# Data Structures-LAB Report

1BM19CS009
AJITHKUMAR G

## Program-1:
## (Stack_Implementation)

### Code:

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#define SIZE 5
int TOP=-1;

void operation(int[]);
void PUSH(int [],int);
int POP (int[]);
void DISPLAY(int[]);

int main()
{
    int stack[SIZE];
    operation(stack);
    return 0;
}

void operation(int stack[])
{
    int choice,ele,popele;
    char flag ;

    do{
        printf("\nEnter The choice: \n\t1.PUSH\t2.POP\t3.DISPLAY\t4.EXIT\n\nChoice : ");
        scanf("%d",&choice);
        switch(choice)
          {
            case 1:if(TOP==SIZE-1)
                   {
                       printf("\nThe Stack is full: OVERFLOW! Try other choices\n");
                       operation(stack);
                   }
                   else
                   {
                       printf("\nEnter The Element to be pushed: ");
                       scanf("%d",&ele);
                       PUSH(stack,ele);
                       printf("\nThe Element %d was pushed successfully",ele);
                   }
                   break;
                   break;
            case 2: if(TOP==-1)
                   {
                       printf("\nThe Stack is Empty: Underflow! Try other choices\n ");
                       operation(stack);
                   }
                   else
                   {
                       popele=stack[TOP];
                       POP(stack);
                       printf("\nThe Element %d was popped successfully ",popele);
                   }
                   break;
            case 3:
                   printf("\nThe Stack Elements are : \n");
                   DISPLAY(stack);
                   break;
           case 4:return;
            default:
                {
                    printf("\nInput choice ERROR! Try again! ");
                    operation(stack);
                }

         }

       printf("\n\nDo you wish to continue operations ? Press Y to continue  : ");
       fflush(stdin);
       flag=getchar();
    }while(toupper(flag)=='Y');

}
  void PUSH(int stack[],int ele)
{
    TOP++;
    stack[TOP]=ele;
}
 int POP(int stack[])
{
    int popele=stack[TOP];
```

```
    int popele=stack[TOP];
    TOP--;
    return popele;
}
void DISPLAY(int stack[])
{
    for(int i=TOP;i>=0;i--)
        printf("\n\t%d ",stack[i]);
}
```

# Output:

```
Enter The choice:
       1.PUSH  2.POP   3.DISPLAY       4.EXIT

Choice : 1

Enter The Element to be pushed: 1

The Element 1 was pushed successfully

Do you wish to continue operations ? Press Y to continue  : y

Enter The choice:
       1.PUSH  2.POP   3.DISPLAY       4.EXIT

Choice : 1

Enter The Element to be pushed: 2

The Element 2 was pushed successfully

Do you wish to continue operations ? Press Y to continue  : y

Enter The choice:
       1.PUSH  2.POP   3.DISPLAY       4.EXIT

Choice : 1

Enter The Element to be pushed: 3

The Element 3 was pushed successfully

Do you wish to continue operations ? Press Y to continue  : y

Enter The choice:
       1.PUSH  2.POP   3.DISPLAY       4.EXIT

Choice : 1

The Stack is full: OVERFLOW! Try other choices

Enter The choice:
       1.PUSH  2.POP   3.DISPLAY       4.EXIT

Choice : 3

The Stack Elements are :

       3
```

```
The Stack Elements are :

       3
       2
       1

Do you wish to continue operations ? Press Y to continue  : y

Enter The choice:
       1.PUSH  2.POP   3.DISPLAY       4.EXIT

Choice : 2

The Element 3 was popped successfully

Do you wish to continue operations ? Press Y to continue  : y

Enter The choice:
       1.PUSH  2.POP   3.DISPLAY       4.EXIT

Choice : 2

The Element 2 was popped successfully

Do you wish to continue operations ? Press Y to continue  : y

Enter The choice:
       1.PUSH  2.POP   3.DISPLAY       4.EXIT

Choice : 2

The Element 1 was popped successfully

Do you wish to continue operations ? Press Y to continue  : y

Enter The choice:
       1.PUSH  2.POP   3.DISPLAY       4.EXIT

Choice : 2

The Stack is Empty: Underflow! Try other choices

Enter The choice:
       1.PUSH  2.POP   3.DISPLAY       4.EXIT

Choice : 4
```

# Program-2:
# (Infix_to_PostFix)

## Code:

```c
#include<stdio.h>
#include<stdlib.h>
#define size 50
int top=-1;
char stack[size];

//Expression Intake
void getData(char *infix)
{
    printf("\nEnter The Infix Expression: ");
    scanf("%s",infix);
}
//PUSH and POP operation
void PUSH(int item)
{
    if(top==size-1)
        printf("\nStack Overflow, the expression exceeds the stack limits!");
    else
    stack[++top]=item;
}
char POP()
{
    if(top==-1)
        printf("\nStack Underflow, Stack is Empty!");
    else
    {
        int popItem=stack[top--];
        return popItem;
    }
}
//checking priority
int check(char ch)
{
    switch(ch)
    {
    case '+':
    case '-': return 1;
    case '*':
    case '/': return 2;
    case '^': return 3;
    default:  return 0;
```

```c
    default:  return 0;
    }
}

void inputChecking(char *infix)
{
    int operators=0,operand=0,brackets=0;
    for(int i=0;infix[i]!='\0';i++)
    {
       switch(infix[i])
        {
            case '(' :  brackets++;

                        break;
            case ')' :  brackets--;
                        break;
            case '+':

            case '-':

            case '*':

            case '^':

            case '/': operators++;
                      break;
            default:operand++;
        }
    }


    if((++operators)==operand && brackets==0)
        printf("\nThe given expression is valid.\n");
        else
        {
            printf("\nThe given expression is Invalid, retry! \n");
            main();
        }

}
```

```c

int main()
{
    char inFix[50],item,postFix[50];
    int k=0;
    getData(inFix);
    inputChecking(inFix);
    for(int i=0;inFix[i]!='\0';i++)
    {
        switch(inFix[i])
        {
            case '(' :  PUSH(inFix[i]);
                        break;
            case ')' :  while((item=POP())!='(')
                        postFix[k++]=item;
                        break;
            case '+':

            case '-':

            case '*':

            case '^':

            case '/':
                        while(check(stack[top])>=check(inFix[i]))
                        {
                            item=POP();
                            postfix[k++]=item;
                        }
                        PUSH(inFix[i]);
                        break;
            default :  postFix[k++]=inFix[i];
                        break;
        }


    }
```

```
    while(top!=-1)
        {
            item=POP();
            postFix[k++]=item;
        }
     postFix[k++]='\0';
    printf("\nThe PostFix Expression is : %s",postFix);
    return 0;
}
```

# Output:

```
Enter The Infix Expression: 2*(3/(4-1))

The PostFix Expression is : 2341-/*
Process returned 0 (0x0)   execution time : 8.314 s
Press any key to continue.
```

```
Enter The Infix Expression: a*(b+c^d-p/r+(t-s))

The PostFix Expression is : abcd^+pr/-ts-+*
Process returned 0 (0x0)   execution time : 37.303 s
Press any key to continue.
```

# Program-3:
# (Queue_Implementation)

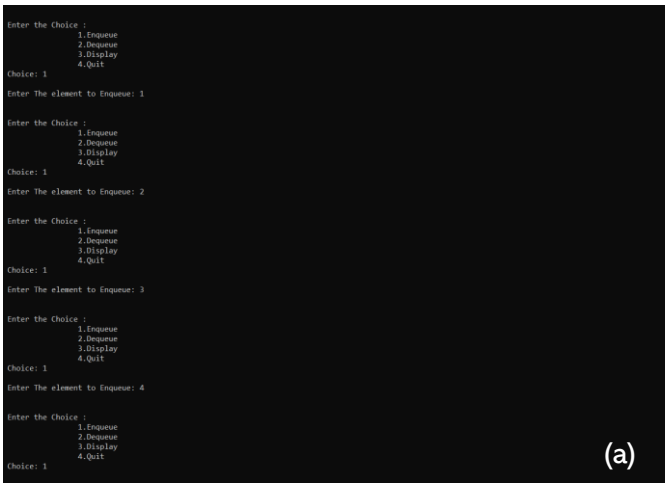
## Code:

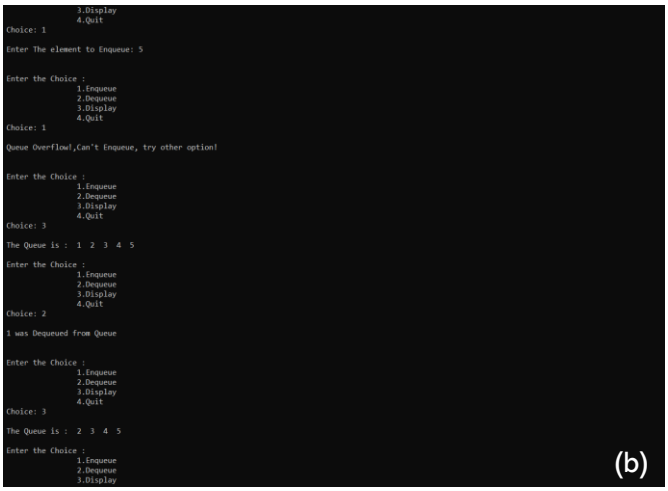
```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 5


void operation(int *,int *,int *);
void enqueue(int ,int *,int *);
int dequeue(int *,int *,int *);
void displayQueue(int *,int *,int *);

int main()
{
    int front=0,rear=-1;
    int queue[MAX];
    operation(queue,&rear,&front);
    return 0;
}

void operation(int *queue,int *rear,int *front)
{
     int choice,element;
    do
    {
        element=0;
        choice=0;
        printf("\n\n Enter the Choice : \n\t\t 1.Enqueue\n\t\t 2.Dequeue\n\t\t 3.Display\n\t\t 4.Quit\n Choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                if(*rear==MAX-1)
                  printf("\n Queue Overflow!,Can't Enqueue, try other option!\n");
                else
                  {
                      printf("\n Enter The element to Enqueue: ");
                      scanf("%d",&element);
                      enqueue(element,queue,rear);
                  }
                  break;
```

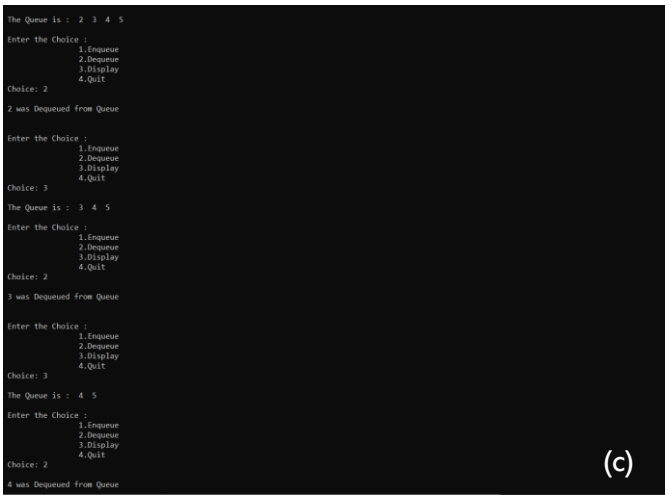
```c
            case 2:
                 if(*rear ==-1)
                   printf("\n Queue Underflow!,Can't Dequeue, try other option!\n");
                 else
                   printf("\n %d was Dequeued from Queue \n",dequeue(queue,rear,front));
                   break;
            case 3:
                   displayQueue(queue,rear,front);
                   break;
            case 4:
                   exit(0);
            default:
                   {
                       element=0;
                       printf("\n Input Error!, press 1 to try again ! :");
                       fflush(stdin);
                       scanf("%d",&element);
                       if(element==1)
                            operation(queue,rear,front);
                        else
                            exit(0);

                   }

        }

    }while(1);
}


void enqueue(int ele,int *queue,int *rear)
{
    queue[++(*rear)]=ele;
}


int dequeue(int *queue,int *rear,int *front)
{
    int ele=queue[*front];
    if((*rear)==(*front))
```

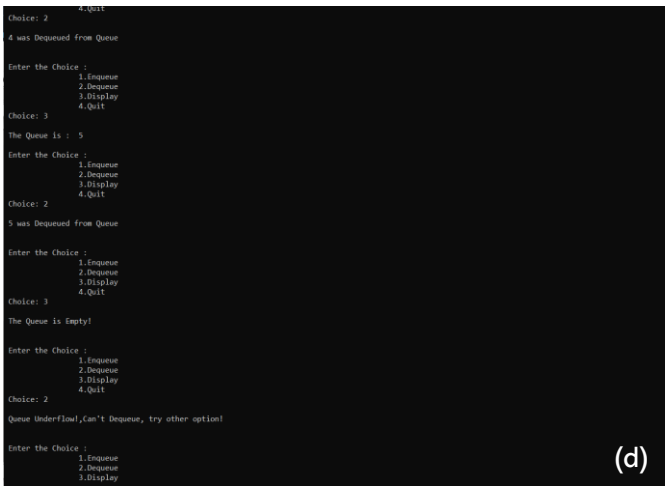
```c
    if((*rear)==(*front))
    {
        *rear=-1;
        *front=0;
    }
    else
        *front+=1;
    return ele;
}


void displayQueue(int *queue,int *rear,int *front)
{
   if(*rear==-1)
       printf("\n The Queue is Empty! \n");
    else
    {
        printf("\n The Queue is : ");
        for(int i=*front;i<=*rear;i++)
        printf(" %d ",queue[i]);
    }

}
```

# Output:


Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
Enter The element to Enqueue: 1
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
Enter The element to Enqueue: 2
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
Enter The element to Enqueue: 3
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
Enter The element to Enqueue: 4
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
(a)


3.Display
4.Quit
Choice: 1
Enter The element to Enqueue: 5
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
Queue Overflow!,Can't Enqueue, try other option!
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is : 1 2 3 4 5
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
1 was Dequeued from Queue
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is : 2 3 4 5
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
(b)


The Queue is : 2 3 4 5
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
2 was Dequeued from Queue
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is : 3 4 5
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
3 was Dequeued from Queue
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is : 4 5
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
4 was Dequeued from Queue
(c)


4.Quit
Choice: 2
4 was Dequeued from Queue
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is : 5
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
5 was Dequeued from Queue
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is Empty!
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
Queue Underflow!,Can't Dequeue, try other option!
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
(d)


Choice: 2
Queue Underflow!,Can't Dequeue, try other option!
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is Empty!
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
Queue Underflow!,Can't Dequeue, try other option!
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 4
Process returned 0 (0x0) execution time : 26.285 s
Press any key to continue.
(e)

# Program-4:
# (CircularQueue_Implementation)

## Code:

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

int front=-1;
int rear=-1;

int queue[MAX];

void Enque(int);
int Deque();
void display();
void operation();

int main()
{
    printf("\n <-----Circular Queue----->\n");
    operation();
    return 0;
}

void operation()
{
    int option=0;
    int item;
    do{
        printf("\n 1. Insert to Queue (EnQueue)");
        printf("\n 2. Delete from the Queue (DeQueue)");
        printf("\n 3. Display the content ");
        printf("\n 4. Exit \n");
        printf("\n Enter the option :");
        scanf("%d",&option);
        switch(option)
        {
            case 1: if(front==(rear+1)%MAX)
                    {
                        printf("\n Queue Overflow! Try other Options");
                        operation();
                    }
                    else
                    {
                        printf("\n Enter the element: ");
                        scanf("%d",&item);
                        Enque(item);
                        break;
                    }
            case 2: if(front==-1 && rear==-1)
                    {
                        printf("\n Queue Underflow, it is empty! Try other Options");
                        operation();
                    }
                    else
                    {
                        item=Deque();
                        printf("\n Removed element from the queue is %d \n",item);
                        break;
                    }
            case 3: display();
                    break;
            case 4: exit(0);

            default:printf("\n Input Error! Try Again \n");
                    operation();
        }

    } while (1);
}

void Enque(int ele)
{
        rear=(rear+1)%MAX;
        queue[rear]=ele;
        if(front ==-1)
        front=0;
}
int Deque()
{
    int item;
    item=queue[front];
    if(front==rear)
    {
        front=-1;
        rear=-1;
    }
    else
    {
        front=(front+1)%MAX;
    }
    return item;
}

void display()
{
    int i;
    if( front==-1 && rear==-1 )
    {
        printf("\n Queue is empty! Try other Options ");
        operation();
    }
    else
    {
        printf("\n Queue contents are: ");
        if(rear>front)
        {
            for(i=front;i!=rear;i=(i+1)%MAX)
            printf(" %d ", queue[i]);
            printf(" %d ", queue[i]);
        }
        else
            if(rear<front)
        {
            for(i=0;i<=rear;i++)
            printf(" %d ", queue[i]);
            for(i=front;i<MAX;i++)
            printf(" %d ", queue[i]);
        }

    }
}
```

# Output:

```
<-----Circular Queue----->

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :1

Enter the element: 1

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :1

Enter the element: 2

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :1

Enter the element: 3

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :1

Queue Overflow! Try other Options
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :3

Queue contents are:  1  2  3
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
```

(a)

```
Enter the option :3

Queue contents are:  1  2  3
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :2

Removed element from the queue is 1

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :3

Queue contents are:  2  3
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :2

Removed element from the queue is 2

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :3

Queue contents are:  3
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :2

Removed element from the queue is 3

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
```

(b)

```
Enter the option :2

Removed element from the queue is 3

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :3

Queue is empty! Try other Options
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :4

Process returned 0 (0x0)   execution time : 17.612 s
Press any key to continue.
```

(c)

# Program-5:
# (LinkedList_Implementation_Insertion)

## Code:

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

struct node
{
    int id;
    char name[20];
    int sem;
    struct node *next;
};

struct node *head=NULL;

void linkedList();
void insertNodeAtBegin();
void insertNodeAtEnd();
void insertNodeAtAny();
void displayList();

int size=0;

int main()
{
    linkedList();
    return 0;
}

void linkedList()
{
    int choice1,choice2;
    printf("\n <---Enter the Operation--->");
    printf("\n 1.Insert Node. \n 2.Display List \n 3.Exit.\n Choice: ");
    scanf("%d",&choice1);
    switch(choice1)
    {
        case 1: printf("\n 1.At the First Position \t 2.At End of list\t 3.At Any Specified Location\n Choice:");
                scanf("%d",&choice2);
                switch(choice2)
                {
                        case 1: insertNodeAtBegin();
                                break;
                        case 2: insertNodeAtEnd();
                                break;
                        case 3: insertNodeAtAny();
                                break;
                        default: printf("\n Input Error, try Again!\n ");
                                 linkedList();

                }
                break;
        case 2: displayList();
                break;
        case 3: exit(0);
        default:printf("\n Input error, Try again!!\n");
                linkedList();
    }
}

void insertNodeAtBegin()
{
    struct node *newnode;
    newnode=(struct node*)malloc(sizeof(struct node));
    printf("\n <--Enter the Details--> ");
    printf("\n ID: ");  scanf("%d",&(newnode->id));
    printf(" Name: ");    scanf("%s",(newnode->name));
    printf(" Sem: ");  scanf("%d",&(newnode->sem));
    newnode->next=head;
    head=newnode;
    size++;
    printf("\n Node created \n");
    linkedList();
}
void insertNodeAtEnd()
{
    struct node *newnode,*temp;
    newnode =(struct node *) malloc (sizeof(struct node));
    printf("\n <--Enter the Details--> ");
    printf("\n ID: ");  scanf("%d",&(newnode->id));
    printf(" Name: ");    scanf("%s",(newnode->name));
    printf(" Sem: ");  scanf("%d",&(newnode->sem));
    if (head==NULL)
    {
      newnode->next=NULL;
      head=newnode;
      printf("Node created\n");
      linkedList();
    }
    for(temp=head;(temp->next)!=NULL;temp=(temp->next));
    newnode->next=NULL;

    {
            temp->next=newnode;
            size++;
            printf("\n Node created  \n");
            linkedList();
    }

}
void insertNodeAtAny()
{
    struct node *newnode,*temp=head;
    newnode=(struct node*)malloc(sizeof(struct node));
    printf("\n <--Enter the Details--> ");
    printf("\n ID: ");  scanf("%d",&(newnode->id));
    printf(" Name: ");    scanf("%s",(newnode->name));
    printf(" Sem: ");  scanf("%d",&(newnode->sem));
    int pos=0,s=0;
            printf("\n Enter the position(pos>=1 and pos <%d) : ",size);
            scanf("%d",&pos);
            if(pos==0)
            {
                printf("\n Error position, check the operation menu!");
                linkedList();
            }

            for(temp;temp->next!=NULL;temp=temp->next)
            {
                if(s==(pos-1))
                {
```

```
                        newnode->next=(temp->next);
                        temp->next=newnode;
                        size++;
                        printf("\n Node created \n");
                        linkedlist();
                    }
                    s++;
                    temp=temp->next;
                }
    }

void displayList()
{
    if(head==NULL)
    {
        printf("\n Empty List!\n");
        linkedList();
    }
    printf("\n The List is :");
    for(struct node *temp=head;temp!=NULL;temp=temp->next)
    {
        printf("\n <---Student Details--->");
        printf("\n ID: %d ",temp->id);
        printf("\n Name: %s ",temp->name);
        printf("\n Sem: %d",temp->sem);
    }
    linkedList();
}

```

# Output:

```
<---Enter the Operation--->
1.Insert Node.
2.Display List
3.Exit.
Choice: 1

1.At the First Position         2.At End of list        3.At Any Specified Location
Choice:1

<--Enter the Details-->
ID: 1
Name: aaa
Sem: 1

Node created

<---Enter the Operation--->
1.Insert Node.
2.Display List
3.Exit.
Choice: 1

1.At the First Position         2.At End of list        3.At Any Specified Location
Choice:1

<--Enter the Details-->
ID: 2
Name: bbb
Sem: 2

Node created

<---Enter the Operation--->
1.Insert Node.
2.Display List
3.Exit.
Choice: 1

1.At the First Position         2.At End of list        3.At Any Specified Location
Choice:2

<--Enter the Details-->
ID: 3
Name: ccc
Sem: 3

Node created
```

(a)

```
<---Enter the Operation--->
1.Insert Node.
2.Display List
3.Exit.
Choice: 1

1.At the First Position         2.At End of list        3.At Any Specified Location
Choice:1

<--Enter the Details-->
ID: 4
Name: ddd
Sem: 4

Node created

<---Enter the Operation--->
1.Insert Node.
2.Display List
3.Exit.
Choice: 2

The List is :
<---Student Details--->
ID: 4
Name: ddd
Sem: 4
<---Student Details--->
ID: 2
Name: bbb
Sem: 2
<---Student Details--->
ID: 1
Name: aaa
Sem: 1
<---Student Details--->
ID: 3
Name: ccc
Sem: 3
<---Enter the Operation--->
1.Insert Node.
2.Display List
3.Exit.
Choice: 1
```

(b)

```
1.At the First Position         2.At End of list        3.At Any Specified Location
Choice:3

<--Enter the Details-->
ID: 5
Name: eee
Sem: 5

Enter the position(pos>=1 and pos <4) : 2

Node created

<---Enter the Operation--->
1.Insert Node.
2.Display List
3.Exit.
Choice: 2

The List is :
<---Student Details--->
ID: 4
Name: ddd
Sem: 4
<---Student Details--->
ID: 2
Name: bbb
Sem: 2
<---Student Details--->
ID: 1
Name: aaa
Sem: 1
<---Student Details--->
ID: 5
Name: eee
Sem: 5
<---Student Details--->
ID: 3
Name: ccc
Sem: 3
<---Enter the Operation--->
1.Insert Node.
2.Display List
3.Exit.
Choice: 3

Process returned 0 (0x0)   execution time : 156.461 s
Press any key to continue.
```

(c)

# Program-6:

## (LinkedList_Implementation_Deletion)

### Code:

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

struct node
{
    int id;
    char name[20];
    int sem;
    struct node *next;
};

struct node *head=NULL;

void linkedList();
void insertNode();
void deleteNode(int);
void deleteNodeAtBegin();
void deleteNodeAtEnd();
void deleteNodeOFGiven();
void displayList();

int size=0;

int main()
{
    linkedList();
    return 0;
}

void linkedList()
{
    int choice1,choice2;
    printf("\n\n <---Enter the Operation--->");
    printf("\n 1.Insert Node. \n 2.Delete Node \n 3.Display List\n 4.Exit.\n Choice: ");
    scanf("%d",&choice1);
    switch(choice1)
    {
        case 1:
                insertNode();
                break;
        case 2:
```

```c
        case 2:
                printf("\n 1.At the First Position \t 2.At End of list\t 3.At Given Element Position\n Choice:");
                scanf("%d",&choice2);
                switch(choice2)
                {
                        case 1: deleteNode(1);
                                break;
                        case 2: deleteNode(2);
                                break;
                        case 3: deleteNode(3);
                                break;
                        default: printf("\n Input Error, try Again!\n ");
                                 linkedList();

                }
                break;

        case 3: displayList();
                break;

        case 4: exit(0);

        default: printf("\n Input error, Try again!!\n");
                 linkedList();
    }
}

void insertNode()
{
    struct node *newnode,*temp;
    newnode =(struct node *) malloc (sizeof(struct node));
    printf("\n <--Enter the Details--> ");
    printf("\n ID: ");  scanf("%d",&(newnode->id));
    printf(" Name: ");    scanf("%s",(newnode->name));
    printf(" Sem: ");  scanf("%d",&(newnode->sem));
    if (head==NULL)
     {
       newnode->next=NULL;
       head=newnode;
       printf("\n Node created\n");
```

```c
       linkedList();
       size++;
     }
    for(temp=head;(temp->next)!=NULL;temp=(temp->next));
    newnode->next=NULL;

    {
            temp->next=newnode;
            size++;
            printf("\n Node created  \n");
            linkedList();
    }

}

void deleteNode(int flag)
{
    if(head==NULL)
    {
        printf("\n The List is Empty! \n");
        linkedList();
    }
    else
    if(head->next==NULL)
    {
         printf("\n Node Deleted. \n Now List Is empty! ");
         free(head);
         head=NULL;
         linkedList();
    }
    else
    {
        switch(flag)
        {
            case 1: deleteNodeAtBegin();
                    break;
            case 2: deleteNodeAtEnd();
                    break;
            case 3: deleteNodeOFGiven();
```

```
            case 3: deleteNodeOFGiven();
                    break;
        }

        linkedList();
    }
}

void deleteNodeAtBegin()
{
    struct node* temp=head;
    head=head->next;
    free(temp);
    printf("\n Node Deleted.");
}
void deleteNodeAtEnd()
{
    struct node* temp=head;
    for(temp;(temp->next)->next!=NULL;temp=temp->next);
    free(temp->next);
    temp->next=NULL;
    printf("\n Node Deleted.");
}
void deleteNodeOFGiven()
{
     struct node* temp1=head,*temp2=temp1;
     int ele;
     printf("\n Enter the Element :");
     scanf("%d",&ele);
     for(temp1;temp1->next!=NULL;temp1=temp1->next)
     {
         temp2=temp1->next;
         if(temp2->id==ele)
         {
             temp1->next=temp2->next;
             free(temp2);
             printf("\n Node Deleted.");
             return;
         }
         if(temp1->id==ele)
```

```
         if(temp1->id==ele)
         {
             head=temp2;
             free(temp1);
             printf("\n Node Deleted.");
             return;
         }
     }
     printf("\n Element Not Present In List! ");
     linkedList();
}

void displayList()
{
   if(head==NULL)
   {
       printf("\n Empty List!\n");
       linkedList();
   }
   printf("\n The List is :");
   for(struct node *temp=head;temp!=NULL;temp=temp->next)
       {
        printf("\n\n <---Student Details--->");
        printf("\n ID: %d ",temp->id);
        printf("\n Name: %s ",temp->name);
        printf("\n Sem: %d",temp->sem);
         printf("\n <---------x--------->");
       }
      linkedList();
}
```

# Output:

```
<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 1

<--Enter the Details-->
ID: 1
Name: 1
Sem: 1

Node created


<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 1

<--Enter the Details-->
ID: 2
Name: 2
Sem: 2

Node created


<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 1

<--Enter the Details-->
ID: 3
Name: 3
Sem: 3

Node created


<---Enter the Operation--->
1.Insert Node.
```

(a)

```
Sem: 3

Node created


<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 1

<--Enter the Details-->
ID: 4
Name: 4
Sem: 4

Node created


<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 3

The List is :

<---Student Details--->
ID: 1
Name: 1
Sem: 1
<---------x--------->

<---Student Details--->
ID: 2
Name: 2
Sem: 2
<---------x--------->

<---Student Details--->
ID: 3
Name: 3
Sem: 3
<---------x--------->

<---Student Details--->
ID: 4
```

(b)

```
ID: 4
Name: 4
Sem: 4
<----------x---------->

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 2

1.At the First Position         2.At End of list        3.At Given Element Position
Choice:1

Node Deleted.

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 3

The List is :

<---Student Details--->
ID: 2
Name: 2
Sem: 2
<----------x---------->

<---Student Details--->
ID: 3
Name: 3
Sem: 3
<----------x---------->

<---Student Details--->
ID: 4
Name: 4
Sem: 4
<----------x---------->

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
```

(c)

```
4.Exit.
Choice: 2

1.At the First Position         2.At End of list        3.At Given Element Position
Choice:2

Node Deleted.

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 3

The List is :

<---Student Details--->
ID: 2
Name: 2
Sem: 2
<----------x---------->

<---Student Details--->
ID: 3
Name: 3
Sem: 3
<----------x---------->

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 2

1.At the First Position         2.At End of list        3.At Given Element Position
Choice:3

Enter the Element :4

Element Not Present In List!

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 2
```

(d)

```
4.Exit.
Choice: 2

1.At the First Position         2.At End of list        3.At Given Element Position
Choice:3

Enter the Element :3

Node Deleted.

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 3

The List is :

<---Student Details--->
ID: 2
Name: 2
Sem: 2
<----------x---------->
```

(e)

# Program-7:
# (LinkedList_Operations)

## Code:

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

struct node
{
    int data;
    struct node *next;
};

struct node *head1=NULL,*head2=NULL;

void linkedList();
int choiceFlag();
void insert(int);
void display(int);
void reverse(int);
void sorting(int);
void swap(struct node*,struct node*);
void concate();

int main()
{
    linkedList();
    return 1;
}

void linkedList()
{
    int choice1,choice2;
    printf("\n 1. Insert  \n 2. Display  \n 3. Reverse \n 4. Sort \n 5. Concatinate and display \n 6. Exit");
    printf("\n Enter your choice : ");
    scanf("%d",&choice1);
    switch(choice1)
    {
        case 1: choice2=choiceFlag();insert(choice2); break;
        case 2: choice2=choiceFlag();display(choice2);break;
        case 3: choice2=choiceFlag();reverse(choice2);break;
        case 4: choice2=choiceFlag();sorting(choice2);break;
        case 5: concate();break;
                break;
        case 6:exit(0);
        default:printf("\n Try Again!\n");
    }
   linkedList();
}

int choiceFlag()
{
    int flag=0;
    do{
    printf("\n 1.First List\n 2.Second List\n Choice: ");
    scanf("%d",&flag);
    if(flag!=1 && flag!=2)
        printf("\n Error choice, Try again! ");
    }
    while(flag!=1 && flag!=2);

    return flag;
}

void insert(int flag)
{
    struct node *newnode,*temp,**head;
    int item;
    newnode =(struct node *) malloc (sizeof(struct node));
    printf(" Enter the data : ");
    scanf("%d",&item);
    newnode->data=item;
    newnode->next=NULL;

    if(flag==1)
    head=&head1;
    else
    head=&head2;


    if (*head==NULL)
     {
       *head=newnode;
       printf("\n First Node created\n");
     }
     else
    {
        temp=*head;

        while(temp->next!=NULL)
        {
                temp=temp->next;
        }
        temp->next=newnode;

        printf("\n Next Node created\n");
    }

}
void reverse(int flag )
{

    struct node **head;

    if(flag==1)
    head=&head1;
    else
    head=&head2;

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

    struct node *pre=NULL,*forw=NULL,*cur=*head;

    while(cur!=NULL)
    {
        forw=cur->next;
        cur->next=pre;
        pre=cur;
        cur=forw;
```

```
        cur=forw;
    }
    *head=pre;
    printf("\n Reversed! \n");
}
void sorting(int flag )
{
    struct node **head;
    int swapFlag;
    if(flag==1)
    head=&head1;
    else
    head=&head2;

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

    struct node *temp1=*head,*temp2=NULL;
    do
    {
        temp1 = *head;
        swapFlag = 0;
        while (temp1->next!=temp2)
        {
            if (temp1->data > (temp1->next)->data)
            {
                swap(temp1, temp1->next);
                swapFlag=1;
            }
            temp1=temp1->next;
        }
        temp2=temp1;
    }
    while (swapFlag);
    printf("\n Sorted! \n");
}
```

```
void swap(struct node *t1, struct node *t2)
{
    int temp = t1->data;
    t1->data = t2->data;
    t2->data = temp;
}

void concate()
{
    struct node *temp1=head1,*temp2=head2;

    while(temp1->next!=NULL)
    temp1=temp1->next;
    temp1->next=temp2;
    display(0);
}

void display(int flag)
{
    struct node *ptr;
    if(flag==1)
    ptr=head1;
    else
    if(flag==2)
    ptr=head2;
    else
    ptr=head1;

    if(ptr==NULL)
    {
        printf("\n List Empty! \n");
    }
    else
    {
        printf("\n");
        while(ptr!=NULL)
        {
        printf(" %d ",ptr->data);
        ptr=ptr->next;
        }
    }
```

# Output:

```
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 1
Enter the data : 1

First Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 1
Enter the data : 2

Next Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 1
Enter the data : 3

Next Node created

1. Insert
2. Display
3. Reverse
```

(a)

```
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 2
Enter the data : 9

Next Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 2

1.First List
2.Second List
Choice: 1

1  2  3  4
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 2

1.First List
2.Second List
Choice: 2

6  2  9
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 3

1.First List
2.Second List
```

(b)

```
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 1
Enter the data : 4

Next Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 2
Enter the data : 6

First Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 2
Enter the data : 2

Next Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
```

(c)

```
2.Second List
Choice: 1

Reversed!

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 2

1.First List
2.Second List
Choice: 1

4  3  2  1
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 4

1.First List
2.Second List
Choice: 1

Sorted!

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 5

1  2  3  4  6  2  9
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 4
```

(d)

```
5. Concatinate and display
6. Exit
Enter your choice : 5

1  2  3  4  6  2  9
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 4

1.First List
2.Second List
Choice: 1

Sorted!

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 2

1.First List
2.Second List
Choice: 1

1  2  2  3  4  6  9
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatinate and display
6. Exit
Enter your choice : 6

Process returned 0 (0x0)   execution time : 69.901
Press any key to continue.
```

(d)

# Program-8:
## (LinkedList_Stack-Queue_Implementation)

### Code:

### Stack:

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

struct node *top=NULL;

struct node
{
    int data;
    struct node *next;
};

void stack();
void push(int);
int pop();
void display();

int main()
{
    stack();
    return 1;
}

void stack()
{
    int choice=0,ele=0;
    do
    {
        printf("\n Enter the choice:\n 1.Push.\n 2.Pop.\n 3.Display.\n 4.Exit\n Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
                printf("\n Enter the element to Push: ");
                scanf("%d",&ele);
                push(ele);
                break;
        case 2:
                ele=pop();
                printf("\n %d was deleted from Queue ",ele);
                break;
        case 3:
                display();
                break;
        case 4:
                exit(0);
        default:
                printf("\n Input Error Try Again! ");
                stack();
        }
    }while(1);
}

void push(int ele)
{
    struct node *newnode;
    newnode =(struct node *) malloc (sizeof(struct node));
    newnode->data=ele;
    newnode->next=NULL;
    if(top==NULL)
    {
        top=newnode;
    }
    else
    {
        newnode->next=top;
        top=newnode;
    }

    printf("\n Element %d was inserted!\n ",ele);

}
int pop()
{
    int ele;
    struct node *temp;
    if(top==NULL)
    {
        printf("\n Stack UnderFlow, The Stack is empty! \n");
        stack();
    }

    ele=top->data;
    temp=top;

    if(top->next==NULL)
    {
        top=NULL;
    }
    else
    {
        top=top->next;
    }
    free(temp);
    return ele;
}

void display()
{
    struct node *i;
    if(top==NULL)
    {
        printf("\n Stack is empty!\n ");
        stack();
    }
    printf("\n The Stack Contains:\n TOP->");
    for(i=top;i!=NULL;i=i->next)
        printf("\t%d ",i->data);

}
```

### Queue:

```c
#include<stdio.h>
#include<stdlib.h>
struct node *front=NULL, *rear=NULL;

struct node
{
    int data;
    struct node *next;
};

void queue();
void insNode(int);
int delNode();
void display();

int main()
{
    queue();

    return 1;
}

void queue()
{
    int choice=0,ele=0;
    do
    {
        printf("\n Enter the choice:\n 1.Insert.\n 2.Delete.\n 3.Display.\n 4.Exit\n Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
                printf("\n Enter the element to Insert: ");
                scanf("%d",&ele);
                insNode(ele);
                break;
        case 2:
                ele=delNode();
                printf("\n %d was deleted from Queue ",ele);
                break;
        case 3:
                display();
                break;
        case 4:
                exit(0);
        default:
                printf("\n Input Error Try Again! ");
                queue();
        }
    }while(1);
}

void insNode(int ele)
{
    struct node *newnode;
    newnode =(struct node *) malloc (sizeof(struct node));
    newnode->data=ele;
    newnode->next=NULL;
    if(rear==NULL)
    {
        front=newnode;
        rear=newnode;
    }
    else
    {
        rear->next=newnode;
        rear=newnode;
    }

    printf("\n Element %d was inserted!\n ",ele);

}
int delNode()
{
    int ele;
    struct node *temp;
    if(front==NULL)
    {
        printf("\n Queue UnderFlow, The queue is empty! \n");
        queue();
    }

    ele=front->data;
    temp=front;

    if(front==rear)
    {
        front=NULL;
        rear=NULL;
    }
    else
    {
        front=front->next;
    }
    free(temp);
    return ele;
}

void display()
{
    struct node *i;
    if(front==NULL && rear==NULL)
    {
        printf("\n Queue is empty!\n ");
        queue();
    }
    printf("\n The Queue Contains:");
    for(i=front;i!=NULL;i=i->next)
        printf("%d ",i->data);

}
```

# Output:

## Stack:

```
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:1

Enter the element to Push: 1

Element 1 was inserted!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:1

Enter the element to Push: 2

Element 2 was inserted!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:1

Enter the element to Push: 3

Element 3 was inserted!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:1

Enter the element to Push: 4

Element 4 was inserted!

Enter the choice:
1.Push.
2.Pop.
```

(a)

```
Element 4 was inserted!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:1

Enter the element to Push: 5

Element 5 was inserted!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:3

The Stack Contains:
TOP->  5       4       3       2       1
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

5 was deleted from Queue
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

4 was deleted from Queue
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

3 was deleted from Queue
Enter the choice:
```

(b)

```
3 was deleted from Queue
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

2 was deleted from Queue
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

1 was deleted from Queue
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

Stack UnderFlow, The Stack is empty!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:3

Stack is empty!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:4

Process returned 0 (0x0)   execution time : 24.445 s
Press any key to continue.
```

(c)

## Queue:

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

Enter the element to Insert: 1

Element 1 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:1

Enter the element to Insert: 2

Element 2 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:1

Enter the element to Insert: 3

Element 3 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:1

Enter the element to Insert: 4

Element 4 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
```

(a)

```
Element 4 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:1

Enter the element to Insert: 5

Element 5 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:3

The Queue Contains:1 2 3 4 5
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:2

1 was deleted from Queue
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:2

2 was deleted from Queue
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:3

The Queue Contains:3 4 5
Enter the choice:
1.Insert.
2.Delete.
```

(b)

```
2.Delete.
3.Display.
4.Exit
Choice:2

3 was deleted from Queue
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:2

4 was deleted from Queue
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:2

5 was deleted from Queue
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:2

Queue UnderFlow, The queue is empty!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:3

Queue is empty!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:4

Process returned 0 (0x0)   execution time : 29.284 s
Press any key to continue.
```

(c)

# Program-9:
# (Queue_Implementation)

## Code:

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

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

Node *head=NULL;

void doublyLinkedList();
void insertNode(int);
void insertNodeToLeft();
void insertNodeToRight();
void deleteSpecifiedValue();
void displayList();

int main()
{
    doublyLinkedList();
    return 0;
}

void doublyLinkedList()
{
    int choice=0;
    printf("\n <--Doubly Linked List-->");
    printf("\n 1.Enter Node\n 2.Enter Node to Left\n 3.Enter Node to Right\n 4.Delete A Node\n 5.DisplayList\n 6.Exit\n Choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1: insertNode(0);
                break;
        case 2: insertNode(1);
                break;
        case 3: insertNode(2);
                break;
        case 4: deleteSpecifiedValue();
                break;
```

```c
                break;
        case 5: displayList();
                break;
        case 6: exit(0);

        default: printf("\n Error choice, Try Again! ");
                 doublyLinkedList();
    }
    doublyLinkedList();
}

void insertNode(int flag)
{
    Node *newnode;
    newnode=(Node*)malloc(sizeof(Node));
    printf("\n Enter the Element: ");
    scanf("%d",&newnode->data);
    if(head==NULL)
    {
        head=newnode;
        newnode->next=NULL;
        newnode->prev=NULL;
        printf("\n First Node created \n");
        doublyLinkedList();
    }
    if(flag==0)
    {
         Node *temp=head;
         for(temp;(temp->next)!=NULL;temp=temp->next);
         temp->next=newnode;
         newnode->prev=temp;
         newnode->next=NULL;
    }
    else
        if(flag==1)
        insertNodeToLeft(newnode);
    else
        insertNodeToRight(newnode);
}
void insertNodeToRight(Node *tempNew)
```

```c
{
    int ele;
    char choice;
    printf("\n Enter the Node element To who's right you want to Insert Node: ");
    scanf("%d",&ele);
    Node *temp=head;
    for(temp;temp!=NULL;temp=temp->next)
        {
            if(temp->data==ele)
            {
                  if(temp->next!=NULL)
                  {
                        tempNew->next=temp->next;
                        tempNew->prev=temp;
                        (temp->next)->prev=tempNew;
                        temp->next=tempNew;
                        printf("\n Node created \n");
                        doublyLinkedList();
                    }
                    else
                    {
                        tempNew->next=NULL;
                        tempNew->prev=temp;
                        temp->next=tempNew;
                        printf("\n Node created \n");
                        doublyLinkedList();
                    }

            }
        }
    printf("\n The given Element was not found! ,press Y to Try again! or press anything to exit: ");
    fflush(stdin);
    scanf("%c",&choice);
    if(choice=='Y' || choice =='y')
    insertNodeToRight(tempNew);
    else
    {
        free(tempNew);
        printf("\n Node creation Failed \n");
```

```
        printf("\n Node creation Failed \n");
        doublyLinkedList();
    }

}
void insertNodeToLeft(Node *tempNew)
{
    int ele;
    char choice;
    printf("\n Enter the Node element To who's left you want to Insert Node: ");
    scanf("%d",&ele);
    Node *temp=head;
    if(head->data==ele)
    {
        tempNew->next=head;
        tempNew->prev=NULL;
        head=tempNew;
        printf("\n Node created \n");
        doublyLinkedList();
    }
    for(temp;temp!=NULL;temp=temp->next)
        {
            if(temp->data==ele)
            {
                    tempNew->next=temp;
                    tempNew->prev=temp->prev;
                    (temp->prev)->next=tempNew;
                    temp->prev=tempNew;
                    printf("\n Node created \n");
                    doublyLinkedList();

            }
        }
    printf("\n The given Element was not found! ,press Y to Try again! or press anything to exit: ");
    fflush(stdin);
    scanf("%c",&choice);
    if(choice=='Y' || choice =='y')
    insertNodeToLeft(tempNew);
    else
    {
```

```
        free(tempNew);
        printf("\n Node creation Failed \n");
        doublyLinkedList();
    }

}
void deleteSpecifiedValue()
{
    if(head==NULL)
    {
        printf("\n Empty List!\n");
        doublyLinkedList();
    }
    int ele;
    printf("\n Enter the Node element to delete: ");
    scanf("%d",&ele);
    Node *temp=head;
    if(head->next==NULL)
    {
        if(head->data==ele)
        {
            free(temp);
            head=NULL;
            printf("\n Node Deleted. \n Now List Is empty! ");
            doublyLinkedList();
        }
    }
     else
     {
         for(temp;temp!=NULL;temp=temp->next)
        {
            if(temp->data==ele)
            {
                    if(temp->next==NULL)
                    {
                        (temp->prev)->next=NULL;
                        free(temp);
                        printf("\n Node Deleted \n");
                        doublyLinkedList();
```

```
                    doublyLinkedList();
                }
                if(temp->prev==NULL)
                {
                    (temp->next)->prev=NULL;
                    head=head->next;
                    printf("\n Node Deleted \n");
                    doublyLinkedList();
                }
                else
                {
                    (temp->prev)->next=temp->next;
                    (temp->next)->prev=temp->prev;
                    free(temp);
                    printf("\n Node Deleted \n");
                    doublyLinkedList();
                }
            }
        }
     }
     printf("\n The given element %d is not present in list!\n ",ele);
}

void displayList()
{
    if(head==NULL)
    {
        printf("\n Empty List!\n");
        doublyLinkedList();
    }
    Node *temp=head;
    printf("\n The List Contains :");
    for(temp;temp!=NULL;temp=temp->next)
    {
        printf(" %d ",temp->data);
       }
    doublyLinkedList();
}
```

# Output:

```
<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 1

Enter the Element: 1

First Node created

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 1

Enter the Element: 2

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 1

Enter the Element: 3

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 5

The List Contains : 1  2  3
<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 2
```

(a)

```
Enter the Element: 10

Enter the Node element To who's left you want to Insert Node: 1

Node created

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 5

The List Contains : 10  1  2  3
<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 3

Enter the Element: 4

Enter the Node element To who's right you want to Insert Node: 2

Node created

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 5

The List Contains : 10  1  2  4  3
<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 4

Enter the Node element to delete: 10

Node Deleted
```

(b)

```
Enter the Node element to delete: 10

Node Deleted

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 5

The List Contains : 1  2  4  3
<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 6

Process returned 0 (0x0)   execution time : 38.214 s
Press any key to continue.
```

(c)

# Program-10:

## (Binary Search Tree_Implementation)

### Code:

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

typedef struct node
{
    int data;
    struct node *left;
    struct node *right;
}Node;
void tree();
Node * create();
Node *insert(Node *,Node *);
void traverse();
void preOrder(Node *);
void inOrder(Node *);
void postOrder(Node *);
void display(Node *,int);

 Node *root;

int main()
{
    tree();
    return 0;
}
void tree()
{
    int choice;
    printf("\n <--Binary Search Tree-->\n 1.Insert Element\n 2.Traverse-All methods\n 3.Display BST\n 4.Exit\n Choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1: insert(root,create()); break;
        case 2: traverse(); break;
        case 3: if(root==NULL)
                printf("\n Tree is Empty!");
                else
                display(root,0);
                break;
        case 4: exit(0);break;
        default: printf("\n Error Choice !\n ");
```

```c
        default: printf("\n Error Choice !\n ");
                 tree();
    }
    tree();
}
Node * create()
{
    Node* newnode=(Node *)malloc(sizeof(Node));
    printf("\n Enter the Element: ");
    scanf("%d",&newnode->data);
    newnode->left=NULL;
    newnode->right=NULL;
    return newnode;
}
Node * insert(Node *Root,Node *newNode)
{
    if(root==NULL)
    {
        root=newNode;
        printf("\n Root Node Created ");
    }
    else
    {
    if(newNode->data>Root->data)
    {
        if(Root->right==NULL)
        {
            Root->right=newNode;
        }
        else
        insert(Root->right,newNode);
    }
    else
    if(newNode->data<Root->data)
    {
        if(Root->left==NULL)
        {
            Root->left=newNode;
        }
        else
```

```c
        else
        insert(Root->left,newNode);
    }
    }
}

void traverse()
{
    if(root==NULL)
    {
        printf("\n The Tree Is Empty! ");
        return;
    }
    printf("\n Pre-Order Traverse: ");
    preOrder(root);
    printf("\n In-Order Traverse: ");
    inOrder(root);
    printf("\n Post-Order Traverse: ");
    postOrder(root);
}
void preOrder(Node *Root)
{
    if(Root!=NULL){
    printf(" %d ",Root->data);
    preOrder(Root->left);
    preOrder(Root->right);
    }
}
void inOrder(Node *Root)
{
    if(Root!=NULL){
    inOrder(Root->left);
    printf(" %d ",Root->data);
    inOrder(Root->right);
    }
}
void postOrder(Node *Root)
{
    if(Root!=NULL){
    postOrder(Root->left);
```

```
120         postOrder(Root->right);
121         printf(" %d ",Root->data);
122         }
123     }
124     void display(Node* root,int i)
125     {
126       int j;
127       if(root!=NULL)
128       {
129         display(root->right,i+1);
130         for(j=0;j<i;j++)
131           printf("----");
132         printf("%d\n",root->data);
133         display(root->left,i+1);
134       }
135     }
```

# Output:

```
<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 100

Root Node Created
<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 150

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 50

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 30

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
```

(a)

```
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 60

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 170

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 3
--------170
----150
100
--------60
----50
--------30

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 2

Pre-Order Traverse:  100  50  30  60  150  170
In-Order Traverse:  30  50  60  100  150  170
Post-Order Traverse:  30  60  50  170  150  100
```

(b)