LAB 1

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 3

int top=-1,stack[MAX];
void push();
void pop();
void display();

void main()
{
   int ch;
   while(1)
   {
         printf("\n1.Push\n2.Pop\n3.Display\n4.Exit");
         printf("\n\nEnter your choice(1-4):");
         scanf("%d",&ch);
         switch(ch)
         {
               case 1: push();
                           break;
               case 2: pop();
                           break;
               case 3: display();
                           break;
               case 4: exit(0);
               default: printf("\nWrong Choice!!");
         }
   }
}

void push()
{
   int val;

   if(top==MAX-1)
   {
```

```
        printf("\nStack overflow!!");
    }
    else
    {
        printf("\nEnter element to push:");
        scanf("%d",&val);
        top=top+1;
        stack[top]=val;
    }
}

void pop()
{
    if(top==-1)
    {
        printf("\nStack underflow!!");
    }
    else
    {
        printf("\nPopped element is %d",stack[top]);
        top=top-1;
    }
}

void display()
{
    int i;
    if(top==-1)
    {
        printf("\nStack is empty!!");
    }
    else
    {
        printf("\nStack is...\n");
        for(i=top;i>=0;--i)
            printf("%d\n",stack[i]);
    }
}
```

```
1.Push
2.Pop
3.Display
4.Exit

Enter your choice(1-4):1

Enter element to push:10

1.Push
2.Pop
3.Display
4.Exit

Enter your choice(1-4):1

Enter element to push:20

1.Push
2.Pop
3.Display
4.Exit

Enter your choice(1-4):1

Enter element to push:30

1.Push
2.Pop
3.Display
4.Exit

Enter your choice(1-4):3
Stack is...
30
20
10

1.Push
2.Pop
3.Display
4.Exit

Enter your choice(1-4):2

Popped element is 30
1.Push
```

```
Enter your choice(1-4):2

Popped element is 20
1.Push
2.Pop
3.Display
4.Exit

Enter your choice(1-4):2

Popped element is 10
```

LAB 2

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

#define SIZE 50

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

void 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);
 }
}

int main()
{
 char infix[50],postfix[50],ch,elem;
 int i=0,k=0;

 printf("Enter the 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("\nPostfix Expression = %s\n",postfix);

    return 0;
  }
```

```
Enter the Infix expression: A+B*(C^D-E)^(F+G*H)-I

Postfix Expression = ABCD^E-FGH*+^*+I-


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

LAB 3

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

#define MAX 50

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

void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("\nQueue is Empty \n");
        return ;
```

```
    }
    else
    {
        printf("\nElement deleted from queue is : %d\n", queueArray[front]);
        front = front + 1;
    }
}

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

```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Insert the element in queue : 4
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Insert the element in queue : 5
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 3
Queue is :
4 5
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 2

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

Element deleted from queue is : 5
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 2

Queue is Empty
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
```

LAB 4

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

int front=-1, rear=-1;

int main()
{
    int ch;
```

```
int item, MAX, i;
printf("Enter the size of queue: ");
scanf("%d",&MAX);
int queue[MAX];
do{
    printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1: if(front==(rear+1)%MAX)
                    printf("Queue is full\n");
                else
                {
                    printf("Enter the element: ");
                    scanf("%d",&item);
                    rear=(rear+1)%MAX;
                    queue[rear]=item;
                    if(front ==-1)
                      front=0;
                }
                break;

        case 2: if(front==-1 && rear==-1)
                    printf("Queue is empty\n");
                else
                {
                    item=queue[front];
                    if(front==rear)
                    {
                        front=-1;
                        rear=-1;
                    }
                    else
                        front=(front+1)%MAX;
                    printf("Removed element is %d \n",item);
                }
                break;
```

```
            case 3: printf("Queue contents are: ");
                    for(i=front;i!=rear;i=(i+1)%MAX)
                        printf(" %d ", queue[i]);
                    printf(" %d ", queue[i]);
                    printf("\n");
                    break;

            case 4: exit(0);
        }
    } while (ch != 4);
    return 0;
}
```

```
Enter the size of queue: 3

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element: 11

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element: 12

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element: 13

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Queue is full
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue contents are:  11  12  13

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Removed element is 11

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Removed element is 12

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
```

```
Enter the element: 14

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element: 15

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue contents are:  13  14  15
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Removed element is 13

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Removed element is 14

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Removed element is 15

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue is empty
```

LAB 5 & 6

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
   int info;
   struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
   NODE x;
   x=(NODE)malloc(sizeof(struct node));
   if(x==NULL)
   {
      printf("mem full\n");
      exit(0);
   }
   return x;
}
void freenode(NODE x)
{
   free(x);
}
NODE insert_front(NODE first,int item)
{
   NODE temp;
```

```
    temp=getnode();
    temp->info=item;
    temp->link=NULL;
    if(first==NULL)
    return temp;
    temp->link=first;
    first=temp;
    return first;
}
NODE delete_front(NODE first)
{
    NODE temp;
    if(first==NULL)
    {
        printf("list is empty cannot delete\n");
        return first;
    }
    temp=first;
    temp=temp->link;
    printf("item deleted at front-end is=%d\n",first->info);
    free(first);
    return temp;
}
NODE insert_rear(NODE first,int item)
{
    NODE temp,cur;
    temp=getnode();
    temp->info=item;
```

```
    temp->link=NULL;
    if(first==NULL)
    return temp;
    cur=first;
    while(cur->link!=NULL)
    cur=cur->link;
    cur->link=temp;
    return first;
}
NODE delete_rear(NODE first)
{
    NODE cur,prev;
    if(first==NULL)
    {
        printf("list is empty cannot delete\n");
        return first;
    }
    if(first->link==NULL)
    {
        printf("item deleted is %d\n",first->info);
        free(first);
        return NULL;
    }
    prev=NULL;
    cur=first;
    while(cur->link!=NULL)
    {
        prev=cur;
```

```
    cur=cur->link;
  }
  printf("item deleted at rear-end is %d",cur->info);
  free(cur);
  prev->link=NULL;
  return first;
}
NODE order_list(int item,NODE first)
{
NODE temp,prev,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
if(item<first->info)
{
  temp->link=first;
  return temp;
  }
  prev=NULL;
  cur=first;
  while(cur!=NULL&&item>cur->info)
  {
    prev=cur;
    cur=cur->link;
  }
  prev->link=temp;
```

```
    temp->link=cur;
    return first;
}
NODE delete_info(int key,NODE first)
{
    NODE prev,cur;
    if(first==NULL)
    {
        printf("list is empty\n");
        return NULL;
    }
    if(key==first->info)
    {
        cur=first;
        first=first->link;
        freenode(cur);
        return first;
    }
    prev=NULL;
    cur=first;
    while(cur!=NULL)
    {
        if(key==cur->info)break;
        prev=cur;
        cur=cur->link;
    }
    if(cur==NULL)
    {
```

```
        printf("search is unsuccessfull\n");
        return first;
    }
    prev->link=cur->link;
    printf("key deleted is %d",cur->info);
    freenode(cur);
    return first;
}
void display(NODE first)
{
    NODE temp;
    if(first==NULL)
    printf("list empty cannot display items\n");
    for(temp=first;temp!=NULL;temp=temp->link)
    {
        printf("%d\t",temp->info);
    }
}
int main()
{
    int item,choice,key;
    NODE first=NULL;
    printf(" 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n 4:Delete_rear\n 5:Order_list\n 6:Delete_info\n
7:Display_list\n 8:Exit\n");
    for(;;)
    {
        printf("\nEnter the choice: ");
        scanf("%d",&choice);
        switch(choice)
```

```
    {
      case 1: printf("enter the item at front-end: ");
              scanf("%d",&item);
              first=insert_front(first,item); break;
      case 2: first=delete_front(first); break;
      case 3: printf("enter the item at rear-end: ");
              scanf("%d",&item);
              first=insert_rear(first,item); break;
      case 4: first=delete_rear(first); break;
      case 5: printf("Enter the item to be inserted in ordered_list: ");
              scanf("%d",&item);
              first=order_list(item,first); break;
      case 6: printf("Enter the key to be deleted: ");
              scanf("%d",&key);
              first=delete_info(key,first); break;
      case 7: display(first); break;
      default:exit(0);
    }
  }
}
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:Delete_info
 7:Display_list
 8:Exit

Enter the choice: 1
enter the item at front-end: 10

Enter the choice: 1
enter the item at front-end: 20

Enter the choice: 1
enter the item at front-end: 30

Enter the choice: 3
enter the item at rear-end: 40
```

```
Enter the choice: 5
Enter the item to be inserted in ordered_list: 50

Enter the choice: 7
30      20      10      40      50
Enter the choice: 2
item deleted at front-end is=30

Enter the choice: 4
item deleted at rear-end is 50
Enter the choice: 6
Enter the key to be deleted: 2
search is unsuccessfull

Enter the choice: 6
Enter the key to be deleted: 20

Enter the choice: 7
10      40
```

LAB 7

```
#include <stdio.h>
#include <stdlib.h>
void sort();
void create();
void reverse();
void create_second();
void concatenate();
void display();
struct node
{
  int data;
  struct node *next;
};
struct node *head=NULL;
struct node *head2= NULL;
int c;

int main()
{
   int choice;
    do
    {
    printf("\n1. Create\n2. Sort\n3. Reverse\n4. Enter second list\n5. Concatenate\n6. Display\n7. Exit\nEnter your choice : ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1: create(); break;
        case 2: sort(); break;
        case 3: reverse(); break;
        case 4: create_second(); break;
        case 5: concatenate(); break;
        case 6: display(); break;
        case 7: exit(0);
    }
    }while(choice != 7);
}
```

```
void create(){
    struct node *newnode;
    struct node *temp;
    int s;
    printf("Enter integer  : ");
    scanf("%d",&s);
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data =s;
    if (head==NULL)
    {
      newnode->next=NULL;
      head=newnode;
      printf("First node created\n");
      c++;
    }
     else
     {
        temp=head;
        while(temp->next!=NULL)
            temp=temp->next;
    temp->next=newnode;
    newnode->next=NULL;
    c++;
    printf("Node created\n");
   }
}

void reverse(){
    struct node *prev=NULL,*current=head, *next=NULL;
    while(current!=NULL)
    {
        next=current->next;
        current->next=prev;
        prev=current;
        current=next;
    }
    head=prev;
    printf("The list is reversed\n");
```

```
}

void display(){
    struct node *ptr=NULL;
    ptr=head;

    if(ptr==NULL)
        printf("List is empty\n");
    else
    {
        printf("\nContents of the Linked List: ");
        while(ptr!=NULL)
       {
          printf("\t%d",ptr->data);
          ptr=ptr->next;
       }
    }
    printf("\n");
}
void create_second() {
  struct node *newnode;
  struct node *temp;
    int s,y;

    printf("Enter integer: ");
    scanf("%d",&s);
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data =s;
    if (head2==NULL)
    {
      newnode->next=NULL;
      head2=newnode;
      printf("First node created\n");
      c++;
    }
     else
     {
    temp=head2;
```

```
        while(temp->next!=NULL)
        {
      temp=temp->next;
        }
    temp->next=newnode;
    newnode->next=NULL;
    c++;
    printf("Node created\n");
   }
}

void concatenate(){
    struct node *ptr;
        if(head==NULL)
                head=head2;
        if(head2==NULL)
            head2=head;
        ptr=head;
        while(ptr->next!=NULL)
            ptr=ptr->next;
        ptr->next=head2;
        printf("The list is concatenated\n");
}

void sort(){
    int swap, i;
    struct node *ptr1;
    struct node *lptr = NULL;

    if (head == NULL)
        return;

    do
    {
        swap = 0;
        ptr1 = head;

        while (ptr1->next != lptr)
```

```
        {
            if (ptr1->data > ptr1->next->data)
            {
                int temp = ptr1->data;
                ptr1->data = ptr1->next->data;
                ptr1->next->data = temp;
                swap = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    }
    while(swap);
    printf("The list is sorted\n");
}
```

```
1. Create
2. Sort
3. Reverse
4. Enter second list
5. Concatenate
6. Display
7. Exit
Enter your choice : 1
Enter integer  : 47
First node created

1. Create
2. Sort
3. Reverse
4. Enter second list
5. Concatenate
6. Display
7. Exit
Enter your choice : 1
Enter integer  : 35
Node created

1. Create
2. Sort
3. Reverse
4. Enter second list
5. Concatenate
6. Display
7. Exit
Enter your choice : 1
```

```
Enter integer  : 70
Node created

1. Create
2. Sort
3. Reverse
4. Enter second list
5. Concatenate
6. Display
7. Exit
Enter your choice : 6

Contents of the Linked List:    47      35      70

1. Create
2. Sort
3. Reverse
4. Enter second list
5. Concatenate
6. Display
7. Exit
Enter your choice : 3
The list is reversed
```

```
Enter your choice : 6

Contents of the Linked List:    70      35      47

1. Create
2. Sort
3. Reverse
4. Enter second list
5. Concatenate
6. Display
7. Exit
Enter your choice : 2
The list is sorted

1. Create
2. Sort
3. Reverse
4. Enter second list
5. Concatenate
6. Display
7. Exit
Enter your choice : 6

Contents of the Linked List:    35      47      70
```

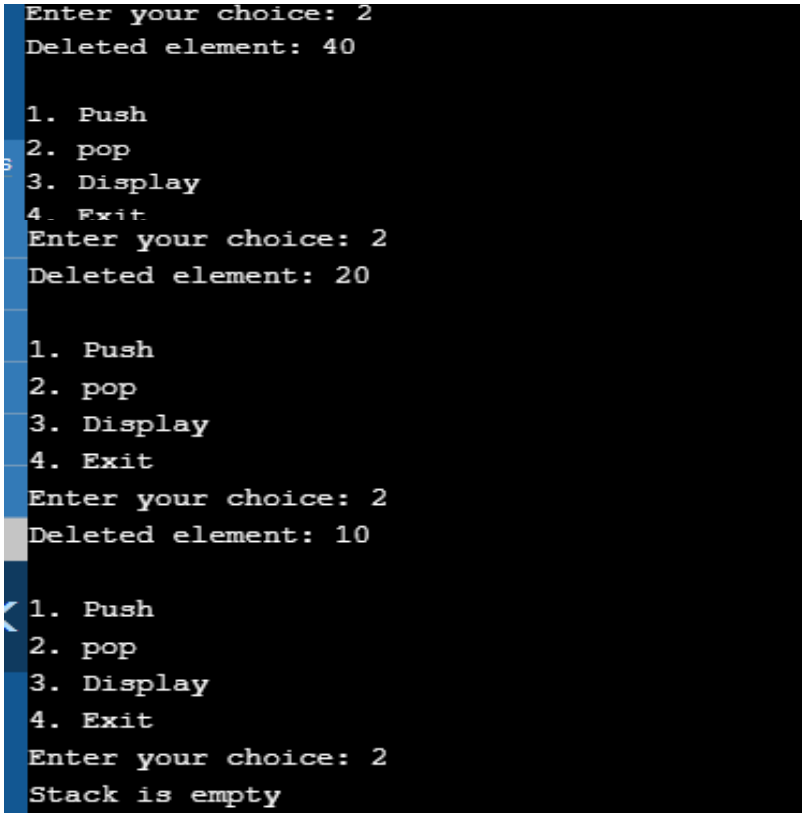
```
Enter your choice : 4
Enter integer: 22
First node created

1. Create
2. Sort
3. Reverse
4. Enter second list
5. Concatenate
6. Display
7. Exit
Enter your choice : 4
Enter integer: 44
Node created

1. Create
2. Sort
3. Reverse
4. Enter second list
5. Concatenate
6. Display
7. Exit
Enter your choice : 5
The list is concatenated

1. Create
2. Sort
3. Reverse
4. Enter second list
5. Concatenate
6. Display
7. Exit
Enter your choice : 6

Contents of the Linked List:    35      47      70      22      44
```

LAB 8

```
//Implementation of Stack using singly linked list
   #include <stdio.h>
   #include <stdlib.h>

   struct node
   {
       int data;
       struct node *next;
   };
   struct node *top=NULL;

   void push()
   {
       struct node *new_node;
       new_node = (struct node* ) malloc (sizeof(struct node));
       printf("Enter the element: ");
       scanf("%d",&new_node -> data);
       new_node->next = NULL;

       if (top == NULL)
           top= new_node;
       else
       {
           new_node->next = top;
           top = new_node;
       }
   }

   void pop()
   {
       if(top == NULL)
       printf("Stack is empty");
       else
       {
           printf("Deleted element: %d\n", top->data);
           top = top->next;
       }
```

```
}

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

void main()
{
    int ch;
    do
    {
        printf("\n1. Push\n2. pop\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", & ch);
        switch (ch)
        {
            case 1: push(); break;
            case 2: pop(); break;
            case 3: display(); break;
            case 4: exit(0);
        }
    }while(ch != 4);
}
```

```
1. Push
2. pop
3. Display
4. Exit
Enter your choice: 1
Enter the element: 10

1. Push
2. pop
3. Display
4. Exit
Enter your choice: 1
Enter the element: 20

1. Push
2. pop
3. Display
4. Exit
Enter your choice: 1
Enter the element: 30
1. Push
2. pop
3. Display
4. Exit
Enter your choice: 3
30      20      10

1. Push
2. pop
3. Display
Enter your choice: 2
Deleted element: 30

1. Push
2. pop
3. Display
4. Exit
Enter your choice: 1
Enter the element: 40

1. Push
2. pop
3. Display
4. Exit
Enter your choice: 3
40      20      10

1. Push
2. pop
3. Display
```




```
//Implementation queue Using Linked list
   #include <stdio.h>
   #include <stdlib.h>
   struct node{
       int data;
       struct node *next;
   };
   struct node *front = NULL,  *rear = NULL;

   void insert(){
       struct node *new_node;
       new_node = (struct node*) malloc(sizeof (struct node));
       printf("Enter the element: ");
       scanf("%d", & new_node->data);
       new_node->next = NULL;
       if (rear == NULL)
       {
           rear = new_node;
           front = new_node;
       }
       else
```

```
    {
        rear->next = new_node;
        rear = new_node;
    }
}

void del(){
    if (front == NULL)
    printf("Queue is empty\n");
    else
    {
        printf("Deleted element: %d\n", front-> data);
        front = front->next;
    }
}

void display() {
    struct node* temp;
    if (front == NULL)
    printf("Queue is empty\n");
    else
    {
        temp = front;
        while (temp!= NULL)
        {
            printf("%d\t", temp->data);
            temp = temp->next;
        }
    }
    printf("\n");
}

void main(){
    int ch;
    do{
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", & ch);
```

```
        switch (ch){
            case 1: insert(); break;
            case 2: del(); break;
            case 3: display(); break;
            case 4: exit(0);
        }
    }while(ch != 4);
}
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element: 10

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element: 20

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element: 30

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
10      20      30

1. Insert
2. Delete
3. Display
```

```
Enter your choice: 2
Deleted element: 10

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted element: 20

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted element: 30

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue is empty
```

LAB 9

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

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

void insert_left()
{
   int listele;
   struct node *new_node,*temp;
   printf("Enter the element in the list: ");
   scanf("%d",&listele);
   new_node=(struct node*)malloc(sizeof(struct node));
   printf("Enter the new node data: ");
   scanf("%d",&new_node->data);
   new_node->next=NULL;
   new_node->prev=NULL;
    if(head==NULL)
    {
        printf("Empty list\n"); return;
    }
   temp=head;
   while(temp->data!=listele)
   {
         temp=temp->next;
         if(temp==NULL)
         {
               printf("Element is  not in the list\n");
               return;
         }
   }
```

```
        if(temp->prev == NULL)
   {
        temp->prev = new_node;
        new_node->prev = NULL;
        new_node->next = temp;
        head = new_node;
   }
   else{
       new_node->prev=temp->prev;
         temp->prev=new_node;
       new_node->next=temp;
         new_node->prev->next = new_node;
   }
}

void insert_right()
{
    int listele;
   struct node *new_node,*temp;
   printf("Enter the element in the list: ");
   scanf("%d",&listele);
   new_node=(struct node*)malloc(sizeof(struct node));
   printf("Enter the new node data: ");
   scanf("%d",&new_node->data);
   new_node->next=NULL;
   new_node->prev=NULL;
    if(head==NULL)
    {
        printf("Empty list\n"); return;
    }
   temp=head;
   while(temp->data!=listele)
   {
        temp=temp->next;
        if(temp==NULL)
        {
              printf("Element is  not in the list\n");
              return;
```

```
        }
    }
    if(temp->next == NULL)
    {
        temp->next = new_node;
        new_node->prev = temp;
    }
    else
    {
        new_node->next=temp->next;
          temp->next=new_node;
        new_node->prev=temp;
        new_node->next->prev=new_node;
    }
}

void insert_end()
{
    struct node *new_node,*temp;
    new_node=(struct node*)malloc(sizeof(struct node));
    printf("Enter the element: ");
    scanf("%d",&new_node->data);
    new_node->next=NULL;
    new_node->prev=NULL;
    if(head==NULL)
    {
        head=new_node;
    }
    else
    {
        temp=head;
        while(temp->next!=NULL)
        temp=temp->next;
        temp->next=new_node;
        new_node->prev=temp;
    }
}
```

```
void del()
{
   struct node *temp;
   int ele;
    if(head==NULL)
    {
        printf("Empty List \n");
        return;
    }
   printf("Enter the element to be deleted: ");
   scanf("%d",&ele);
   temp=head;
   while(temp->data!=ele)
   {
         temp=temp->next;
         if(temp==NULL)
         {
          printf("Element is not in the list\n");
          return;
         }
   }
   if(temp==head)
   {
       head=head->next;
   }
    else if(temp->next==NULL)
   {
         temp=temp->prev;
         temp->next=NULL;
   }

   else
   {
       temp->prev->next=temp->next;
         temp->next->prev=temp->prev;
   }
}
```

```
   void display()
   {
      struct node *temp;
      temp=head;
       if(head==NULL)
       {
           printf("Empty List\n");
           return;
       }
      while(temp!=NULL)
      {
             printf("%d\t",temp->data);
             temp=temp->next;
      }
      printf("\n");
   }

   int main()
   {
      int ch;
      do
      {
            printf("\n1. Insert left\n2. Insert right\n3. Create\n4. Delete\n5.
Display\n6. Exit\n");
            printf("Enter your choice: ");
            scanf("%d",&ch);
            switch(ch)
            {
                   case 1: insert_left(); break;
                   case 2: insert_right(); break;
                   case 3: insert_end(); break;
                   case 4: del(); break;
                   case 5: display(); break;
                   case 6: exit(0);
            }
      }while (ch!= 6);
   }
```

```
1. Insert left
2. Insert right
3. Create
4. Delete
5. Display
6. Exit
Enter your choice: 3
Enter the element: 10

1. Insert left
2. Insert right
3. Create
4. Delete
5. Display
6. Exit
Enter your choice: 3
Enter the element: 20

1. Insert left
2. Insert right
3. Create
4. Delete
5. Display
6. Exit
Enter your choice: 3
Enter the element: 30
6. Exit
Enter your choice: 1
Enter the element in the list: 20
Enter the new node data: 15

1. Insert left
2. Insert right
3. Create
4. Delete
5. Display
6. Exit
Enter your choice: 2
Enter the element in the list: 20
Enter the new node data: 25

1. Insert left
2. Insert right
3. Create
4. Delete
5. Display
6. Exit
Enter your choice: 5
10      15      20      25      30
```

```
Enter your choice: 4
Enter the element to be deleted: 20

1. Insert left
2. Insert right
3. Create
4. Delete
5. Display
6. Exit
Enter your choice: 5
10      15      25      30
```

LAB 10

```
#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;

    printf("\nOPERATIONS ---");
    printf("\n1 - 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");
    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  ");
        }
    }
}

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))      /* value more than root node
value insert at right */
            search(t->r);
        else if ((temp->value > t->value) && (t->r == NULL))
            t->r = temp;
        else if ((temp->value < t->value) && (t->l != NULL))    /* value less than root
node value insert at left */
            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\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 a tree to display\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 a tree to display\n");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d -> ", t->value);
}
```

```
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 : 34

Enter your choice : 1
Enter data of node to be inserted : 45

Enter your choice : 1
Enter data of node to be inserted : 46

Enter your choice : 1
Enter data of node to be inserted : 67

Enter your choice : 2
34 -> 45 -> 46 -> 67 ->
Enter your choice : 3
34 -> 45 -> 46 -> 67 ->
Enter your choice : 4
67 -> 46 -> 45 -> 34 ->
Enter your choice : 5


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