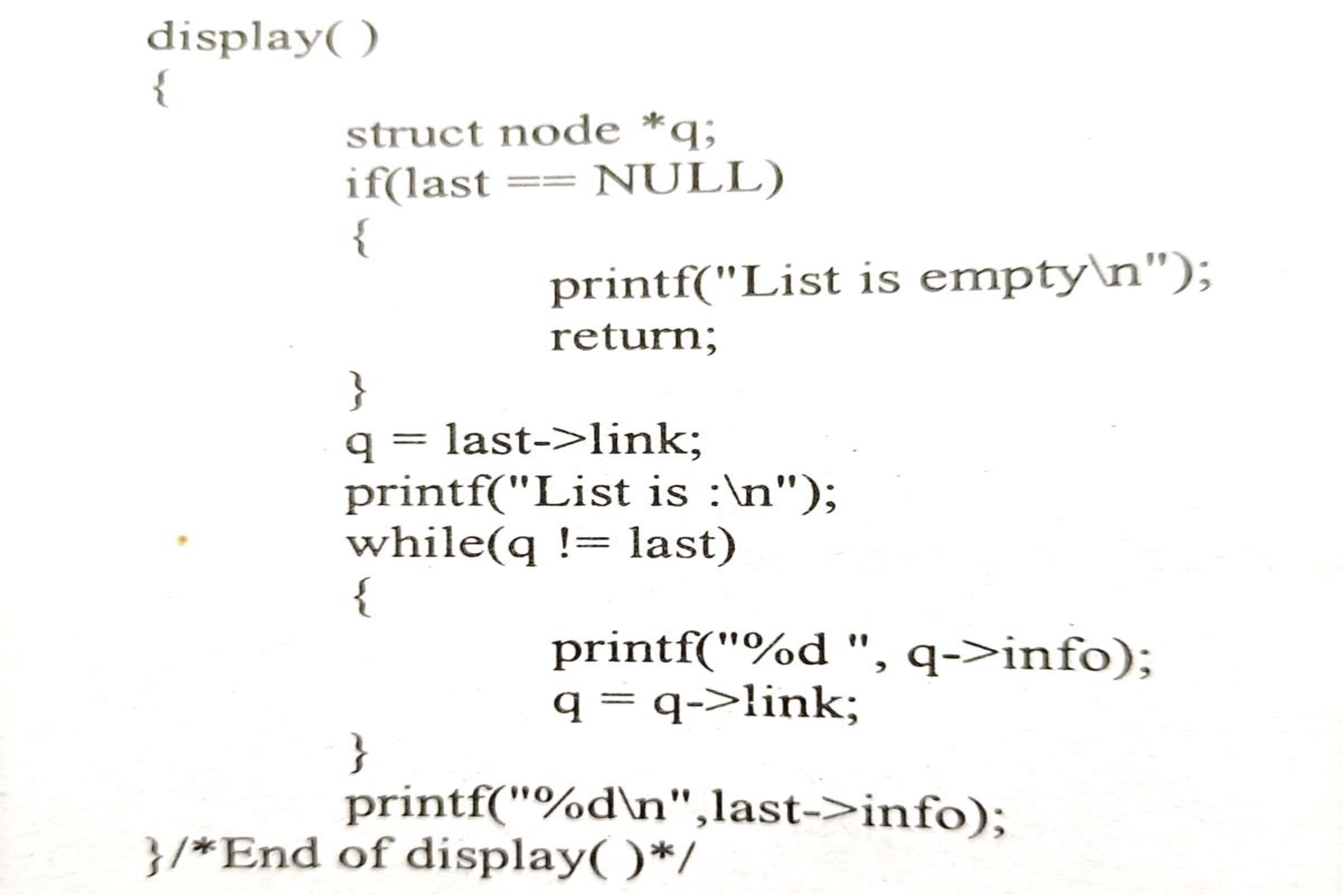
Traversal of circular linked list



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

Deletion from a circular linked list :-

Deletion in a circular linked list may be possible in four ways-

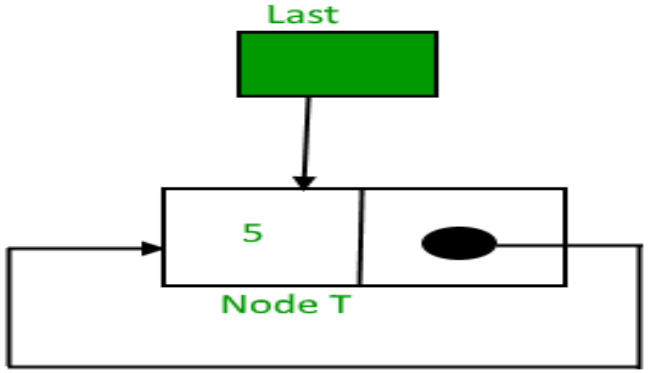
- ***If list has only one element***
- ***Node to be deleted is the first node of list***
- ***Deletion in between***
- ***Node to be deleted is last node of list***

Deletion from a circular linked list :-

Case 1:- ***If list has only one element***

- Here we check the condition for only one element of list
- then assign NULL value to last pointer because after deletion no node will be in list.

```
if(last->link == last && last->info == num) /* Only one
element */
{
        tmp = last;
       last = NULL;
       free(tmp);
}
```

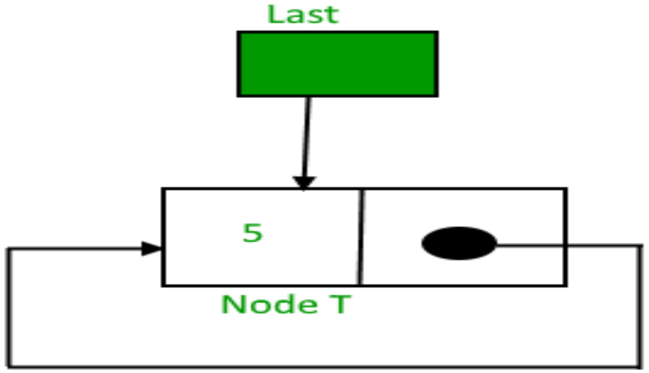


Deletion from a circular linked list :-

Case 1:- ***If list has only one element***

- Here we check the condition for only one element of list
- then assign NULL value to last pointer because after deletion no node will be in list.

```
if(last->link == last && last->info == num) /* Only one
element */
{
         tmp = last;
        last = NULL;
        free(tmp);
}
```

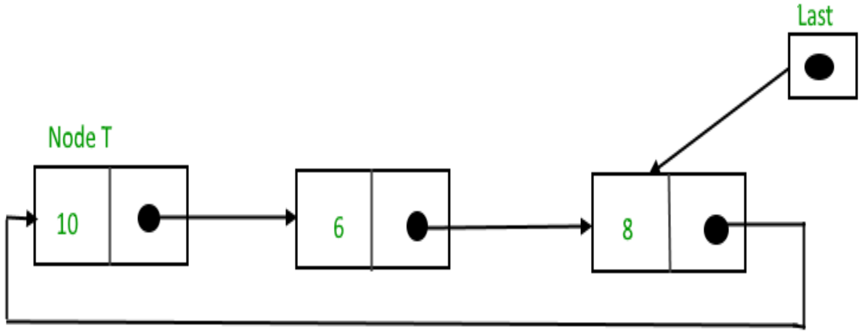


Deletion from a circular linked list :-

Case 2:- ***Node to be deleted is the first node of list***

- Assign the link part of deleted node to the link part of pointer last.
- So now link part of last pointer will point to the next node which is now first node of list after deletion.

```
q = last->link; /*q is pointing to the first node of list*/
if(q->info == num)
{
        tmp = q;
        last->link = q->link;
        free(tmp);
}
```



Deletion from a circular linked list :-

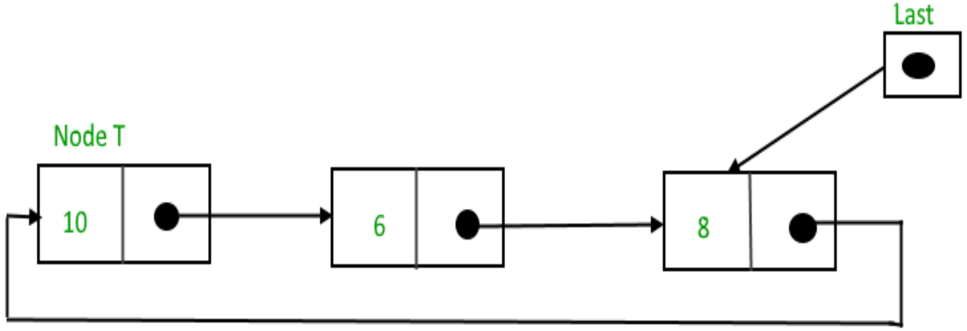
Case 3:- **Deletion in between is same as in single linked list.**

Deletion of node in between will be as-

```
q = last->link;
while(q->link != last)
{
        if(q->link->info == num) /* Element deleted in between */
        {
                tmp = q->link;
                q->link = tmp->link;
                free(tmp);
        }
        q = q->link;
} /* End of while */
```

- First we are traversing the list, when we find the element to be deleted,
- then q points to the previous node.
- We assign the link part of node to be deleted to the link part of previous node and
- then we free the address of node to be deleted from memory.

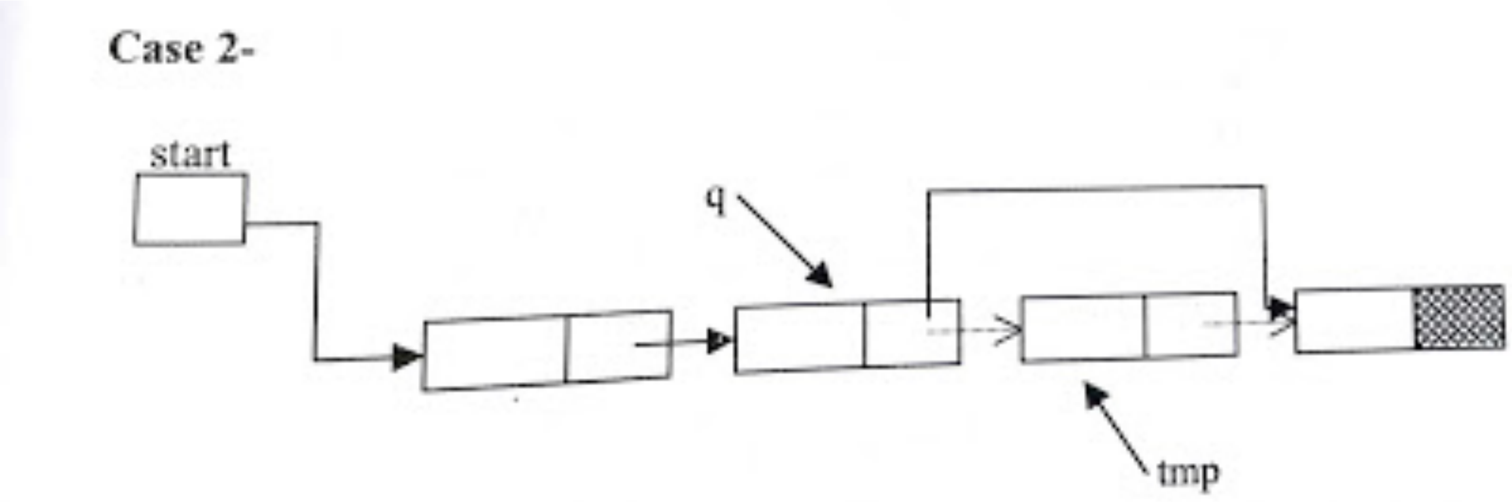
As we need to stop at the previous node for deletion



Same as Deletion in between in Singly Linked List

- 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);
```

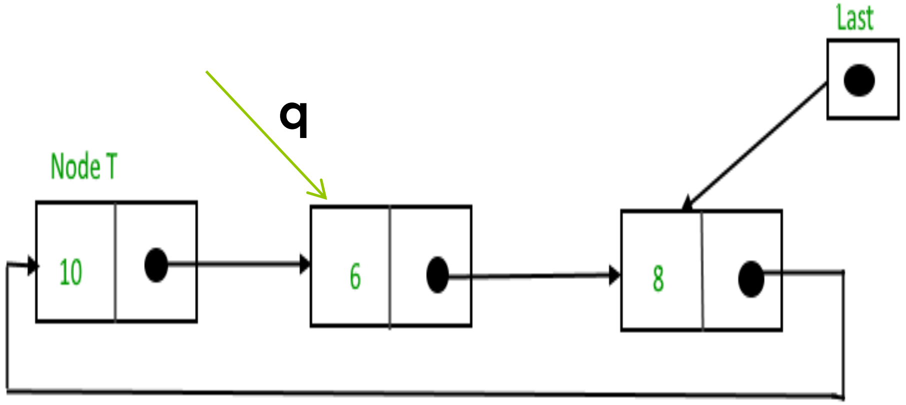


Deletion from a circular linked list :-

Case 4:- **Deletion of last node**

- Assign the link part of last node to the link part of previous node.
- So link part of previous node will point to the first node of list.
- Then assign the value of previous node to the pointer variable last **because after deletion of last node , pointer variable last should point to the previous node.**

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



Doubly Linked List

Doubly Linked List

Drawback of single linked list-

- 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 singly linked list and
  - **To do operation with just previous node then we have no way to go on previous node,**
  - We will again traverse from starting node.

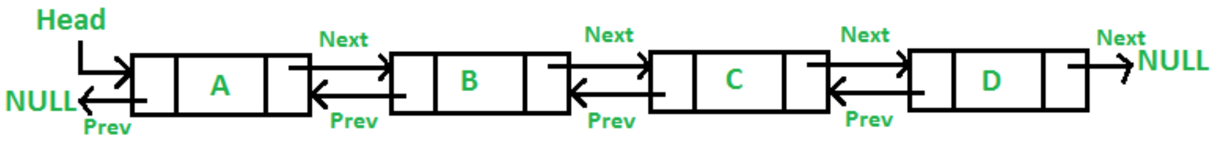
Doubly Linked List

Drawback of single linked list-

Solution-

- Doubly linked list, in this each node has address of both previous and next node.

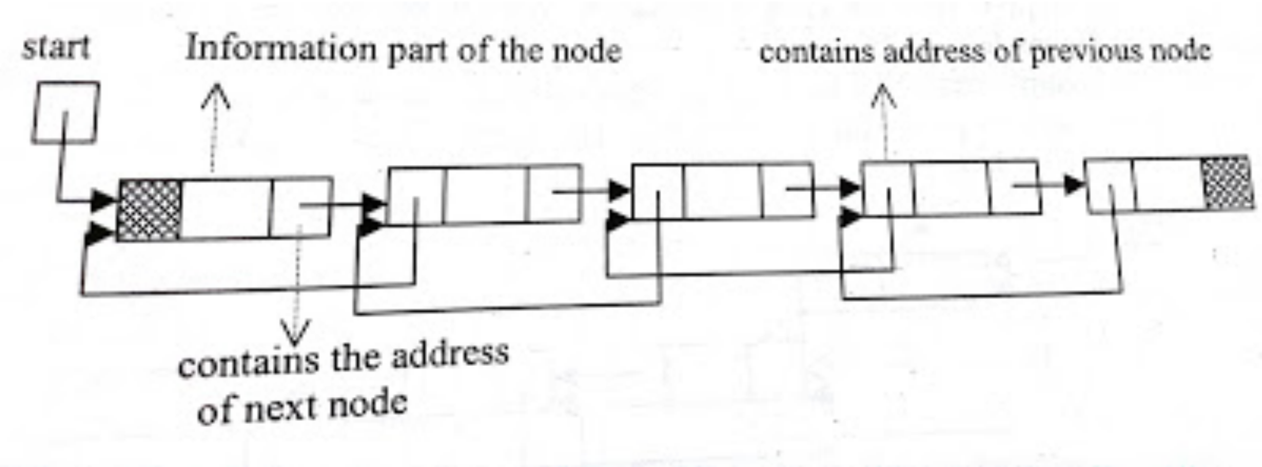
Doubly Linked List



Doubly Linked List

The data structure for doubly linked list will be as-

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


start
Information part of the node
contains address of previous node
contains the address of next node

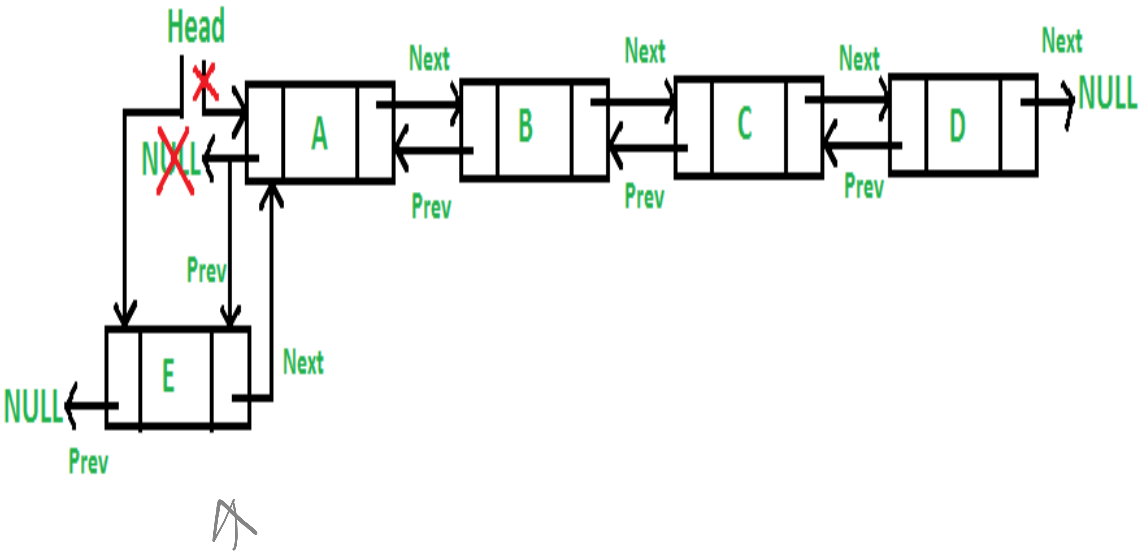
Doubly Linked List

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

- *struct node *previous is a pointer to structure, which will contain the address of previous node*
- *struct node * next will contain the address of next node in the list.*
- *Traversal in both directions at any time.*

Doubly Linked List-Insertion at beginning

A 5 steps process

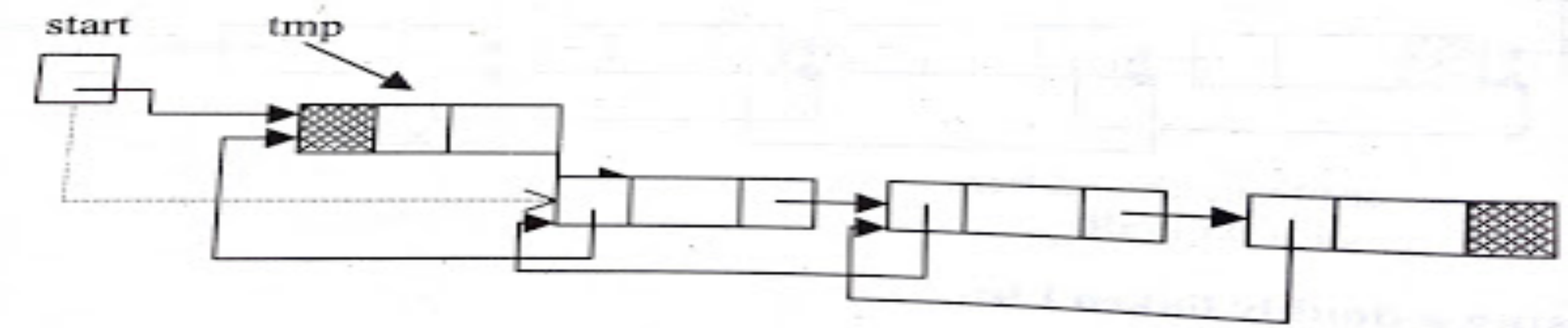


Doubly Linked List-Insertion at beginning

- Start points to the first node of doubly linked list.
- Assign the value of start to the next part of inserted node and address of inserted node to the prev part of start as-

1) **tmp->next=start;**
2) **tmp->info=data**
3) **start->prev = tmp;**

1) Now inserted node points to the next node, which was beginning node of the doubly linked list and
3) prev part of second node will point to the new inserted node. Now inserted node is the first node of the doubly linked list.



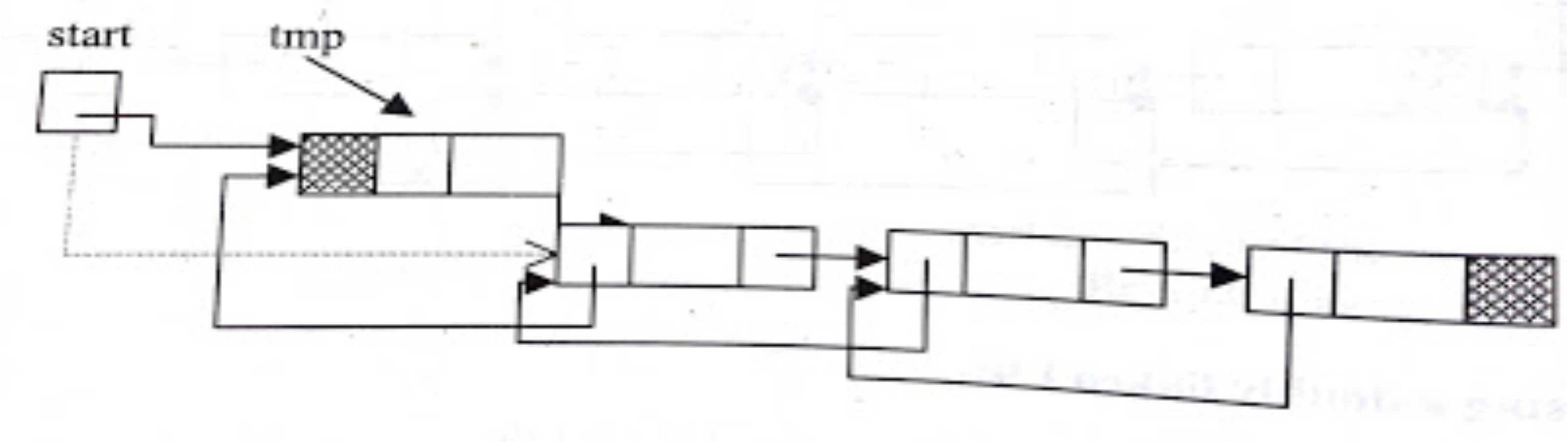
Doubly Linked List-Insertion at beginning

- So start will be reassigned as-

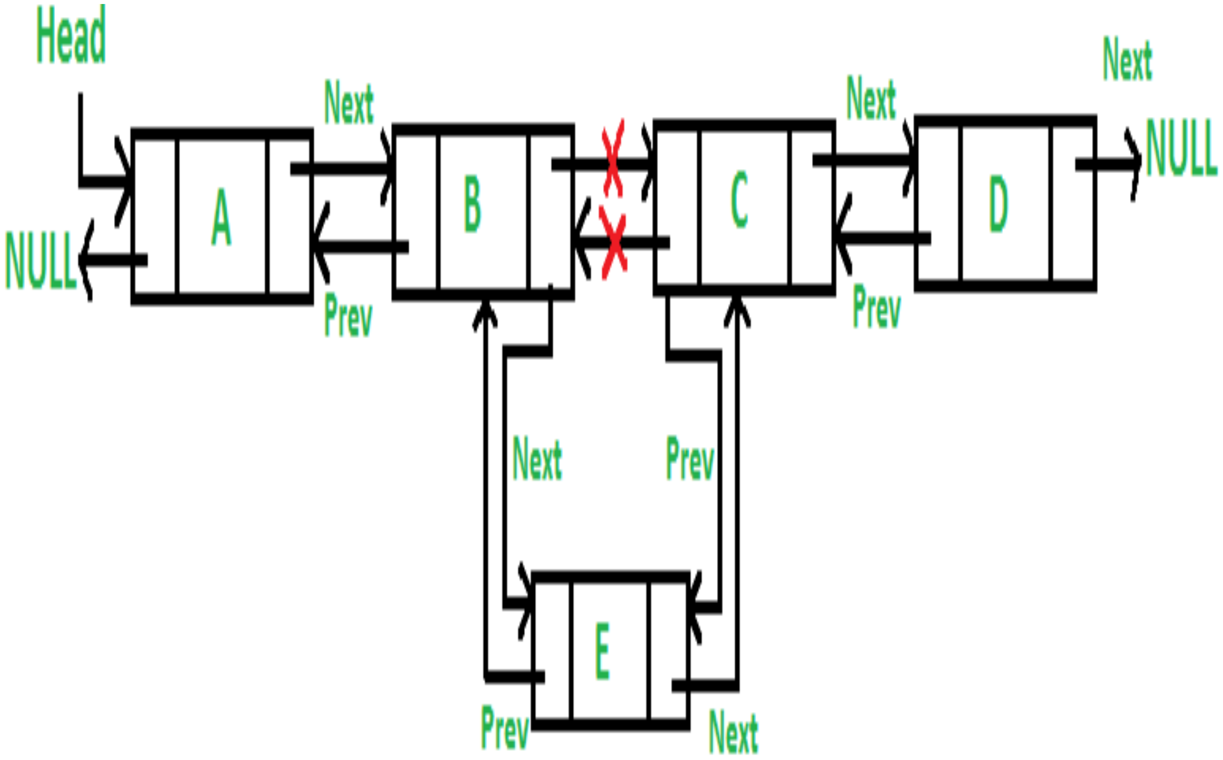
  start= tmp;

- 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;



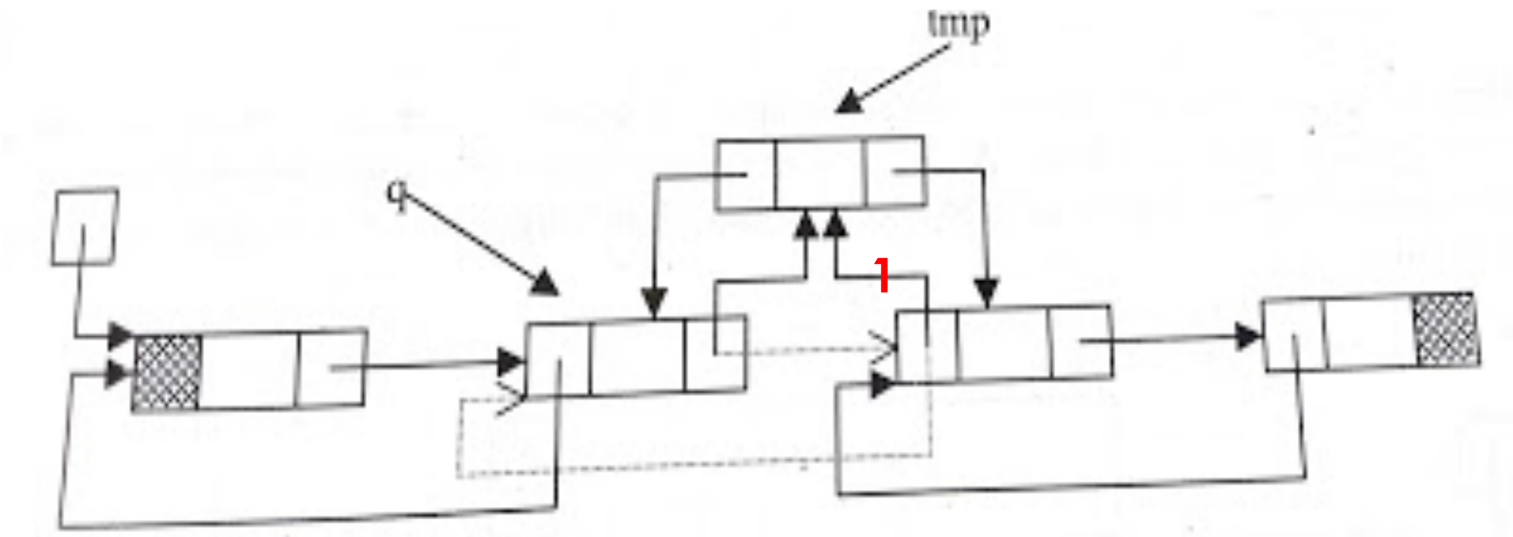
Doubly Linked List-Insertion in between



Doubly Linked List-Insertion in between

- *Traverse to obtain the* ***node (q) after which we want to insert the element.***
- *Assign the address of* ***inserted node(tmp****) to the prev part of next node.*

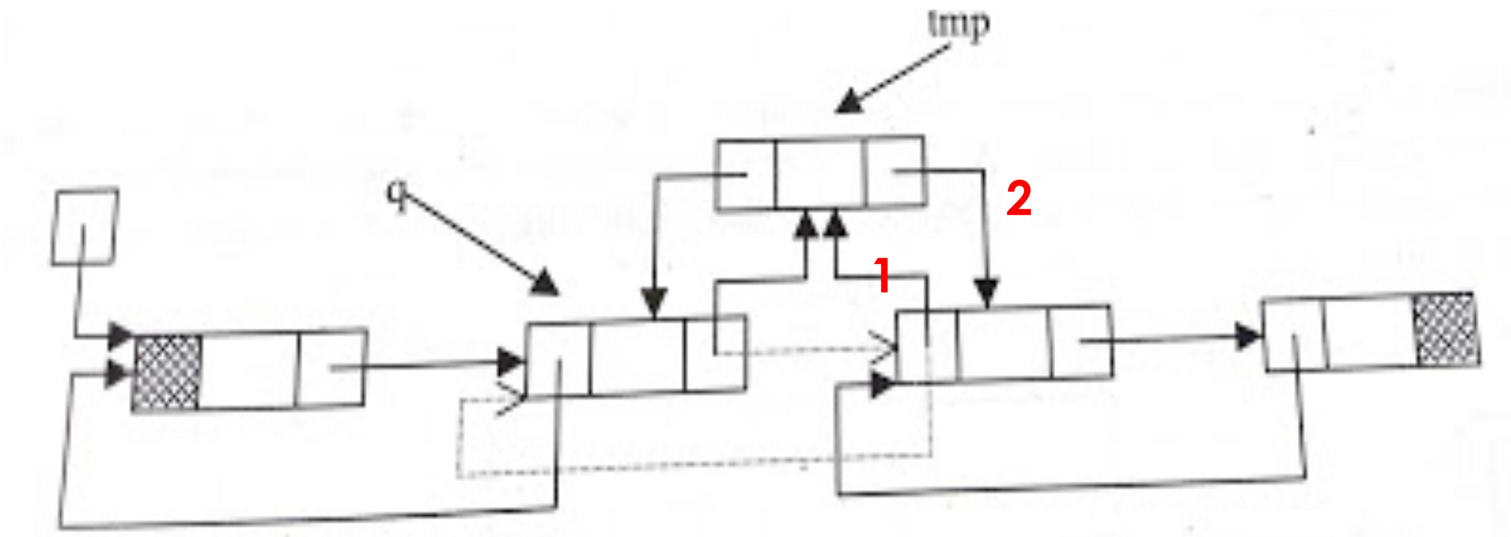
q->next->prev=tmp;



Doubly Linked List-Insertion in between

- *Assign the next part of previous node to the next part of inserted node.*

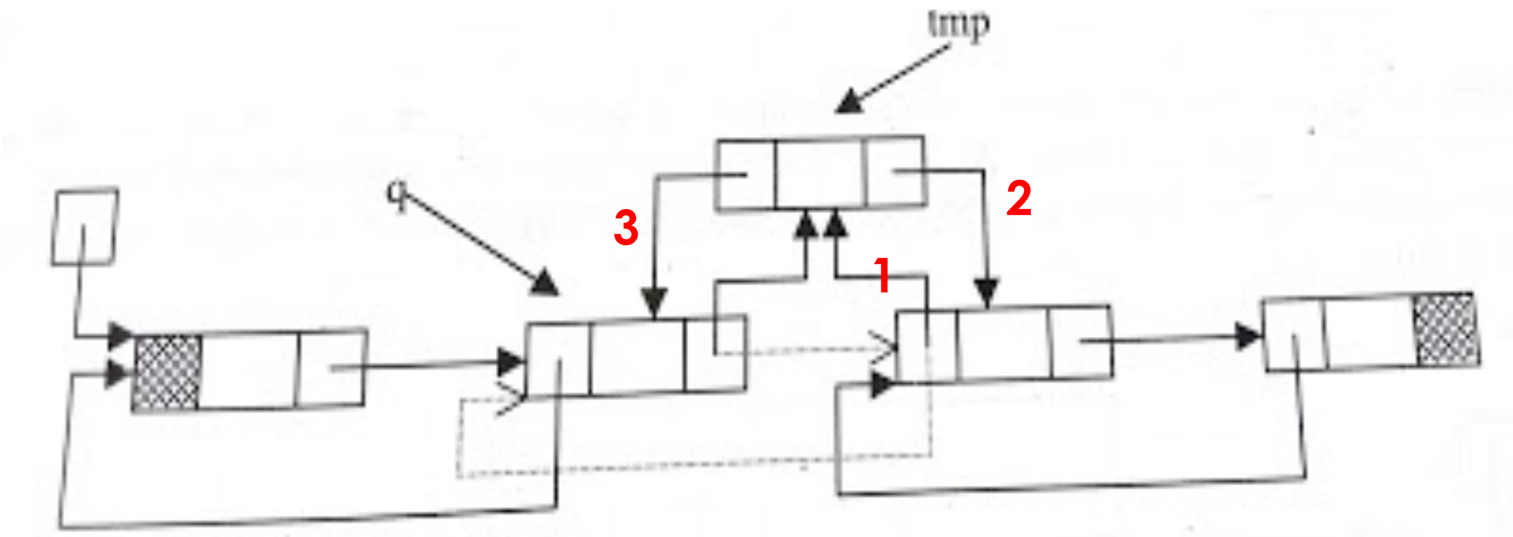
  tmp->next=q->next;



Doubly Linked List-Insertion in between

- *Address of previous node will be assigned to prev part of inserted node*

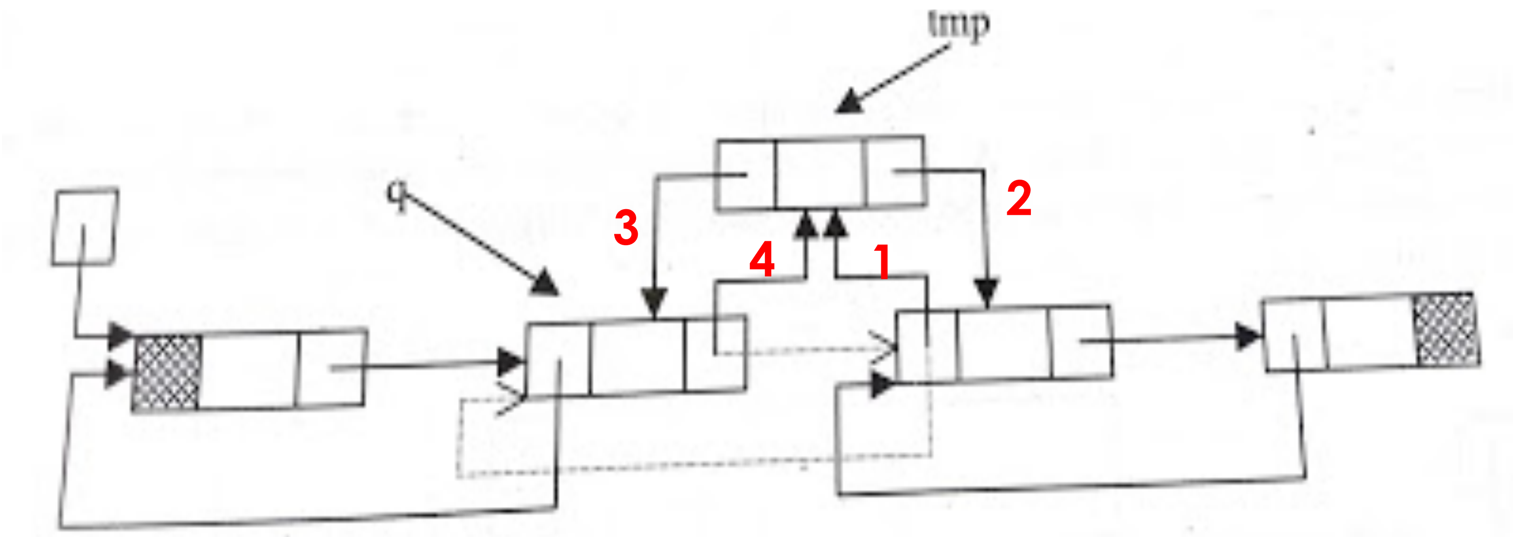
  tmp->prev=q;



Doubly Linked List-Insertion in between

- *Address of inserted node will be assigned to next part of previous node.*

  q->next=tmp;



Doubly Linked List-Insertion in between

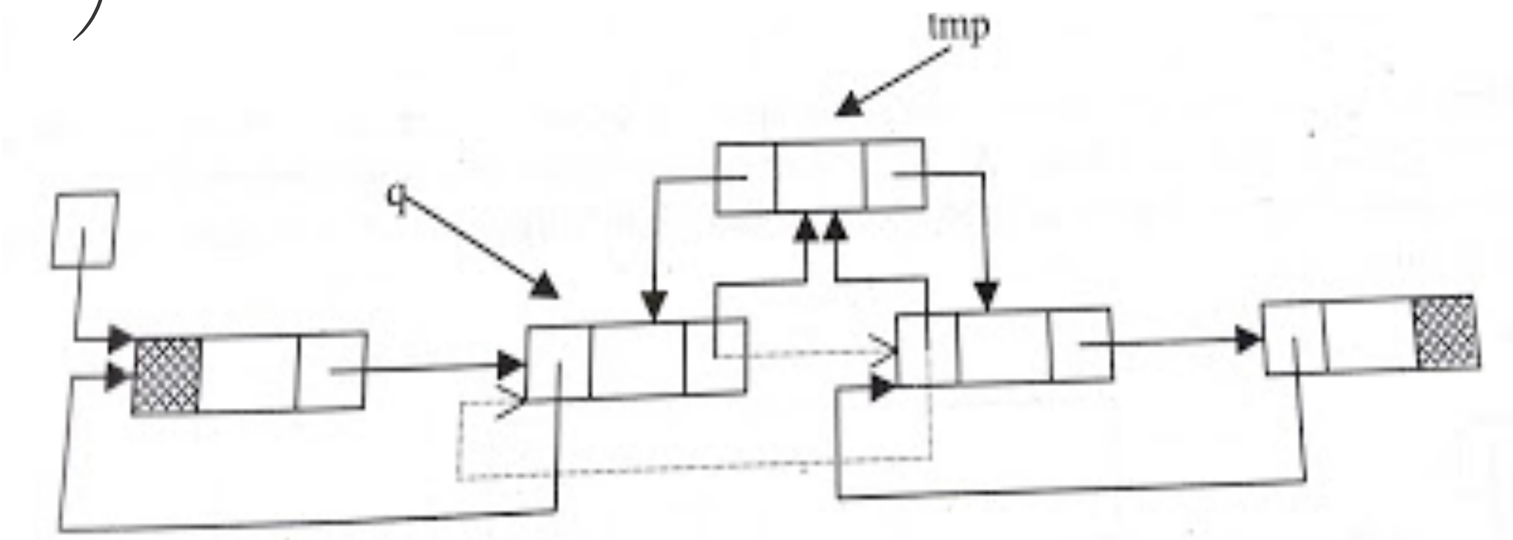
- *Traverse to obtain the* ***node (q) after which we want to insert the element.***
- *Assign the address of* ***inserted node(tmp****) to the prev part of next node.*

  1) ***q->next->prev=tmp;***
- *Assign the next part of previous node to the next part of inserted node.*

  2) ***tmp->next=q->next;***
- *Address of previous node will be assigned to prev part of inserted node*

  3) ***tmp->prev=q;***
- *Address of inserted node will be assigned to next part of previous node.*

  4) ***q->next=tmp;***



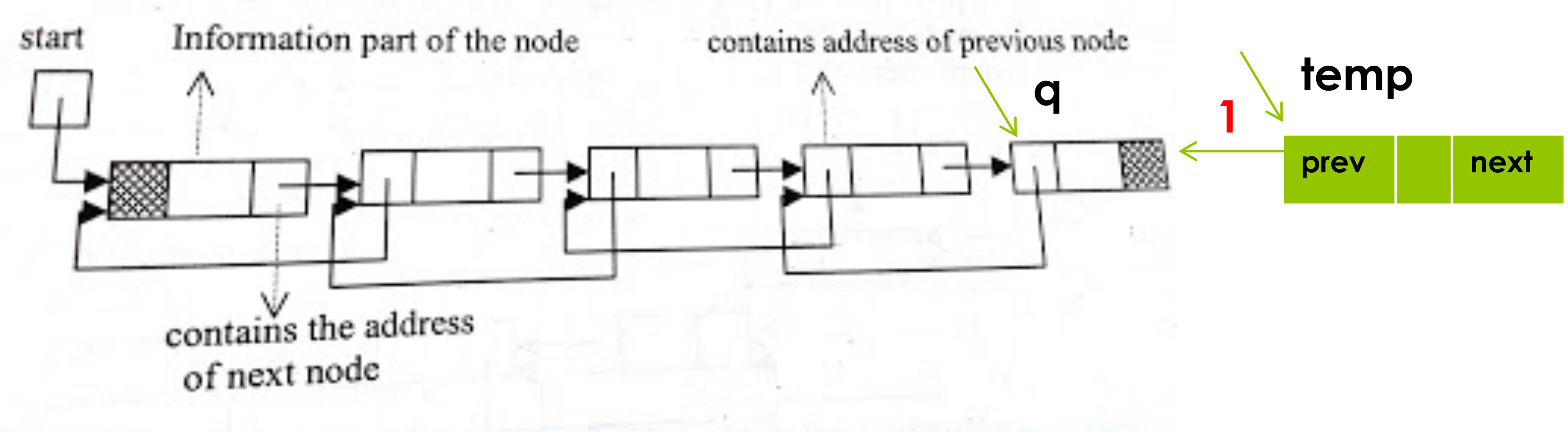
Doubly Linked List-Insertion at the last

- *Traverse to obtain the* ***node (q) after which we want to insert the element.***

  tmp->next=NULL;

- *Address of previous node will be assigned to prev part of inserted node*

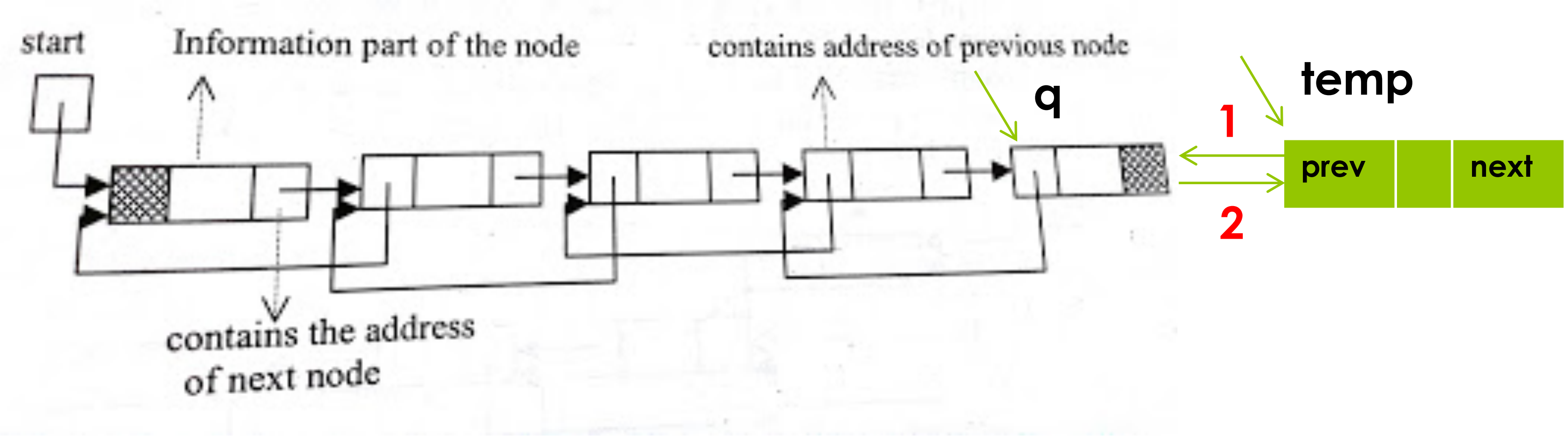
  tmp->prev=q;


start
Information part of the node
contains address of previous node
q
1
temp
prev
next
contains the address of next node

Doubly Linked List-Insertion at the last

- *Address of inserted node will be assigned to next part of previous node.*

  q->next=tmp;

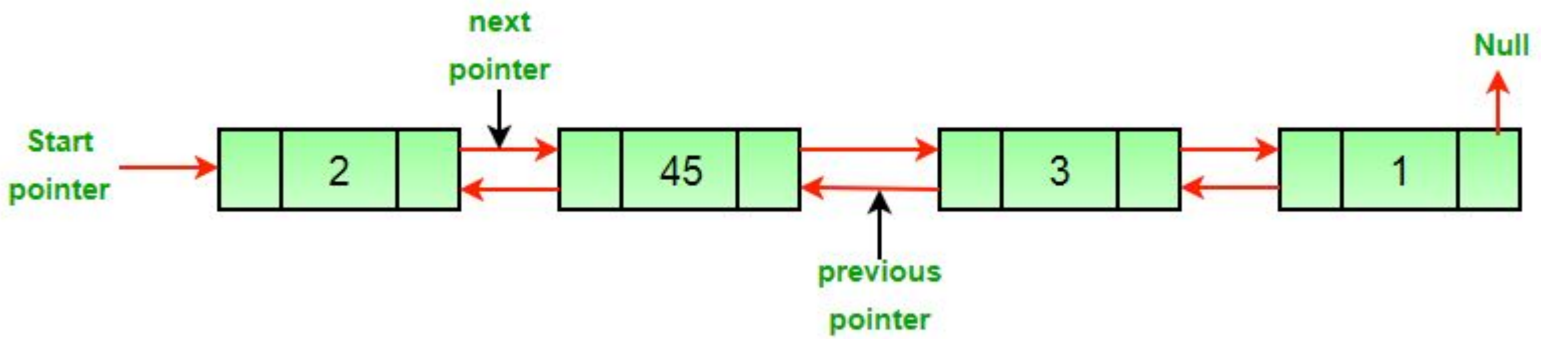

start
Information part of the node
contains address of previous node
q
temp
1
2
prev
next
contains the address of next node

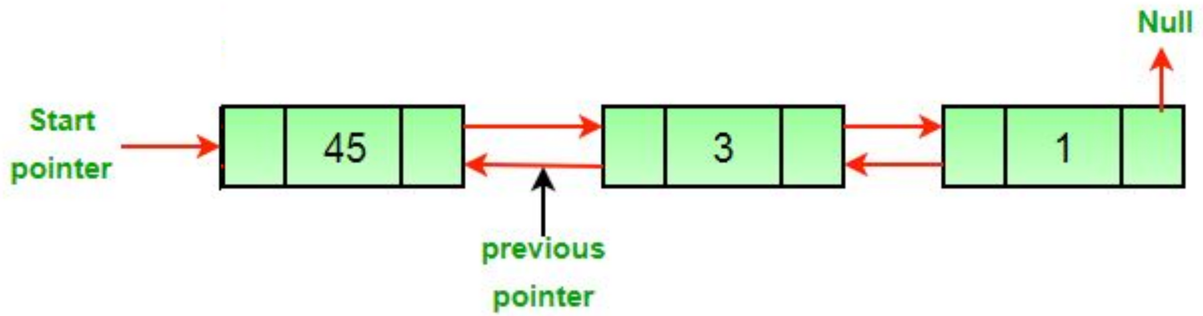
Deletion from doubly linked list

Traverse the linked list and compare with each element. After finding the element there may be three cases for deletion-

- **Deletion at beginning**
- **Deletion in between**
- **Deletion of last node**

Deletion from doubly linked list-Deletion at beginning





Doubly linked list-Deletion at beginning

- *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;

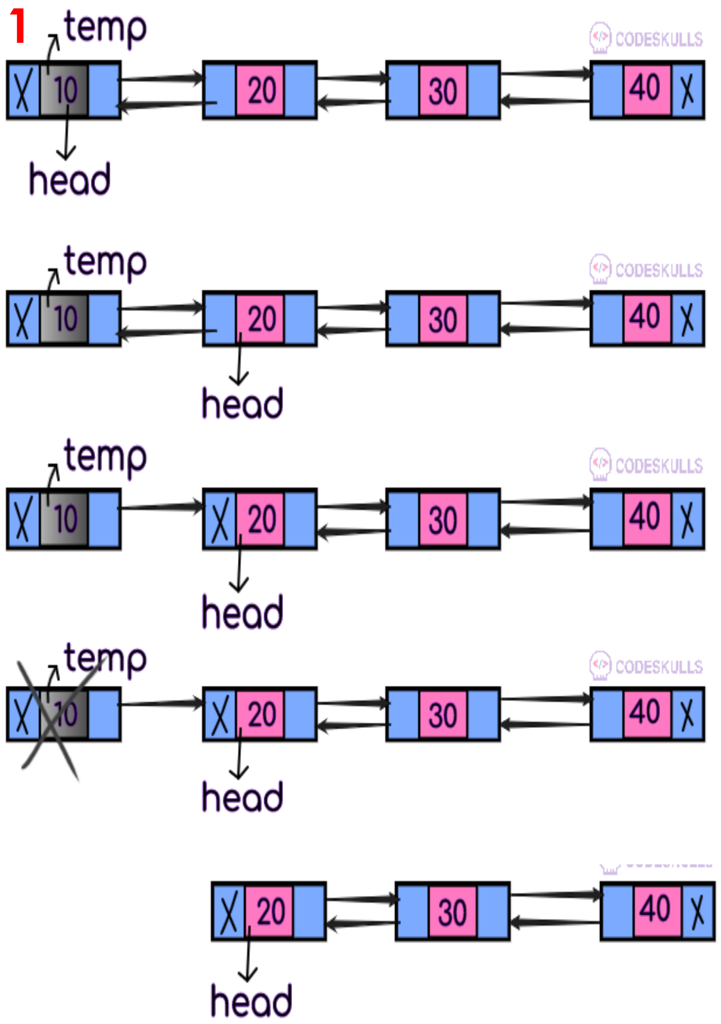
- *Since start points to the first node of linked list, so start->next will point to the second node of list.*
- *Then NULL will be assigned to start->prev.*

  start->prev = NULL;

- *Now we should free the node to be deleted which is pointed by tmp.*

  free(tmp);

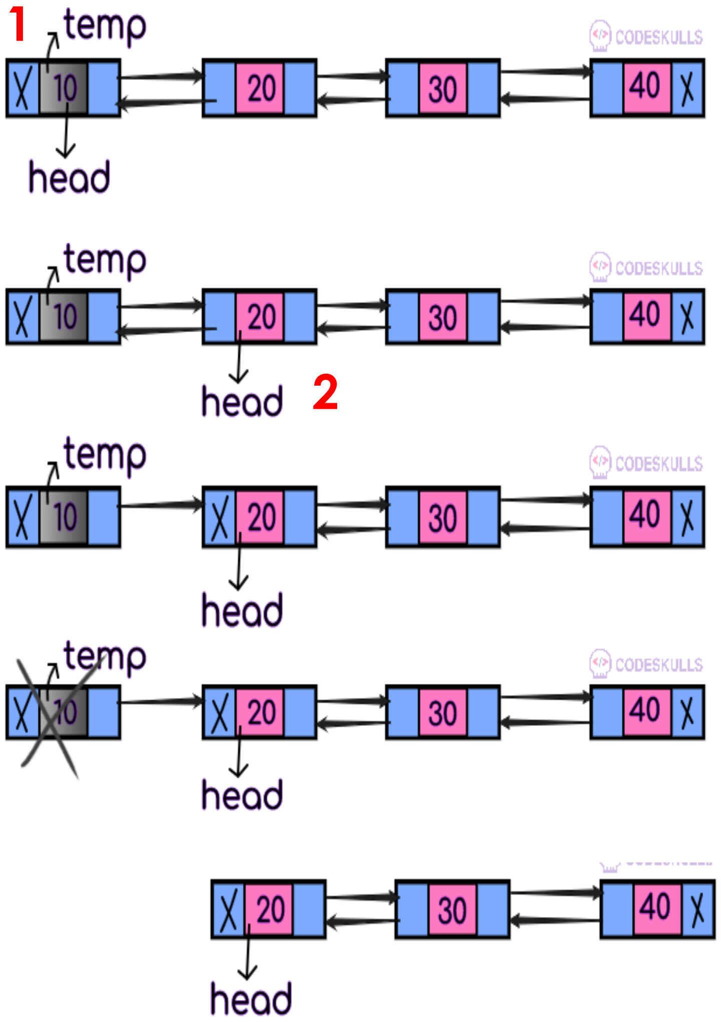
Doubly linked list-Deletion at beginning



- *Assign the value of start to tmp as-*

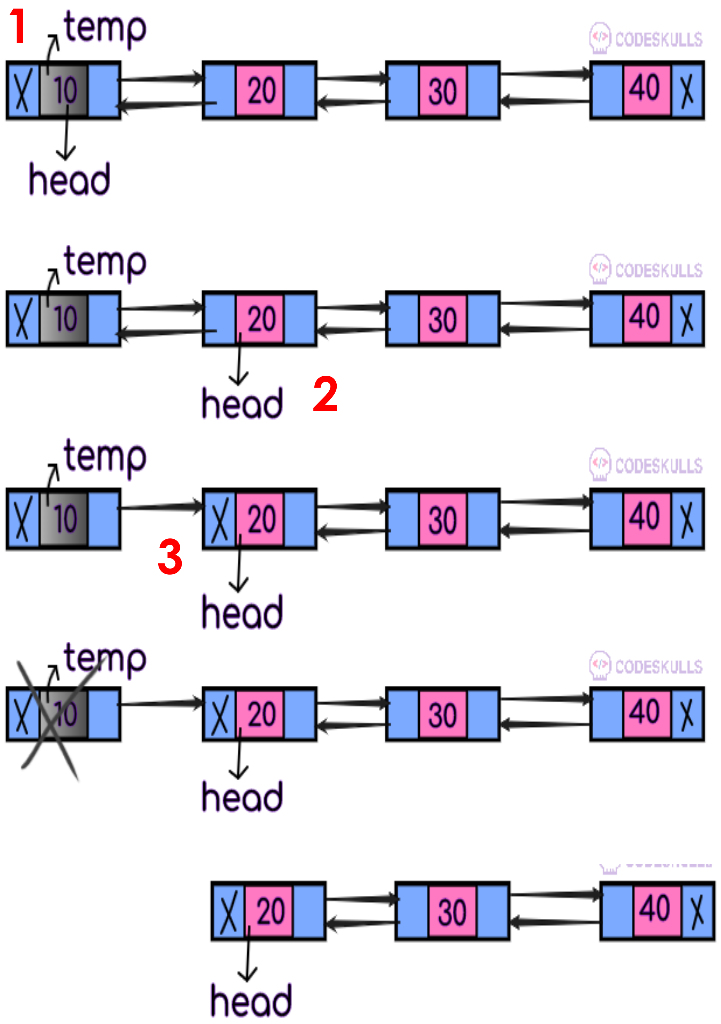
  1)tmp = start;

Doubly linked list-Deletion at beginning



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

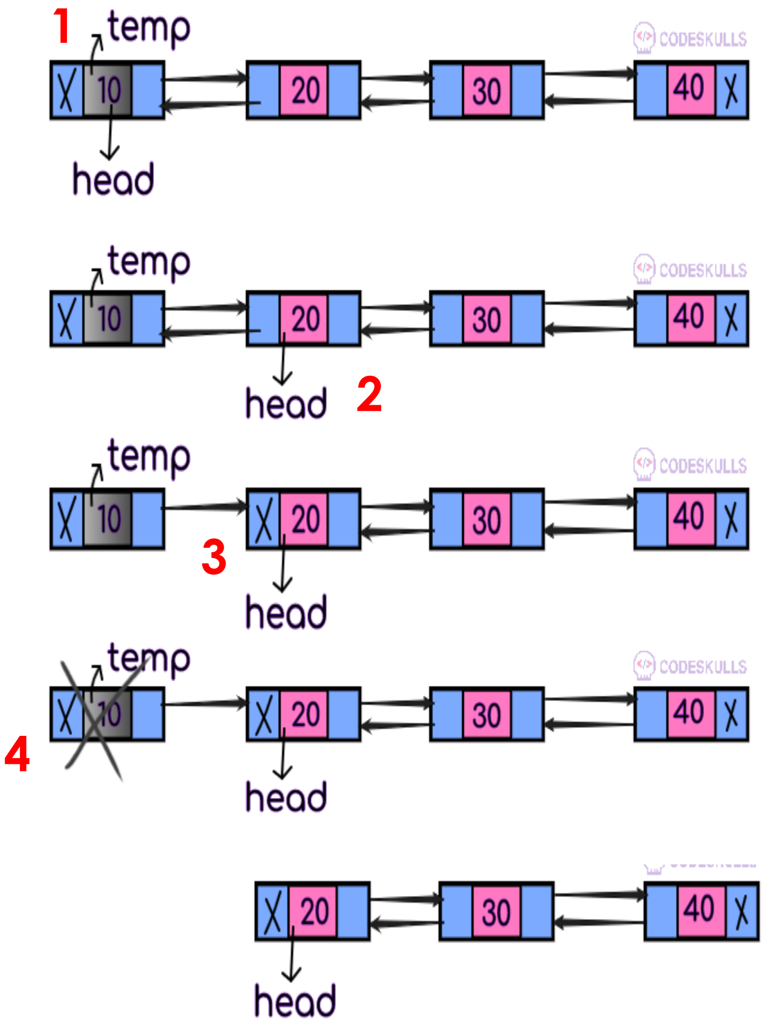
  2)start=start->next;



- *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.*

3)start->prev = NULL;

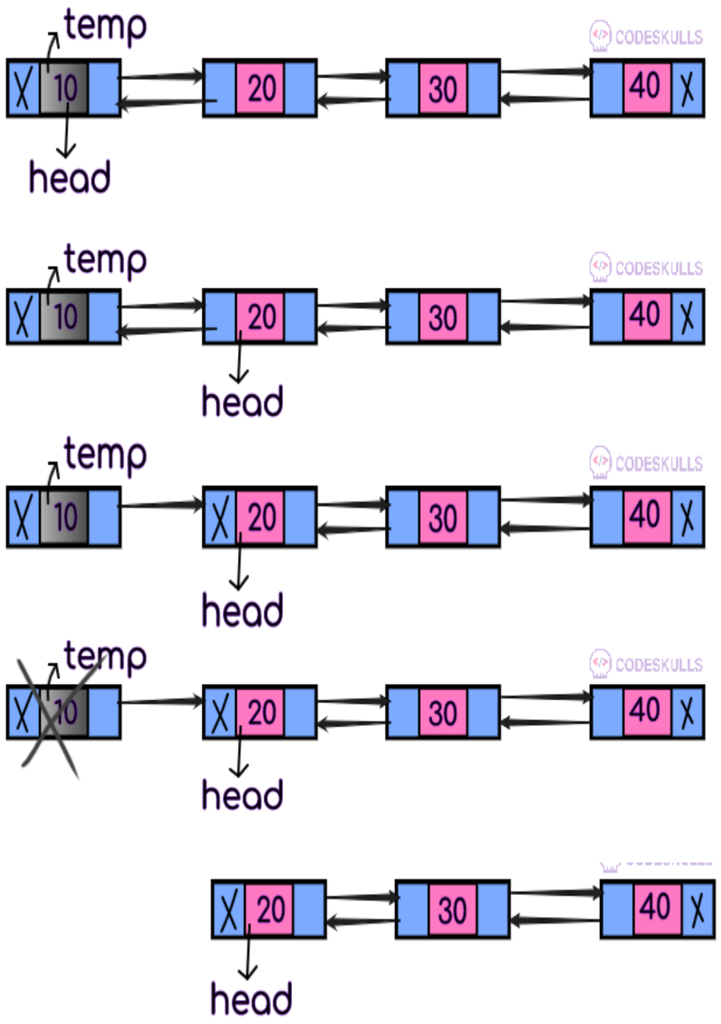
Doubly linked list-Deletion at beginning



- *Now we should free the node to be deleted which is pointed by tmp.*

4)free(tmp);

Doubly linked list-Deletion at beginning



- *Assign the value of start to tmp as-*

  1)tmp = start;

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

  2)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.*

  3)start->prev = NULL;

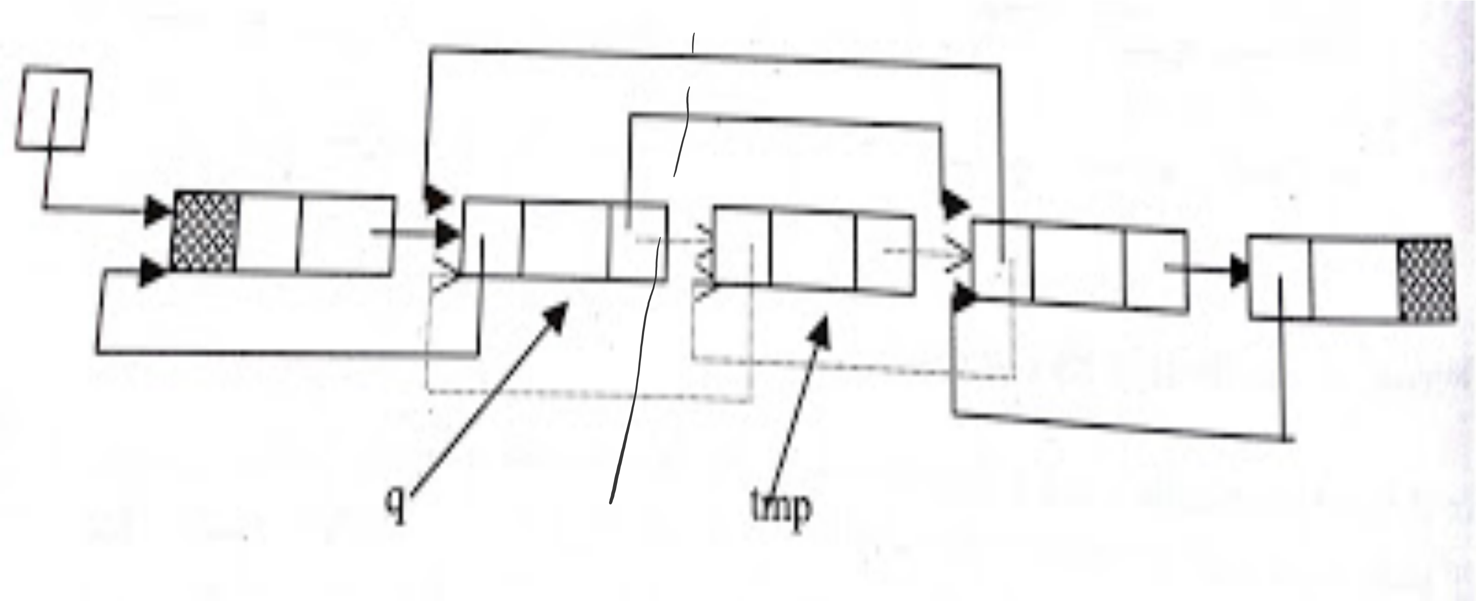
- *Now we should free the node to be deleted which is pointed by tmp.*

  4)free(tmp);

Deletion from doubly linked list-Deletion in between

- ***Concept-***
- *We assign the next part of the deleted node to the next part of the previous node*
  - *address of the previous node to prev part of next node.*

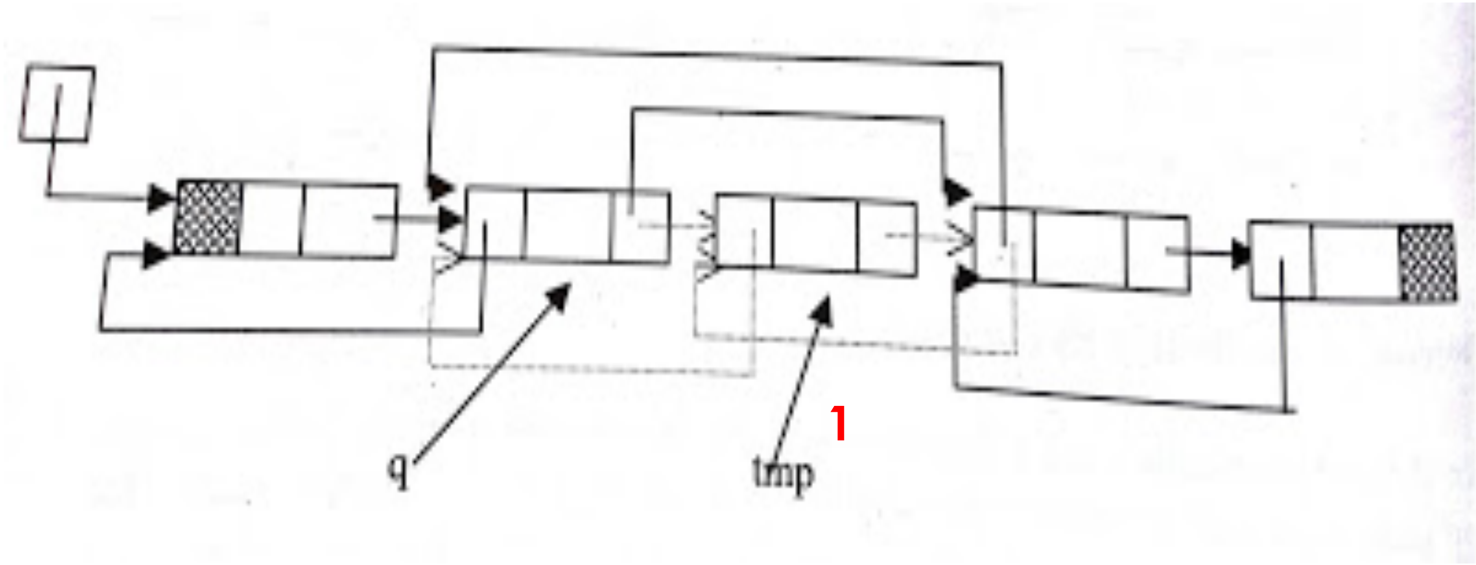
MrQ



Deletion from doubly linked list-Deletion in between

1) ***tmp=q->next;***

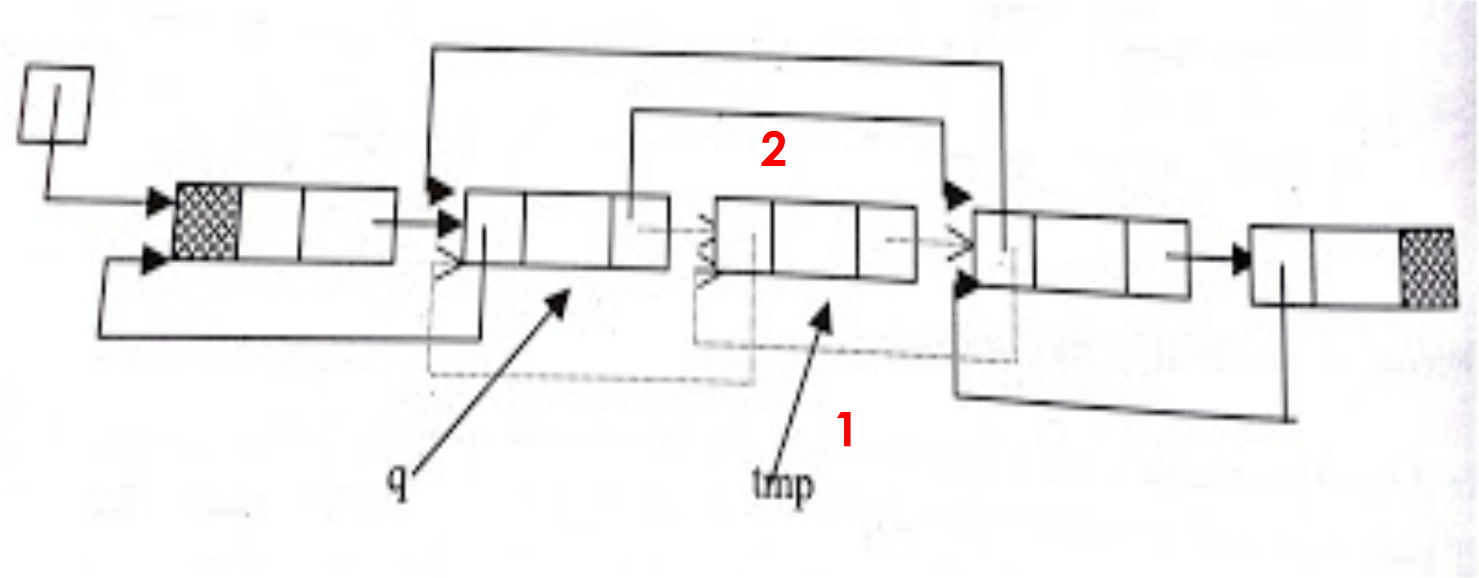
- *Here q is pointing to the previous node of node to be deleted.*
  - *After statement 1 tmp will point to the node to be deleted.*



Deletion from doubly linked list-Deletion in between

1) ***tmp=q->next;***
2) ***q->next=tmp->next;***

- *After statement 2 next part of previous node will point to next node of the node to be deleted*

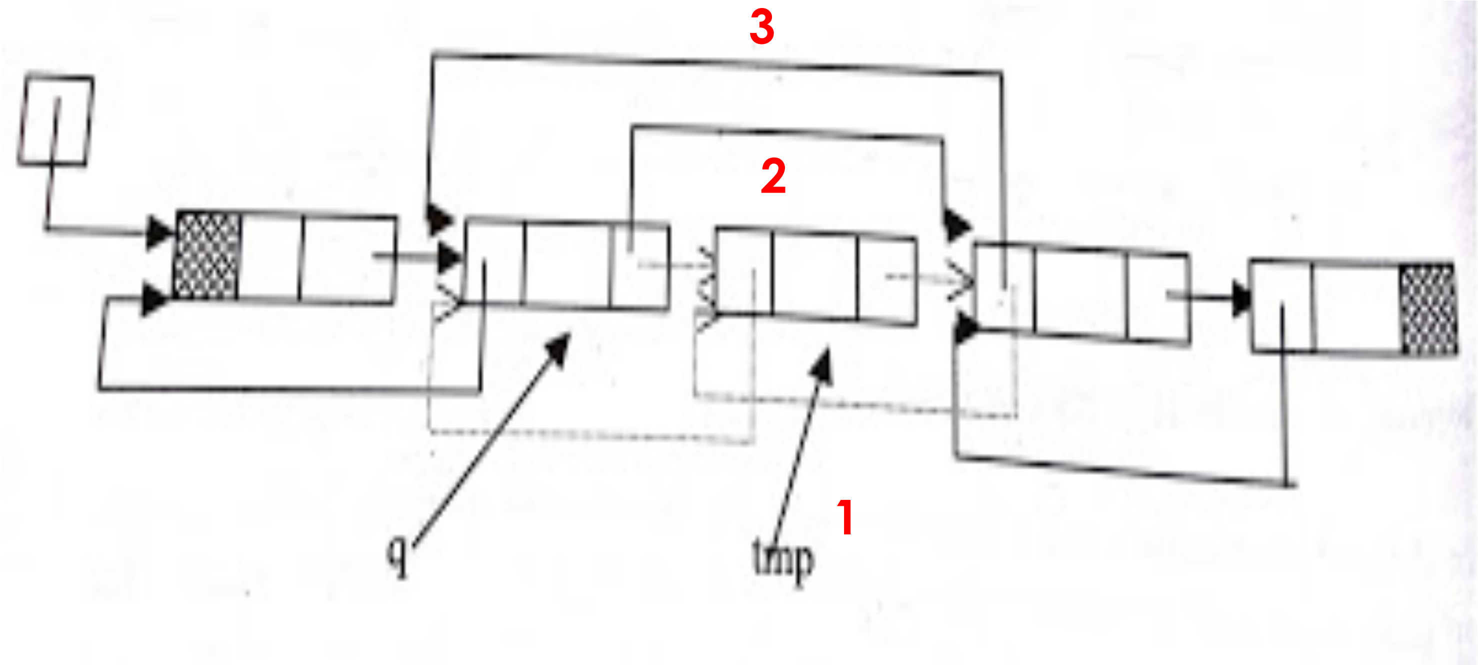


Deletion from doubly linked list-Deletion in between

1

1) ***tmp=q->next;***
2) ***q->next=tmp->next;***
3) ***tmp->next->prev=q;***

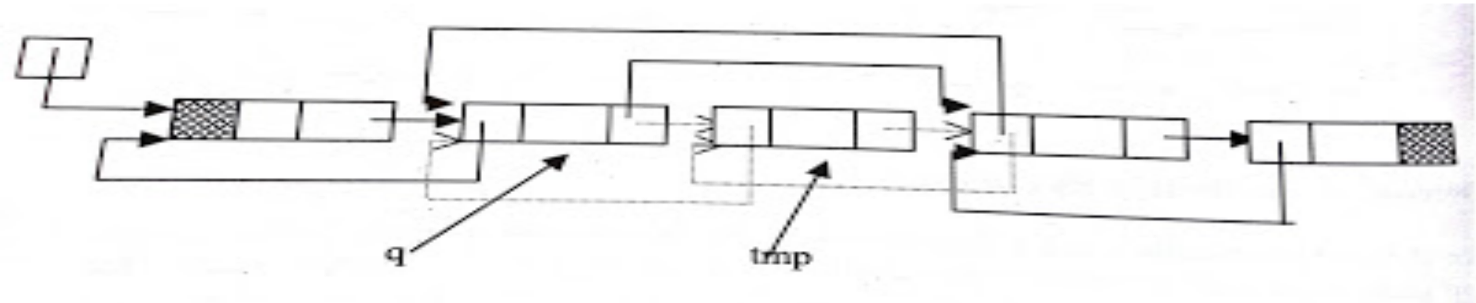
- *After statement 3 prev part of next node will point to previous node.*



Deletion from doubly linked list-Deletion in between

1) ***tmp=q->next;***
2) ***q->next=tmp->next;***
3) ***tmp->next->prev=q;***
4) ***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*
  - *After statement 3 prev part of next node will point to previous node.*

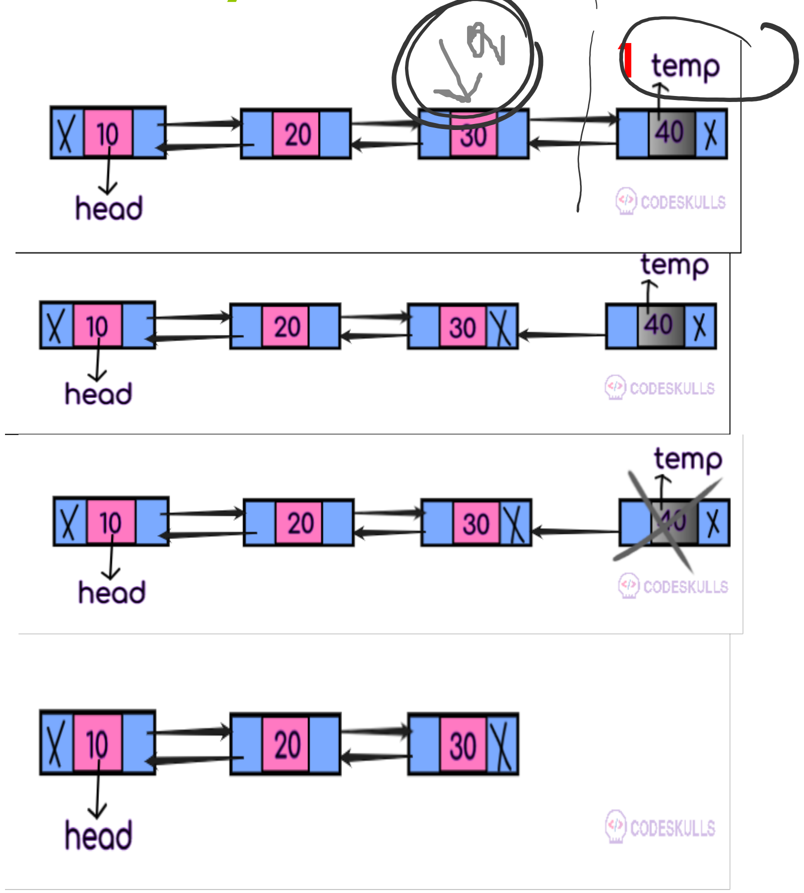


Doubly linked list-Deletion of last node

Concept-

- *If node to be deleted is last node of doubly linked list then*
  - *we will just free the last node*

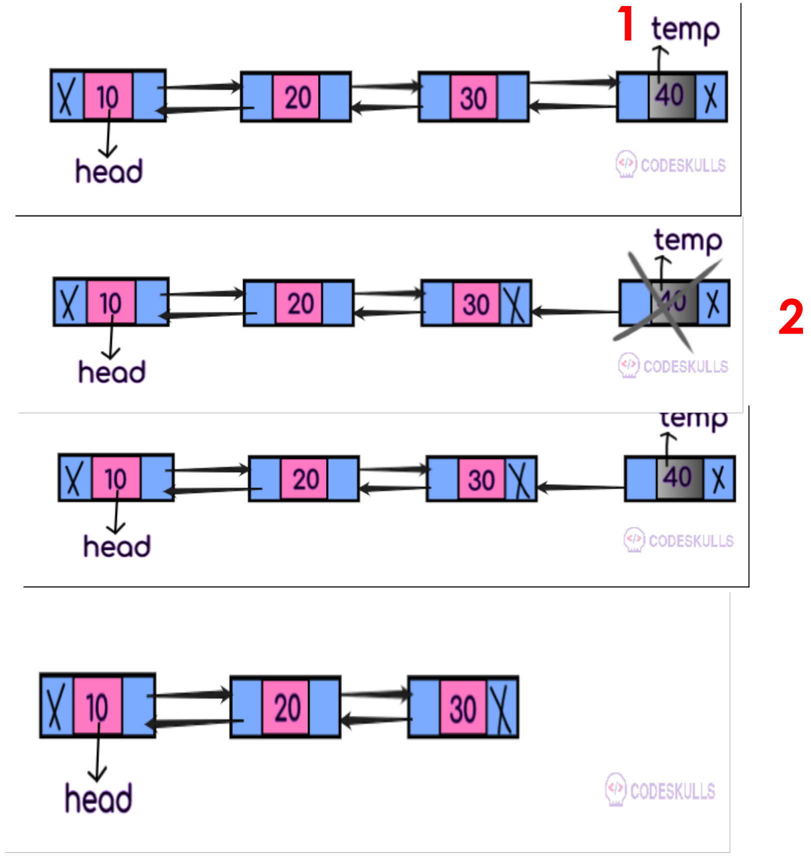
Doubly linked list-Deletion of last node



1) tmp=q->next;

- *Here q is second last node*
- *After statement 1, tmp will point to last node*

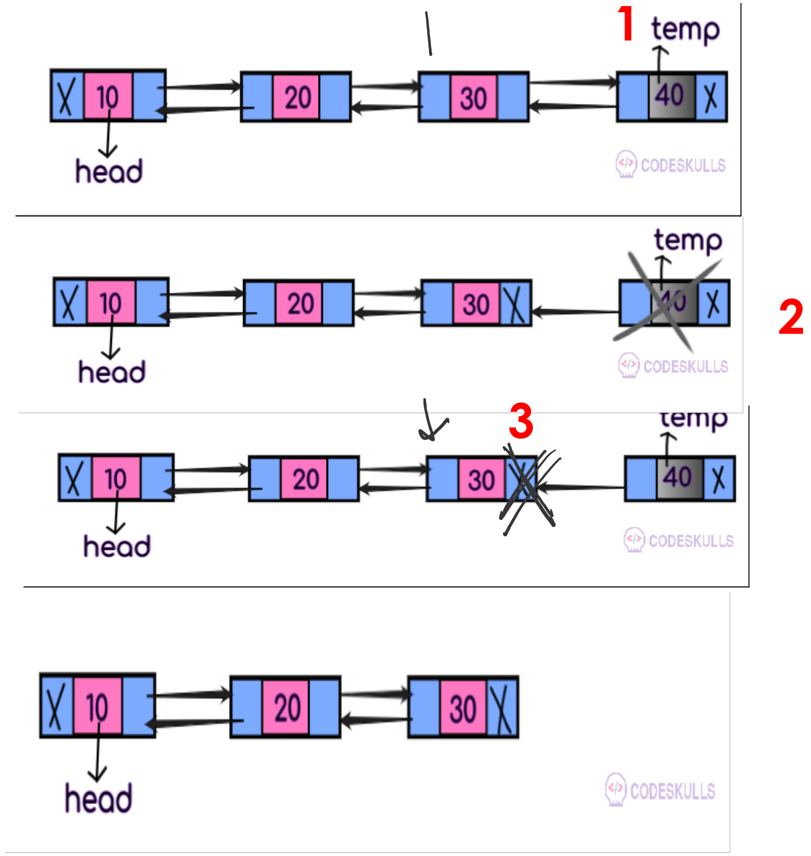
Doubly linked list-Deletion of last node



2) free(tmp);

- *After statement 2, last node will be deleted*

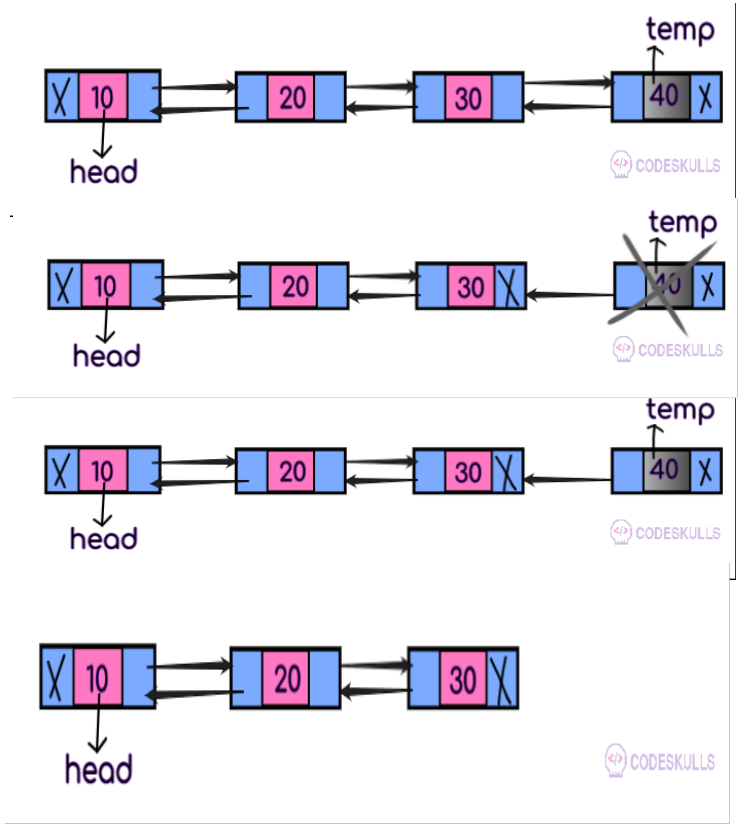
Doubly linked list-Deletion of last node



- *Next part of second last node i.e q will be NULL.*

3) q->next=NULL;

Doubly linked list-Deletion of last node



- *If node to be deleted is last node of doubly linked list then*
  - *we will just free the last node and*

    tmp=q->next;

    free(tmp);
  - *next part of second last node will be NULL.*

    q->next=NULL;
- *Here q is second last node*
- *After statement 1, tmp will point to last node*
- *After statement 2 last node will be deleted and after statement 3 second last node will become the last node of list.*

Application of Linked List

- ***Polynomial Representation and Addition***

Polynomial arithmetic with linked list

- *Linked list can be used*
  - *to represent polynomial expression*
  - *for arithmetic operations also.*

Representation of Polynomial

- *Let us take a polynomial expression-*

$$\mathbf{5x^4 + x^3 - 6x + 2}$$

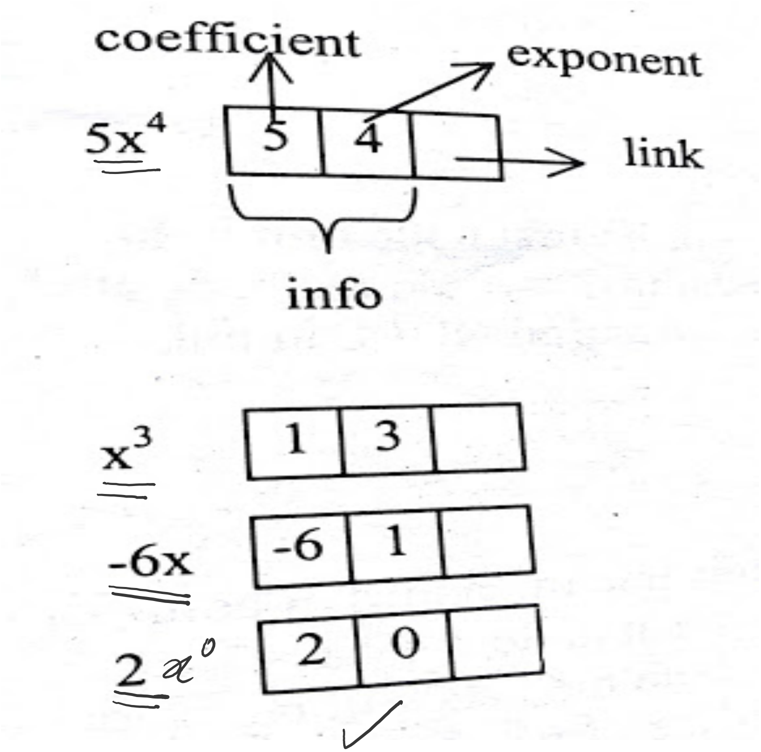
Representation of Polynomial

- *Let us take a polynomial expression-*

  $$\mathbf{5x^4 + x^3 - 6x + 2}$$

- *Here we can see every symbol x is attached with two things,*
  - *coefficient*
  - *exponent.*

Representation of Polynomial



- *As in $5x^4$, coefficient is 5 and exponent is 4.*

- ***$5x^4 + x^3 - 6x + 2$***

- ***Each node of list will contain the coefficient and exponent of each term of polynomial expression.***

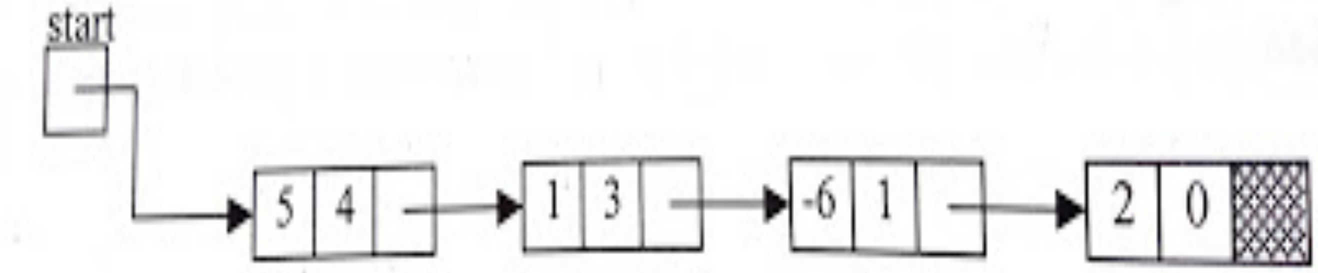
Representation of Polynomial

- *So the data structure for polynomial expression will be-*

```
struct node{
            int coefficient;
            int exponent;
            struct node *link;
        }
```

- ***Descending sorted linked list is used based on the exponent***
- *As it will be easier for arithmetic operation with polynomial linked list.*
- *Otherwise, we have a need to traverse the full list for every arithmetic operation.*

- *Now we can represent polynomial expression*
- ***5x*^{4} + x^3 - *6x* + 2** *as-*



Creation of polynomial linked list

- *Creation of polynomial linked list will be same as*
  - *creation of sorted linked list but*
  - ***it be in descending order and based on exponent of symbol.***

Creation of polynomial linked list

- ***Insertion at the Start***

- *In if condition we are checking for the node to be added will be first node or not.*

```
/* list empty or exp greater than first one */
if(start==NULL || ex> start->expo)
{
            tmp->link=start;
            start=tmp;
            return start;
}
```

Creation of polynomial linked list

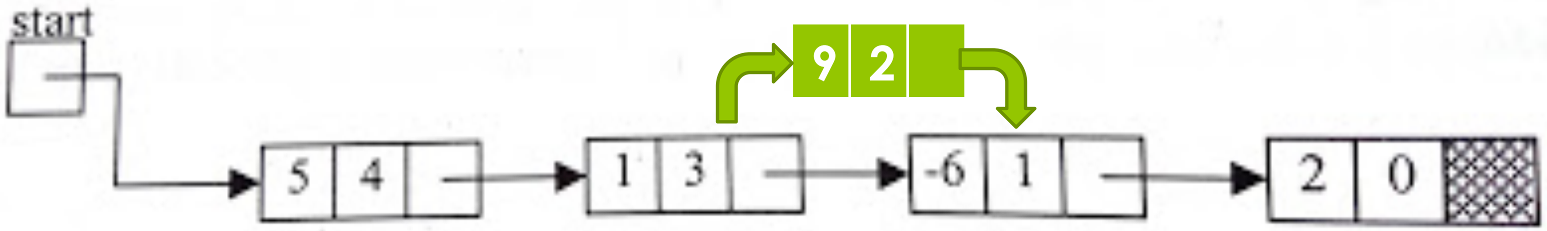
```
else
{
            ptr=start;
           while(ptr->link!=NULL && ptr-
>link->expo > ex)
                  {
                        ptr=ptr->link;
                 }
          tmp->link=ptr->link;
          ptr->link=tmp;



            if(ptr->link==NULL)  /* item to
be added in the end */
                     tmp->link=NULL;
}
```

- *In else part*
  - *we traverse the list and*
  - *check the condition for exponent then*
  - *we add the node at proper place in list.*

- ***If node will be added at the end of list then***
  - *we assign NULL in link part of added node.*

Creation of polynomial linked list

Lets insert 9x^2
Let ex=2
Ptr=1st Node
Check if ptr->link->expo > ex
Check if 2nd node's expo >ex
If 3>2, yes
Then
Ptr=ptr->link
Ptr=2nd node
Check if ptr->link->expo>ex
Check if 3rd node's expo>ex
If 1>2, no
Insert the new node here



Addition with polynomial linked list

Input :

$p1= 13x^8 + 7x^5 + 32x^2 + 54$

$p2= 3x^{12} + 17x^5 + 3x^3 + 98$

Output : $3x^{12} + 13x^8 + 24x^5 + 3x^3 + 32x^2 + 152$

Addition with polynomial linked list

- For addition of two polynomial linked list, we have a need to traverse the nodes of both the lists.
  - **If the node has exponent value higher than another, then**
    - that node will be added to the resultant list

or

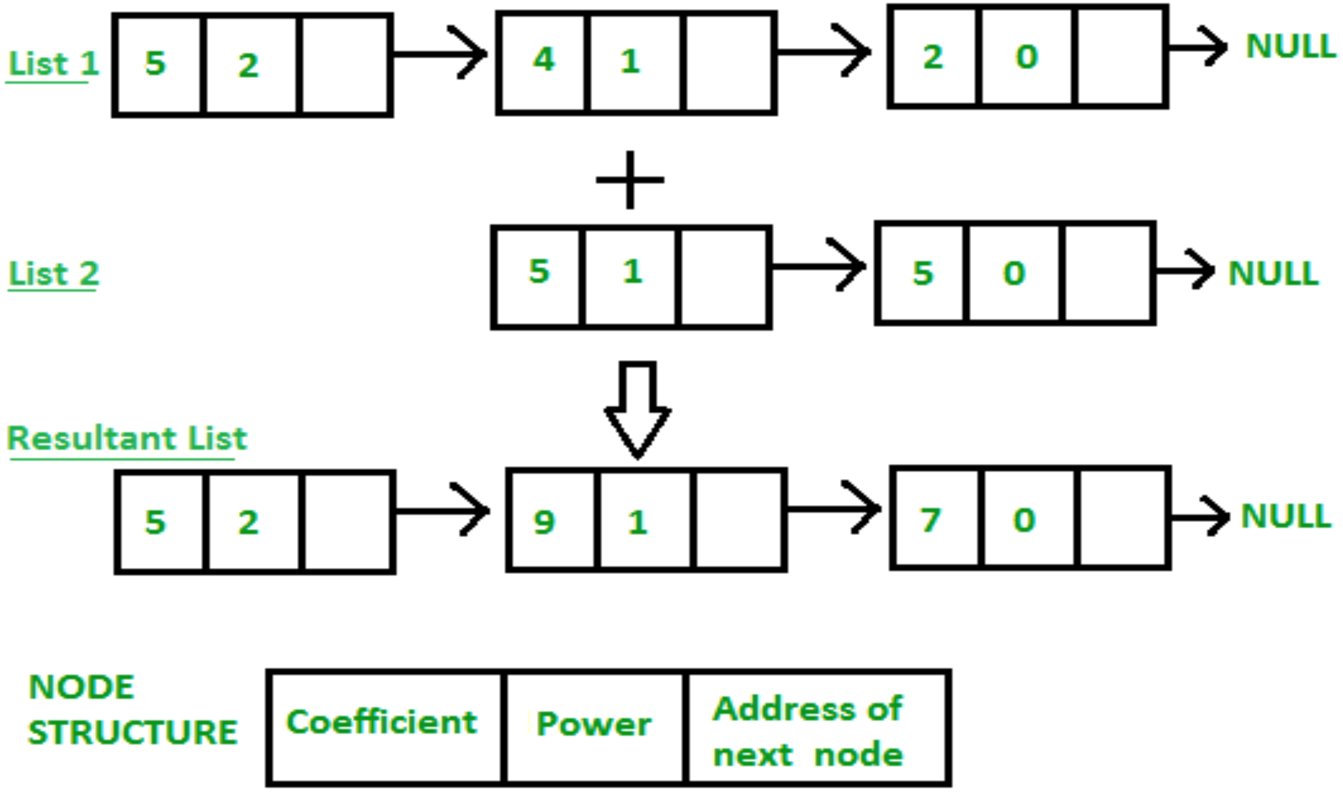
  - **Nodes which have unique exponent values in both the lists will be added in the resultant list.**

Addition with polynomial linked list

- If the nodes have **same exponent** value then
  - first the coefficient of both nodes will be added
  - then the result will be added to the resultant list.

Addition with polynomial linked list

- Suppose in traversing one list is finished then
  - **remaining node of the another list will be added to the resultant list.**


List 1
5
2
4
1
2
0
NULL
+
List 2
5
1
5
0
NULL
Resultant List
5
2
9
1
7
0
NULL
NODE STRUCTURE
Coefficient
Power
Address of next node