Module 2.1

Linear Data Structure: Linked List

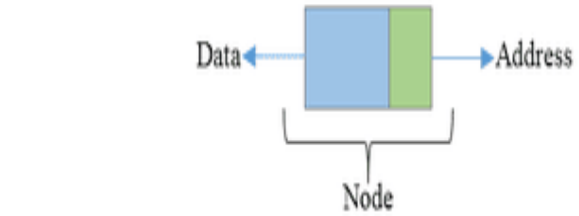
Outline

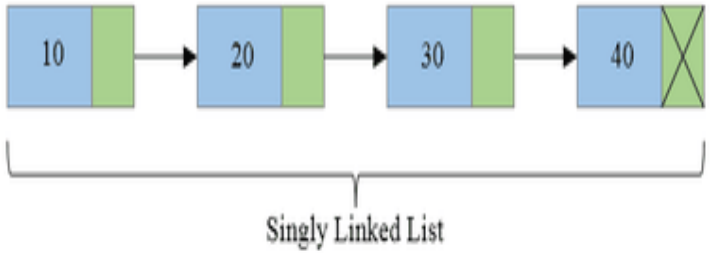
- Introduction and Representation of Linked List
- Linked List v/s Array
- Implementation of Linked List
- Circular Linked List
- Doubly Linked List,
- Application – Polynomial Representation and Addition
- Other additional applications/Case study.

Linked List

Linked Lists

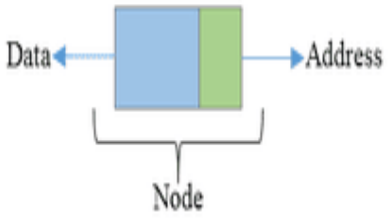
- **Linear Collection of data elements called Nodes**
- **Linear order is given by means of pointers.**





Linked Lists

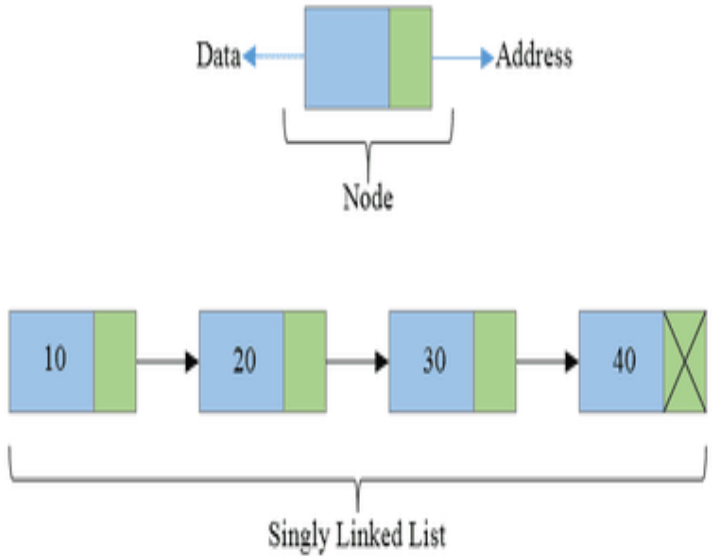
- **Each node may be divided into atleast two fields for :**
  - **Storing Data**
  - **Storing Address of next element.**



Linked Lists

- **The Last node's Address field contains Null rather than a valid address.**

- **It's a NULL Pointer and indicates the end of the list.**



Comparison between Array and Linked List

Advantages of Linked List

- **Linked are Dynamic Data Structures**
  - Grow and shrink during execution of the program
- **Efficient Memory Utilization**
  - As memory is not preallocated.
  - Memory can be allocated whenever required and deallocated when not needed.
- **Insertion and deletions are easier and efficient**
  - Provide flexibility in inserting a data item at a specified position and deletion of a data item from the given position
- **Many complex applications can be easily carried out with linked lists**

Disadvantages of Linked List

- Access to an arbitary data item is little bit cumbersome and also time consuming

- **More memory**
  - If the number of fields are more, then more memory space is needed.

Advantages of Arrays

- Simple to use and define
- Supported by almost all programming languages
- **Constant access time**
  - Array element can be accessed a[i]
- Mapping by compiler
  - Compiler maps a[i] to its physical location in memory.
  - **This mapping is carried out in constant time, irrespective of which element is accessed**

Disadvantages of Arrays

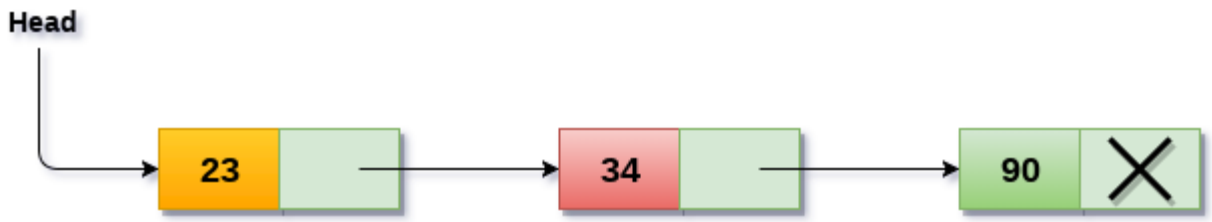
- **Static Data Structure-**
  - **Size of an array is defined at the time of programming**
- **Insertion and Deletion is time consuming**
- Requires Contiguous memory

Types of Linked List

- Singly Linked List
- Doubly Linked List
- Circular Linked List

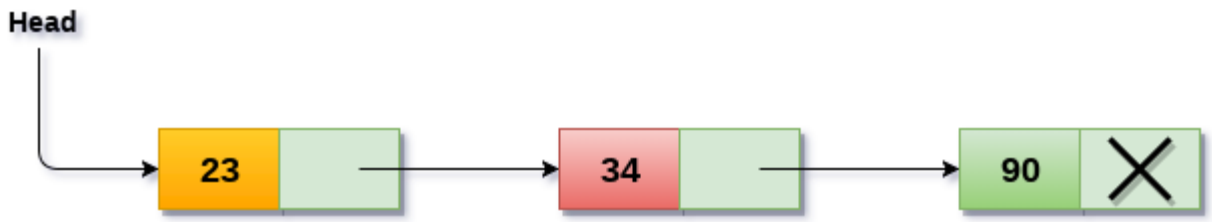
Singly Linked List

- All nodes are linked in sequential manner
- Linear Linked List
- One way chain
- It has beginning and end



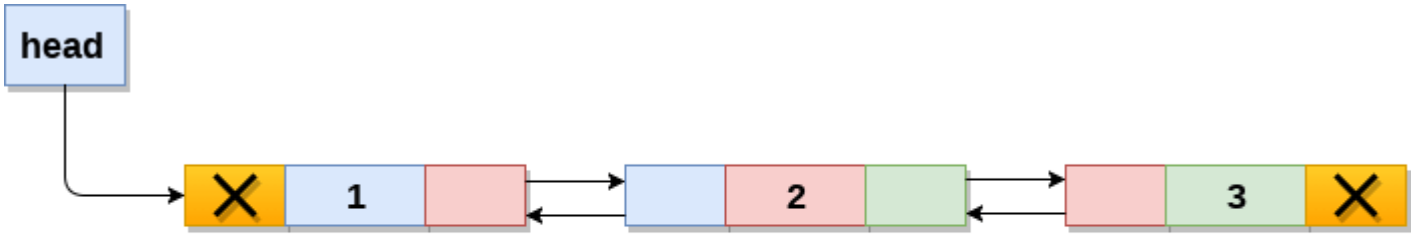
Singly Linked List

- Problem-
  - The predecessor of a node cannot be accessed from the current node.
  - This can be overcome in doubly linked list.



Doubly Linked List

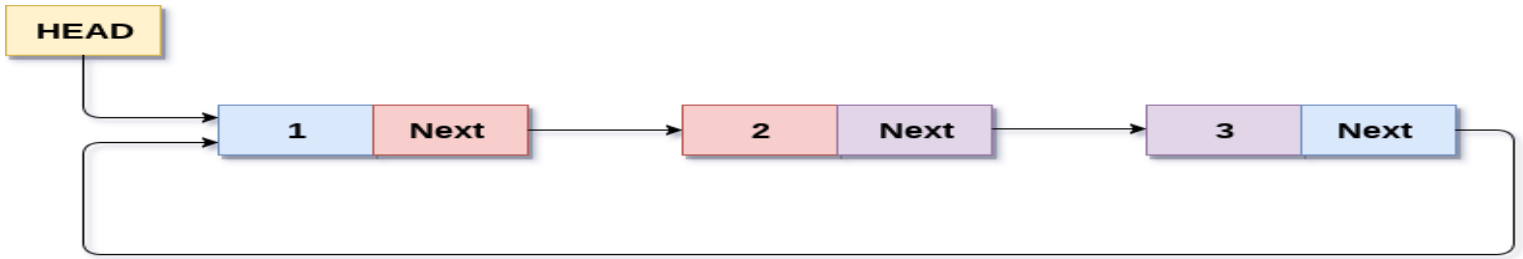
- Linked List holds two pointer fields
- Addresses of next as well as preceding elements are linked with current node.
- This helps to traverse in both Forward or Backward direction



Doubly Linked List

Circular Linked List

- The first and last elements are adjacent.
- A linked list can be made circular by
  - Storing the address of the first node in the link field of the last node.



Circular Singly Linked List

Linked List Operations

- Creation
- Insertion
- Deletion
- Traversal
- Searching

Implementation of Linked Lists

- Structures in C are used to define a node
- Address of a successor node can be stored in a pointer type variable

Linked Lists

Struct Basics

Struct Syntax

Struct examples

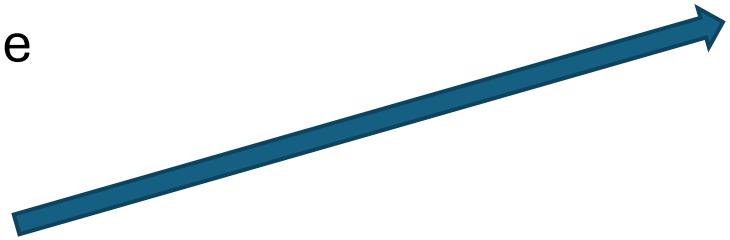
Pointer to structure

Sample Programs for Pointer to structures

Initialization of Pointers to Structures

Linked Lists

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



Information field

Pointer that points to the structure itself,
Thus Linked List.

Singly Linked List

Creation of a new node

```
struct node{
                type1 member1;
                type2 member2;
                ..........
                ..........
                struct node *link;
            };
```

Info part

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


```
struct node *start = NULL;
```

Creation of a new node

New node=temp

```
struct node *tmp;
tmp= (struct node *) malloc(sizeof(struct node));
tmp->info=data;
tmp->link=NULL;
```

Creating a Linked List

```
create_list(int data)
{
        struct node *q,*tmp;
        tmp= (struct node *) 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()*/
```

Explanation-

```
if(start==NULL) /*If list is empty */
        {
        start=tmp;
        }
```

Explanation-

```
else
        {     /*Element inserted at the end */
                q=start;
        while(q->link!=NULL)
                q=q->link;
                q->link=tmp;
        }
```

Traversing a Linked List

- Visit the node and print the data value

Traversing a Linked List

- Assign the Value of start to another pointer say q

struct node *q=start;

- Now q also points to the first element of linked list.

- For processing the next element, we assign the address of the next element to the pointer q as-

q=q->link;

- Traverse each element of the Linked list through this assignment until pointer q has NULL address, which is link part of last element.

```
while(q!=NULL)
    {
        q=q->link;
    }
```

Explanation-

```
while(q!=NULL)
        {
                q=q->link;
        }
```

Explanation-

Algorithm for Traversing a Linked List

Step 1:[INITIALIZE] SET PTR=START

Step 2: Repeat Steps 3 and 4 while PTR!=NULL
Step 3: Print PTR->INFO

Step 4: Set PTR=PTR->LINK

[End of Loop]
Step 5 :EXIT

Algorithm for Counting the number of elements in a Linked List

Step 1:[INITIALIZE] SET COUNT=0

Step 2:[INITIALIZE] SET PTR=START

Step 3: Repeat Steps 4 and 5 while PTR!=NULL
Step 4: Set COUNT=COUNT +1

Step 5: Set PTR=PTR->LINK

[End of Loop]
Step 6:Print COUNT

Step 7:EXIT

Searching a Linked List

- First traverse the linked list
- While traversing compare the info part of each element with the given element

Searching a Linked List

```
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()*/
```

Algorithm for Searching a Linked List

Step 1:[INITIALIZE] SET POSITION=1

Step 2:[INITIALIZE] SET PTR=START

Step 3: Repeat Steps 4 while PTR!=NULL

Step 4: If DATA=PTR->INFO

Print POSITION

Exit

[End of If]

Set PTR=PTR->LINK

Set POSITION=POSITION +1

[End of Loop]

Step 5:If PTR=NULL

Print Search Unsuccessful

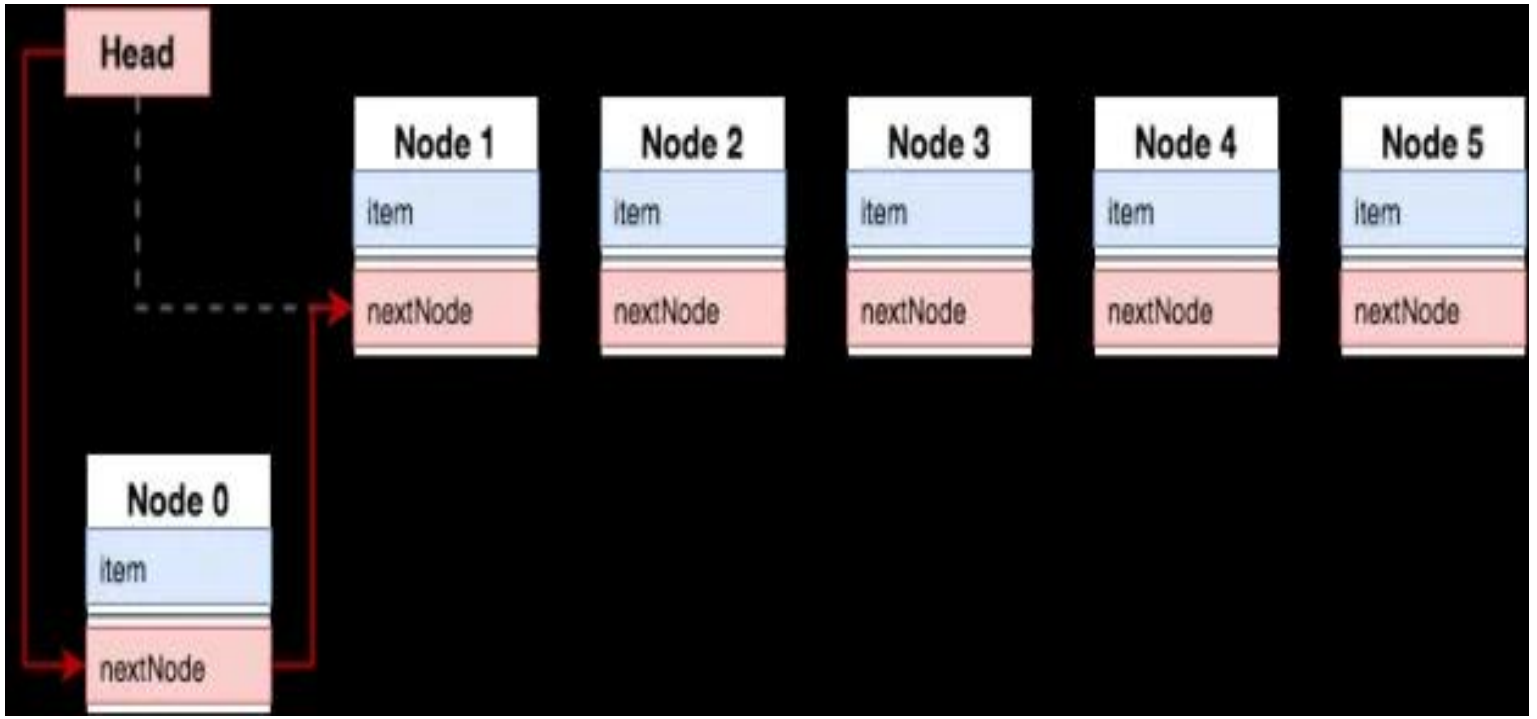
[End of If]

Step 6: Exit

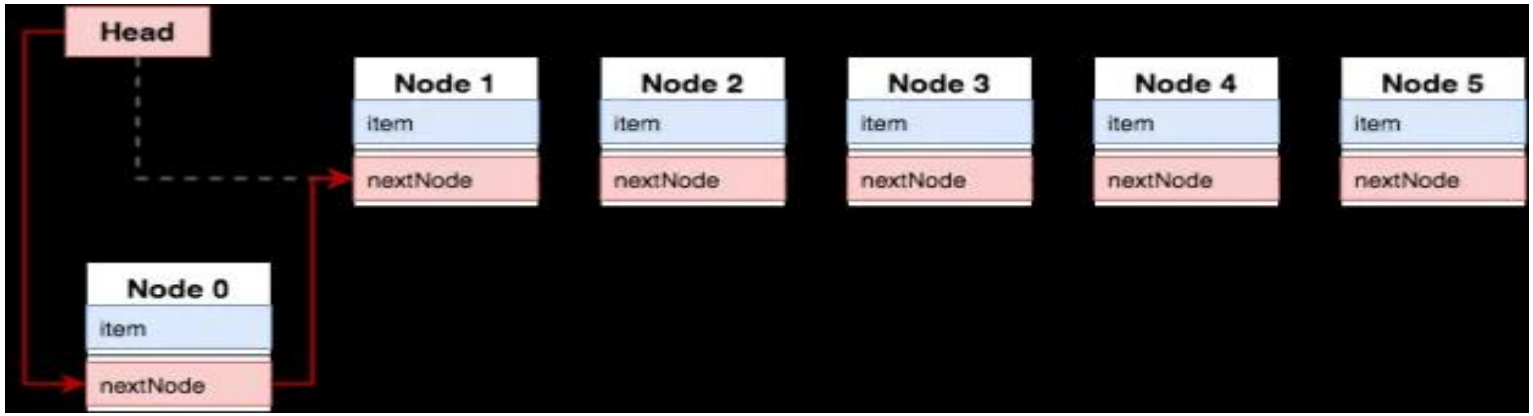
Insertion into a Linked List

- Insertion is possible in two ways:
  - Insertion at Beginning
  - Insertion in Between

Case 1- Insertion at Beginning

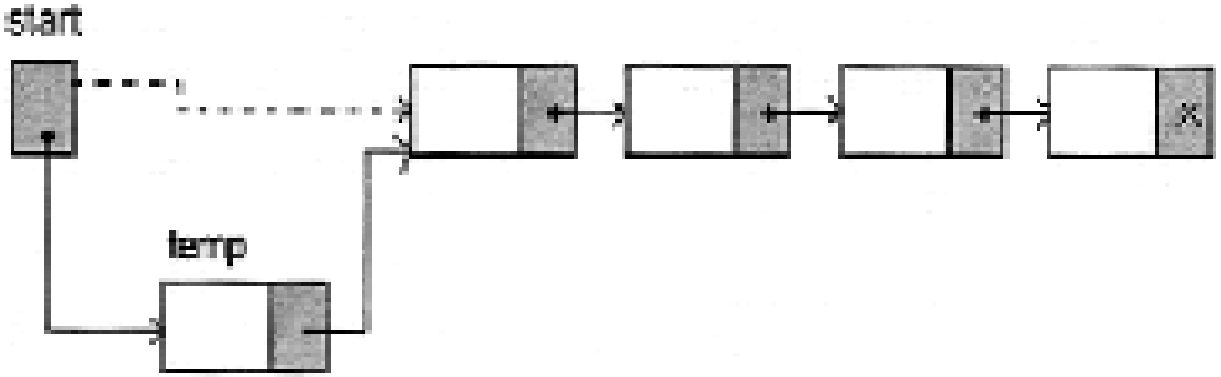


Case 1- Insertion at Beginning



CREATE THE NEW NODE,
CONNECT NEW NODE TO THE OLD FIRST NODE
CONNECT THE START POINTER TO THE NEW NODE,
……

Case 1- Insertion at Beginning



?

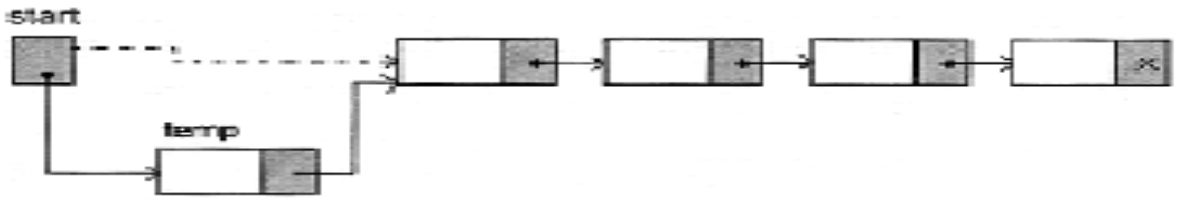
Case 1- Insertion at Beginning

- Lets say tmp is the pointer which points to the node that has to be inserted
- Assign data to the new node

  tmp->info=data;
- Start points to the first element of linked list
- Assign the value of start to the link part of the inserted node as

  tmp->link=start;
- Now inserted node points beginning of the linked list.
- To make the newly inserted node the first node of the linked list:

  start=temp



Algorithm for Insertion at Beginning in a Linked List

- First check whether Memory is available for the new node.
- If the memory has exhausted then an Overflow message is printed
- Else We allocate memory for the new node

Algorithm for Insertion at Beginning in a Linked List

Step 1:If AVAIL=NULL Then

WRITE OVERFLOW

Go to Step 7

[End of If]

Step 2: Set TEMP=AVAIL

Step 3: Set AVAIL=AVAIL->LINK

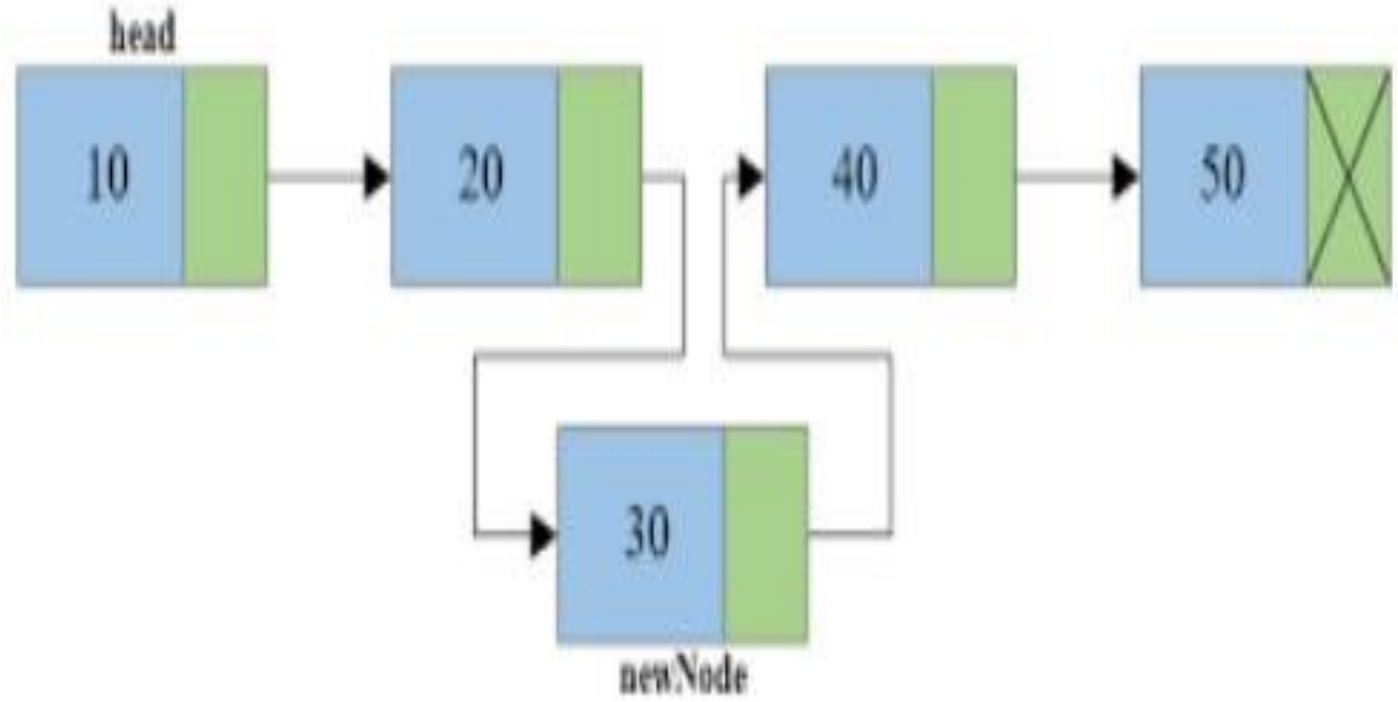
Step 4: Set TEMP->INFO=DATA

Step 5: Set TEMP->LINK=START

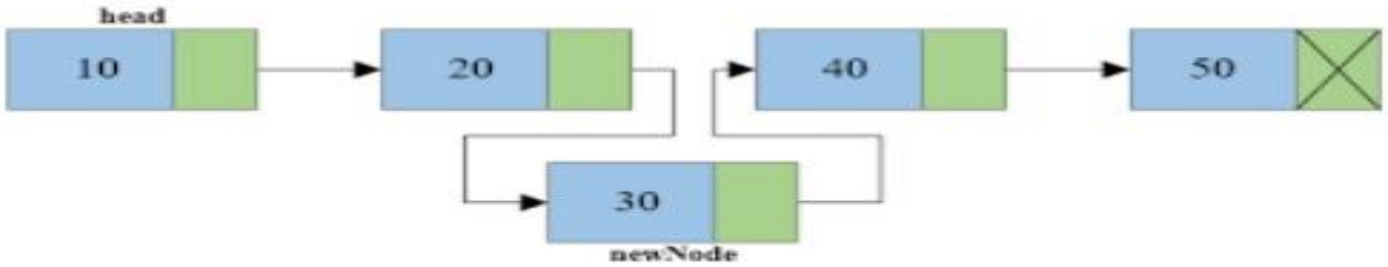
Step 6 : Set START=TEMP

Step 7 :EXIT

Case 2- Insertion in Between



Case 2- Insertion in Between

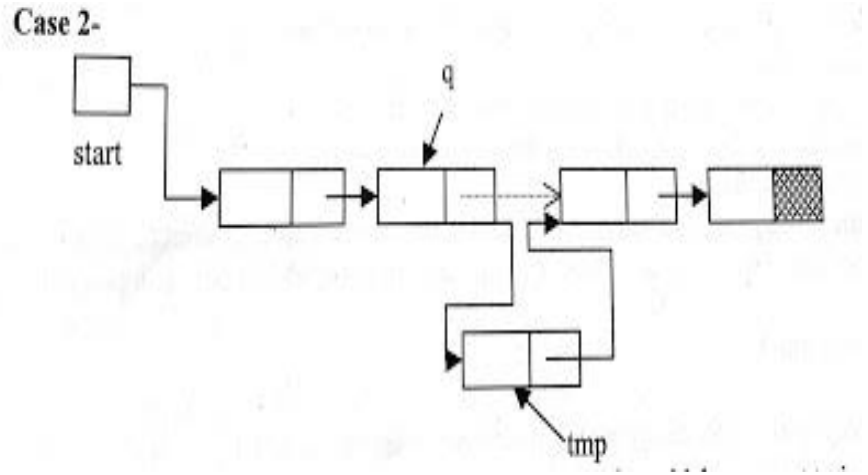


CREATE THE NEW NODE ,
CONNECT THE NEW NODE TO THE NEXT NODE
CONNECT THE PREVIOUS TO THE NEW NODE,
……

Case 2- Insertion in Between

- First we traverse the linked list for obtaining the node after which we want to insert the element

```
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*/
```



Explanation-

```
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*/
```

Explanation-

Algorithm for Insertion in Between

Step 1:If AVAIL=NULL Then

WRITE OVERFLOW

Go to Step 13

[End of If]

Step 2: Set TEMP=AVAIL

Step 3: Set AVAIL=AVAIL->LINK

Step 4: Set TEMP->INFO=DATA

Step 5: Set TEMP->LINK=NULL

Step 6 : Read POSITION from User

Step 7 : Set Q=START

Step 8 : Set I=0

Step 9 : Repeat step 10 till I<POS-1

Step 10: Set Q=Q->LINK

If Q=NULL

Print Less Number of Elements

Exit

[End of If]

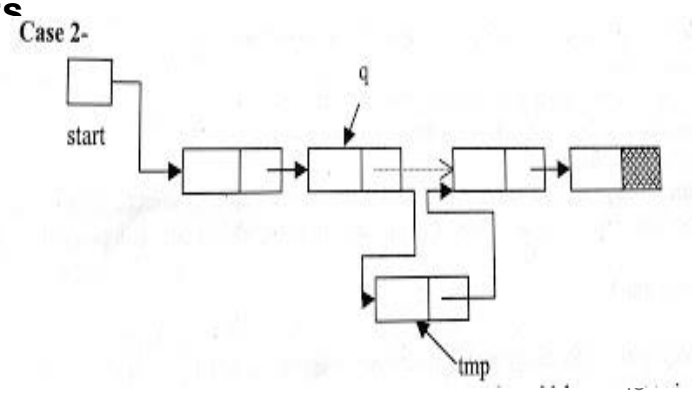
SET I=I+1

[End of Loop]

Step 11:Set TEMP->LINK=Q->LINK

Step 12:Set Q->LINK=TEMP

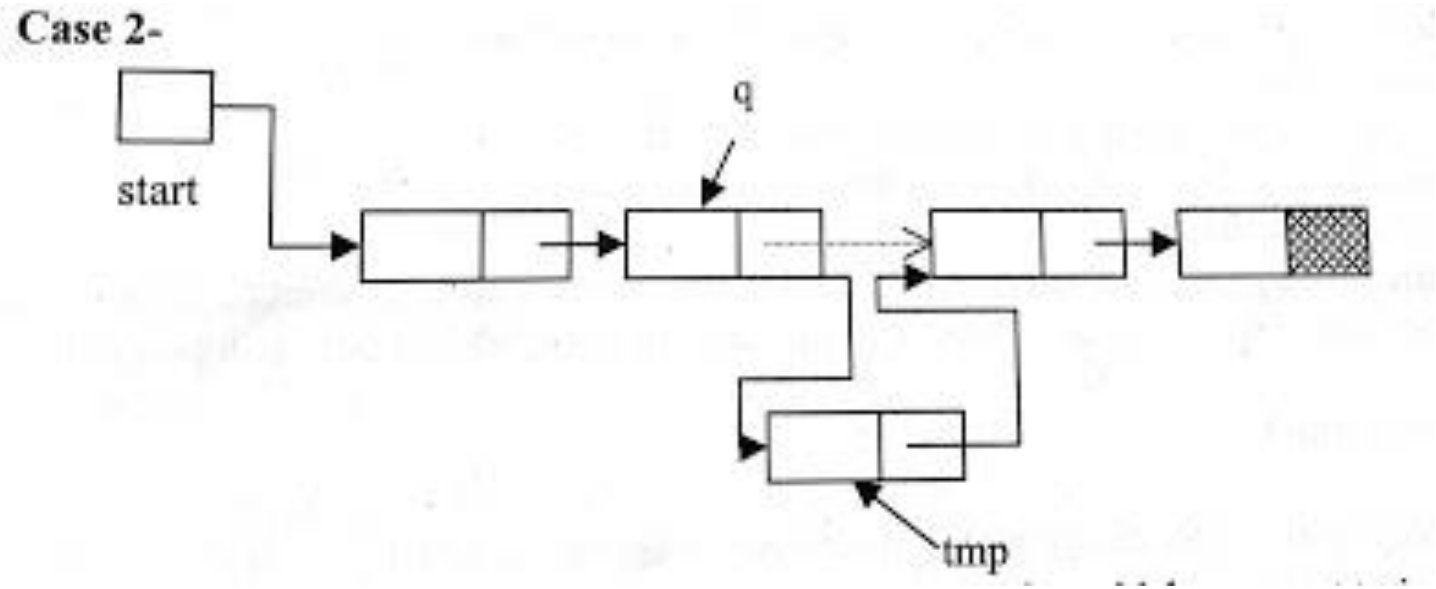
```
q=start;
for(i=0;i<pos-1;i++)
        {
                q=q->link;

        if(q==NULL)
                {

        printf("There are less
than %d elements",pos);

        return;
                }
        }/*End of for*/
```



Case 2- Insertion in Between

- Then we add the new node by adjusting address fields

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



Case 2- Insertion at the end

- **Without Using Position ?**
- **At the end?**

Algorithm for Insertion at the end of a Linked List

Step 1: If AVAIL=NULL Then

WRITE OVERFLOW

Go to Step 10

[End of If]

Step 2: Set TEMP=AVAIL

Step 3: Set AVAIL=AVAIL->LINK

Step 4: Set TEMP->INFO=DATA

Step 5: Set TEMP->LINK=NULL

Step 6 : Set Q=START

Step 7 : Repeat step 8 while PTR->LINK!=NULL

Step 8: SET Q=Q->LINK

[End of Loop]

Step 9 : SET Q->LINK=TEMP

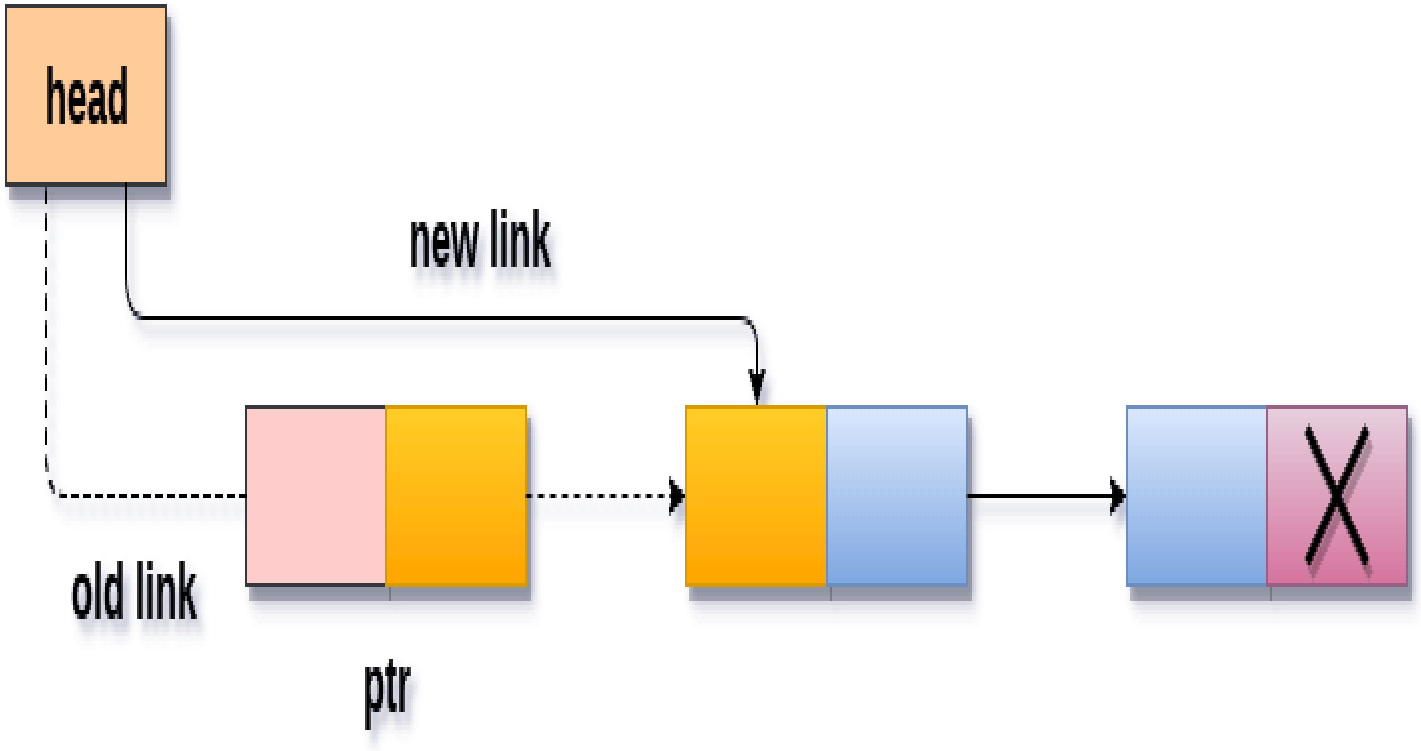
Step 10 : EXIT

Deletion from a Linked List

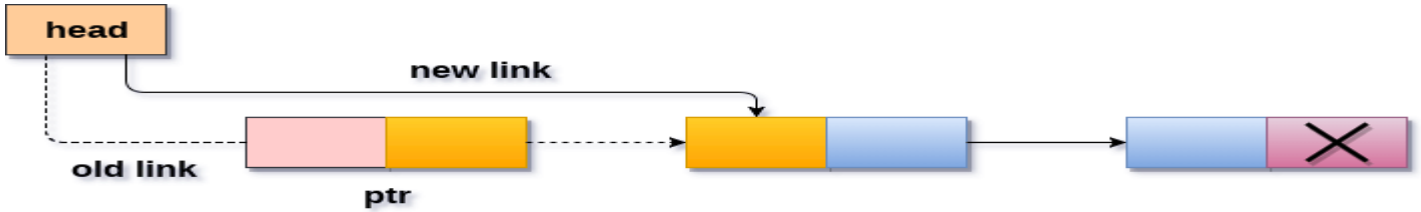
- For deleting the node from a linked list, first we traverse the linked list and compare with each element.

- After finding the element there may be two cases for deletion-
  - **Deletion in beginning**
  - **Deletion in between**

Deletion in beginning



Deletion in beginning



CONNECT
START POINTER TO THE SECOND NODE……
DELETE THE FIRST NODE

Deletion in beginning

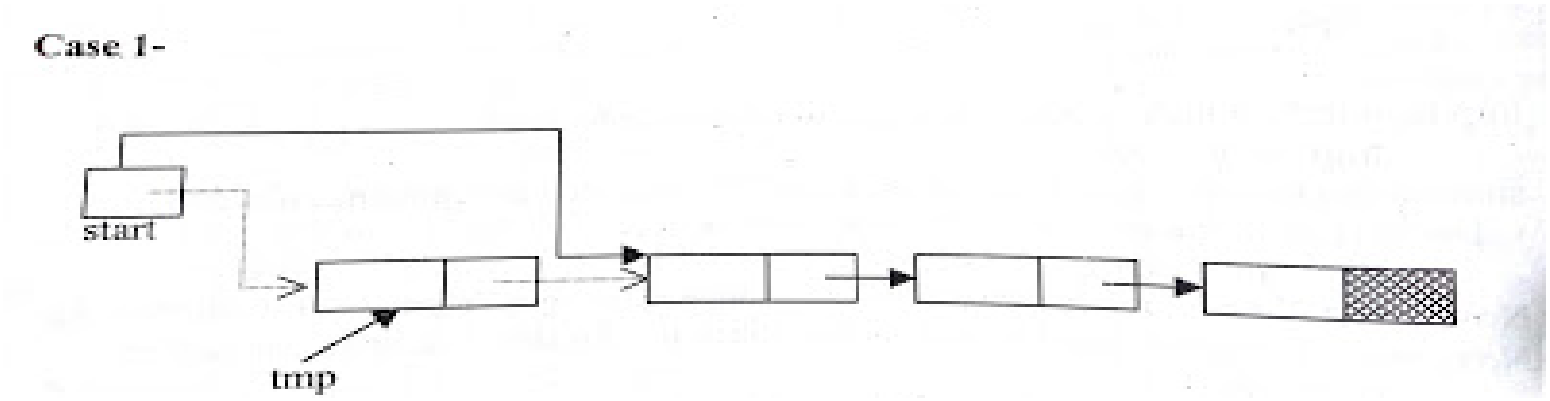
- Start points to the first element of linked list.
- If element to be deleted is the first element of linked list then we assign the value of start to tmp as-

  tmp = start;

- So tmp points to the first node which has to be deleted.
- Now assign the link part of the deleted node to start as-

  start=start->link;

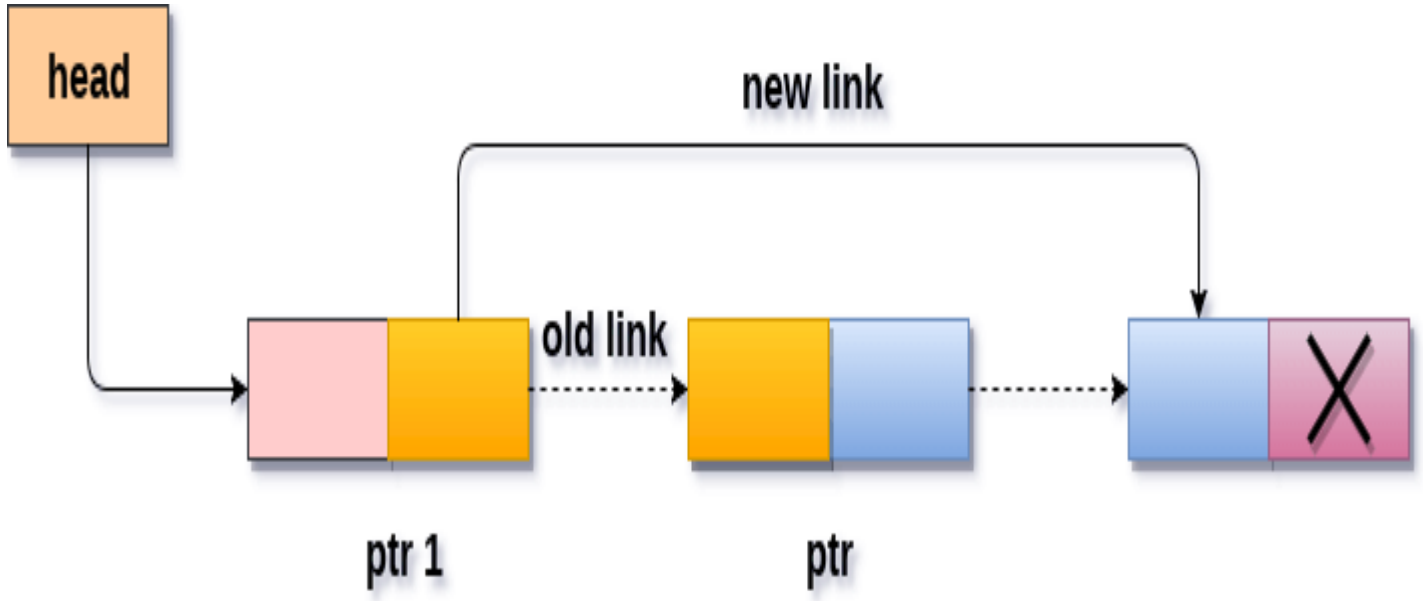
- **<u>Since start points to the first element of linked list, so start->link will point to the second element of linked list.</u>**
- Now we should free the element to be deleted which is pointed by tmp.
  free(tmp);



Algorithm for Deletion in the beginning of the Linked List

Step 1:If START=NULL
Step 2: Write UNDERFLOW
Step 3: Go to Step 7
[End of If]
Step 4: Set TEMP=START
Step 5: Set START=START->LINK
Step 6:FREE TEMP
Step 7:EXIT

Deletion in between



Deletion a node from specified position

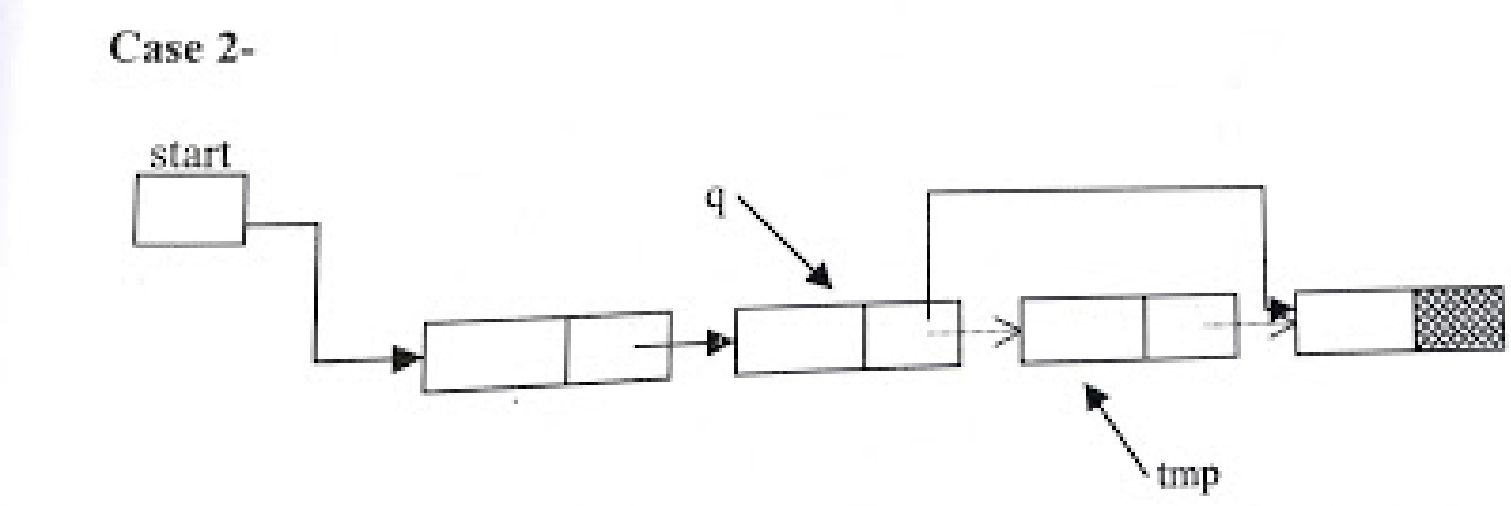
Deletion in between

DELETE THE NODE AND CONNECT THE PREVIOUS AND THE NEXT NODE.....

Deletion in between

- 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);
```



Deletion at the end

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

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

Circular Linked List

Why Circular?

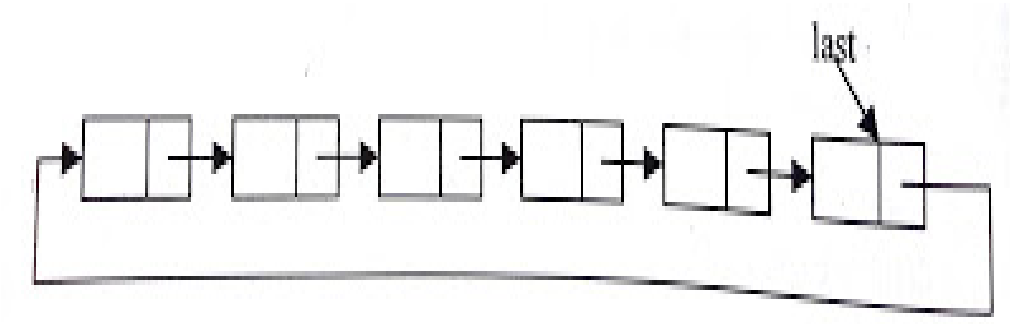
- In a singly linked list,
  - **If we are at any node in the middle of the list, then it is not possible to access nodes that precede the given node.**
  - This problem can be solved by slightly altering the structure of singly linked list.

How?

- In a singly linked list, next part (pointer to next node) of the last node is NULL,
  - if we utilize this link to point to the first node then we can reach preceding nodes.

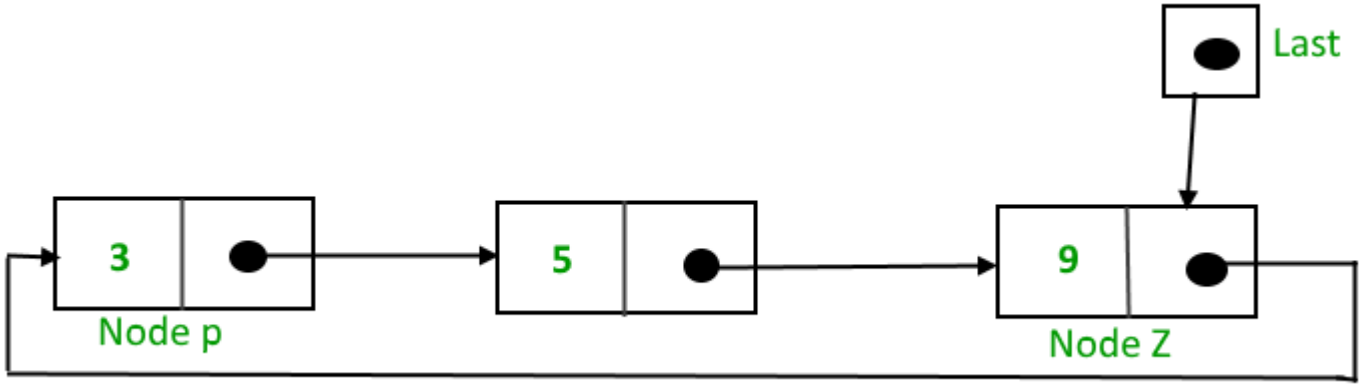
Implementation of circular linked list

- Creation of circular linked list is same as single linked list.
- **Last node will always point to first node instead of NULL.**



Implementation of circular linked list

- **One pointer last,**
  - **which points to last node of list and link part of this node points to the first node of list.**



Advantages of a Circular linked list

- In circular linked list, **we can easily traverse to its previous node**, which is not possible in singly linked list.

- **Entire list can be traversed from any node.**
  - If we are at a node, then we can go to any node. But in linear linked list it is not possible to go to previous node.

Advantages of a Circular linked list

- In Single Linked List, for insertion at the end, the whole list has to be traversed.
- In Circular Linked list,
  - **with pointer to the last node there won't be any need to traverse the whole list.**
  - So insertion in the begging or at the end takes constant time irrespective of the length of the list i.e O(1).
  - **It saves time when we have to go to the first node from the last node.**
    - **It can be done in single step because there is no need to traverse the in between nodes**

Disadvantages of Circular linked list

- Circular list are complex
  - as compared to singly linked lists.

- **Reversing of circular list is a complex**
  - **as compared to singly or doubly lists.**

- If not traversed carefully,
  - then we could end up in an infinite loop.

- Like singly and doubly lists circular linked lists also **doesn't supports direct accessing of elements.**

Insertion into a circular linked list :-

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

- ***Insertion in an empty list***
- **Insertion at the end of the list**
- ***Insertion at beginning***
- ***Insertion in between***

Insertion in an empty list

New element can be added as-

- *If linked list is empty:*

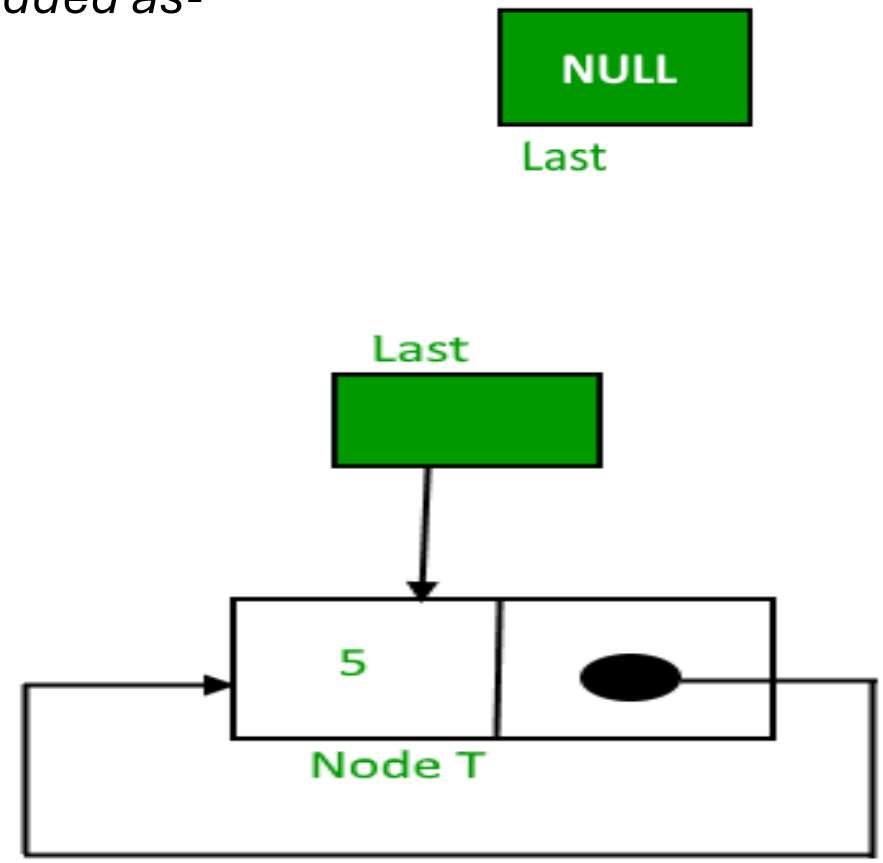
If (last==NULL)

{

last=tmp;

tmp->link=last

}



Insertion at the end of circular linked list

- *If linked list is not empty:*

Insertion at the end of the list

{

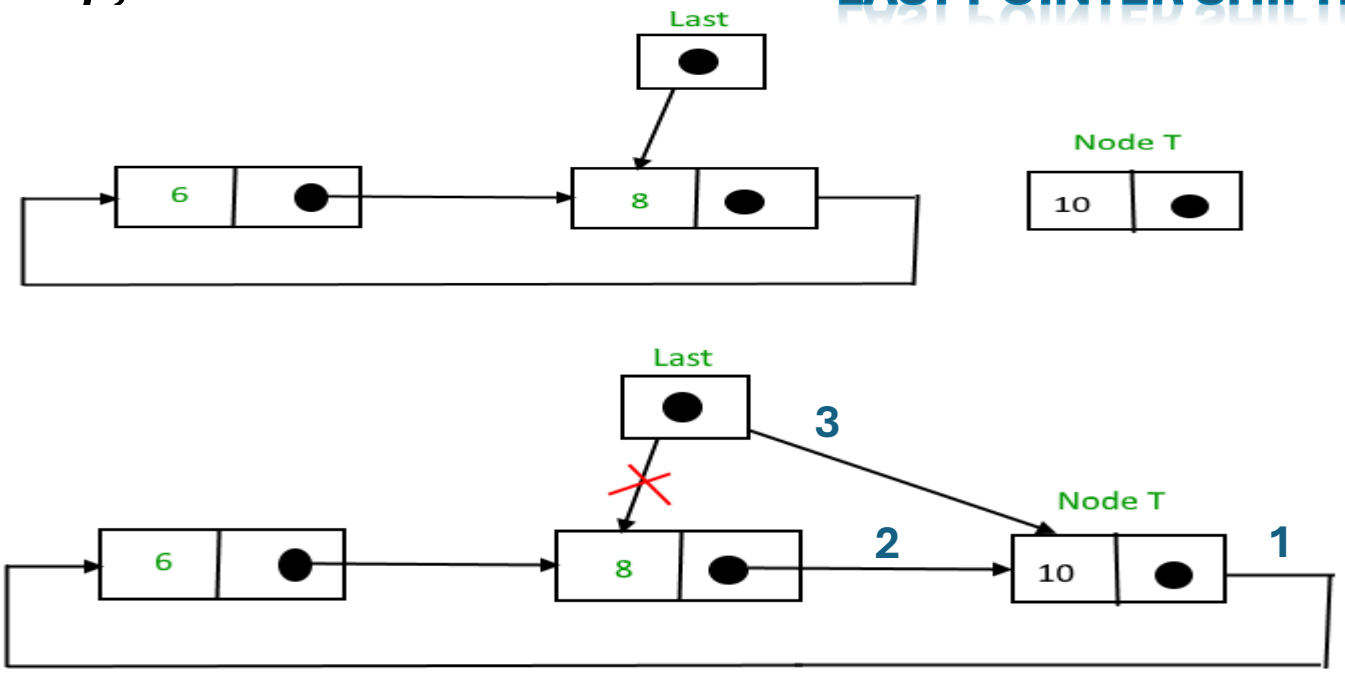
tmp->link = last->link; /* added at the end of list*/

last->link = tmp;

last = tmp;

}

LAST POINTER SHIFTED



Insertion at the end of circular linked list

- *If linked list is not empty:*

Insertion at the end of the list

{

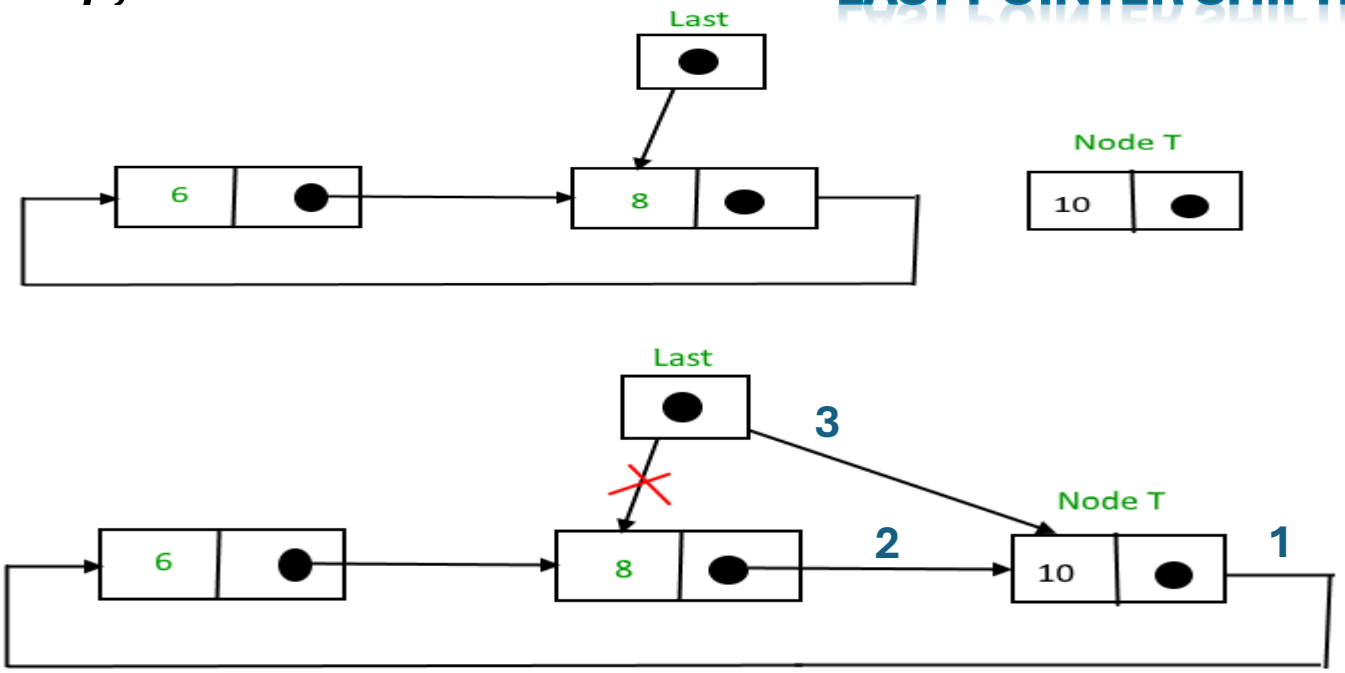
tmp->link = last->link; /* added at the end of list*/

last->link = tmp;

last = tmp;

}

LAST POINTER SHIFTED



Insertion at the beginning of circular linked list

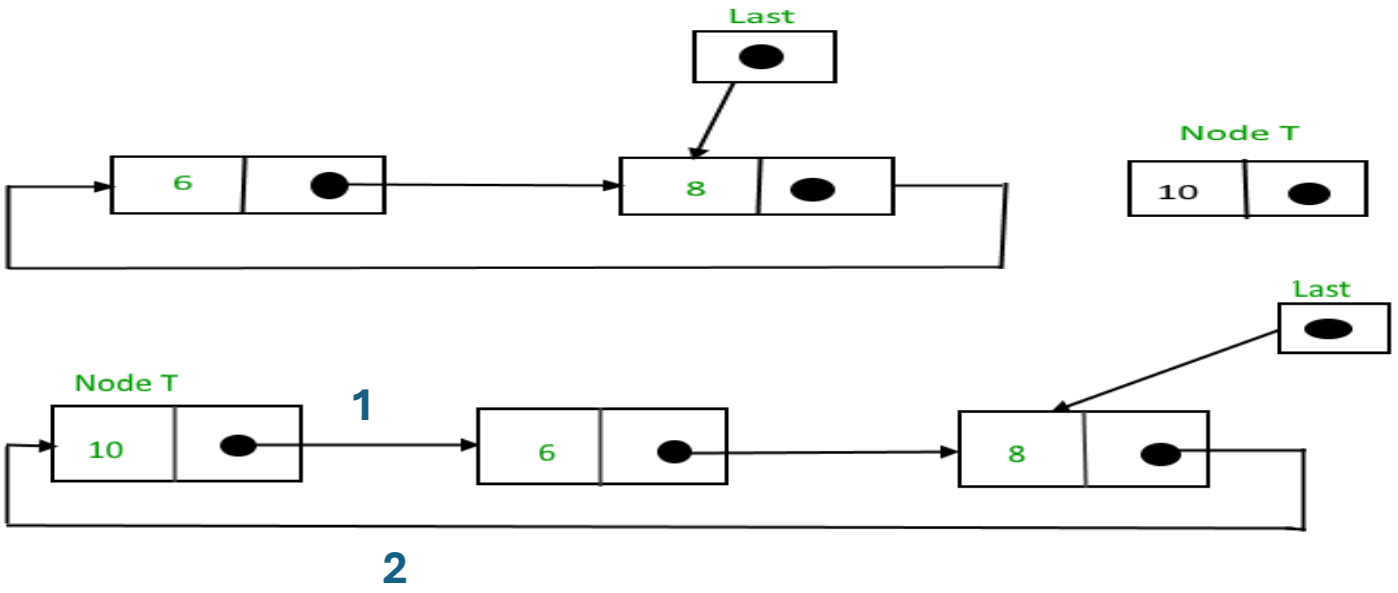
- *If linked list is not empty:*

Insertion at the beginning of the list

Follow these step:

1. Create a node, say tmp.
2. Make tmp-> next = last -> next.
3. last -> next = tmp.

LAST POINTER NOT SHIFTED !!



Insertion at the beginning of circular linked list

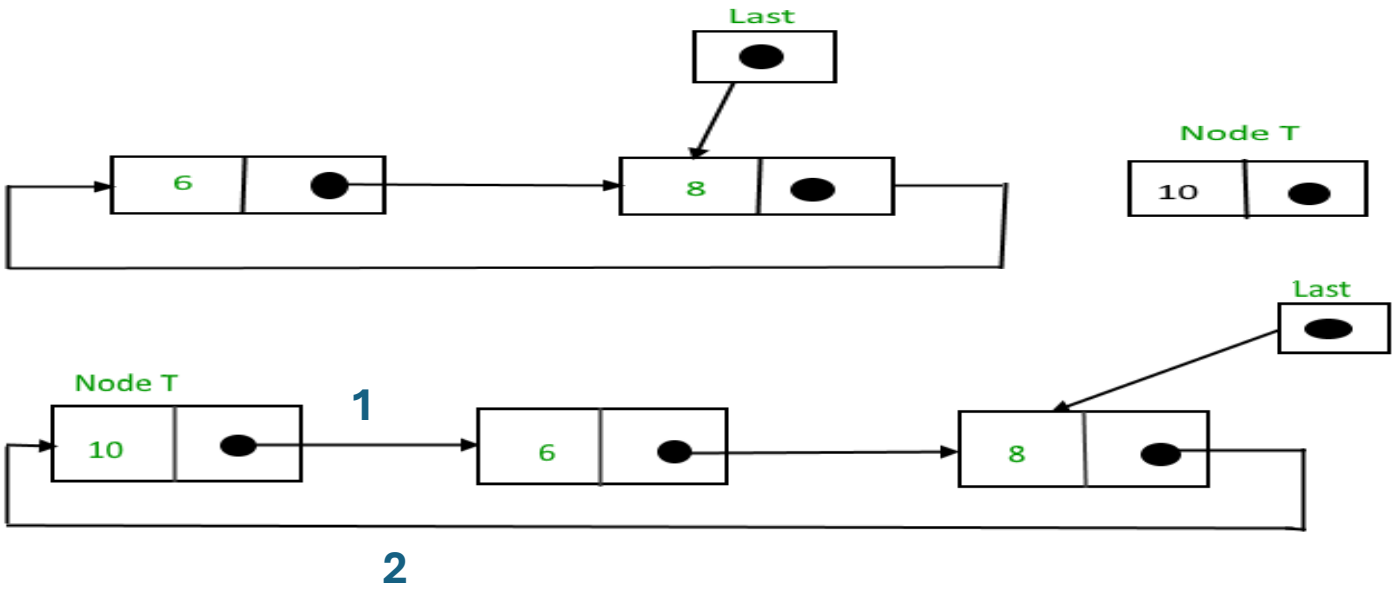
- *If linked list is not empty:*

Insertion at the beginning of the list

Follow these step:

1. Create a node, say tmp.
2. Make tmp-> next = last -> next.
3. last -> next = tmp.

LAST POINTER NOT SHIFTED !!



Insertion in between of circular linked list

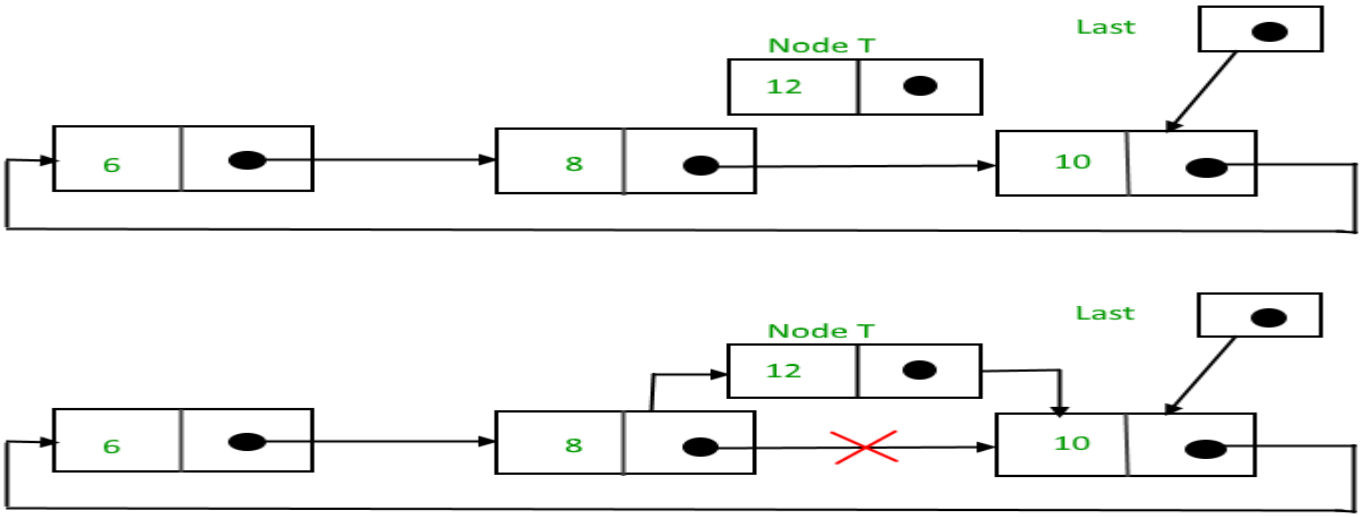
- *Insertion in between is same as in single linked list. This can be as-*

tmp->link = q->link;

tmp->info = num;

q->link =tmp;

- *Here q points to the node after which new node will be inserted.*



Insertion in between of circular linked list

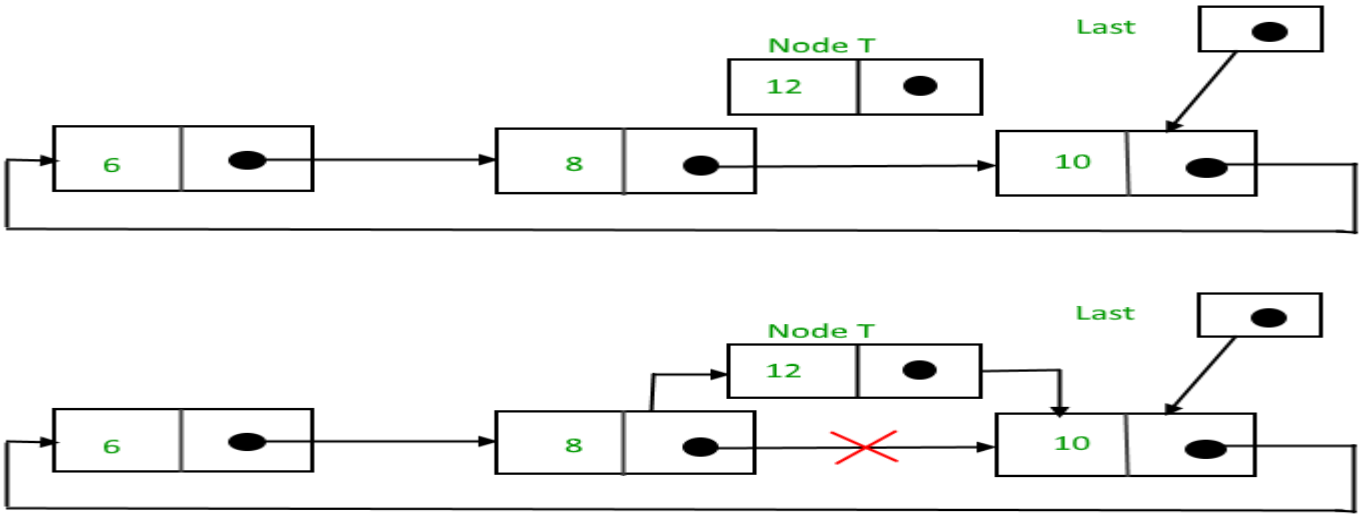
- *Insertion in between is same as in single linked list. This can be as-*

tmp->link = q->link;

tmp->info = num;

q->link =tmp;

- *Here q points to the node after which new node will be inserted.*



Creation of CLL

```
create_list(int num)
{
        struct node *q,*tmp;
        tmp= malloc(sizeof(struct node));
        tmp->info = num;

        if(last == NULL)
        {
                last = tmp;
                tmp->link = last;
        }
        else
        {
                tmp->link = last->link; /*added at the end of list*/
                last->link = tmp;
                last = tmp;
        }
}/*End of create_list( )*/
```