DATA STRUCTURES LAB-PROGRAMS

YASHWANTH KIRAN S

1BM19CS187

'3D'- Batch-2

Sept-Dec 2020

LAB -1

Write a program to simulate the working of stack using an array with the following :

a) Push b) Pop c) Display

```
#include <stdio.h>
#define SIZE 5
int top=-1;
int stack[SIZE];
void push(int ele)
 {
   if(isFull())
   {
      printf("The stack is full\n");
   }
   else
```

```
  {
     top++;
     stack[top]=ele;
  }
}
int pop()
{
   if(isEmpty())
   {
      return 0;
   }
   else
   {
      return stack[top--];
   }
}
int isEmpty()
{
   if(top==-1)
      return 1;
   else
      return 0;
}
```

```
int isFull()
{
    if(top==SIZE-1)
        return 1;
    else
        return 0;
}
void display()
{
    if(isEmpty())
        printf("The stack is empty\n");
    else
    {
        printf("The elements are\n");
        for(int i=0;i<=top;i++)
        {
            printf("%d\n",stack[i]);
        }
    }
}

int main()
{
```

```
int c,d,p;
while(1)
{
printf("Enter command\n1-push\n2-pop\n3-Display\n");
scanf("%d",&c);
switch(c)
{
  case 1:printf("Enter an element\n");
         scanf("%d",&d);
         push(d);
         break;
  case 2:p=pop();
         if(p==0)
          printf("Stack is empty\n");
          else
          printf("Element removed succesfully\n");
          break;
  case 3:display();
         break;
  case 4:exit(0);
  default: printf("Invalid input\n");
}
}
```

```
return 0;

}
```

OUTPUT:

```
Enter command
1-push
2-pop
3-Display
1
Enter an element
45
Enter command
1-push
2-pop
3-Display
1
Enter an element
18
Enter command
1-push
2-pop
3-Display
3
The elements are
45
18
Enter command
1-push
2-pop
3-Display
2
Element removed succesfully
```

```
Enter command
1-push
2-pop
3-Display
3
The elements are
45
Enter command
1-push
2-pop
3-Display
2
Element removed succesfully
Enter command
1-push
2-pop
3-Display
3
The stack is empty
Enter command
1-push
2-pop
3-Display
```

LAB-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

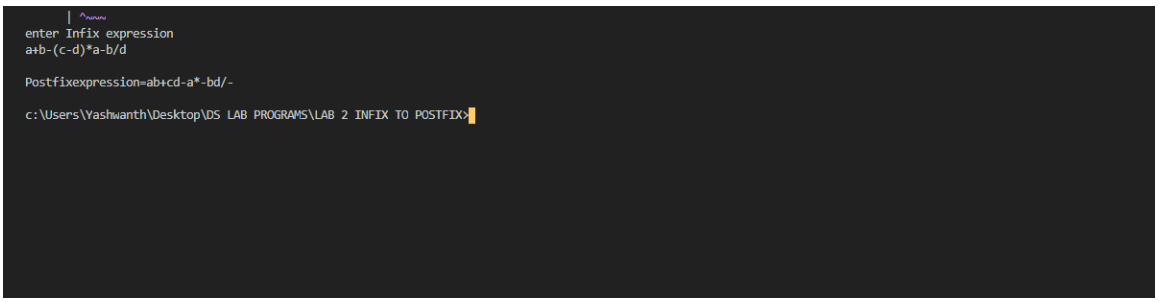
+ (plus), - (minus), * (multiply) and / (divide)

```
#include<stdio.h>
#include<ctype.h>
#define SIZE 50
char stack[SIZE];
int top=-1;
push(char elem)
{
stack[++top]=elem;
}
char pop()
{
 return(stack[top--]);
 }
 int pr(char symbol)
 {
 if(symbol== '^')
 {
 return(3);
```

```
}
else if(symbol== '*' || symbol== '/' )
{
return(2);
}
else if(symbol== '+' || symbol== '-')
{
return(1);
}
else {
return(0);
}
}
void main()
{
char infix[50],postfix[50],ch,elem;
int i=0,k=0;
printf("enter Infix expression");
scanf("%s",infix);
push('#');
while((ch=infix[i++])!='\0' )
{
if(ch == '(') push(ch);
else
if(isalnum(ch)) postfix[k++]=ch;
```

```
else
if(ch==')')
{ while(stack[top] !='(')
postfix[k++]=pop();
elem=pop();
}
else
{
while(pr(stack[top]) >=pr(ch))
 postfix[k++]=pop();
 push(ch);
 }
 }
 while(stack[top]!='#')
  postfix[k++]=pop();
  postfix[k]='\0';
  printf("\nPostfixexpression=%s\n",postfix);
  }
```

OUTPUT:



LAB-3

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

operations

a) Insert b) Delete c) Display

```
#include <stdio.h>

#define MAX 10

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
   int choice;
   while (1)
   {
      printf("1.Insert element to queue \n");
      printf("2.Delete element from queue \n");
      printf("3.Display all 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 add_item;
    if (rear == MAX - 1)
```

```
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
}

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

OUTPUT:

LAB-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

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 6
int a[SIZE],t;
int front=-1;
int rear=-1;

int IsEmpty()
{
 if(rear==-1 && front==-1)
    return 1;
  else
    return 0;
}


int IsFull()
{
 if(front==(rear+1)%SIZE)
 return 1;
```

```
 else
return 0;
}


void Enqueue(int x)
{
  if(IsFull())
   printf("The queue is full\n");
  else if(IsEmpty())
  {
    front=0;
    rear=0;
    a[rear]=x;
   }
   else
   {
       rear=(rear+1)%SIZE;
       a[rear]=x;
    }
}

int Dequeue()
{    int x;
    if(IsEmpty())
```

```
    printf("The queue is empty.\n");
    else if(front==rear)
    {
        x=a[front];
        front=-1;
        rear=-1;
         printf("The element was removed\n");
    }
    else
    {
        x=a[front];
        front=(front+1)%SIZE;
         printf("The element was removed\n");
    }
    return x;
}


void display()

{
    if (front == -1)
    {
        printf("\nQueue is Empty");
        return;
```

```
    }
    printf("\nElements in Circular Queue are:\n");
    if (rear >= front)
    {
        for (int i = front; i <= rear; i++)
            printf("%d\n",a[i]);
    }
    else
    {
        for (int i = front; i < SIZE; i++)
            printf("%d\n", a[i]);

        for (int i = 0; i <= rear; i++)
            printf("%d\n", a[i]);
    }
}



int main()
{
 int n,a;
 while(1)
  {
 printf("Enter the operation.\n1-Insert\n2-Delete\n3-Display\n4-Exit\n");
```

```
scanf("%d",&n);
switch(n)
{
  case 1: printf("Enter the element\n");
            scanf("%d",&a);
            Enqueue(a);
             break;

 case 2 :   Dequeue();

            break;
 case 3:  display();
          break;
  case 4: exit(0);
   default :   printf("There is no such operation\n");
 }
 }
 return 0;
}
```

OUTPUT:

```
C:\Users\Yashwanth\Desktop>cd "c:\Users\Yashwanth\Desktop\DS LAB PROGRAMS\LAB 4 CIRCULAR QUEUE\" && gcc cq.c -o cq && "c:\Users\Yashwanth\Desktop\DS LAB PROGRAMS\LAB 4 CIRCULAR QUEUE
\"cq
Enter the operation.
1-Insert
2-Delete
3-Display
4-Exit
1
Enter the element
45
Enter the operation.
1-Insert
2-Delete
3-Display
4-Exit
1
Enter the element
18
Enter the operation.
1-Insert
2-Delete
3-Display
4-Exit
1
Enter the element
7
Enter the operation.
1-Insert
2-Delete
3-Display
4-Exit
3
```

```
Elements in Circular Queue are:
45
18
7
Enter the operation.
1-Insert
2-Delete
3-Display
4-Exit
2
The element was removed
Enter the operation.
1-Insert
2-Delete
3-Display
4-Exit
3

Elements in Circular Queue are:
18
7
Enter the operation.
1-Insert
2-Delete
3-Display
4-Exit
2
The element was removed
Enter the operation.
1-Insert
2-Delete
3-Display
4-Exit
3
```

```
Elements in Circular Queue are:
7
Enter the operation.
1-Insert
2-Delete
3-Display
4-Exit
2
The element was removed
Enter the operation.
1-Insert
2-Delete
3-Display
4-Exit
3

Queue is EmptyEnter the operation.
1-Insert
2-Delete
3-Display
4-Exit
```

LAB-5 & LAB-6

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

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.

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

typedef struct node{
  int data;
  struct node *link;
}node;

node *root=NULL;

void add_at_end()
{
   node *temp;
   temp=(node *)malloc(sizeof(node));

   printf("Enter the node element\n");
```

```
   scanf("%d",&temp->data);
    temp->link=NULL;
   if(root==NULL)
  {
  root=temp;
  }
  else
  {
    node *p=root;
    while(p->link!=NULL)
   {
    p=p->link;
   }
   p->link=temp;
  }
}

void add_at_begin()
{
 node *temp;
temp=(node *)malloc(sizeof(node));
 printf("Enter node element\n");
 scanf("%d",&temp->data);
 temp->link=NULL;
```

```
 if(root==NULL)
 {
    root=temp;
 }
 else
 {
  temp->link=root;
  root=temp;
 }
}

int length()
{
  node *p;
  p=root;
  int i=0;

  while(p!=NULL)
  {
   i++;
   p=p->link;
  }
return i;
}
```

```
void add_after(){

node *p,*temp;
int loc,i=1;
printf("Enter the location");
scanf("%d",&loc);

if(loc>length())
{
  printf("Invalid location. The list has %d nodes",length());
}
else
{

p=root;
while(i<loc)
{
 p=p->link;
 i++;
}

temp=(node *)malloc(sizeof(node));
printf("Enter the node element\n");
scanf("%d",&temp->data);
temp->link=NULL;
```

```
temp->link=p->link;
p->link=temp;
}
}

void delete()
{
int loc;
node *temp;
printf("Enter the locatin of node to be deleted\n");
scanf("%d",&loc);

if (loc>length())
{
  printf("There is no such node\n");
}
else if (loc==1)
{
 temp=root;
root=temp->link;
temp->link=NULL;
free(temp);
}
else
```

```
{
node *p=root,*q;
int i=1;
while(i<loc-1)
{
  p=p->link;
  i++;
}
q=p->link;
p->link=q->link;
q->link=NULL;
free(q);
}
}


void display()
{
 node *temp=root;
 if(temp==NULL)
 {
   printf("No nodes in the list\n");
 }
 else
 {
```

```
   while(temp!=NULL)
   {
     printf("%d\n",temp->data);
     temp=temp->link;
   }
 }
}



int main()
{


 int op,len;
 while(1)
{ printf("Enter the operation\n1.Add in begin\n2.Add at end\n");
  printf("3.Add after a node\n4.Delete node\n5.Display\n6.Length of
list\n7.Exit\n");
  scanf("%d",&op);
  switch (op)
  {
  case 1:add_at_begin();
    break;
  case 2: add_at_end();
    break;
```

```
  case 3: add_after();
    break;
  case 4: delete();
    break;
  case 5: display();
    break;
  case 6: len=length();
          printf("The length is %d\n",len);
    break;
  case 7: exit(0);
     break;
  default: printf("No such operation\n");
  }
 }
return 0;
}
```

OUTPUT:

```
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
6.Length of list
7.Exit
1
Enter node element
123
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
6.Length of list
7.Exit
2
Enter the node element
789
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
6.Length of list
7.Exit
3
Enter the location1
Enter the node element
456
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
6.Length of list
7.Exit
```

```
5
123
456
789
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
6.Length of list
7.Exit
4
Enter the locatin of node to be deleted
1
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
6.Length of list
7.Exit
5
456
789
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
6.Length of list
7.Exit
6
The length is 2
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
```

```
6.Length of list
7.Exit
4
Enter the locatin of node to be deleted
2
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
6.Length of list
7.Exit
5
456
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
6.Length of list
7.Exit
4
Enter the locatin of node to be deleted
1
Enter the operation
1.Add in begin
2.Add at end
3.Add after a node
4.Delete node
5.Display
6.Length of list
7.Exit
5
No nodes in the list
```

LAB-7

WAP Implement Single Link List with following operations

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

sort and reverse:

```
#include <stdio.h>
#include <stdlib.h>
void count();
void putBeg(int value);
void put(int pos, int value);
void putEnd(int value);

void delFirst();
void delLast();
void delPos();
void display();
void inputValue(int *add);
void inputPos(int *add);

void reverse();
void sort();

typedef struct node {
    int value;
   struct node* next;
```

```
}node;
void swap(node*, node*);
node* head = NULL;

void putBeg(int value)
{
    node* ptr = (node*) malloc((sizeof(node)));
    ptr -> value = value;
    ptr->next = head;
    head = ptr;
}
void put(int pos, int value)
{
    if(pos==0)
    {
        putBeg(value);
        return;
    }
    int i  = 0;
    node* tmp = head;
    while(i != pos-1 && tmp != NULL)
    {
        i++;
        tmp = tmp->next;
    }
```

```
    if(i != pos-1 || tmp == NULL)
    {
        printf("\n\nERROR\nEnter Correct Index\n\n");
        return;
    }
    node* ptr = (node*) malloc((sizeof(node)));
    ptr -> value = value;
    ptr->next = (tmp->next);
    tmp->next = ptr;
}

void putEnd(int value) {

    node* ptr = (node*) malloc((sizeof(node)));
    ptr -> value = value;
    ptr->next=NULL;
    if(head==NULL)
    {
        head=ptr;
        return;
    }
    node* tmp = head;
    for(;tmp->next!=NULL;tmp=tmp->next);
    tmp->next=ptr;
}
```

```
void display() {

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

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

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

```
        return;
    }
    if(head->next == NULL)
    {
        free(head);
        head=NULL;
        return;
    }
    node* tmp = head;
    for(;(tmp->next)->next!=NULL;tmp=tmp->next);
    free(tmp->next);
    tmp->next = NULL;
}

void delPos(int pos){

    if(head==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
    if(pos==0)
    {
        delFirst();
        return;
    }
```

```
    int i  = 0;
    node* tmp = head;
    while(i!=pos-1&&tmp!=NULL)
    {
        i++;
        tmp = tmp->next;
    }
    if(i!=pos-1||tmp->next==NULL)
    {
        printf("\n\nERROR\nEnter Correct Index\n\n");
        return;
    }
    node* tmp1 = tmp->next;
    tmp->next = (tmp->next)->next;
    free(tmp1);
}
void reverse() {
    if(head == NULL) {
        printf("\n\nLinked List is empty\n\n");
        return;
    }
    if(head -> next == NULL){
        printf("\n\nReversed\n\n");
        return;
    }
```

```
    node* tmp;
    node* current = head -> next;
    node* previous = head;
    while(current != NULL) {
        tmp = current->next;
        current -> next = previous;
        previous = current;
        current = tmp;
    }
    head->next=NULL;
    head = previous;
    printf("\n\nReversed\n\n");
    return;
}
void sort()
{
    int flag, i;
    node *ptr1;
    node *ptr2 = NULL;

    if (head == NULL)
        return;

    do
    {
```

```
        flag = 0;
        ptr1 = head;

        while (ptr1->next != ptr2)
        {
            if (ptr1->value > ptr1->next->value)
            {
                swap(ptr1, ptr1->next);
                flag = 1;
            }
            ptr1 = ptr1->next;
        }
        ptr2 = ptr1;
    }
    while (flag);
    printf("\n\nSorted\n\n");
}
void swap(node *a, node *b)
{
    int temp = a->value;
    a->value = b->value;
    b->value = temp;
}
void inputValue(int *add)
{
```

```
    printf("Enter element to be added : ");
    scanf("%d", add);
}
void inputPos(int* add){
    printf("Enter index : ");
    scanf("%d", add);
}

void main() {
    int choice = 0, input, pos;
    while(1)
    {
        count();
        printf("Enter 1 to add at beginning\n");
        printf("Enter 2 to add at end\n");
        printf("Enter 3 to add at given index\n");
        printf("Enter 4 to delete first element\n");
        printf("Enter 5 to delete last element\n");
        printf("Enter 6 to delete element at given index\n");
        printf("Enter 7 to display\n");
        printf("Enter 8 to reverse\n");
        printf("Enter 9 to sort\n");
        printf("Enter -1 to quit\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
```

```
if(choice==-1)
    break;
switch(choice)
{
    case 1:
        inputValue(&input);
        putBeg(input);
        break;
    case 2:
        inputValue(&input);
        putEnd(input);
        break;
    case 3:
        inputValue(&input);
        inputPos(&pos);
        put(pos, input);
        break;
    case 4:
        delFirst();
        break;
    case 5:
        delLast();
        break;
    case 6:
        inputPos(&pos);
```

```
            delPos(pos);
            break;
        case 7:
            display();
            break;
        case 8:
            reverse();
            break;
        case 9:
            sort();
            break;
        default:
            printf("\n\nIncorrect Choice\n\n");
            break;
      }
   }
   printf("\n\n-----DONE-----\n\n");
}
void count() {
   int i = 0;
   node* tmp=head;
   while(tmp!=NULL) {
      i++;
      tmp = tmp->next;
   }
```

```
    printf("\n\nCount : %d\n\n", i);
}
```

OUTPUT:

```
Count : 0

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at given index
Enter 4 to delete first element
Enter 5 to delete last element
Enter 6 to delete element at given index
Enter 7 to display
Enter 8 to reverse
Enter 9 to sort
Enter -1 to quit
Enter your choice : 1
Enter element to be added : 10


Count : 1

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at given index
Enter 4 to delete first element
Enter 5 to delete last element
Enter 6 to delete element at given index
Enter 7 to display
Enter 8 to reverse
Enter 9 to sort
Enter -1 to quit
Enter your choice : 1
Enter element to be added : 20


Count : 2
```

```
Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at given index
Enter 4 to delete first element
Enter 5 to delete last element
Enter 6 to delete element at given index
Enter 7 to display
Enter 8 to reverse
Enter 9 to sort
Enter -1 to quit
Enter your choice : 1
Enter element to be added : 20


Count : 2

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at given index
Enter 4 to delete first element
Enter 5 to delete last element
Enter 6 to delete element at given index
Enter 7 to display
Enter 8 to reverse
Enter 9 to sort
Enter -1 to quit
Enter your choice : 2
Enter element to be added : 30


Count : 3
```

```
Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at given index
Enter 4 to delete first element
Enter 5 to delete last element
Enter 6 to delete element at given index
Enter 7 to display
Enter 8 to reverse
Enter 9 to sort
Enter -1 to quit
Enter your choice : 7


Linked List Contains : 20  10  30



Count : 3

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at given index
Enter 4 to delete first element
Enter 5 to delete last element
Enter 6 to delete element at given index
Enter 7 to display
Enter 8 to reverse
Enter 9 to sort
Enter -1 to quit
Enter your choice : 9


Sorted
```

```
Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at given index
Enter 4 to delete first element
Enter 5 to delete last element
Enter 6 to delete element at given index
Enter 7 to display
Enter 8 to reverse
Enter 9 to sort
Enter -1 to quit
Enter your choice : 7


Linked List Contains : 10  20  30



Count : 3

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at given index
Enter 4 to delete first element
Enter 5 to delete last element
Enter 6 to delete element at given index
Enter 7 to display
Enter 8 to reverse
Enter 9 to sort
Enter -1 to quit
Enter your choice : 8


Reversed
```

```
Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at given index
Enter 4 to delete first element
Enter 5 to delete last element
Enter 6 to delete element at given index
Enter 7 to display
Enter 8 to reverse
Enter 9 to sort
Enter -1 to quit
Enter your choice : 7


Linked List Contains : 30  20  10



Count : 3

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at given index
Enter 4 to delete first element
Enter 5 to delete last element
Enter 6 to delete element at given index
Enter 7 to display
Enter 8 to reverse
Enter 9 to sort
Enter -1 to quit
Enter your choice : -1
```

merge , contatenate:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
   int value;
   struct node* next;
}node;
void swap(node*, node*);
void add_beg(int, node**);
node* head3 = NULL;
node* head1 = NULL;
node* head2 = NULL;
node* head4 = NULL;
void del_beg(node** head) {
   if(*head==NULL){
      printf("\n\nLinked List is Empty\n\n");
      return;
   }
   node *tmp = (*head)->next;
   free(*head);
   *head = tmp;
}
void del_end(node** head) {
   if(*head==NULL){
      printf("\n\nLinked List is Empty\n\n");
```

```
        return;
    }
    if((*head)->next == NULL)
    {
        free(*head);
        *head=NULL;
        return;
    }
    node* tmp = *head;
    for(;(tmp->next)->next!=NULL;tmp=tmp->next);
    free(tmp->next);
    tmp->next = NULL;
}
void add_end(int value, node** head) {
    node* ptr = (node*) malloc(sizeof(node));
    ptr -> value = value;
    ptr->next=NULL;
    if(*head == NULL) {
        *head = ptr;
        return;
    }
    node* tmp = *head;
    while(tmp->next!=NULL) {
        tmp = tmp -> next;
    }
```

```
    tmp -> next = ptr;
}
void del_pos(int pos, node** head){

    if(*head==NULL){
        printf("\n\nLinked List is Empty\n\n");
        return;
    }
    if(pos==0)
    {
        del_beg(head);
        return;
    }
    int i  = 0;
    node* tmp = *head;
    while(i!=pos-1&&tmp!=NULL)
    {
        i++;
        tmp = tmp->next;
    }
    if(i!=pos-1||tmp->next==NULL)
    {
        printf("\n\nERROR\nEnter Correct Index\n\n");
        return;
    }
```

```
    node* tmp1 = tmp->next;
    tmp->next = (tmp->next)->next;
    free(tmp1);
}

void add_pos(int pos, int value, node** head)
{
    if(pos==0)
    {
        add_beg(value, head);
        return;
    }
    int i  = 0;
    node* tmp = *head;
    while(i != pos-1 && tmp != NULL)
    {
        i++;
        tmp = tmp->next;
    }
    if(i != pos-1 || tmp == NULL)
    {
        printf("\n\nERROR\nEnter Correct Index\n\n");
        return;
    }
    node* ptr = (node*) malloc((sizeof(node)));
```

```
    ptr -> value = value;
    ptr->next = (tmp->next);
    tmp->next = ptr;
}
void add_beg(int value, node** head) {
    node* ptr = (node*) malloc(sizeof(node));
    ptr -> value = value;
    ptr -> next = *head;
    *head = ptr;
}
void sort(node* start)
{
    int flag, i;
    node *ptr1;
    node *ptr2 = NULL;

    if (start == NULL)
        return;
    do
    {
        flag = 0;
        ptr1 = start;

        while (ptr1->next != ptr2)
        {
```

```
            if (ptr1->value > ptr1->next->value)
            {
                swap(ptr1, ptr1->next);
                flag = 1;
            }
            ptr1 = ptr1->next;
        }
        ptr2 = ptr1;
    }
    while (flag);
}
void swap(node *a, node *b)
{
    int temp = a->value;
    a->value = b->value;
    b->value = temp;
}


void merge(node* curr1, node* curr2, node* prev) {
    int flag1 = (curr1 == NULL);
    int flag3 = (curr2 == NULL);
    if(flag1 && flag3)
        return;
    node* newNode = (node*) malloc(sizeof(node));
```

```
    newNode -> next = NULL;
    if(prev==NULL) {
        sort(head1);
        sort(head2);
        head3 = newNode;
    }
    int flag2 = 1;
    if(!flag1 && !flag3){
        flag2 = curr1->value >= curr2->value;
    }
    else if(flag3)
        flag2 = 0;
    if(flag1 || flag2) {
        newNode->value = curr2->value;
        curr2 = curr2->next;
    }
    else {
        newNode->value = curr1->value;
        curr1 = curr1->next;
    }
    if(prev!=NULL)
        prev->next = newNode;
    prev = newNode;
    merge(curr1, curr2, prev);
}
```

```
void concatenate() {
    if(head1==NULL&&head2==NULL)
        return;
    node* tmp = head1;
    if(head1!=NULL)
    {
        while(tmp->next!=NULL) {
            tmp = tmp->next;
        }
    }
    else {
        head1 = head2;
        return;
    }
    tmp->next = head2;
}
void display(node* head) {

    if(head == NULL) {
        printf("Empty\n");
        return;
    }
    node* tmp = head;

    while(tmp != NULL)
```

```
    {
        printf("%d  ", tmp->value);
        tmp = tmp->next;
    }
    printf("\n\n");
}
void main() {
    int input, f, ch, pos;
    while(1) {
        printf("\nEnter 1 to add at beginning\n");
        printf("Enter 2 to add at end\n");
        printf("Enter 3 to add at any index\n");
        printf("Enter 4 to delete at beginning\n");
        printf("Enter 5 to delete at end\n");
        printf("Enter 6 to delete at any index\n");
        printf("Enter 7 to display\n");
        printf("Enter 8 to concatenate\n");
        printf("Enter -1 to merge\n");
        printf("Enter your choice : ");
        scanf("%d", &ch);
        if(ch==-1 || ch==8)break;
        switch(ch) {
            case 1:
                printf("Enter 1 for first list 2 for second list : ");
                scanf("%d", &f);
```

```
    printf("Enter element to add : ");
    scanf("%d", &input);
    if(f==1)
       add_beg(input,&head1);
    else if(f==2)
       add_beg(input, &head2);
    else
       printf("\n\nwrong input\n\n");
    break;
case 2:
    printf("Enter 1 for first list 2 for second list : ");
    scanf("%d", &f);
    printf("Enter element to add : ");
    scanf("%d", &input);
    if(f==1)
       add_end(input,&head1);
    else if(f==2)
       add_end(input, &head2);
    else
       printf("\n\nwrong input\n\n");
    break;
case 3:
    printf("Enter 1 for first list 2 for second list : ");
    scanf("%d", &f);
    printf("Enter element to add : ");
```

```
    scanf("%d", &input);
    printf("Enter index : ");
    scanf("%d", &pos);
    if(f==1)
       add_pos(pos, input,&head1);
    else if(f==2)
       add_pos(pos, input, &head2);
    else
       printf("\n\nwrong input\n\n");
    break;
case 4:
    printf("Enter 1 for first list 2 for second list : ");
    scanf("%d", &f);
    if(f==1)
       del_beg(&head1);
    else if(f==2)
       del_beg(&head2);
    else
       printf("\n\nwrong input\n\n");
    break;
case 5:
    printf("Enter 1 for first list 2 for second list : ");
    scanf("%d", &f);
    if(f==1)
       del_end(&head1);
```

```
    else if(f==2)
        del_end(&head2);
    else
        printf("\n\nwrong input\n\n");
    break;
case 6:
    printf("Enter 1 for first list 2 for second list : ");
    scanf("%d", &f);
    printf("Enter index : ");
    scanf("%d", &pos);
    if(f==1)
        del_pos(pos,&head1);
    else if(f==2)
        del_pos(pos, &head2);
    else
        printf("\n\nwrong input\n\n");
    break;
case 7:
    printf("Enter 1 for first list 2 for second list : ");
    scanf("%d", &f);
    if(f==1){
        printf("Linked List contains : ");
        display(head1);
    }
    else if(f==2){
```

```
                printf("Linked List contains : ");
                display(head2);
            }
            else
                printf("\n\nwrong input\n\n");
            break;
        default:
            printf("\n\nwrong input\n\n");
      }

   }
   if(ch==8) {
      concatenate();
      printf("\nConcatenated List ");
      display(head1);
      return;
   }
   merge(head1, head2, head3);
   printf("\nFirst List : ");
   display(head1);
   printf("\nSecond List : ");
   display(head2);
   printf("\nMerged Linked List : ");
   display(head3);
}
```

OUTPUT:

```
Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at any index
Enter 4 to delete at beginning
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : 1
Enter 1 for first list 2 for second list : 1
Enter element to add : 30

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at any index
Enter 4 to delete at beginning
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : 1
Enter 1 for first list 2 for second list : 2
Enter element to add : 45

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at any index
Enter 4 to delete at beginning
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : 8
```

```
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : 1
Enter 1 for first list 2 for second list : 1
Enter element to add : 30

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at any index
Enter 4 to delete at beginning
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : 1
Enter 1 for first list 2 for second list : 2
Enter element to add : 45

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at any index
Enter 4 to delete at beginning
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : 8

Concatenated List 30  45
```

```
Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at any index
Enter 4 to delete at beginning
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : 1
Enter 1 for first list 2 for second list : 1
Enter element to add : 18

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at any index
Enter 4 to delete at beginning
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : 1
Enter 1 for first list 2 for second list : 2
Enter element to add : 45

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at any index
Enter 4 to delete at beginning
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : -1
```

```
Enter 1 for first list 2 for second list : 1
Enter element to add : 18

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at any index
Enter 4 to delete at beginning
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : 1
Enter 1 for first list 2 for second list : 2
Enter element to add : 45

Enter 1 to add at beginning
Enter 2 to add at end
Enter 3 to add at any index
Enter 4 to delete at beginning
Enter 5 to delete at end
Enter 6 to delete at any index
Enter 7 to display
Enter 8 to concatenate
Enter -1 to merge
Enter your choice : -1

First List : 18


Second List : 45


Merged Linked List : 18  45


c:\Users\Yashwanth\Desktop\DS LAB PROGRAMS\LAB 7 SINGLY LINKED LIST>
```

LAB-8

WAP to implement Stack & Queues using Linked Representation.

Stack

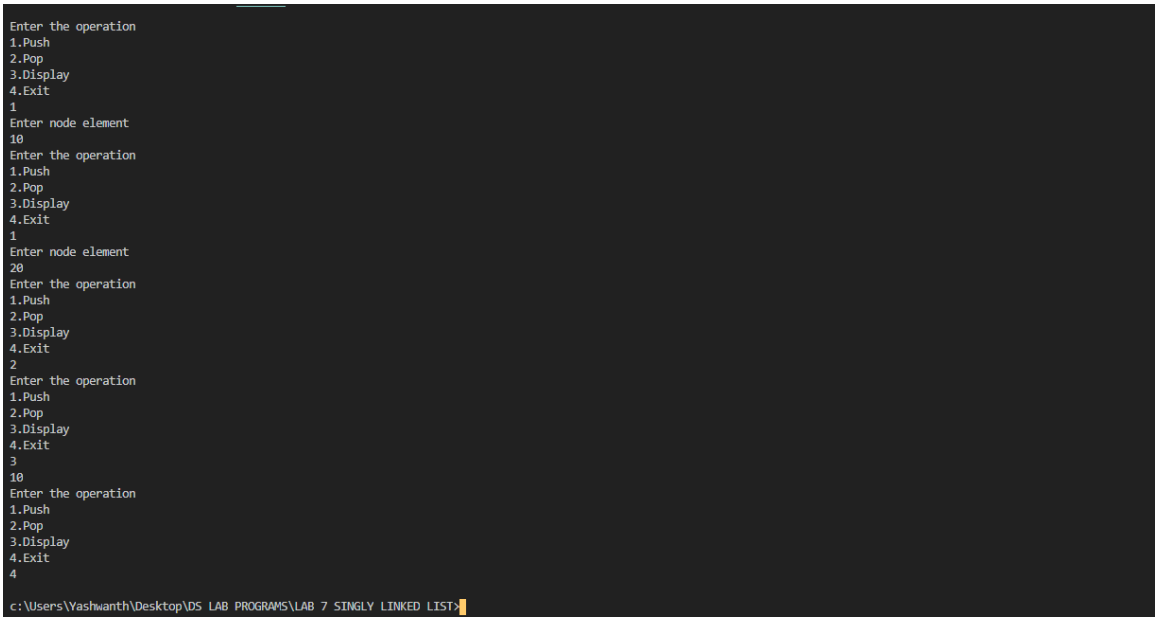
```
#include <stdio.h>
#include <stdlib.h>

typedef struct node{
  int data;
  struct node *link;
}node;

node *top=NULL;



void push()
{
 node *temp;
temp=(node *)malloc(sizeof(node));
 printf("Enter node element\n");
 scanf("%d",&temp->data);
 temp->link=NULL;

 if(top==NULL)
 {
```

```
    top=temp;
 }
 else
 {
  temp->link=top;
  top=temp;
 }
}

void pop()
{

node *temp;

if(top==NULL)
{
  printf("Stack is empty\n");
}

else
{
 temp=top;
top=temp->link;
temp->link=NULL;
free(temp);
```

```
}

}


void display()
{
 node *temp=top;
 if(temp==NULL)
 {
   printf("Stack is empty\n");
 }
 else
 {
   while(temp!=NULL)
   {
     printf("%d\n",temp->data);
     temp=temp->link;
   }
 }
}

int main()
{
```

```
 int op,len;
 while(1)
{ printf("Enter the operation\n1.Push\n");
  printf("2.Pop\n3.Display\n4.Exit\n");
  scanf("%d",&op);
  switch (op)
  {
  case 1:push();
    break;
  case 2: pop();
    break;
  case 3: display();
    break;
  case 4: exit(0);
     break;
  default: printf("No such operation\n");
  }
 }
return 0;
}
```

OUTPUT:



Queue:

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

typedef struct node{
  int data;
  struct node *link;
}node;

node *root=NULL;

void enqueue()
{
   node *temp;
```

```
   temp=(node *)malloc(sizeof(node));

   printf("Enter the node element\n");
   scanf("%d",&temp->data);
    temp->link=NULL;
   if(root==NULL)
  {
  root=temp;
  }
  else
  {
    node *p=root;
    while(p->link!=NULL)
   {
    p=p->link;
   }
   p->link=temp;
  }
}


void dequeue()
{
node *temp;
```

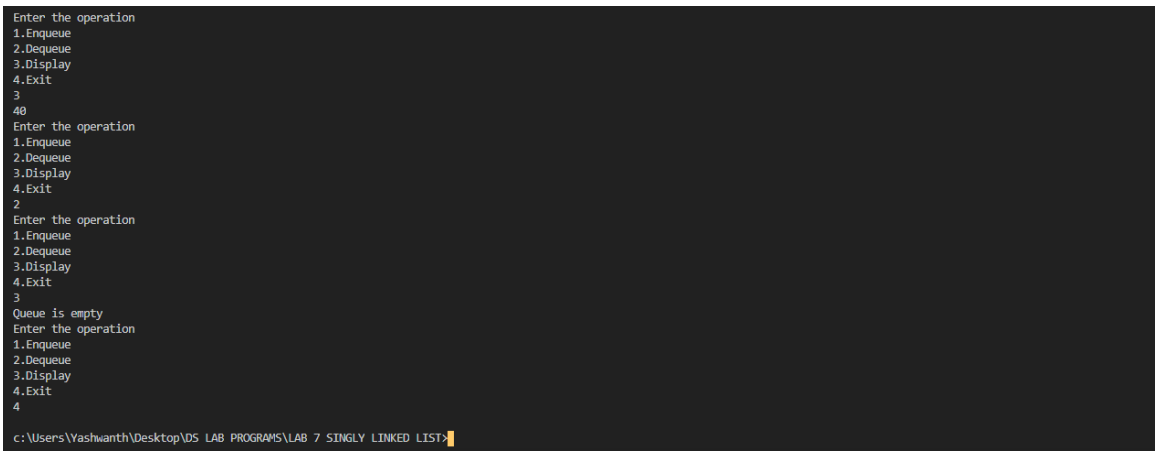
```
if(root==NULL)
{
  printf("Queue is empty\n");
}

else
{
 temp=root;
root=temp->link;
temp->link=NULL;
free(temp);
}
}


void display()
{
 node *temp=root;
 if(temp==NULL)
 {
   printf("Queue is empty\n");
 }
 else
 {
   while(temp!=NULL)
```

```
   {
     printf("%d\n",temp->data);
     temp=temp->link;
   }
 }
}

int main()
{

 int op,len;
 while(1)
{ printf("Enter the operation\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
  scanf("%d",&op);
  switch (op)
  {
  case 1:enqueue();
    break;
  case 2: dequeue();
    break;
  case 3: display();
    break;
  case 4: exit(0);
     break;
  default: printf("No such operation\n");
```

```
  }
 }
return 0;
}
```

OUTPUT:

```
C:\Users\Yashwanth>cd "c:\Users\Yashwanth\Desktop\DS LAB PROGRAMS\LAB 7 SINGLY LINKED LIST\" && gcc QUEUEusingLL.c -o QUEUEusingLL && "c:\Users\Yashwanth\Desktop\DS LAB PROGRAMS\LAB
7 SINGLY LINKED LIST\"QUEUEusingLL
Enter the operation
1.Enqueue
2.Dequeue
3.Display
4.Exit
1
Enter the node element
20
Enter the operation
1.Enqueue
2.Dequeue
3.Display
4.Exit
1
Enter the node element
40
Enter the operation
1.Enqueue
2.Dequeue
3.Display
4.Exit
3
20
40
Enter the operation
1.Enqueue
2.Dequeue
3.Display
4.Exit
2
```



```
Enter the operation
1.Enqueue
2.Dequeue
3.Display
4.Exit
3
40
Enter the operation
1.Enqueue
2.Dequeue
3.Display
4.Exit
2
Enter the operation
1.Enqueue
2.Dequeue
3.Display
4.Exit
3
Queue is empty
Enter the operation
1.Enqueue
2.Dequeue
3.Display
4.Exit
4

c:\Users\Yashwanth\Desktop\DS LAB PROGRAMS\LAB 7 SINGLY LINKED LIST>
```

LAB-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.

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

typedef struct Node {
    int value;
    struct Node* prev;
    struct Node* next;
}node;

node* head = NULL;

void add_beg(int value) {
    node* ptr = (node*) malloc(sizeof(node));
    ptr->value = value;
    ptr->prev = NULL;
    ptr->next = head;
    if(head!=NULL) {
        head->prev = ptr;
    }
```

```
    head = ptr;
}

void add_key(int value, int key) {
    node* tmp = head;
    while(tmp!=NULL) {
        if(tmp->value == key) {
            break;
        }
        tmp = tmp->next;
    }
    if(tmp==NULL) {
        printf("\nNo Match\n");
        return;
    }
    if(tmp==head) {
        add_beg(value);
        return;
    }
    node* ptr = (node*) malloc(sizeof(node));
    ptr->value = value;
    ptr->prev = tmp->prev;
    ptr->next = tmp;
    (tmp->prev)->next = ptr;
    tmp->prev = ptr;
```

```
}
void del_key(int key) {
    if(head == NULL) {
        printf("\nList is Empty\n");
        return;
    }
    node* tmp = head;
    while(tmp != NULL) {
        if(tmp -> value == key) {
            break;
        }
        tmp = tmp->next;
    }
    if(tmp==head) {
        if(head->next==NULL)
        {
            free(head);
            head=NULL;
            return;
        }
        head = head->next;
        free(head->prev);
        head->prev = NULL;
        return;
    }
```

```
    if(tmp==NULL) {
        printf("\nNo Match\n");
        return;
    }
    if(tmp->next==NULL) {
        tmp->prev->next = NULL;
        free(tmp);
        return;
    }
    tmp->next->prev = tmp->prev;
    tmp->prev->next = tmp->next;
    free(tmp);
}
void display() {
    if(head == NULL) {
        printf("\nList is Empty\n");
        return;
    }
    node* tmp = head;
    printf("\nLinked list contains : ");
    while(tmp!= NULL) {
        printf("%d ", tmp->value);
        tmp = tmp->next;
    }
    printf("\n");
```

```
}
void main() {
    int choice, value, key;
    while(1) {
        printf("Enter 1 to add at beginning\n");
        printf("Enter 2 to add at left of a node\n");
        printf("Enter 3 to delete a node\n");
        printf("Enter 4 to display\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 insert : ");
                scanf("%d", &value);
                add_beg(value);
                break;
            case 2:
                printf("\nEnter value to insert : ");
                scanf("%d", &value);
                printf("\nEnter value of key node : ");
                scanf("%d", &key);
                add_key(value, key);
```

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

OUTPUT:

```
C:\Users\Yashwanth>cd "c:\Users\Yashwanth\Desktop\DS LAB PROGRAMS\LAB 8 DOUBLY LINKED LIST\" && gcc LAB8_DLL.c -o LAB8_DLL && "c:\Users\Yashwanth\Desktop\DS LAB PROGRAMS\LAB 8 DOUBLY
 LINKED LIST\"LAB8_DLL
Enter 1 to add at beginning
Enter 2 to add at left of a node
Enter 3 to delete a node
Enter 4 to display
Enter -1 to quit
Enter your choice :
1

Enter value to insert : 100
Enter 1 to add at beginning
Enter 2 to add at left of a node
Enter 3 to delete a node
Enter 4 to display
Enter -1 to quit
Enter your choice : 2

Enter value to insert : 50

Enter value of key node : 100
Enter 1 to add at beginning
Enter 2 to add at left of a node
Enter 3 to delete a node
Enter 4 to display
Enter -1 to quit
Enter your choice : 4

Linked list contains : 50 100
Enter 1 to add at beginning
Enter 2 to add at left of a node
Enter 3 to delete a node
Enter 4 to display
Enter -1 to quit
```

```
Linked list contains : 50 100
Enter 1 to add at beginning
Enter 2 to add at left of a node
Enter 3 to delete a node
Enter 4 to display
Enter -1 to quit
Enter your choice : 3

Enter value of node to be deleted : 50
Enter 1 to add at beginning
Enter 2 to add at left of a node
Enter 3 to delete a node
Enter 4 to display
Enter -1 to quit
Enter your choice : 4

Linked list contains : 100
Enter 1 to add at beginning
Enter 2 to add at left of a node
Enter 3 to delete a node
Enter 4 to display
Enter -1 to quit
Enter your choice : -1


-----DONE-----


c:\Users\Yashwanth\Desktop\DS LAB PROGRAMS\LAB 8 DOUBLY LINKED LIST>
```

LAB-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.

```
#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);

int flag = 1;
```

```
void main()
{
   int ch;

   while(1)
   {
      printf("\n**MENU**\n");
      printf("1 - Insert an element into tree\n");
      printf("2 - Inorder Traversal\n");
      printf("3 - Preorder Traversal\n");
      printf("4 - Postorder Traversal\n");
      printf("5 - Exit\n");
      printf("\nEnter your choice : ");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         insert();
         break;
      case 2:
         printf("\nINORDER TRAVERSAL:\n");
         inorder(root);
         break;
      case 3:
```

```
            printf("\nPREORDER TRAVERSAL:\n");
            preorder(root);
            break;
        case 4:
            printf("\nPOSTORDER TRAVERSAL:\n");
            postorder(root);
            break;
        case 5:
            exit(0);
        default :
            printf("Invalid Choice!\n");
            break;
        }
    }
}

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 insert()
{
   create();
   if (root == NULL)
      root = temp;
   else
      search(root);
}

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 the tree!\n");
        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 the tree!\n");
        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 the tree!\n");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d -> ", t->value);
}
```

OUTPUT:

```
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 : 20

**MENU**
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 : 30

**MENU**
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 : 40

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

```
4 - Postorder Traversal
5 - Exit

Enter your choice : 2

INORDER TRAVERSAL:
20 -> 30 -> 40 ->
**MENU**
1 - Insert an element into tree
2 - Inorder Traversal
3 - Preorder Traversal
4 - Postorder Traversal
5 - Exit

Enter your choice : 3

PREORDER TRAVERSAL:
20 -> 30 -> 40 ->
**MENU**
1 - Insert an element into tree
2 - Inorder Traversal
3 - Preorder Traversal
4 - Postorder Traversal
5 - Exit

Enter your choice : 4

POSTORDER TRAVERSAL:
40 -> 30 -> 20 ->
**MENU**
1 - Insert an element into tree
2 - Inorder Traversal
3 - Preorder Traversal
4 - Postorder Traversal
5 - Exit

Enter your choice :
```