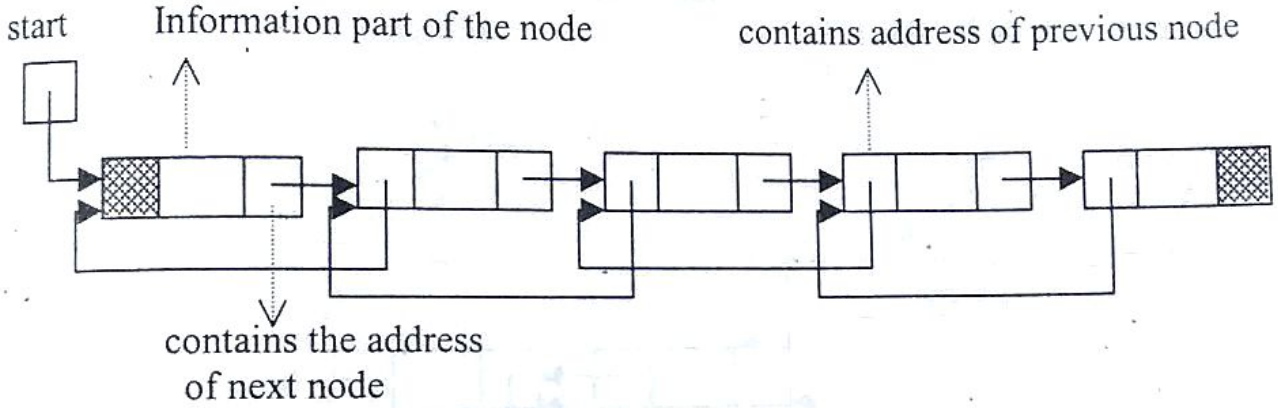
Double linked list-

We have seen in single linked list we can traverse only in one direction because each node has address of next node only. Suppose we are in the middle of linked list and we want to do operation with just previous node then we have no way to go on previous node, we will again traverse from starting node. So this is a drawback of single linked list. We implement another data structure doubly linked list, in this each node has address of previous and next node also. The data structure for doubly linked list will be as-

```
struct node{
        struct node *previous;
        int info;
        struct node *next;
      }
```

Here struct node *previous is a pointer to structure, which will contain the address of previous node and struct node *next will contain the address of next node in the list. So we can traverse in both directions at any time.

The only drawback is that each node has to contain the address information of previous node.



Traversing a doubly linked List-

In doubly linked list info is the information part, next is the address of the next node and prev is the address of previous node. We take ptr as a pointer variable. Here start points to the first element of list. Initially we assign the value of start to ptr. So ptr also points to the first node of list. For processing the next element we assign the address of next node to ptr as –

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

Now ptr has the address of next node. We can traverse each element of list through this assignment until ptr has NULL value which is next part value of last element. So the doubly linked list can be traversed as-

```
while( ptr != NULL )
        ptr=ptr->next;
```

Insertion into a doubly linked List-

Insertion in a doubly linked list may be possible in two ways-

1. Insertion at begining
2. Insertion in between

Case 1-

start points to the first node of doubly linked list. For insertion at beginning, we assign the value of start to the next part of inserted node and address of inserted node to the prev part of start as –

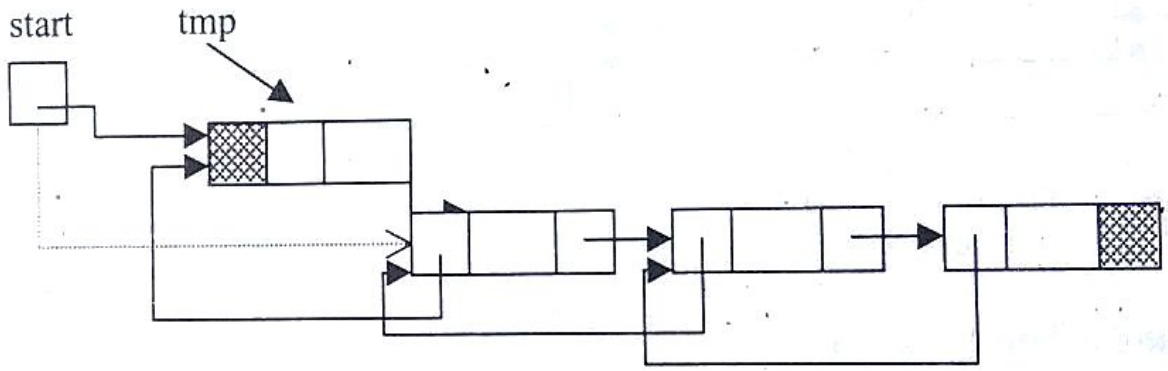
```
tmp->next=start;
start->prev = tmp;
```

Now inserted node points to the next node, which was beginning node of the doubly linked list and prev part of second node will point to the new inserted node. Now inserted node is the first node of the doubly linked list. So start will be reassigned as-

```
start= tmp;
```

Now start will point to the inserted node which is first node of the doubly linked list. Assign NULL to prev part of inserted node since now it will become the first node and prev part of first node is NULL

```
tmp->prev=NULL;
```

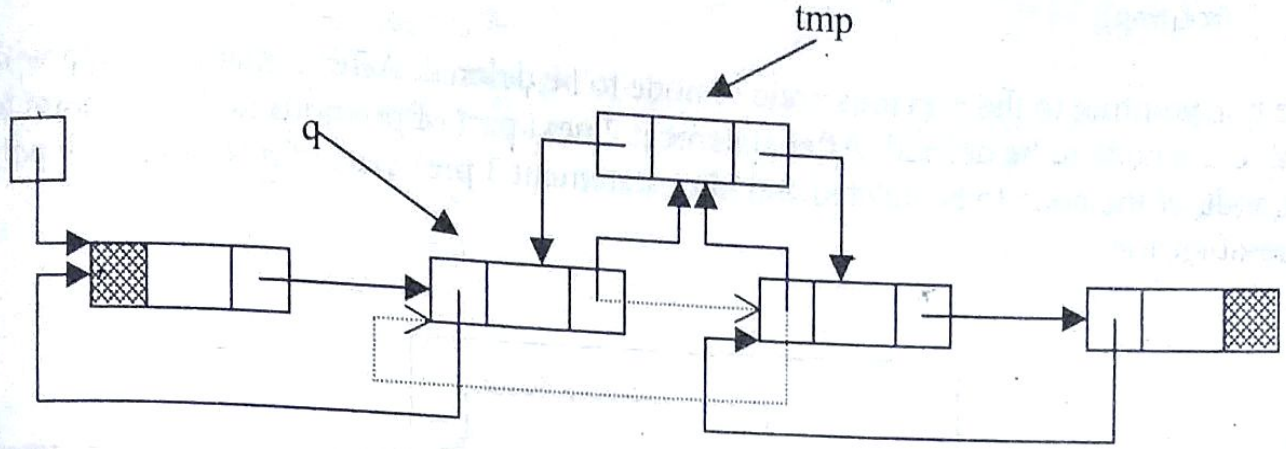


Case 2-

First we traverse the doubly linked list for obtaining the node after which we want to insert the element. For inserting the element after the node we assign the address of inserted node to the prev part of next node. Then we assign the next part of previous node to the next part of inserted node. Address of previous node will be assigned to prev part of inserted node and address of inserted node will be assigned to next part of previous node.

```
q->next->prev=tmp;
tmp->next=q->next;
tmp->prev=q;
q->next=tmp;
```

Here q points to the previous node (after that new node will be inserted), After statement 1, prev part of next node will point to inserted node. After statement 2, next part of inserted node will point to next node. After statement 3, prev part of inserted node will point to it's previous node and after statement 4, next part of previous node will point to inserted node.



Deletion from doubly linked list -

For deleting the node from a doubly linked list ,first we traverse the linked list and compare with each element. After finding the element there may be three cases for deletion-

1. Deletion at beginning
2. Deletion in between
3. Deletion of last node

Case 1-

Here start points to the first node of doubly linked list. If node to be deleted is the first node of list then we assign the value of start to tmp as-

```
tmp = start;
```

Now we assign the next part of deleted node to start as-

```
start=start->next;
```

Since start points to the first node of linked list, so start->next will point to the second node of list. Then NULL will be assigned to start->prev. Now we should free the node to be deleted which is pointed by tmp.

```
free( tmp );
```

So the whole process for deletion of first node of doubly linked list will be-

```
tmp=start;
start=start->next;  /*first element deleted*/
start->prev = NULL;
free(tmp);
```

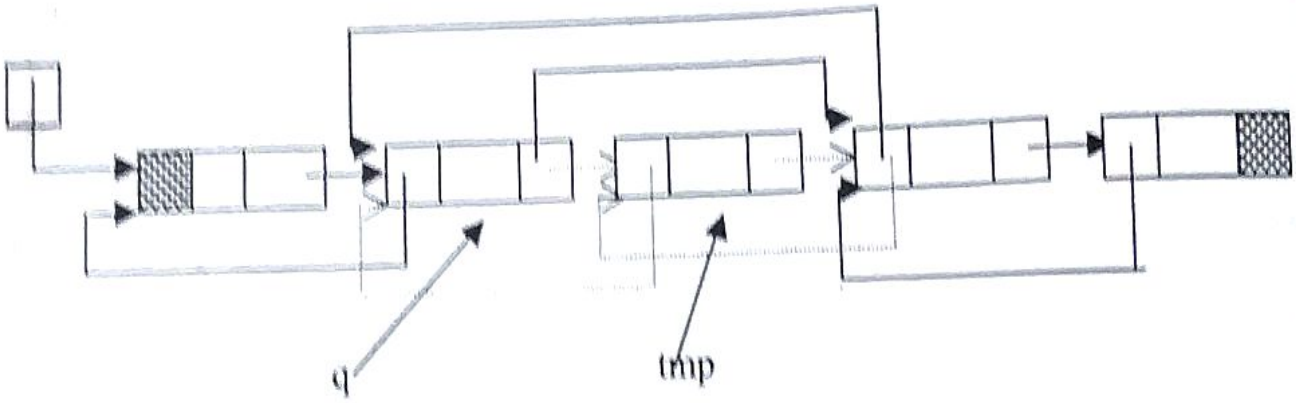
Case 2-

If the element is other than the first element of linked list then we assign the next part of the deleted node to the next part of the previous node and address of the previous node to prev part of next node. This can be as-

```
tmp=q->next;
q->next=tmp->next;
```

```
tmp->next->prev=q;
free(tmp);
```

Here q is pointing to the previous node of node to be deleted. After statement 1 tmp will point to the node to be deleted. After statement 2 next part of previous node will point to next node of the node to be deleted and after statement 3 prev part of next node will point to previous node.



Case 3-

If node to be deleted is last node of doubly linked list then we will just free the last node and next part of second last node will be NULL.

```
tmp=q->next;
free(tmp);
q->next=NULL;
```

Here q is pointing to the previous node of node to be deleted. After statement 1 tmp will point to the node to be deleted. After statement 2 last node will be deleted and after statement 3 second last node will become the last node of list.

```
/* Program of double linked list*/
# include <stdio.h>
# include <malloc.h>

struct node
{
        struct node *prev;
        int info;
        struct node *next;
}*start;

main( )
{
        int choice,n,m,po,i;
        start=NULL;
```

```
while(1)
{
        printf("1.Create List\n");
        printf("2.Add at begining\n");
        printf("3.Add after\n");
        printf("4.Delete\n");
        printf("5.Display\n");
        printf("6.Count\n");
        printf("7.Reverse\n");
        printf("8.exit\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
         case 1:
                printf("How many nodes you want : ");
                scanf("%d",&n);
                for(i=0;i<n;i++)
                {
                        printf("Enter the element : ");
                        scanf("%d",&m);
                        create_list(m);
                }
                break;
        case 2:
                printf("Enter the element : ");
                scanf("%d",&m);
                addatbeg(m);
                break;
        case 3:
                printf("Enter the element : ");
                scanf("%d",&m);
                printf("Enter the position after which this element is inserted : ");
                scanf("%d",&po);
                addafter(m,po);
                break;
        case 4:
                printf("Enter the element for deletion : ");
                scanf("%d",&m);
                del(m);
                break;
        case 5:
                display( );
                break;
        case 6:
                count( );
                break;
        case 7:
                rev( );
```

```
                        break;
                case 8:
                        exit( );
                default:
                        printf("Wrong choice\n");
        }/*End of switch*/
  }/*End of while*/
}/*End of main( )*/

create_list(int num)
{
        struct node *q,*tmp;
        tmp= malloc(sizeof(struct node));
        tmp->info=num;
        tmp->next=NULL;
        if(start==NULL)
        {
                tmp->prev=NULL;
                start->prev=tmp;
                start=tmp;
        }
        else
        {
                q=start;
                while(q->next!=NULL)
                        q=q->next;
                q->next=tmp;
                tmp->prev=q;
        }
}/*End of create_list( )*/

addatbeg(int num)
{
        struct node *tmp;
        tmp=malloc(sizeof(struct node));
        tmp->prev=NULL;
        tmp->info=num;
        tmp->next=start;
        start->prev=tmp;
        start=tmp;
}/*End of addatbeg( )*/

addafter(int num,int c)
{
        struct node *tmp,*q;
        int i;
        q=start;
```

```
        for(i=0;i<c-1;i++)
        {
                q=q->next;
                if(q==NULL)
                {
                        printf("There are less than %d elements\n",c);
                        return;
                }
        }
        tmp=malloc(sizeof(struct node) );
        tmp->info=num;
        q->next->prev=tmp;
        tmp->next=q->next;
        tmp->prev=q;
        q->next=tmp;
}/*End of addafter( ) */

del(int num)
{
        struct node *tmp,*q;
        if(start->info==num)
        {
                tmp=start;
                start=start->next;  /*first element deleted*/
                start->prev = NULL;
                free(tmp);
                return;
        }
        q=start;
        while(q->next->next!=NULL)
        {
                if(q->next->info==num)    /*Element deleted in between*/
                {
                        tmp=q->next;
                        q->next=tmp->next;
                        tmp->next->prev=q;
                        free(tmp);
                        return;
                }
                q=q->next;
        }
        if(q->next->info==num)    /*last element deleted*/
        {       tmp=q->next;
                free(tmp);
                q->next=NULL;
                return;
        }
        printf("Element %d not found\n",num);
}/*End of del( )*/
```

```
display( )
{
        struct node *q;
        if(start==NULL)
        {
                printf("List is empty\n");
                return;
        }
        q=start;
        printf("List is :\n");
        while(q!=NULL)
        {
                printf("%d ", q->info);
                q=q->next;
        }
        printf("\n");
}/*End of display( ) */

count( )
{       struct node *q=start;
        int cnt=0;
        while(q!=NULL)
        {
                q=q->next;
                cnt++;
        }
        printf("Number of elements are %d\n",cnt);
}/*End of count( )*/

rev( )
{
        struct node *p1,*p2;
        p1=start;
        p2=p1->next;
        p1->next=NULL;
        p1->prev=p2;
        while(p2!=NULL)
        {
                p2->prev=p2->next;
                p2->next=p1;
                p1=p2;
                p2=p2->prev; /*next of p2 changed to prev */
        }
        start=p1;
}/*End of rev( )*/
```