Sri SAI RAM ENGINEERING COLLEGE

INSTITUTE OF TECHNOLOGY

West Tambaram, Chennai - 44

YEAR	SEM
II	III

CS 8391

DATASTRUCTURES (COMMON TO CSE &IT)

UNIT No.1

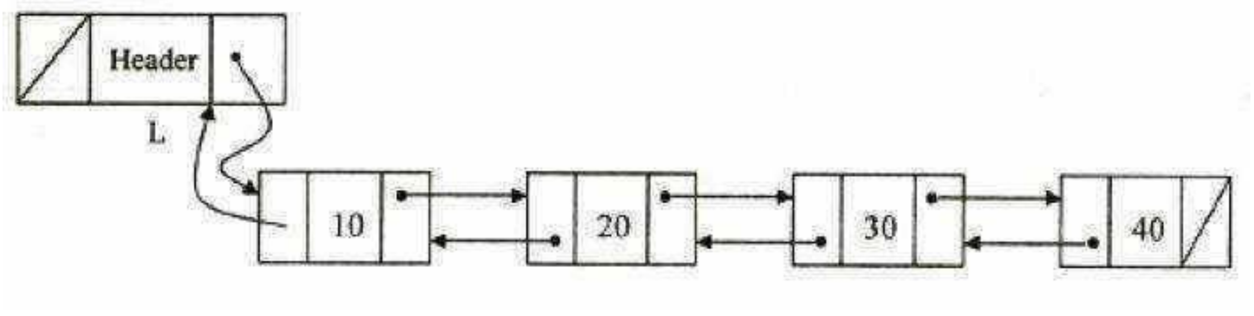
LINEAR DATA STRUCTURES – LIST

1.5 DOUBLY-LINKED LIST

DOUBLY-LINKED LIST

A doubly linked list is a linked list in which each node has three fields namely Data, Next, Prev.

- Data-This field stores the value of the element
- Next-This field points to the successor node in the list
- Prev-This field points to the predecessor node in the list



DOUBLY LINKED LISTS-DECLARATION

- The PREV field of the first node and the NEXT field of the last node will contain NULL.
- The PREV field is used to store the address of the preceding node, which enables us to traverse the list in the backward direction.
- A doubly linked list provides the ease to manipulate the elements of the list as it maintains pointers to nodes in both the directions (forward and backward).
- Advantage
  doubly linked list is that it makes searching twice as efficient

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

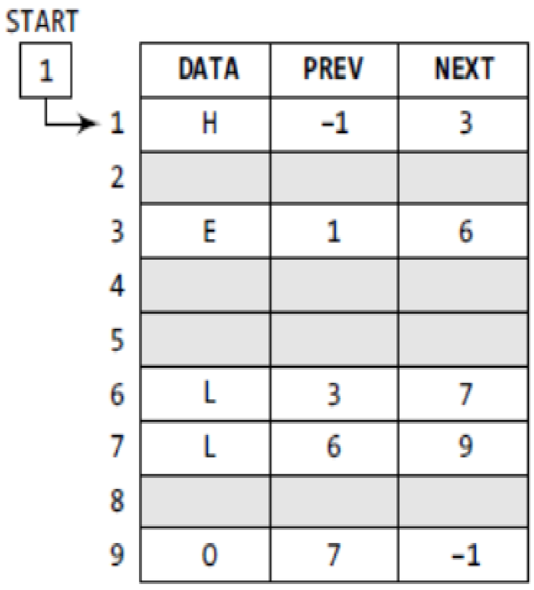
Basic operations of a doubly -linked list are:

- Insert – Inserts a new element at the end of the list.
- Delete – Deletes any node from the list.
- Find – Finds any node in the list.
- Print – Prints the list

Declaration of DLL Node

```
typedef struct node *position ; struct
node
{
int data; position
prev; position
next;
};
```

- Let us view how a doubly linked list is maintained in the memory. Consider Fig. In the figure, we see that a variable START is used to store the address of the first node.
- In this example, START = 1, so the first data is stored at address 1, whichis
H. Since this is the first node, it has no previous node and hence stores NULL or –1 in the PREV field.
- We will traverse the list until we reach a position where the NEXT entry contains –1 or NULL. This denotes the end of the linked list. When we traverse the DATA and NEXT in this manner, we will finally see that the linked list in the above example stores characters that when put together form the word HELLO



	DATA	PREV	NEXT
1	H	-1	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	O	7	-1

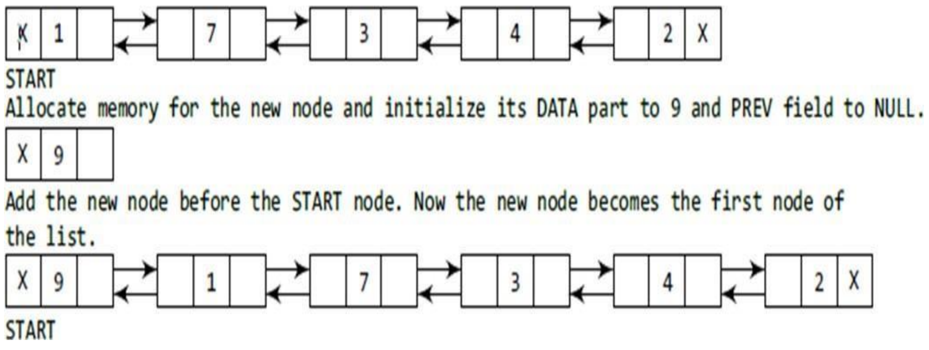
Inserting a New Node in a Doubly Linked List

In this section, we will discuss how a new node is added into an already existing doubly linked list.

Case 1: The new node is inserted at the beginning. Case 2: The new node is inserted at the end.
Case 3: The new node is inserted after a given node. Case 4: The new node is inserted before a given node.

Inserting a Node at the Beginning of a Doubly Linked List

Consider the doubly linked list shown in Fig. Suppose we want to add a new node with data 9 as the first node of the list. Then the following changes will be done in the linked list.


X 1 7 3 4 2 X
START
Allocate memory for the new node and initialize its DATA part to 9 and PREV field to NULL.
X 9
Add the new node before the START node. Now the new node becomes the first node of the list.
X 9 1 7 3 4 2 X
START

Inserting a Node at the Beginning of a Doubly Linked List

Algorithm to insert a new node at the beginning of a doubly linked list.

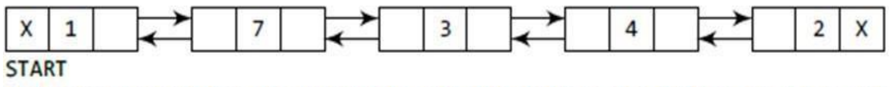
- In Step 1, we first check whether memory is available for the new node. If the free memory has exhausted, then an OVERFLOW message is printed.
- Otherwise, if free memory cell is available, then we allocate space for the new node. Set its DATA part with the given VAL and the NEXT part is initialized with the address of the first node of the list, which is stored in START.
- Now, since the new node is added as the first node of the list, it will now be known as the START node, that is, the START pointer variable will now hold the address of NEW_NODE.

Routine - Inserting a Node at the Beginning of a Doubly Linked List

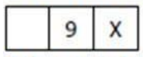
```
struct node *insert_beg(struct node *start)
{
        struct node *new_node; int
        num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node)); new_node ->
        data = num;
        start -> prev = new_node;
        new_node -> next = start;
        new_node -> prev = NULL; start =
        new_node;
        return start;
}
```

Inserting a Node at the end of a Doubly Linked List

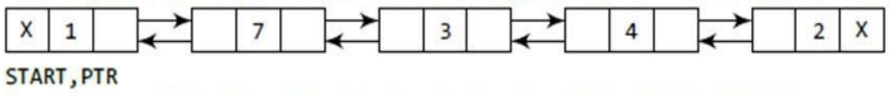
- Suppose we want to add a new node with data 9 as the last node of the list. Then the following changes will be done in the linked list.



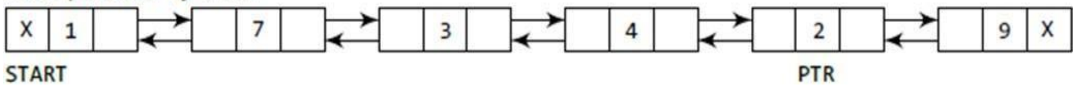
Allocate memory for the new node and initialize its DATA part to 9 and its NEXT field to NULL.



Take a pointer variable PTR and make it point to the first node of the list.



Move PTR so that it points to the last node of the list. Add the new node after the node pointed by PTR.



Algorithm to insert a new node at the end of a doubly linked list.

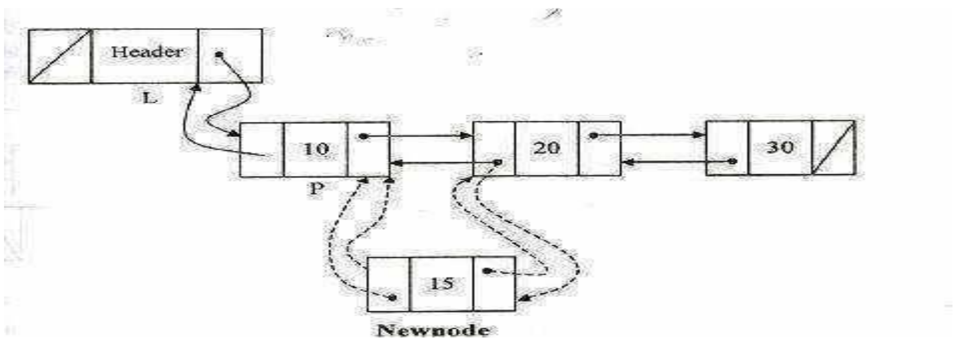
- In Step 6, we take a pointer variable PTR and initialize it with START. In the while loop, we traverse through the linked list to reach the last node. Once we reach the last node,

- In Step 9, we change the NEXT pointer of the last node to store the address of the new node. Remember that the NEXT field of the new node contains NULL which signifies the end of the linked list.
- The PREV field of the NEW_NODE will be set so that it points to the node pointed by PTR (now the second last node of the list).

```
struct node *insert_end(struct node *start)
{
        struct node *ptr, *new_node; int
        num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node)); new_node ->
        data = num;
        ptr=start;
        while(ptr -> next != NULL) ptr =
        ptr -> next;
        ptr -> next = new_node;
        new_node -> prev = ptr;
        new_node -> next = NULL; return
        start;
}
```

Routine to insert an element in a DLL any position :

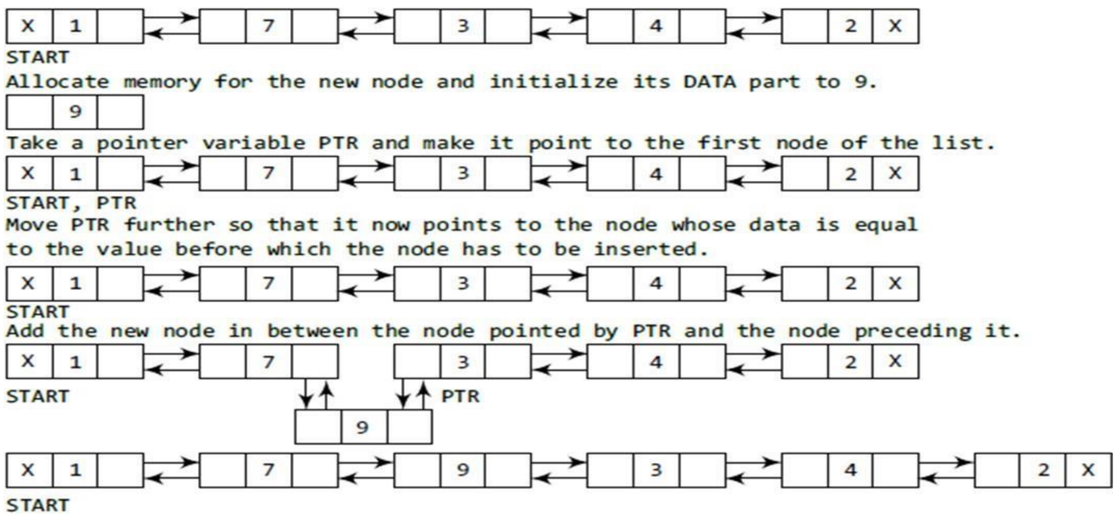
```
void Insert (int x, list L, position P)
{
            struct node *Newnode;
            Newnode = (struc node*)malloc (sizeof(struct node)); if (Newnode!
            = NULL)
            Newnode->data= X;
             Newnode ->next= P ->next; P->next ->prev=Newnode P ->next =
             Newnode;
            Newnode ->prev = P:
}
```



Inserting a Node Before a Given Node in a Doubly Linked List

Consider the doubly linked list shown in Fig. Suppose we want to add a new node with value 9 before the node containing 3. Before discussing the changes that will be done in the linked list.

- In Step 1, we first check whether memory is available for the new node.
- In Step 5, we take a pointer variable PTR and initialize it with START.
- That is, PTR now points to the first node of the linked list.
- In the while loop, we traverse through the linked list to reach the node that has its value equal to NUM.
- We need to reach this node because the new node will be inserted before this node.
- Once we reach this node, we change the NEXT and PREV fields in such a way that the new node is inserted before the desired node.


X 1 7 3 4 2 X
START
Allocate memory for the new node and initialize its DATA part to 9.
9
Take a pointer variable PTR and make it point to the first node of the list.
X 1 7 3 4 2 X
START, PTR
Move PTR further so that it now points to the node whose data is equal to the value before which the node has to be inserted.
X 1 7 3 4 2 X
START
Add the new node in between the node pointed by PTR and the node preceding it.
X 1 7 3 4 2 X
START
PTR
9
X 1 7 9 3 4 2 X
START

```
struct node *insert_before(struct node *start)
{
        struct node *new_node, *ptr;                int num, val;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        printf("\n Enter the value before which the data has to be inserted : ");
        scanf("%d", &val);
        new_node = (struct node *)malloc(sizeof(struct node)); new_node ->
        data = num;
        ptr = start;
        while(ptr -> data != val) ptr =
        ptr -> next; new_node -> next
        = ptr;
        new_node -> prev = ptr-> prev; ptr ->
        prev -> next = new_node; ptr -> prev =
        new_node;
        return start;
}
```

Deleting a Node from a Doubly Linked List

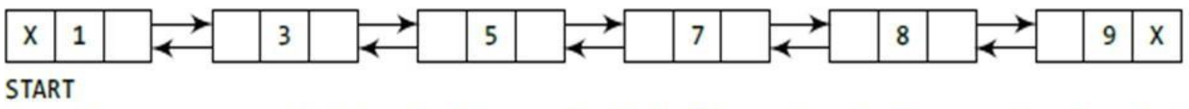
A node is deleted from an already existing doubly linked list.

Case 1: The first node is deleted. Case 2:
The last node is deleted.

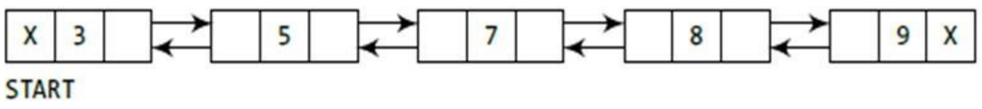
Case 3: The node after a given node is deleted

Deleting the First Node from a Doubly Linked List

When we want to delete a node from the beginning of the list, then the following changes will be done in the linked list.



Free the memory occupied by the first node of the list and make the second node of the list as the START node.



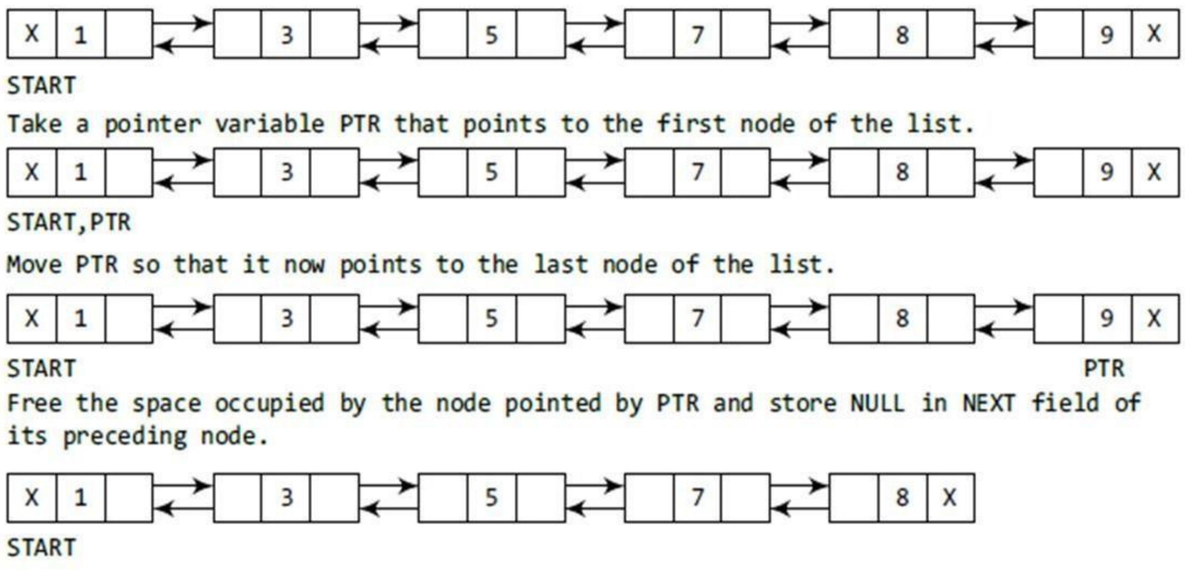
Algorithm to delete the first node of a doubly linked list.

- In Step 1 of the algorithm, we check if the linked list exists or not. If START = NULL, then it signifies that there are no nodes in the list and the control is transferred to the last statement of the algorithm.
- If there are nodes in the linked list, then we use a temporary pointer variable PTR that is set to point to the first node of the list. initialize PTR with START that stores the address of the first node of the list.
- In Step 3, START is made to point to the next node in sequence and finally the memory occupied by PTR (initially the first node of the list) is freed and returned to the free pool.

```
struct node *delete_beg(struct node *start)
{
        struct node *ptr; ptr
        = start;
        start = start -> next; start -
        > prev = NULL; free(ptr);
        return start;
}
```

Deleting the Last Node from a Doubly Linked List

Consider the doubly linked list shown in Fig. Suppose we want to delete the last node from the linked list, then the following changes will be done in the linked list.


X 1 3 5 7 8 9 X
START
Take a pointer variable PTR that points to the first node of the list.
X 1 3 5 7 8 9 X
START,PTR
Move PTR so that it now points to the last node of the list.
X 1 3 5 7 8 9 X
START
PTR
Free the space occupied by the node pointed by PTR and store NULL in NEXT field of its preceding node.
X 1 3 5 7 8 X
START

Algorithm to delete the last node of a doubly linked list

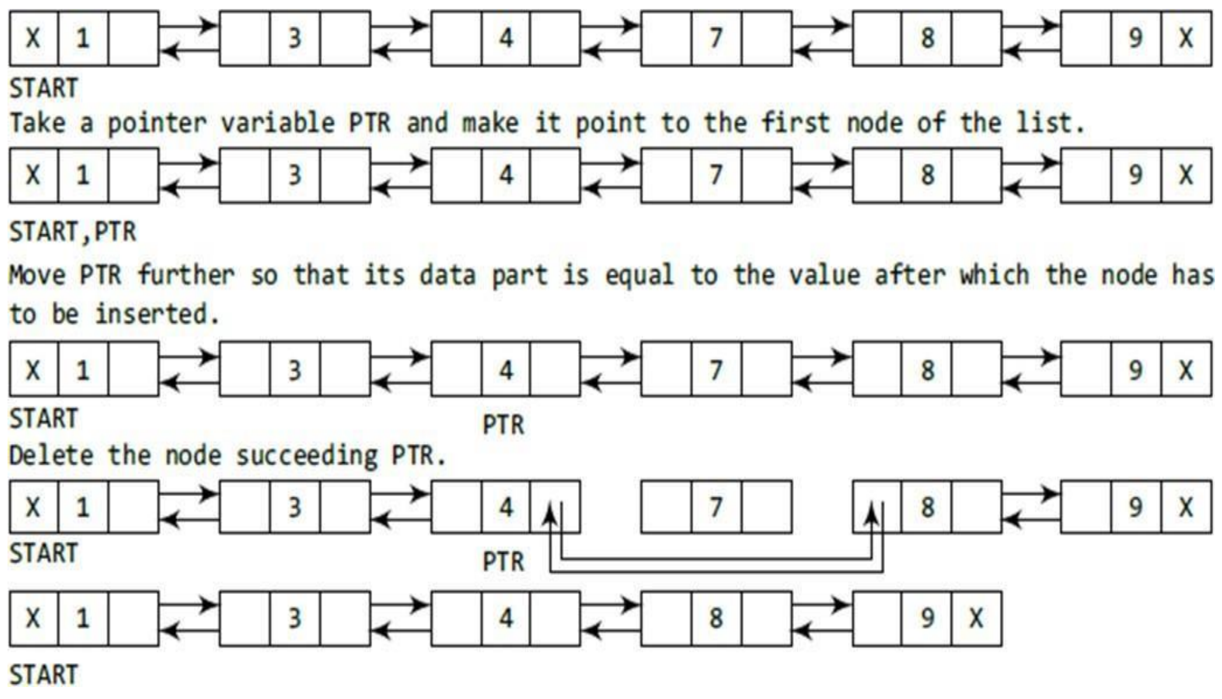
- In Step 2, we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the linked list. The while loop traverses through the list to reach the last node. Once we reach the last node, we can also access the second last node by taking its address from the PREV field of the last node.
- To delete the last node, we simply have to set the next field of second last node to NULL, so that it now becomes the (new) last node of the linked list. The memory of the previous last node is freed and returned to the free pool.

Deleting the Last Node from a Doubly Linked List

```
struct node *delete_end(struct node *start)
{
    struct node *ptr; ptr
    = start;
    while(ptr -> next != NULL) ptr =
    ptr -> next;
    ptr -> prev -> next = NULL;
    free(ptr);
    return start;
}
```

Deleting the Node After a Given Node in a Doubly Linked List

Consider the doubly linked list shown in Fig. Suppose we want to delete the node that succeeds the node which contains data value 4. Then the following changes will be done in the linked list.



Take a pointer variable PTR and make it point to the first node of the list.

Move PTR further so that its data part is equal to the value after which the node has to be inserted.

Delete the node succeeding PTR.

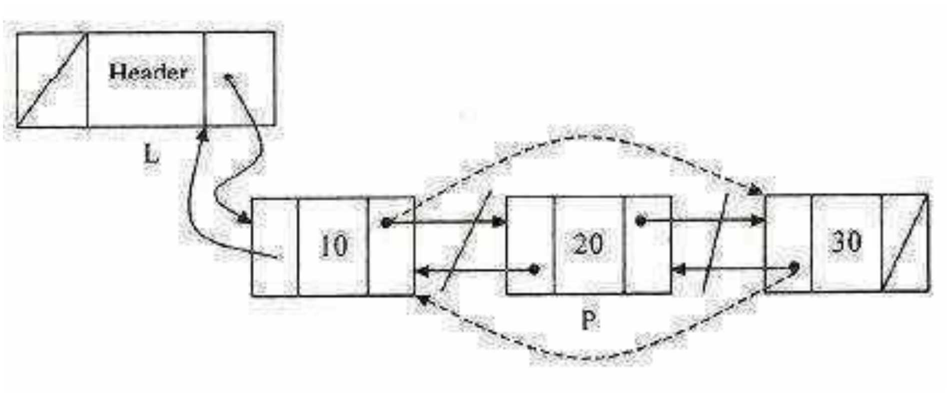
Algorithm to delete a node after a given node of a doubly linked list

- In Step 2,we take a pointer variable PTR and initialize it with START. That is, PTR now points to the first node of the doubly linked list.
- The while loop traverses through the linked list to reach the given node.Once we reach the node containing VAL, the node succeeding it can be easily accessed by using the address stored in its NEXT field.
- The NEXT field of the given node is set to contain the contents in the NEXT field of the succeeding node. Finally, the memory of the node succeeding the given node is freed and returned to the free pool.

```
struct node *delete_after(struct node *start)
{
struct node *ptr, *temp;                    int val;
printf("\n Enter the value after which the node has to deleted : "); scanf("%d",
&val);                                 ptr = start;
while(ptr -> data != val)
ptr = ptr -> next;  temp = ptr -> next;
ptr -> next = temp -> next;
temp -> next -> prev = ptr;
free(temp);
return start;
}
```

Routine for deleting an element:

```
void Delete (int x ,List L)
{ Position p , temp;
 P = Find( x, L );
 if(P==L->next) temp=L;
 L->next=temp->next;
 temp->next->prev=L;
free(temp);
elseif( IsLast( p, L ) )
{ temp = p;
p -> prev -> next = NULL;
 free(temp);
}
Else
{
temp = p;
p -> prev -> next = p -> next; p ->
next -> prev = p -> prev; free(temp);
}
```



Routine to display the elements in the list:

```
void Display( List L )
{
        P = L -> next ;
        while ( p != NULL)          {
        printf("%d", p -> data ;
p = p -> next ;             }
printf(" NULL");
  }
```

Routine to search whether an element is present

```
void find()
{
int a,flag=0,count=0;
if(L==NULL)
printf("\nThe list is empty");
Else
{
printf("\nEnter the elements to be searched");
scanf("%d",&a);
 for(P=L;P!=NULL;P=P->next)
{  count++;
if(P->data==a)
{
flag=1;
printf("\nThe element is found");
printf("\nThe position is
%d",count);
break; }
 }
if(flag==0)
printf("\nThe element is not found");
}}
```