a value to be pushed:3  
  
Enter the Choice:3  
  
The elements in STACK  
3  
2  
1  
Press Next Choice  
Enter the Choice:1  
Enter a value to be pushed:4  
  
Enter the Choice:1  
Enter a value to be pushed:5  
  
Enter the Choice:1  
  
          Stack is over flow
```

```

Enter the Choice:2

        The popped elements is 5
Enter the Choice:2

        The popped elements is 4
Enter the Choice:2

        The popped elements is 3
Enter the Choice:3

The elements in STACK

2
1
Press Next Choice
Enter the Choice:4

        EXIT POINT

...Program finished with exit code 0
Press ENTER to exit console.

```

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 the binary operators + (plus), - (minus), * (multiply) and / (divide).

```

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

```

```

#define SIZE 100

```

```

char stack[SIZE];
int top = -1;

```

```

void push(char item)
{
    if(top >= SIZE-1)
    {

```

```

        printf("\nStack Overflow.");
    }
    else
    {
        top = top+1;
        stack[top] = item;
    }
}

```

```

char pop()
{
    char item ;

    if(top <0)
    {
        printf("stack under flow: invalid infix expression");
        getchar();
        exit(1);
    }
    else
    {
        item = stack[top];
        top = top-1;
        return(item);
    }
}

```

```

int is_operator(char symbol)
{
    if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

```

```

int precedence(char symbol)
{
    if(symbol == '^')
    {
        return(3);
    }
    else if(symbol == '*' || symbol == '/')
    {

```

```

        return(2);
    }
    else if(symbol == '+' || symbol == '-')
    {
        return(1);
    }
    else
    {
        return(0);
    }
}

```

```

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{

```

```

    int i, j;
    char item;
    char x;

```

```

    push('(');
    strcat(infix_exp, "");

```

```

    i=0;
    j=0;
    item=infix_exp[i];

```

```

    while(item != '\0')
    {

```

```

        if(item == '(')
        {
            push(item);

```

```

        }
        else if( isdigit(item) || isalpha(item))
        {
            postfix_exp[j] = item;
            j++;

```

```

        }
        else if(is_operator(item) == 1)
        {

```

```

            x=pop();
            while(is_operator(x) == 1 && precedence(x)>= precedence(item))
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
            push(x);

```

```

            push(item);

```

```

        }
    }

```

```

        else if(item == ')')
        {
            x = pop();
            while(x != '(')
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
        }
        else
        {
            printf("\nInvalid infix Expression.\n");
            getchar();
            exit(1);
        }
        i++;

        item = infix_exp[i];
    }
    if(top>0)
    {
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
    }
    if(top>0)
    {
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
    }

    postfix_exp[j] = '\0';

}

int main()
{
    char infix[SIZE], postfix[SIZE];

    printf("\nEnter Infix expression : ");
    gets(infix);

    InfixToPostfix(infix,postfix);
    printf("Postfix Expression: ");

```



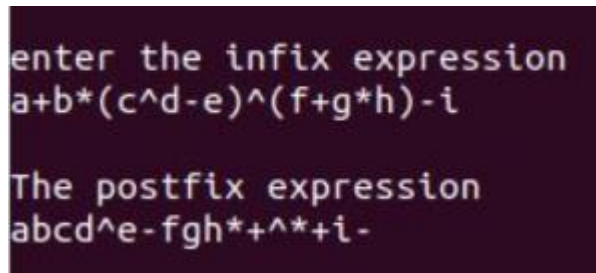
```

        puts(postfix);

        return 0;
}

```

OUTPUT:



```

enter the infix expression
a+b*(c^d-e)^(f+g*h)-i

The postfix expression
abcd^e-fgh*+^*+i-

```

PROGRAM-3

Write a program 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

```

#include <stdio.h>
#include <stdlib.h>
#define MAX 100
void insert ();
void delete ();
void display ();
int queue_arr[MAX];
int rear = -1;
int front = -1;
int main ()
{
    int choice;
    while (1)
    {
        printf ("1.Insert element to queue \n");
        printf ("2.Delete element from queue \n");
        printf ("3.Display elements of queue \n");
        printf ("4.Quit \n");
        printf ("Enter your choice : ");
        scanf ("%d", &choice);
        switch (choice)
        {

```

```

        case 1:
            insert ();
            break;
        case 2:
            delete ();
            break;
        case 3:
            display ();
            break;
        case 4:
            exit (1);
        default:
            printf ("Wrong choice \n");
    }
}
}

```

```

void insert ()
{
    int item;
    if (rear == MAX - 1)
        printf ("Queue Overflow \n");
    else
    {
        if (front == -1)
        {
            front = 0;
        }
        printf ("Inset the element in queue : ");
        scanf ("%d", &item);
        rear = rear + 1;
        queue_arr[rear] = item;
    }
}

```

```

void delete ()
{
    if (front == -1 || front > rear)
    {
        printf ("Queue Underflow \n");
        return;
    }
    else
    {
        printf ("Element deleted from queue is : %d\n", queue_arr[front]);
        front = front + 1;
    }
}

```

```

void display ()

```

```

{
    int i;
    if (front == -1)
        printf ("Queue is empty \n");
    else
    {
        printf ("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf ("%d ", queue_arr[i]);
        printf ("\n");
    }
}

```

OUTPUT:

```

1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 1
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 2
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 3
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 3
Queue is :
1 2 3
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 2
Element deleted from queue is : 1

```

```

Element deleted from queue is : 1
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 3
Queue is :
2 3
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 4

...Program finished with exit code 1
Press ENTER to exit console.

```

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

```
#include<stdio.h>
```

```
#define QUE_SIZE 3
```

```
int item,front=0,rear=-1,q[QUE_SIZE],count=0;
```

```
void insertrear()
```

```
{
```

```
if(count==QUE_SIZE)
```

```
{
```

```
printf("\nqueue overflow\n");
```

```
return;
```

```
}
```

```
rear=(rear+1)%QUE_SIZE;
```

```
q[rear]=item;
```

```
count++;
```

```
}
```

```
int deletefront()
```

```
{
```

```
if(count==0) return -1;
```

```
item=q[front];
```

```
front=(front+1)%QUE_SIZE;
```

```

count--;
return item;
}
void displayQ()
{
int i,f;
if(count==0)
{
printf("\nqueue is empty\n");
return;
}
f=front;
printf("\nContents of queue \n");
for(i=1;i<=count;i++)
{
printf("%d\n",q[f]);
f=(f+1)%QUE_SIZE;
}
}
int main()
{
int choice;

for(;;)
{
printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
printf("enter choice\n");
scanf("%d",&choice);

switch(choice)
{
case 1:printf("enter the item to be inserted\n");
scanf("%d",&item);
insertrear();
break;
case 2:item=deletefront();
if(item== -1)
printf("queue is empty\n");
else
printf("item deleted =%d\n",item);
break;
case 3:displayQ();
break;

```

```
default: return 0;
}
}
```

```
}
```

OUTPUT :

```
enter the item to be inserted
6
```

```
1:insertrear
2:deletefront
3:display
4:exit
```

enter choice

1

enter the item to be inserted

4

queue overflow

```
1:insertrear
2:deletefront
3:display
4:exit
```

enter choice

2

item deleted =5

```
1:insertrear
2:deletefront
3:display
4:exit
```

enter choice

3

Contents of queue

3

4

5

```
1:insertrear
2:deletefront
3:display
4:exit
```

enter choice

2

queue is empty

PROGRAM-5&6

Write a program to implement Singly Linked List with following operations

- a) Create a linked list.**
- b) Insertion of a node at first position, at any position and at end of list.**
- c) Deletion of first element, specified element and last element in the list.**
- d) Display the contents of the linked list**

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

```
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_begin();
void delete_end();
void delete_pos();
```

```
struct node
{
    int data;
    struct node *next;
};
struct node *head=NULL;
```

```
void insert_begin()
{
    int value;
    printf("Enter Element to be Inserted : ");
    scanf("%d",&value);
    struct node* ptr =(struct node*) malloc((sizeof(struct node)));
    ptr -> data = value;
    if(head==NULL){
        head = ptr;
        head->next=NULL;
        return;
    }
    ptr->next = head;
    head = ptr;
}
```

```
void insert_pos()
{
    int value,pos,i;
    printf("Enter Element to be Inserted : ");
    scanf("%d",&value);
    printf("Enter Position : ");
    scanf("%d", &pos);
```

```

struct node* ptr = (struct node*) malloc(sizeof(struct node));
ptr -> data = value;
struct node* temp=head;
if(pos==1)
{
    ptr->next=temp;
    head=ptr;
    return;
}
for(i=1;i<pos;i++)
{
    temp=temp->next;
}
ptr->next=temp->next;
temp->next=ptr;
}

```

```

void insert_end()
{
    int value;
    printf("Enter Element to be Inserted : ");
    scanf("%d",&value);
    struct node* ptr = (struct node*) malloc(sizeof(struct node));
    ptr -> data = value;
    ptr->next=NULL;
    if(head==NULL)
    {
        head=ptr;
        return;
    }
    struct node* temp = head;
    for(;temp->next!=NULL;temp=temp->next);
    temp->next=ptr;
}

```

```

void display() {
    if(head==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
    printf("\n\nLinked List Contains : ");
    for(struct node* temp=head;temp!=NULL;temp = temp->next)
        printf("%d ", temp->data);
    printf("\n\n");
}

```

```

void delete_begin() {
    if(head==NULL){
        printf("\n\nLinked List is Empty\n\n");
    }
}

```



```

        return;
    }
    if(head->next==NULL)
    {
        free(head);
        head=NULL;
        return;
    }
    struct node *temp = head->next;
    free(head);
    head = temp;
}

void delete_end() {
    if(head==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
    if(head->next == NULL)
    {
        free(head);
        head=NULL;
        return;
    }
    struct node* temp = head;
    for(;(temp->next)->next!=NULL;temp=temp->next);
    struct node* temp1 = temp->next;
    temp->next = NULL;
    free(temp1);
}

```

```

void delete_pos(){

    int pos;
    printf("Enter Position : ");
    scanf("%d", &pos);
    if(head==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
    if(pos==0)
    {
        delete_begin();
        return;
    }
    int i = 0;
    struct node* temp = head;
    while(i!=pos-1&&temp!=NULL)
    {
        i++;
    }
}

```

```

        temp = temp->next;
    }
    if(i!=pos-1)
    {
        printf("\n\nERROR\nEnter Correct Index\n\n");
        return;
    }
    if((temp->next)->next==NULL)
    {
        delete_end();
        return;
    }
    struct node* temp1 = temp->next;
    temp->next = (temp->next)->next;
    free(temp1);
}

void main(){
    int choice;
    while(1){

        printf("\n 1.Insert at the beginning ");
        printf("\n 2.Insert at the end ");
        printf("\n 3.Insert at specified position ");
        printf("\n 4.Delete from beginning ");
        printf("\n 5.Delete from the end ");
        printf("\n 6.Delete from specified position ");
        printf("\n 7.Display ");
        printf("\n 8.Exit ");
        printf("\n Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {

            case 1:
                insert_begin();
                break;

            case 2:
                insert_end();
                break;

            case 3:
                insert_pos();
                break;

            case 4:
                delete_begin();
                break;

            case 5:
                delete_end();
                break;

            case 6:

```

```
        delete_pos();
        break;
    case 7:
        display();
        break;
    case 8: exit(1);

    default:
        printf("\n Wrong Choice:n");
        break;
}

}

}
```

OUTPUT:

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Exit

Enter your choice:1

Enter Element to be Inserted : 1

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Exit

Enter your choice:2

Enter Element to be Inserted : 2

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Exit

Enter your choice:2

Enter Element to be Inserted : 3

```
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Delete from beginning
5.Delete from the end
6.Delete from specified position
7.Display
8.Exit
Enter your choice:7
```

```
Linked List Contains : 1 2 3
```

```
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Delete from beginning
5.Delete from the end
6.Delete from specified position
7.Display
8.Exit
Enter your choice:3
```

```
Enter Element to be Inserted : 4
```

```
Enter Position : 2
```

```
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Delete from beginning
5.Delete from the end
6.Delete from specified position
7.Display
8.Exit
Enter your choice:7
```

Linked List Contains : 1 2 4 3

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Exit

Enter your choice:4

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Exit

Enter your choice:5

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Exit

Enter your choice:1

Enter Element to be Inserted : 5

```

3.Insert at specified position
4.Delete from beginning
5.Delete from the end
6.Delete from specified position
7.Display
8.Exit
Enter your choice:6
Enter Position : 2

1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Delete from beginning
5.Delete from the end
6.Delete from specified position
7.Display
8.Exit
Enter your choice:7

Linked List Contains : 5 2

1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Delete from beginning
5.Delete from the end
6.Delete from specified position
7.Display
8.Exit
Enter your choice:8

...Program finished with exit code 1
Press ENTER to exit console.

```

PROGRAM-7

Write a program to implement Single Link List with following operations

- a) **Sort the linked list.**
- b) **Reverse the linked list.**
- c) **Concatenation of two linked lists**

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

```
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_begin();
void delete_end();
void delete_pos();
void swap();
void sort();
void reverse();
void display1();
void insert_begin1();
void insert_end1();
void insert_pos1();
void delete_begin1();
void delete_end1();
void delete_pos1();
void swap1();
void sort1();
void reverse1();
```

```
struct node
{
    int data;
    struct node *next;
};
struct node *head=NULL;
struct node *head1=NULL;
```

```
void insert_begin()
{
    int value;
    printf("Enter Element to be Inserted : ");
    scanf("%d",&value);
    struct node* ptr =(struct node*) malloc(sizeof(struct node));
    ptr -> data = value;
    if(head==NULL){
        head = ptr;
        head->next=NULL;
    }
    return;
```



```

    }
    ptr->next = head;
    head = ptr;
}
void insert_pos()
{
    int value,pos,i;
    printf("Enter Element to be Inserted : ");
    scanf("%d",&value);
    printf("Enter Position : ");
    scanf("%d", &pos);
    struct node* ptr = (struct node*) malloc(sizeof(struct node));
    ptr -> data = value;
    struct node* temp=head;
    if(pos==1)
    {
        ptr->next=temp;
        head=ptr;
        return;
    }
    for(i=1;i<pos;i++)
    {
        temp=temp->next;
    }
    ptr->next=temp->next;
    temp->next=ptr;
}

```

```

void insert_end()
{
    int value;
    printf("Enter Element to be Inserted : ");
    scanf("%d",&value);
    struct node* ptr = (struct node*) malloc(sizeof(struct node));
    ptr -> data = value;
    ptr->next=NULL;
    if(head==NULL)
    {
        head=ptr;
        return;
    }
    struct node* temp = head;
    for(;temp->next!=NULL;temp=temp->next);
    temp->next=ptr;
}

```

```

void display() {
    if(head==NULL){
        printf("\n\nLinked List is Empty\n\n");
    }
}

```

```

        return;
    }
    printf("\n\nLinked List Contains : ");
    for(struct node* temp=head;temp!=NULL;temp = temp->next)
        printf("%d ", temp->data);
    printf("\n\n");
}

```

```

void delete_begin() {
    if(head==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
    if(head->next==NULL)
    {
        free(head);
        head=NULL;
        return;
    }
    struct node *temp = head->next;
    free(head);
    head = temp;
}

```

```

void delete_end() {
    if(head==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
    if(head->next == NULL)
    {
        free(head);
        head=NULL;
        return;
    }
    struct node* temp = head;
    for(;(temp->next)->next!=NULL;temp=temp->next);
    struct node* temp1 = temp->next;
    temp->next = NULL;
    free(temp1);
}

```

```

void delete_pos(){

    int pos;
    printf("Enter Position : ");
    scanf("%d", &pos);
    if(head==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
}

```

```

    }
    if(pos==0)
    {
        delete_begin();
        return;
    }
    int i = 0;
    struct node* temp = head;
    while(i!=pos-1&&temp!=NULL)
    {
        i++;
        temp = temp->next;
    }
    if(i!=pos-1)
    {
        printf("\n\nERROR\nEnter Correct Index\n\n");
        return;
    }
    if((temp->next)->next==NULL)
    {
        delete_end();
        return;
    }
    struct node* temp1 = temp->next;
    temp->next = (temp->next)->next;
    free(temp1);
}

void sort()
{
    int flag, i;
    struct node *ptr1;
    struct node *ptr2 = NULL;

    if (head == NULL)
        return;

    do
    {
        flag = 0;
        ptr1 = head;

        while (ptr1->next != ptr2)
        {
            if (ptr1->data > ptr1->next->data)
            {
                swap(ptr1, ptr1->next);
                flag = 1;
            }
            ptr1 = ptr1->next;
        }
    }

```

```

    }
    ptr2 = ptr1;
}
while (flag);
printf("\nLinked List Sorted");
}
void swap(struct node *a,struct node *b)
{
    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}

```

```

void reverse() {
    if(head == NULL) {
        printf("\n\nLinked List is empty\n\n");
        return;
    }
    if(head -> next == NULL){
        printf("\n\nReversed\n\n");
        return;
    }
    struct node* temp;
    struct node* current = head -> next;
    struct node* previous = head;
    while(current != NULL) {
        temp = current->next;
        current -> next = previous;
        previous = current;
        current = temp;
    }
    head->next=NULL;
    head = previous;
    printf("\nLinked List Reversed");
    return;
}

```

```

void insert_begin1()
{
    int value;
    printf("Enter Element to be Inserted : ");
    scanf("%d",&value);
    struct node* ptr =(struct node*) malloc((sizeof(struct node)));
    ptr -> data = value;
    if(head1==NULL){
        head1 = ptr;
        head1->next=NULL;
        return;
    }
    ptr->next = head1;
}

```

```

    head1 = ptr;
}
void insert_pos1()
{
    int value,pos,i;
    printf("Enter Element to be Inserted : ");
    scanf("%d",&value);
    printf("Enter Position : ");
    scanf("%d", &pos);
    struct node* ptr = (struct node*) malloc(sizeof(struct node));
    ptr -> data = value;
    struct node* temp=head1;
    if(pos==1)
    {
        ptr->next=temp;
        head1=ptr;
        return;
    }
    for(i=1;i<pos;i++)
    {
        temp=temp->next;
    }
    ptr->next=temp->next;
    temp->next=ptr;
}

void insert_end1()
{
    int value;
    printf("Enter Element to be Inserted : ");
    scanf("%d",&value);
    struct node* ptr = (struct node*) malloc(sizeof(struct node));
    ptr -> data = value;
    ptr->next=NULL;
    if(head1==NULL)
    {
        head1=ptr;
        return;
    }
    struct node* temp = head1;
    for(;temp->next!=NULL;temp=temp->next);
    temp->next=ptr;
}

void display1() {

    if(head1==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
}

```

```

printf("\n\nLinked List Contains : ");
for(struct node* temp=head1;temp!=NULL;temp = temp->next)
    printf("%d ", temp->data);
printf("\n\n");
}

```

```

void delete_begin1() {
    if(head1==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
    if(head1->next==NULL)
    {
        free(head1);
        head1=NULL;
        return;
    }
    struct node *temp = head1->next;
    free(head1);
    head1 = temp;
}

```

```

void delete_end1() {
    if(head1==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
    if(head1->next == NULL)
    {
        free(head1);
        head1=NULL;
        return;
    }
    struct node* temp = head1;
    for(;(temp->next)->next!=NULL;temp=temp->next);
    struct node* temp1 = temp->next;
    temp->next = NULL;
    free(temp1);
}

```

```

void delete_pos1(){
    int pos;
    printf("Enter Position : ");
    scanf("%d", &pos);
    if(head1==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
    if(pos==0)

```

```

{
    delete_begin();
    return;
}
int i = 0;
struct node* temp = head1;
while(i!=pos-1&&temp!=NULL)
{
    i++;
    temp = temp->next;
}
if(i!=pos-1)
{
    printf("\n\nERROR\nEnter Correct Index\n\n");
    return;
}
if((temp->next)->next==NULL)
{
    delete_end();
    return;
}
struct node* temp1 = temp->next;
temp->next = (temp->next)->next;
free(temp1);
}

```

```

void sort1()
{
    int flag, i;
    struct node *ptr1;
    struct node *ptr2 = NULL;

    if (head1 == NULL)
        return;

    do
    {
        flag = 0;
        ptr1 = head1;

        while (ptr1->next != ptr2)
        {
            if (ptr1->data > ptr1->next->data)
            {
                swap(ptr1, ptr1->next);
                flag = 1;
            }
            ptr1 = ptr1->next;
        }
        ptr2 = ptr1;
    }
}

```

```

    }
    while (flag);
    printf("\nLinked List Sorted");
}
void swap1(struct node *a,struct node *b)
{
    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}

void reverse1() {
    if(head1 == NULL) {
        printf("\n\nLinked List is empty\n\n");
        return;
    }
    if(head1 -> next == NULL){
        printf("\n\nReversed\n\n");
        return;
    }
    struct node* temp;
    struct node* current = head1 -> next;
    struct node* previous = head1;
    while(current != NULL) {
        temp = current->next;
        current -> next = previous;
        previous = current;
        current = temp;
    }
    head1->next=NULL;
    head1 = previous;
    printf("\nLinked List Reversed");
    return;
}

struct node *concat( struct node *head,struct node *head1)
{
    struct node *ptr;
    if(head==NULL)
    {
        head=head1;
        return head;
    }
    if(head1==NULL)
    {
        return head;
    }
    ptr=head;
    while(ptr->next!=NULL)
    {

```



```

        ptr=ptr->next;
    }
    ptr->next=head1;
    return head;
    printf("Concatenated to Linked List 1");
}

void main(){
    int choice,n;
    while(1){

        printf("\n 1.Insert at the beginning ");
        printf("\n 2.Insert at the end ");
        printf("\n 3.Insert at specified position ");
        printf("\n 4.Delete from beginning ");
        printf("\n 5.Delete from the end ");
        printf("\n 6.Delete from specified position ");
        printf("\n 7.Display ");
        printf("\n 8.Sort the Linked List");
        printf("\n 9.Reverse the Linked List ");
        printf("\n 10.Concatenate the Linked List");
        printf("\n 11.Exit ");
        printf("\n Enter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {

            case 1: printf("Enter 1 for Linked List 1 and 2 for Linked List 2\n");
                    scanf("%d",&n);
                    if(n==1){
                        insert_begin();
                        break;
                    }
                    else{
                        insert_begin1();
                        break;
                    }
                    }
            case 2: printf("Enter 1 for Linked List 1 and 2 for Linked List 2\n");
                    scanf("%d",&n);
                    if(n==1){
                        insert_end();
                        break;
                    }
                    else{
                        insert_end1();
                        break;
                    }
                    }
            case 3: printf("Enter 1 for Linked List 1 and 2 for Linked List 2\n");
                    scanf("%d",&n);

```

```

        if(n==1){
            insert_pos();
            break;
        }
        else{
            insert_pos1();
            break;
        }

case 4:printf("Enter 1 for Linked List 1 and 2 for Linked List 2\n");
scanf("%d",&n);
if(n==1){
    delete_begin();
    break;
}
else{
    delete_begin1();
    break;
}

case 5:printf("Enter 1 for Linked List 1 and 2 for Linked List 2\n");
scanf("%d",&n);
if(n==1){
    delete_end();
    break;
}
else{
    delete_end1();
    break;
}

case 6:printf("Enter 1 for Linked List 1 and 2 for Linked List 2\n");
scanf("%d",&n);
if(n==1){
    delete_pos();
    break;
}
else{
    delete_pos1();
    break;
}

case 7:printf("Enter 1 for Linked List 1 and 2 for Linked List 2\n");
scanf("%d",&n);
if(n==1){
    display();
    break;
}
else{
    display1();

```

```

        break;
    }

case 8:printf("Enter 1 for Linked List 1 and 2 for Linked List 2\n");
    scanf("%d",&n);
    if(n==1){
        sort();
        break;
    }
    else{
        sort1();
        break;
    }

case 9:printf("Enter 1 for Linked List 1 and 2 for Linked List 2\n");
    scanf("%d",&n);
    if(n==1){
        reverse();
        break;
    }
    else{
        reverse1();
        break;
    }
case 10: concat(head,head1);
    break;

case 11: exit(1);

default:
    printf("n Wrong Choice:n");
    break;
}

}

}

```

OUTPUT:

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:1

Enter 1 for Linked List 1 and 2 for Linked List 2

1

Enter Element to be Inserted : 10

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:2

Enter 1 for Linked List 1 and 2 for Linked List 2

1

Enter Element to be Inserted : 1

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:1

Enter 1 for Linked List 1 and 2 for Linked List 2

1

Enter Element to be Inserted : 3

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:1

Enter 1 for Linked List 1 and 2 for Linked List 2

1

Enter Element to be Inserted : 5

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:7

Enter 1 for Linked List 1 and 2 for Linked List 2

1

Linked List Contains : 5 3 10 1

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:9

Enter 1 for Linked List 1 and 2 for Linked List 2

1

Linked List Reversed

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:7

Enter 1 for Linked List 1 and 2 for Linked List 2

1

Linked List Contains : 1 10 3 5

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:8

Enter 1 for Linked List 1 and 2 for Linked List 2

1

Linked List Sorted

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:7

Enter 1 for Linked List 1 and 2 for Linked List 2

1

Linked List Contains : 1 3 5 10

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:1

Enter 1 for Linked List 1 and 2 for Linked List 2

2

Enter Element to be Inserted : 13

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:1

Enter 1 for Linked List 1 and 2 for Linked List 2

2

Enter Element to be Inserted : 14

- 1.Insert at the beginning
- 2.Insert at the end
- 3.Insert at specified position
- 4.Delete from beginning
- 5.Delete from the end
- 6.Delete from specified position
- 7.Display
- 8.Sort the Linked List
- 9.Reverse the Linked List
- 10.Concatenate the Linked List
- 11.Exit

Enter your choice:2

Enter 1 for Linked List 1 and 2 for Linked List 2

2

Enter Element to be Inserted : 17

```

1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Delete from beginning
5.Delete from the end
6.Delete from specified position
7.Display
8.Sort the Linked List
9.Reverse the Linked List
10.Concatenate the Linked List
11.Exit
Enter your choice:7
Enter 1 for Linked List 1 and 2 for Linked List 2
2

Linked List Contains : 14 13 17

1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Delete from beginning
5.Delete from the end
6.Delete from specified position
7.Display
8.Sort the Linked List
9.Reverse the Linked List
10.Concatenate the Linked List
11.Exit
Enter your choice:10

1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Delete from beginning
5.Delete from the end
6.Delete from specified position
7.Display
8.Sort the Linked List
9.Reverse the Linked List
10.Concatenate the Linked List
11.Exit
Enter your choice:7
Enter 1 for Linked List 1 and 2 for Linked List 2
1

Linked List Contains : 1 3 5 10 14 13 17

```

PROGRAM-8

Write a program to implement Stack & Queue using Linked Representation.

Queue using linked list

```
#include<stdlib.h>
```

```
#include <stdio.h>
```

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node *next;
```

```
};
```

```
struct node *head=NULL;
```

```
void display();
```

```
void enqueue();
```

```
void dequeue();
```

```
void enqueue()
```

```
{
```

```
    int value;
```

```
    printf("Enter Element to be Inserted : ");
```

```
    scanf("%d",&value);
```

```
    struct node* ptr = (struct node*) malloc(sizeof(struct node));
```

```
    ptr -> data = value;
```

```
    ptr->next=NULL;
```

```
    if(head==NULL)
```

```
    {
```

```
        head=ptr;
```

```
        return;
```

```
    }
```

```
    struct node* temp = head;
```

```
    for(;temp->next!=NULL;temp=temp->next);
```

```
    temp->next=ptr;
```

```
    printf("Element Inserted To Queue\n");
```

```
}
```

```
void dequeue() {
```

```
    if(head==NULL){
```

```
        printf("\n\nQueue is Empty\n\n");
```

```
        return;
```

```
    }
```

```
    if(head->next==NULL)
```

```
    {
```

```
        free(head);
```

```
        head=NULL;
```

```
        return;
```

```
    }
```

```
    struct node *temp = head->next;
```

```
    free(head);
```

```
    head = temp;
```

```
    printf("Element Deleted from Queue\n");
```

```

}

void display() {

    if(head==NULL){
        printf("\n\nQueue is Empty\n\n");
        return;
    }
    printf("\n\nQueue Contains : ");
    for(struct node* temp=head;temp!=NULL;temp = temp->next)
        printf("%d ", temp->data);
    printf("\n\n");
}

void main ()
{
    int choice;
    while (1)
    {
        printf ("1.Insert element to queue \n");
        printf ("2.Delete element from queue \n");
        printf ("3.Display elements of queue \n");
        printf ("4.Quit \n");
        printf ("Enter your choice : ");
        scanf ("%d", &choice);
        switch (choice)
        {
            case 1:
                enqueue ();
                break;
            case 2:
                dequeue ();
                break;
            case 3:
                display ();
                break;
            case 4:
                exit (1);
            default:
                printf ("Wrong choice \n");
        }
    }
}

```

OUTPUT:

```
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 1
Enter Element to be Inserted : 1
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 1
Enter Element to be Inserted : 2
Element Inserted To Queue
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 1
Enter Element to be Inserted : 3
Element Inserted To Queue
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 1
Enter Element to be Inserted : 4
Element Inserted To Queue
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 3

Queue Contains : 1  2  3  4

1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 2
Element Deleted from Queue
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 2
Element Deleted from Queue
```

```
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice : 3
```

```
Queue Contains : 3 4
```

```
1.Insert element to queue
2.Delete element from queue
3.Display elements of queue
4.Quit
Enter your choice :
```

Stack using linked list

```
#include<stdlib.h>
#include <stdio.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head=NULL;

void display();
void push();
void pop();

void push()
{
    int value;
    printf("Enter Element to be Inserted : ");
    scanf("%d",&value);
    struct node* ptr =(struct node*) malloc(sizeof(struct node));
    ptr -> data = value;
    if(head==NULL){
        head = ptr;
        head->next=NULL;
        return;
    }
    ptr->next = head;
    head = ptr;
    printf("Element Pushed To the Stack\n");
}

void display() {
```

```

if(head==NULL){
    printf("\n\nStack is Empty\n\n");
    return;
}
printf("\n\nStack Contains : ");
for(struct node* temp=head;temp!=NULL;temp = temp->next)
    printf("%d ", temp->data);
printf("\n\n");
}

```

```

void pop() {
    if(head==NULL){
        printf("\n\nStack is Empty\n\n");
        return;
    }
    if(head->next==NULL)
    {
        free(head);
        head=NULL;
        return;
    }
    struct node *temp = head->next;
    free(head);
    head = temp;
    printf("Element Popped from the Stack\n");
}

```

```

void main()
{
    int choice;
    do
    {
        printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
            }
        }
    }
}

```

```
        break;
    }
    case 4:
    {
        printf("\n\t EXITING ");
        break;
    }
    default:
    {
        printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
    }

}
}
while(choice!=4);

}
```

OUTPUT:

Enter Element to be Inserted : 1

- 1.PUSH
- 2.POP
- 3.DISPLAY
- 4.EXIT

Enter the Choice:1

Enter Element to be Inserted : 2

Element Pushed To the Stack

- 1.PUSH
- 2.POP
- 3.DISPLAY
- 4.EXIT

Enter the Choice:1

Enter Element to be Inserted : 3

Element Pushed To the Stack

- 1.PUSH
- 2.POP
- 3.DISPLAY
- 4.EXIT

Enter the Choice:3

Stack Contains : 3 2 1

- 1.PUSH
- 2.POP
- 3.DISPLAY
- 4.EXIT

Enter the Choice:2

Element Popped from the Stack

- 1.PUSH
- 2.POP
- 3.DISPLAY
- 4.EXIT

Enter the Choice:2

Element Popped from the Stack

- 1.PUSH
- 2.POP
- 3.DISPLAY
- 4.EXIT

Enter the Choice:3

Stack Contains : 1

PROGRAM-9

Write a program to implement Doubly Link 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**

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

```
void createList();
void addLeft();
void delete();
void display();
```

```
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};
```

```
struct Node* head=NULL;
```

```
void createList(struct Node** head_ref, int new_value)
{
    struct Node *temp=(struct Node*) malloc(sizeof (struct Node));
    temp->data = new_value;
    temp->prev = NULL;
    temp->next = NULL;
    if (*head_ref==NULL)
    {
        (*head_ref)= temp;
    }

    else
    {
        (*head_ref)->prev = temp;
        temp->next=(*head_ref);
        (*head_ref)=temp;
    }
}
```

```
void display(struct Node* node)
{
    struct Node* temp;
    printf("Linked List contains : \n");
    while (node != NULL)
    {
        printf(" %d ", node->data);
        temp = node;
```



```

        node = node->next;
    }
    printf("\n");
}

void addLeft(int new_value,int key)
{
    struct Node* temp = head;
    while(temp!=NULL) {
        if(temp->data == key)
        {
            break;
        }
        temp = temp->next;
    }
    if(temp==NULL)
    {
        printf("\nNo Match\n");
        return;
    }
    if(temp==head)
    {
        createList(&head,new_value);
        return;
    }
    struct Node* ptr = (struct Node*) malloc(sizeof(struct Node));
    ptr->data = new_value;
    ptr->prev = temp->prev;
    ptr->next = temp;
    (temp->prev)->next = ptr;
    temp->prev = ptr;
}

void delete(int key)
{
    if(head == NULL)
    {
        printf("\nList is Empty\n");
        return;
    }
    struct Node* temp = head;
    while(temp != NULL)
    {
        if(temp->data == key)
        {
            break;
        }
        temp = temp->next;
    }
}

```

```

if(temp==head)
{
    if(head->next==NULL)
    {
        free(head);
        head=NULL;
        return;
    }
    head = head->next;
    free(head->prev);
    head->prev = NULL;
    return;
}
if(temp==NULL)
{
    printf("\nNo Match\n");
    return;
}
if(temp->next==NULL)
{
    temp->prev->next = NULL;
    free(temp);
    return;
}
temp->next->prev = temp->prev;
temp->prev->next = temp->next;
free(temp);

}

void main()
{
    int choice, value, key;
    while(1)
    {
        printf("1.Add Node at beginning\n");
        printf("2.Add at left of a node\n");
        printf("3.Delete a node\n");
        printf("4.Display Linked List\n");
        printf("Enter -1 to quit\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        if(choice==-1)
            break;
        switch(choice)
        {
            case 1:
                printf("\nEnter value to add : ");
                scanf("%d", &value);
                createList(&head,value);

```

```

        break;
    case 2:
        printf("\nEnter value to insert : ");
        scanf("%d", &value);
        printf("\nEnter value of key node : ");
        scanf("%d", &key);
        addLeft(value,key);
        break;
    case 3:
        printf("\nEnter value of node to be deleted : ");
        scanf("%d", &key);
        delete(key);
        break;
    case 4:
        display(head);
        break;
    default:
        printf("\n\nWrong Input\n\n");
    }
}
printf("\n\n-----DONE-----\n\n");
}

```

OUTPUT:

```
1.Add Node at beginning
2.Add at left of a node
3.Delete a node
4.Display Linked List
Enter -1 to quit
Enter your choice : 1

Enter value to add : 1
1.Add Node at beginning
2.Add at left of a node
3.Delete a node
4.Display Linked List
Enter -1 to quit
Enter your choice : 1

Enter value to add : 2
1.Add Node at beginning
2.Add at left of a node
3.Delete a node
4.Display Linked List
Enter -1 to quit
Enter your choice : 1

Enter value to add : 3
1.Add Node at beginning
2.Add at left of a node
3.Delete a node
4.Display Linked List
Enter -1 to quit
Enter your choice : 1

Enter value to add : 5
1.Add Node at beginning
2.Add at left of a node
3.Delete a node
4.Display Linked List
Enter -1 to quit
Enter your choice : 4
Linked List contains :
5 3 2 1
1.Add Node at beginning
2.Add at left of a node
3.Delete a node
4.Display Linked List
Enter -1 to quit
Enter your choice : 2

Enter value to insert : 4

Enter value of key node : 3
```

```

1.Add Node at beginning
2.Add at left of a node
3.Delete a node
4.Display Linked List
Enter -1 to quit
Enter your choice : 4
Linked List contains :
 5  4  3  2  1
1.Add Node at beginning
2.Add at left of a node
3.Delete a node
4.Display Linked List
Enter -1 to quit
Enter your choice : 3

Enter value of node to be deleted : 3
1.Add Node at beginning
2.Add at left of a node
3.Delete a node
4.Display Linked List
Enter -1 to quit
Enter your choice : 4
Linked List contains :
 5  4  2  1
1.Add Node at beginning
2.Add at left of a node
3.Delete a node
4.Display Linked List
Enter -1 to quit
Enter your choice :

```

PROGRAM-10

Write a program to

- **To construct a binary search tree.**
- **To transverse the tree using all the methods ie, in-order, pre-order, post-order**
- **Display contents of the list.**

```

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

```

```

struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;

```

```
void insert();
void inorder(struct btnode *t);
void create();
void search(struct btnode *t);
void preorder(struct btnode *t);
void postorder(struct btnode *t);
```

```
void main()
{
    int ch;

    printf("\nOPERATIONS ---");
    printf("\n1 : Insert an element into tree\n");
    printf("\n2 : Inorder Traversal\n");
    printf("\n3 : Preorder Traversal\n");
    printf("\n4 : Postorder Traversal\n");
    printf("\n5 : Exit\n");
    while(1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                insert();
                break;
            case 2:
                inorder(root);
                break;
            case 3:
                preorder(root);
                break;
            case 4:
                postorder(root);
                break;
            case 5:
                exit(0);
            default :
                printf("Wrong choice, Please enter correct choice ");
                break;
        }
    }
}
```

```
void insert()
{
    create();
    if (root == NULL)
```

```

        root = temp;
    else
        search(root);
}

```

```

void create()
{
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}

```

```

void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
    else if ((temp->value < t->value) && (t->l != NULL))
        search(t->l);
    else if ((temp->value < t->value) && (t->l == NULL))
        t->l = temp;
}

```

```

void inorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    if (t->l != NULL)
        inorder(t->l);
    printf("%d -> ", t->value);
    if (t->r != NULL)
        inorder(t->r);
}

```

```

void preorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
    }
}

```

```

        return;
    }
    printf("%d -> ", t->value);
    if (t->l != NULL)
        preorder(t->l);
    if (t->r != NULL)
        preorder(t->r);
}

```

```

void postorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display ");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d -> ", t->value);
}

```

OUTPUT:


```
OPERATIONS ---
1 : Insert an element into tree
2 : Inorder Traversal
3 : Preorder Traversal
4 : Postorder Traversal
5 : Exit

Enter your choice : 1
Enter data of node to be inserted : 6

Enter your choice : 1
Enter data of node to be inserted : 5

Enter your choice : 1
Enter data of node to be inserted : 7

Enter your choice : 1
Enter data of node to be inserted : 9

Enter your choice : 1
Enter data of node to be inserted : 8

Enter your choice : 1
Enter data of node to be inserted : 10

Enter your choice : 1
Enter data of node to be inserted : 4

Enter your choice : 1
Enter data of node to be inserted : 3

Enter your choice : 1
Enter data of node to be inserted : 1

Enter your choice : 1
Enter data of node to be inserted : 2

Enter your choice : 1
Enter data of node to be inserted : 0

Enter your choice : 2
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 ->
Enter your choice : 3
6 -> 5 -> 4 -> 3 -> 1 -> 0 -> 2 -> 7 -> 9 -> 8 -> 10 ->
Enter your choice : 4
0 -> 2 -> 1 -> 3 -> 4 -> 5 -> 8 -> 10 -> 9 -> 7 -> 6 ->
Enter your choice :
```