# Chapter 3

## Linked List

### What is List-

As the name implies, list is a collection of short pieces of information, such as names of people, usually written with a single item on each line and often ordered in a way that makes a particular item easy to find. It is usually used in the daily life routine. Some other examples of list are birthday list of friends, shopping list.
Here we will take the name and age of persons in the list.

| Name | Age |
|---|---|
| Suresh | 26 |
| Ranjana | 32 |
| Reeta | 30 |
| Reena | 28 |
| Ankit | 10 |

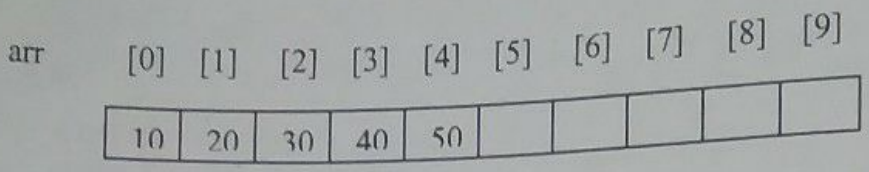
We can take three operations on the list-
1. Addition
2. Deletion
3. Searching

| Name | Age |
|---|---|
| Suresh | 26 |
| Reeta | 32 |
| Reena | 28 |
| Ankit | 10 |
| Rajesh | 37 |
| Madhu | -35 |

Here we have deleted the name and age of Ranjana and added the name and age of Madhu and Rajesh in the list.
There are two ways of maintaining this list in computer memory. First is to take an array to store the elements and second is the linked list.

### Array Implementation of List-

Let us take an array of size 10, which has 5 elements.
int arr[10];




**Traversing an array List-**

In array, we can find the next element through the index number of array because each next element of array has index number incremented by 1 with previous index number. So we can traverse and access the array elements through array index.
Let us take index value is 0.
Now the contents of arr[index] is 10 which is first element of array. We can process the next element of array by incrementing the index by 1.

index = index + 1

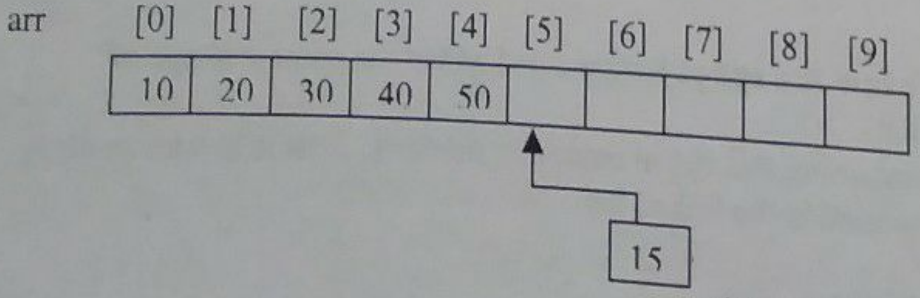
Now array[index] is 20 which is second element of array. So we can traverse each element of array by incrementing the index by 1 until the index is not greater than number of elements.

**Searching in an array list-**

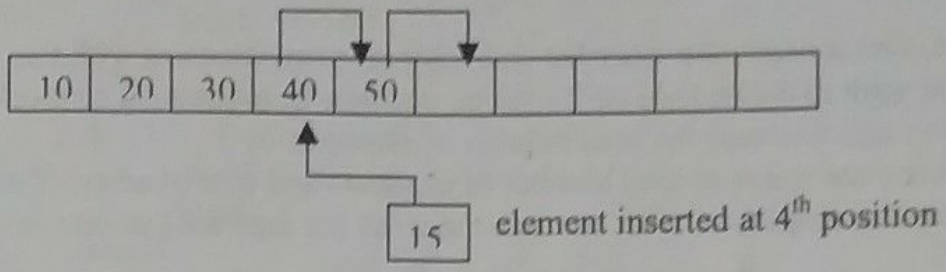
Searching refers to search an element into an array list. For searching the element, we first traverse the array list and with traversing, we compare each element of array with the given element.
We can search the element by another methods, which we will describe in the topic searching.

**Insertion into an array list-**

Insertion into an array list may be possible in two ways-
1. Insertion at end
2. Insertion in between




Case I element inserted at 6th position




Case II

**Case I-**

In the first case we have to set the array index to the total number of element and then insert the element.

index = Total number of elements ( i.e. 5 )

arr[ index ] = value of inserted element

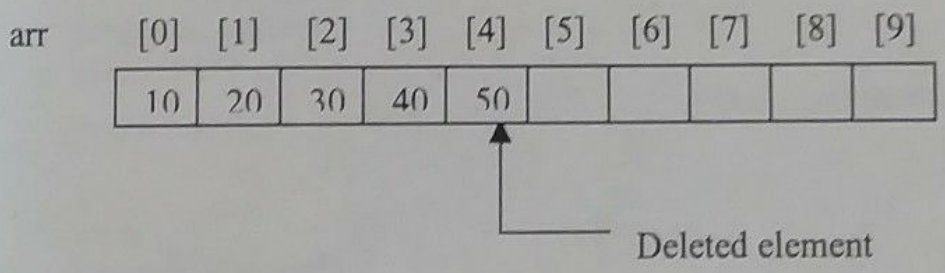
**Case II-**

In the second case, we have to shift right one position all array elements from last array element to the array element before which we want to insert the element.

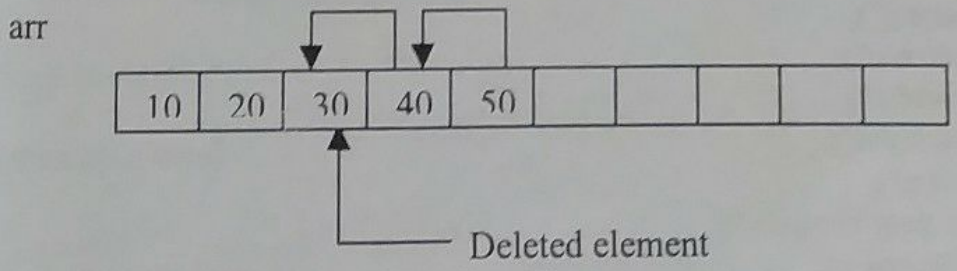
**Deletion from an array list-**

Deletion into an array list may be possible in two ways.

1. Deletion of the last element
2. Deletion in between




Case I




Case II

**Case I-**

First traverse the array list and compare array element with the element which you want to delete. If the item is last item of the array then delete that element and decrease the total number of elements by 1.

**Case II-**

In the second case, first traverse the array list and compare array element with the element which you want to delete then shift left one position from the next element to the last element of array and decrease the total number of elements by 1.

It is necessary to keep the status of total number of elements and size of array. If we want to keep a record such as employee code, name in the array list then we can take array of structure as-

Example-

```
struct {
        int code;
        char name[30];
        }emp[10];
```

Here each element of structure array emp[10] has member variables code and name.

The advantage of the array list is that we can easily compute the address of the array through index and we can also access the array element through index.

The disadvantage of the array list is that we have to keep the total number of elements and array size. We cannot take elements more than array size because array size is fixed. It also requires much processing in insertion and deletion because of shifting of array elements.

```
/* Program of list using array */
# include<stdio.h>
# define MAX 20
int arr[MAX];
int n;  /*Total number of elements in the list */

main( )
{
        int  choice,item,pos;
        while(1)
        {
                printf("1.Input list\n");
                printf("2.Insert\n");
                printf("3.Search\n");
                printf("4.Delete\n");
                printf("5.Display\n");
                printf("6.Quit\n");
                printf("Enter your choice : ");
                scanf("%d",&choice);

                switch(choice)
                {
                case 1:
                        printf("Enter the number of elements to be entered : ");
```

```
                scanf("%d",&n);
                input(n);
                break;
        case 2:
                insert( );
                break;
        case 3:
                printf("Enter the element to be searched : ");
                scanf("%d", &item);
                pos = search(item);
                if(pos >= 1)
                        printf("%d found at position %d\n",item,pos);
                else
                        printf("Element not found\n");
                break;
        case 4:
                del( );
                break;
        case 5:
                display( );
                break;
        case 6:
                exit( );
                break;
        default:
                printf("Wrong choice\n");
        } /*End of switch */
    }/*End of while */
}/*End of main( ) */

input( )
{
    int i;
    for(i = 0; i< n ; i++)
    {
        printf("Input value for element %d : ", i+1);
        scanf("%d", &arr[i]);
    }
}/*End of input( )*/

int search(int item)
{
    int i;
    for(i=0; i < n; i++)
    {
        if(item = = arr[i])
            return(i+1);
    }
    return(0);  /* If element not found */
}/*End of search( )*/
```

```
insert( )
{
        int temp,item,position;
        if(n = = MAX)
        {
                printf("List overflow\n");
                return;
        }
        printf("Enter position for insertion : ");
        scanf("%d", &position);
        printf("Enter the value : ");
        scanf("%d",&item);
        if(position > n+1 )
        {
                printf("Enter position less than or equal to %d\n",n+1);
                return;
        }
        if( position = = n+1 )  /*Insertion at the end */
        {
                arr[n] = item;
                n = n+1;
                return;
        }
        /* Insertion in between */
        temp=n-1;
        while( temp >= position-1)
        {
                arr[temp+1] = arr[temp];  /* shifting right */
                temp --;
        }
        arr[position-1] = item;
        n = n +1 ;
}/*End of insert( )*/

del( )
{
        int temp,position,item;
        if(n = = 0)
        {
                printf("List underflow\n");
                return;
        }
        printf("Enter the element to be deleted : ");
        scanf("%d",&item);
        if(item = = arr[n-1]) /*Deletion at the end*/
        {
                n = n-1;
                return;
        }
```

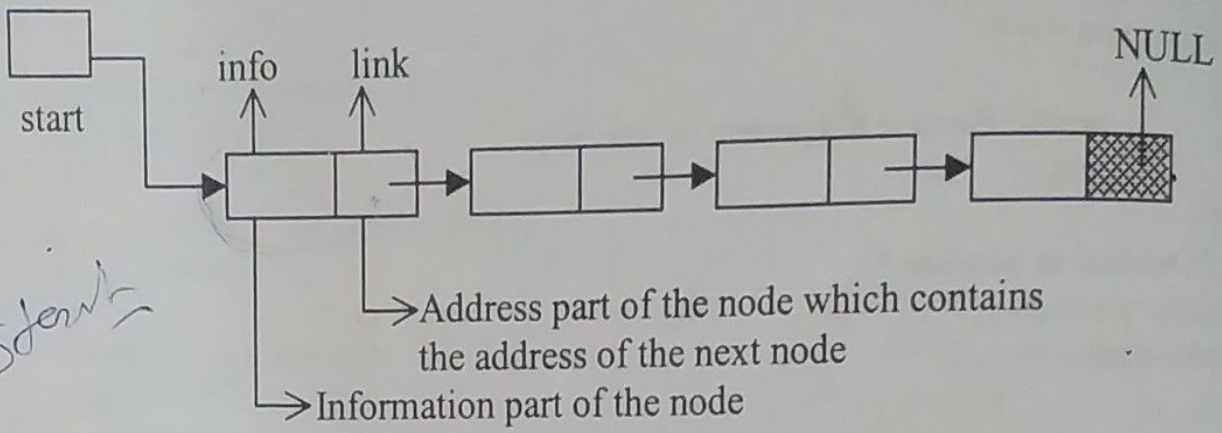
```
        position=search(item);
        if(position = = 0)
        {
                printf("Element not present in array\n");
                return;
        }
        /*Deletion in between */
        temp=position-1;
        while(temp <= n-1)
        {
                arr[temp] = arr[temp+1];  /* Shifting left */
                temp ++;
        }
        n = n - 1 ;
}/*End of del( )*/

display( )
{
        int i;
        if(n = = 0)
        {
                printf("List is empty\n");
                return;
        }
        for(i = 0; i< n; i++)
                printf("Value at position %d : %d\n", i+1, arr[i]);
}/*End of display( )*/
```

**Linked List**

The second way for implementing a list in memory is to use a structure which is self referential structure .One member of this structure is a pointer that points to the structure itself.

```
struct node {
        int data;
        struct node *link;
    };
```

Here member of the structure struct node *link points to the structure itself. This type of structure is called linked list. A linked list is a collection of elements called nodes. Each node has two parts, first part contains the information field and second part contains the address of the next node .The address part of last node of linked list will have NULL value.




**Traversing a Linked List-**

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

```
ptr=ptr->link;
```

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

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

**Searching into a Linked List-**

Searching refers to search an element in a linked list .For searching the element we first traverse the linked list and with traversing we compare the info part of each element with the given element. It can be written as-

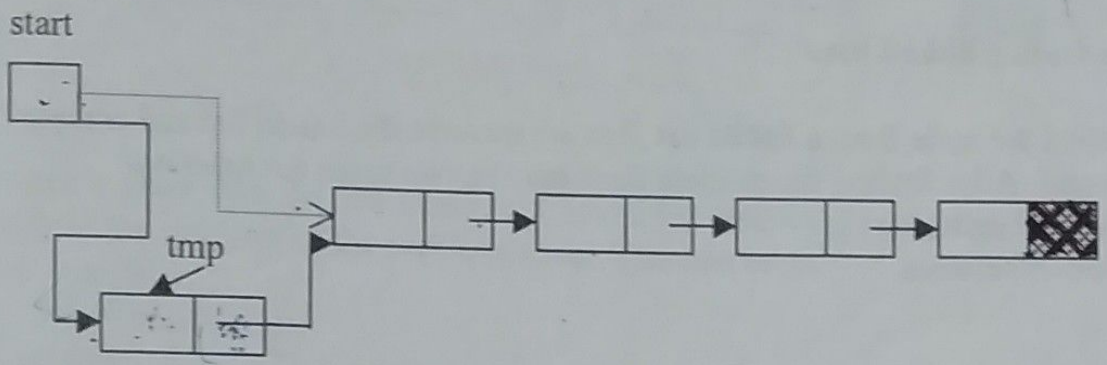
```
while( ptr != NULL )
{
        if ( ptr->info = = data)
                ----------
        else
                ptr=ptr->link;
}
```

Here data is the element which we want to search.

**Insertion into a Linked List-**

Insertion in a linked list may be possible in two ways-
1. Insertion at beginning
2. Insertion in between

**Case 1-**




tmp is pointer which points to the node that has to be inserted.

```
tmp->info=data;
```

start points to the first element of linked list. For insertion at beginning, we assign the value of start to the link part of inserted node as –

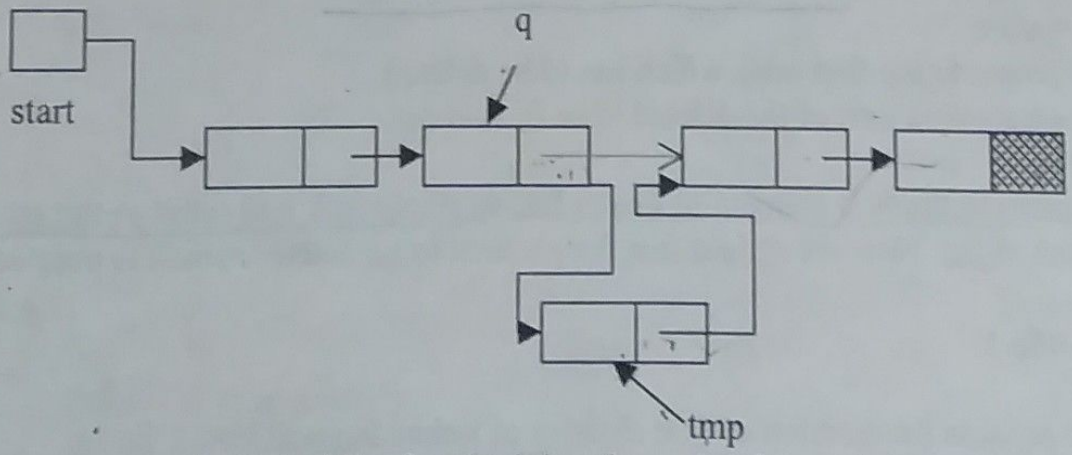
```
tmp->link=start;
```

Now inserted node points to the next node which was beginning node of the linked list.
Now inserted node is the first node of the linked list. So start will be reassigned as-

```
start = tmp;
```

Now start will point to the inserted node which is first node of the linked list.
Dotted line represents the link before insertion.

**Case 2-**




First we traverse the linked list for obtaining the node after which we want to insert the element. We obtain pointer q which points to the element after which we have to insert new node. For inserting the element after the node, we give the link part of that node to the link part of inserted node and the address of the inserted node is placed into the link part of the previous node.

```
tmp->info=data;
tmp->link=q->link;
q->link=tmp;
```

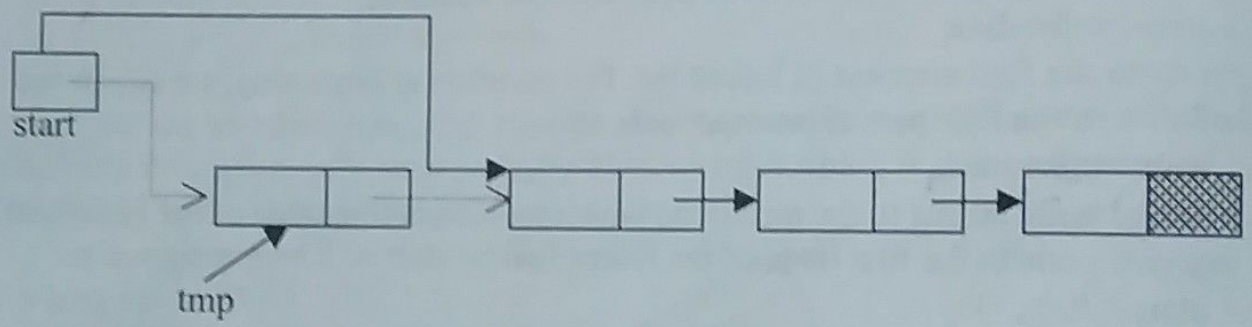
Here q is pointing the previous node. After statement 2, link of inserted node will point to the next node and after statement 3 link of previous node will point to the inserted node.

**Deletion from a linked list -**

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

1. Deletion at beginning
2. Deletion in between

**Case 1-**




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

```
tmp = start;
```

So now tmp points to the first node which has to be deleted.

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

```
start=start->link;
```

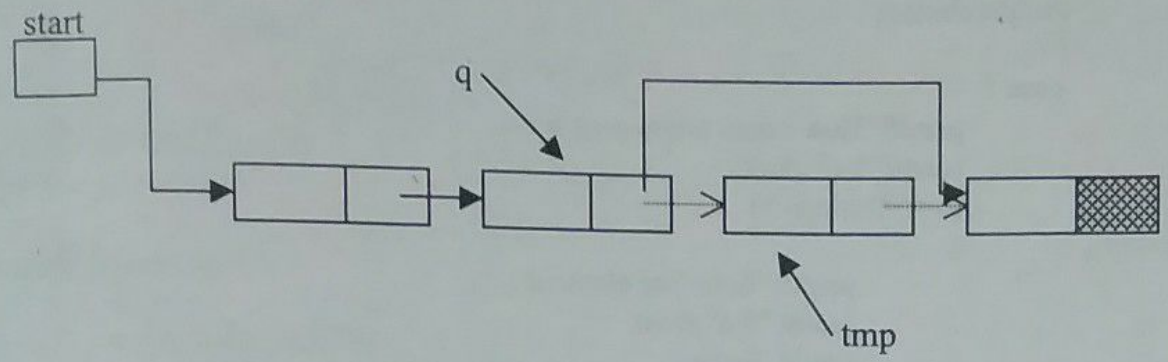
Since start points to the first element of linked list, so start->link will point to the second element of linked list. Now we should free the element to be deleted which is pointed by tmp.

```
free( tmp );
```

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

```
tmp = start;
start=start->link;
free( tmp );
```

**Case 2-**




If the element is other than the first element of linked list then we give the link part of the deleted node to the link part of the previous node. This can be as-

```
tmp = q->link;
q->link = tmp->link;
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 link of previous node will point to next node of the node to be deleted.

If node to be deleted is last node of linked list then statement 2 will be as-

```
q->link = NULL;
```

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

struct node
{
        int info;
        struct node *link;
}*start;

main( )
{
        int choice,n,m,position,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.Search\n");
                printf("9.Quit\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",&position);
        addafter(m,position);
        break;
 case 4:
        if(start==NULL)
        {
                printf("List is empty\n");
                continue;
        }
        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:
        printf("Enter the element to be searched : ");
        scanf("%d",&m);
        search(m);
```

```
                    break;
                case 9:
                    exit( );
                default:
                    printf("Wrong choice\n");
                }/*End of switch */
        }/*End of while */
}/*End of main( )*/

create_list(int data)
{
        struct node *q,*tmp;
        tmp= malloc(sizeof(struct node));
        tmp->info=data;
        tmp->link=NULL;

        if(start==NULL) /*If list is empty */
                start=tmp;
        else
        {       /*Element inserted at the end */
                q=start;
                while(q->link!=NULL)
                        q=q->link;
                q->link=tmp;
        }
}/*End of create_list( )*/

addatbeg(int data)
{
        struct node *tmp;
        tmp=malloc(sizeof(struct node));
        tmp->info=data;
        tmp->link=start;
        start=tmp;
}/*End of addatbeg( )*/

addafter(int data,int pos)
{
        struct node *tmp,*q;
        int i;
        q=start;
        for(i=0;i<pos-1;i++)
        {
                q=q->link;
                if(q==NULL)
                {
                        printf("There are less than %d elements",pos);
                        return;
                }
        }/*End of for*/
```

```
        tmp=malloc(sizeof(struct node) );
        tmp->link=q->link;
        tmp->info=data;
        q->link=tmp;
}/*End of addafter( )*/

del(int data)
{
        struct node *tmp,*q;
        if(start->info == data)
        {
                tmp=start;
                start=start->link;  /*First element deleted*/
                free(tmp);
                return;
        }
        q=start;
        while(q->link->link != NULL)
        {
                if(q->link->info==data)    /*Element deleted in between*/
                {
                        tmp=q->link;
                        q->link=tmp->link;
                        free(tmp);
                        return;
                }
                q=q->link;
        }/*End of while */
        if(q->link->info==data)    /*Last element deleted*/
        {
                tmp=q->link;
                free(tmp);
                q->link=NULL;
                return;
        }
        printf("Element %d not found\n",data);
}/*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->link;
        }
        printf("\n");
}/*End of display( ) */

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

rev( )
{
        struct node *p1,*p2,*p3;
        if(start->link==NULL)    /*only one element*/
                return;
        p1=start;
        p2=p1->link;
        p3=p2->link;
        p1->link=NULL;
        p2->link=p1;
        while(p3!=NULL)
        {
                p1=p2;
                p2=p3;
                p3=p3->link;
                p2->link=p1;
        }
        start=p2;
}/*End of rev( )*/

search(int data)
{
        struct node *ptr = start;
        int pos = 1;
        while(ptr!=NULL)
        {
                if(ptr->info==data)
                {
                        printf("Item %d found at position %d\n",data,pos);
```

```
                return;
            }
            ptr = ptr->link;
            pos++;
        }
        if(ptr == NULL)
            printf("Item %d not found in list\n",data);
}/*End of search( )*/
```

**Reverse linked list**

Let us take a linked list-

start