**AIM:**  To implement doubly linked list.

There are two cases in doubly linked list. They are

1. Insertion
2. Insertion in beginning
3. Insertion in middle
4. Insertion in the end
5. Deletion
6. Deletion in beginning
7. Deletion in middle
8. Deletion in the end

**CASE 1:**

**ALGORITHM:**

InsertBeg(List, e)

**Step 1** : Start.

**Step 2** : Set NewNode = addressof(Node).

**Step 3** : If List = NULL, then goto Step 4 else goto Step 8.

**Step 4** : Set NewNode -> Element = e.

**Step 5** : Set NewNode -> Next = NULL.

**Step 6 :** Set NewNode -> Prev = List.

**Step 7 :** Set List ->Next = NewNode and goto Step 13.

**Step 8** : Set NewNode -> Element = e.

**Step 9** : Set NewNode -> Next = List -> Next.

**Step 10:** Set NewNode ->Next -> Prev = NewNode.

**Step 11**: Set NewNode -> Prev = List.

**Step 12**: Set List -> Next = NewNode.

**Step 13:** Stop.

InsertMid(List, p, e)

**Step 1** : Start.

**Step 2 :** Set NewNode = addressof(Node).

**Step 3** : Set Position = Find(List, p).

**Step 4 :** Set NewNode -> Element = e.

**Step 5 :** Set NewNode ->Next = Position -> Next.

**Step 6 :** Set Position -> Next -> Prev = NewNode.

**Step 7** : Set Position -> Next = NewNode.

**Step 8** : Set NewNode -> Prev = Position.

**Step 9 :** Stop.

InsertLast(List, e)

**Step 1 :** Start.

**Step 2 :** Set NewNode = addressof(Node).

**Step 3 :** If List = NULL, then goto Step 4 else goto Step 8.

**Step 4** : Set NewNode -> Element = e.

**Step 5 :** Set NewNode -> Next = NULL.

**Step 6** : Set NewNode ->Prev = List.

**Step 7 :** Set List -> Next = NewNode and goto Step 15.

**Step 8 :** Set Position = List.

**Step 9** : Repeat the Step 10 until Position -> Next != NULL.

**Step 10:** Set Position = Position -> Next.

**Step 11**: Set NewNode -> Element = e.

**Step 12**: Set Position -> Next = NewNode.

**Step 13:** Set NewNode ->Prev = Position.

**Step 14**: Set NewNode ->Next = NULL.

**Step 15:** Stop.

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

struct node{

struct node \*prev;

int data;

struct node \*next;

};

typedef struct node Node;

int isempty(Node\*);

void insertbeginning(Node\*);

void insertmiddle(Node\*);

void insertlast(Node\*);

void traverse(Node\*);

int main()

{

Node \*head=malloc(sizeof(Node));

int choice;

do{

printf("\n1.Insert beginning node \n2.Insert middle node \n3.Insert last node \n4.traverse ");

printf("Enter your choice ");

scanf("%d", &choice);

switch(choice){

case 1:

insertbeginning(head);

traverse(head->next);

break;

case 2:

insertmiddle(head);

traverse(head->next);

break;

case 3:

insertlast(head);

traverse(head->next);

break;

case 4:

traverse(head->next);

break;

}

}while(choice<=4);

return 0;

}

int isempty(Node \*temp)

{

if(temp->next==NULL)

return 1;

else

return 0;

}

void insertbeginning(Node\*temp){

Node \*newnode=malloc(sizeof(Node));

printf("Enter the data ");

scanf("%d", &newnode->data);

if(!isempty(temp))

{

newnode->next=temp->next;

newnode->prev=temp;

temp->next->prev=newnode;

temp->next=newnode;

}

else

{

temp->next=newnode;

newnode->prev=temp;

newnode->next=NULL;

}

}

void insertmiddle(Node \*temp){

Node \*newnode=malloc(sizeof(Node));

printf("Enter the data ");

scanf("%d", &newnode->data);

int key;

printf("Enter the data after which you want to add the new node ");

scanf("%d", &key);

while(temp->next!=NULL&&temp->data!=key){

temp=temp->next;

}

newnode->next=temp->next;

newnode->prev=temp;

temp->next->prev=newnode;

temp->next=newnode;

}

void insertlast(Node \*temp){

Node \*newnode=malloc(sizeof(Node));

printf("Enter the data ");

scanf("%d", &newnode->data);

while(temp->next!=NULL){

temp=temp->next;

}

newnode->next=NULL;

newnode->prev=temp;

temp->next=newnode;

}

void traverse(Node \*temp){

while(temp!=NULL){

printf("%d->", temp->data);

temp=temp->next;

}

printf("NULL");

}

**OUTPUT FOR THE ABOVE PROGRAM:**

1.Insert beginning node

2.Insert middle node

3.Insert last node

4.traverse Enter your choice 1

Enter the data 22

22->NULL

1.Insert beginning node

2.Insert middle node

3.Insert last node

4.traverse Enter your choice 1

Enter the data 33

33->22->NULL

1.Insert beginning node

2.Insert middle node

3.Insert last node

4.traverse Enter your choice 1

Enter the data 44

44->33->22->NULL

1.Insert beginning node

2.Insert middle node

3.Insert last node

4.traverse Enter your choice 2

Enter the data 55

Enter the data after which you want to add the new node 33

44->33->55->22->NULL

1.Insert beginning node

2.Insert middle node

3.Insert last node

4.traverse Enter your choice 3

Enter the data 66

44->33->55->22->66->NULL

1.Insert beginning node

2.Insert middle node

3.Insert last node

4.traverse Enter your choice 4

44->33->55->22->66->NULL

**CASE-2:**

**ALGORITHM:**

DelBeg(List, e)

**Step 1** : Start.

**Step 2** : If !IsEmpty = True, then goto Step 3 else goto Step 9.

**Step 3** : Set TempNode = List -> Next.

**Step 4 :** Set List -> Next = TempNode -> Next.

**Step 5** : If List -> Next != NULL, then goto Step 6 else goto Step 7.

**Step 6** : Set TempNode -> Next -> Prev = List.

**Step 7** : Display the TempNode -> Element.

**Step 8** : Delete TempNode and goto Step 10.

**Step 9** : Display “List is Empty”.

**Step 10:** Stop.

DeleteMid(List, e)

**Step 1** : Start.

**Step 2** : If !IsEmpty = True, then goto Step 3 else goto Step 10.

**Step 3** : Set Position = Find (List, e).

**Step 4** : If !Islast(Position) = True, then goto Step 5 else goto Step 11.

**Step 5** : Set TempNode = Position.

**Step6**:Set Position–>Prev->Next = Position->Next. **Step 7 :**SetPosition->Next->Prev = Position->Prev.

**Step 8** : Display the TempNode->Element.

**Step 9** : Delete TempNode and goto Step 11.

**Step 10**: Display “List is Empty”.

**Step 11:** Stop.

DeleteEnd(List)

**Step 1** : Start.

**Step 2** : If !IsEmpty = True, then goto Step 3 else goto Step 10.

**Step 3** : Set Position = List.

**Step 4** : Repeat the Step 5 until Position->Next != NULL.

**Step 5** : Set Position = Position->Next.

**Step 6** : Set TempNode = Position.

**Step 7** : Set Position->Prev->Next = NULL.

**Step 8** : Display the TempNode->Element.

**Step 9** : Delete TempNode and goto Step 11.

**Step 10:** Display “List is Empty”.

**Step 11**: Stop.

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

struct node{

struct node \*prev;

int data;

struct node \*next;

};

typedef struct node Node;

int isempty(Node\*);

void create(Node\*);

void deletebeginning(Node\*);

void deletemiddle(Node\*);

void deletelast(Node\*);

void traverse(Node\*);

int main()

{

Node \*head=malloc(sizeof(Node));

int choice;

do{

printf("\n1. Creation\n2.Delete beginning node \n3.Delete middle node \n4.Delete last node \n5.traverse \n");

printf("Enter your choice ");

scanf("%d", &choice);

switch(choice){

case 1:

create(head);

traverse(head->next);

break;

case 2:

deletebeginning(head);

traverse(head->next);

break;

case 3:

deletemiddle(head);

traverse(head->next);

break;

case 4:

deletelast(head);

traverse(head->next);

break;

case 5:

traverse(head->next);

break;

}

}while(choice<=4);

return 0;

}

int isempty(Node \*temp)

{

if(temp->next==NULL)

return 1;

else

return 0;

}

void create(Node\*temp){

Node \*newnode=malloc(sizeof(Node));

printf("Enter the data ");

scanf("%d", &newnode->data);

if(!isempty(temp))

{

newnode->next=temp->next;

newnode->prev=temp;

temp->next->prev=newnode;

temp->next=newnode;

}

else

{

temp->next=newnode;

newnode->prev=temp;

newnode->next=NULL;

}

}

void deletebeginning(Node \*temp){

if(!isempty(temp))

{

temp=temp->next;

temp->prev->next=temp->next;

temp->next->prev=temp->prev;

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

free(temp);

}

else

{

printf("The DLL is empty");

}

}

void deletemiddle(Node \*temp){

Node \*pos;

int key;

printf("Enter the key ");

scanf("%d", &key);

while(temp->next!=NULL&&temp->data!=key){

pos=temp;

temp=temp->next;

}

pos->next=temp->next;

temp->next->prev=pos;

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

free(temp);

}

void deletelast(Node \*temp){

while(temp->next!=NULL){

temp=temp->next;

}

temp->prev->next=NULL;

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

free(temp);

}

void traverse(Node \*temp){

while(temp!=NULL){

printf("%d->", temp->data);

temp=temp->next;

}

printf("NULL");

}

**OUTPUT FOR THE ABOVE PROGRAM:**

1. Creation

2.Delete beginning node

3.Delete middle node

4.Delete last node

5.traverse

Enter your choice 1

Enter the data 22

22->NULL

1. Creation

2.Delete beginning node

3.Delete middle node

4.Delete last node

5.traverse

Enter your choice 1

Enter the data 33

33->22->NULL

1. Creation

2.Delete beginning node

3.Delete middle node

4.Delete last node

5.traverse

Enter your choice 1

Enter the data 44

44->33->22->NULL

1. Creation

2.Delete beginning node

3.Delete middle node

4.Delete last node

5.traverse

Enter your choice 2

The deleted node is 44

33->22->NULL

1. Creation

2.Delete beginning node

3.Delete middle node

4.Delete last node

5.traverse

Enter your choice 3

Enter the key 33

The deleted node is 33

22->NULL

1. Creation

2.Delete beginning node

3.Delete middle node

4.Delete last node

5.traverse

Enter your choice 1

Enter the data 66

66->22->NULL

1. Creation

2.Delete beginning node

3.Delete middle node

4.Delete last node

5.traverse

Enter your choice 4

The deleted node is 22

66->NULL

1. Creation

2.Delete beginning node

3.Delete middle node

4.Delete last node

5.traverse

Enter your choice 5

66->NULL

**RESULT:** Hence, it is implemented using doubly linked list.