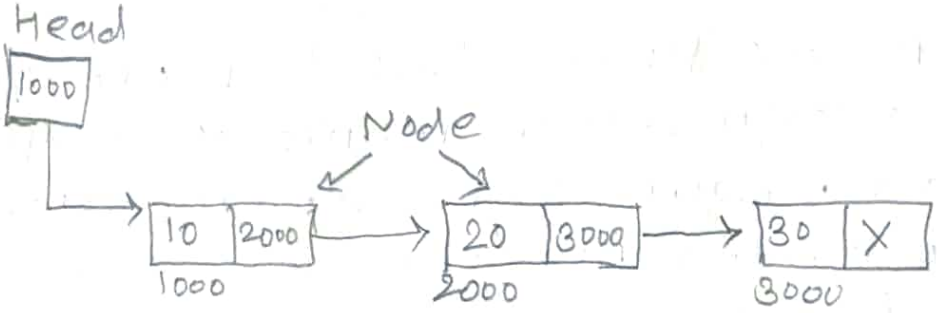
*Linked List:-

- Linked list is a linear data structure. Linked list is a one-way list. in linked list we can store data element in sequential manner called node.
- Linked list can be defined as collection of objects called Nodes that are randomly stored in the memory.
- Linked list is as shown in fig. below.



Linked List

- In linked list each node contain two parts, first part is data and second part is link to next node.
- The last node in the list contain a NULL pointer to indicate that end of the list.

*some characteristics of linked list.

- Dynamic memory allocation.
- Insertion and deletion can be possible at any position and not require any data movement.
- element can be placed anywhere in the memory.
- for dynamic allocation we can use pointer.
- each node have two field. first field contain Data and second field contain Address to next node.

* we can perform different operations on linked list like.
  - Create a linked list.
  - Traverse a linked list.
  - Insert a node in list.
  - Delete a node from list.

* **Advantages:**
- we can increase or decrease the size of linked list during the execution of program
- unlike Array, linked list do proper/efficient memory utilization. memory is allocated whenever require and de-allocated when it is no longer needed.
- In linked list we can insert element at any position or we can delete element from any position.
- many complex application can be easily carried out with linked list.

* **Disadvantages:**
- A linked list element require more memory space because it also has to store Address of next element in the list.
- As compair to Array, it is quite difficult to Access linked list elements. because index is not associated with each element.

* Types of linked list.

1) singly linked List
2) Doubly Linked list
3) circular Linked List
   - ↳ singly circular linked List
   - ↳ Doubly circular Linked List.

* singly Linked List:-

- singly Linked List can be defined as the collection of ordered set of elements.
- singly Linked list is a way to represent a linear list, where elements are store in the form of node.
- a node contain two parts. First part is data and second part contain address of next node. This representation is called a one-way chain or singly linked List.
- In singly linked list we can travers only in one direction. Here every node contain only one pointer which points to next node. so we can travel only in forward direction.

- we can create a structure, to create a new node as,

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

- In this structure, we can create a node by declaring two variable, 'data' variable will store element and '*next' variable will store Address of next structure (next node).
- figure below shows singly linked list.

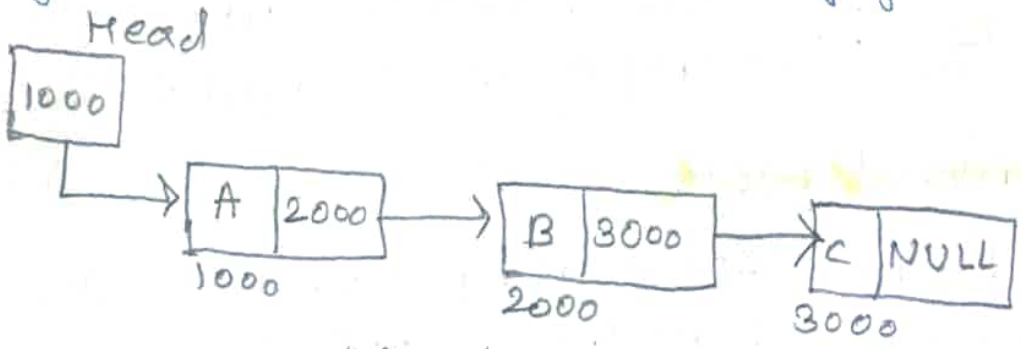


fig. singly Linked List:

- As shown in fig. node is made up of two parts.
- Address Field of last node contain value NULL

---

* operations on singly Linked List.

1) Create a Linked List.

step 1:

- To create a linked list, we need to create a node first. we will create a node one by one and append it to the list.

Initialize head = NULL;

Here we must note that, head is not a node. Head is the pointer variable which stores the Address of first node in the linked list.

Step 2: create a new node

- To create a new node follow the following steps.

* Allocate memory for new node
* store data in the new node's data field.
* set next field to NULL.

Step 3: Add newly created node at the end of linked list.

case 1: if list is empty

~~head~~ set

head = newnode;

case 2: if list is not empty

set

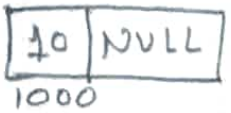
last → next = newnode;

last = new node;

Example: consider {10, 20, 30} data for linked list creation.

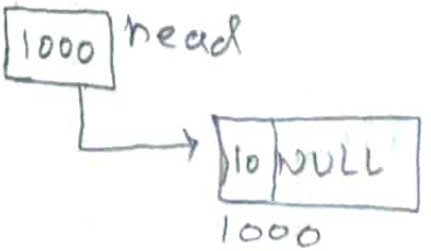
Step 1: Initialize head = NULL:

Step 2: create a new node;



newnode → data = 10

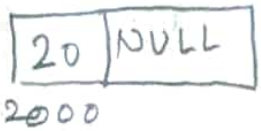
Step 3: Append this node in the linked list



newnode = 1000

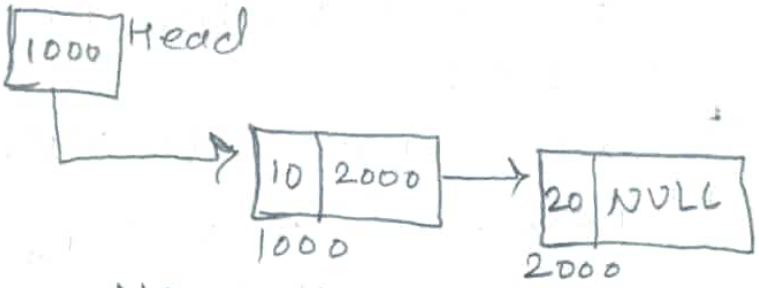
last = 1000

step 4: create a new node



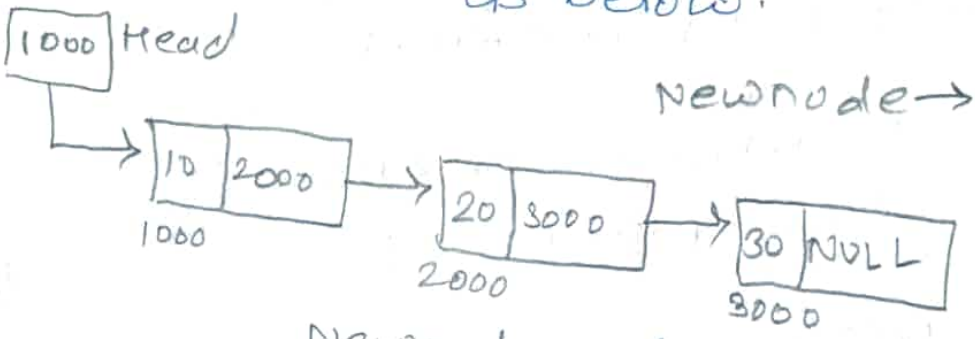
newnode → data = 20

step 5: Append this node in the linked list.



Newnode = 2000
last = 2000

step 6 & 7: similarly we can create a new node with element 30 and can append in list as below.



Newnode → data = 30

Newnode = 3000
last = 3000.

* in this way we can insert any number of nodes in the linked list.

---

2) Insertion of node :-

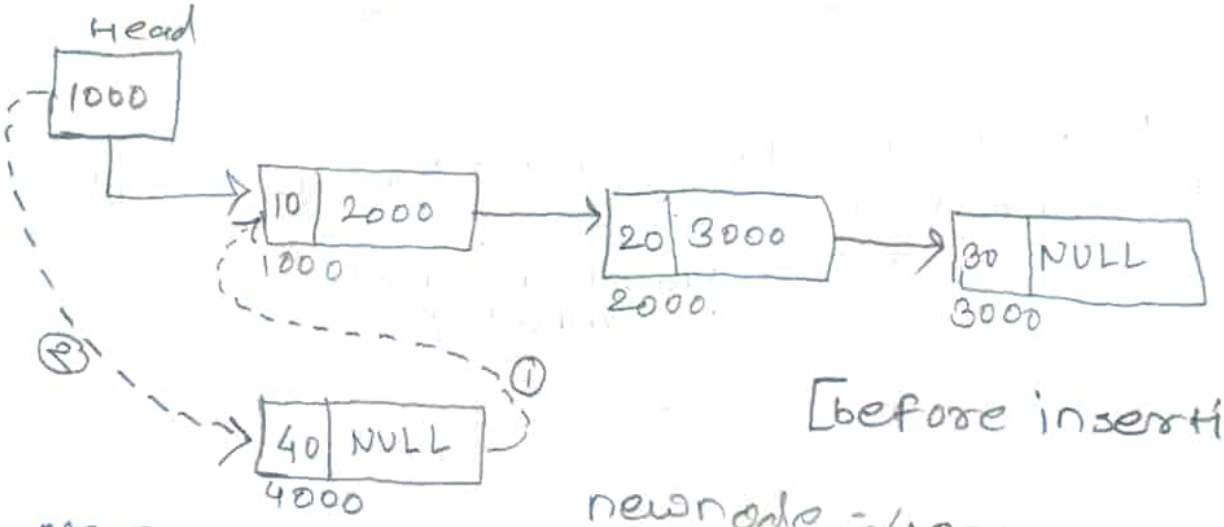
The insertion of element into a singly linked list can be performed at different position

1) Insertion at beginning.
2) Insertion at end
3) Insertion at any given position.

1. To insert node at first position

It involves insertion of any element at the front of the list. We just need a few link adjustment to make the new node as the first node of the list.

- in fig. below dotted line represent link manipulation needed to add new node at the first position.

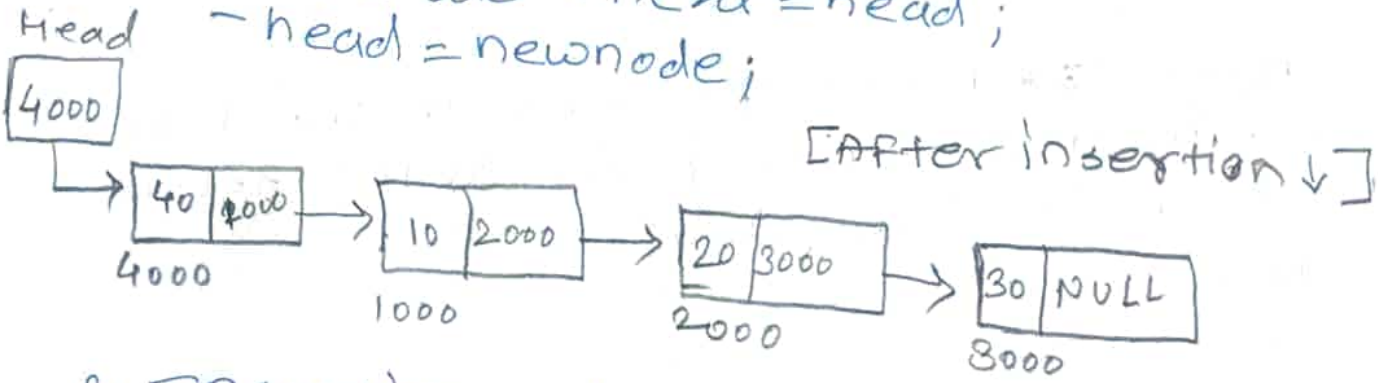


[before insertion]

newnode = 4000

Now,
Using following steps, we will get list as shown below.

- newnode → next = head;
- head = newnode;



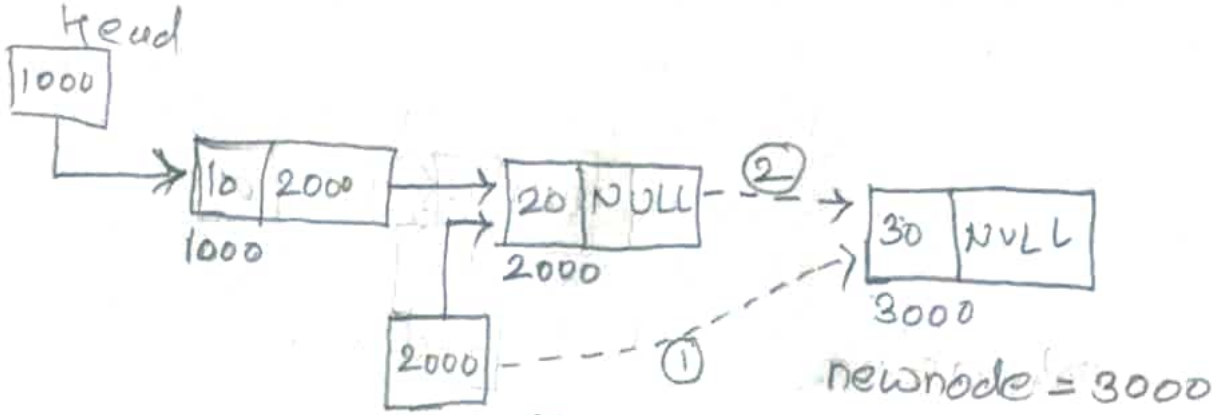
[After insertion ↓]

2. Inserting node at end (last) :-

- It involves insertion of any element at the end of list

- we can insert element at ~~last~~ end of list as ~~bel~~ explained below

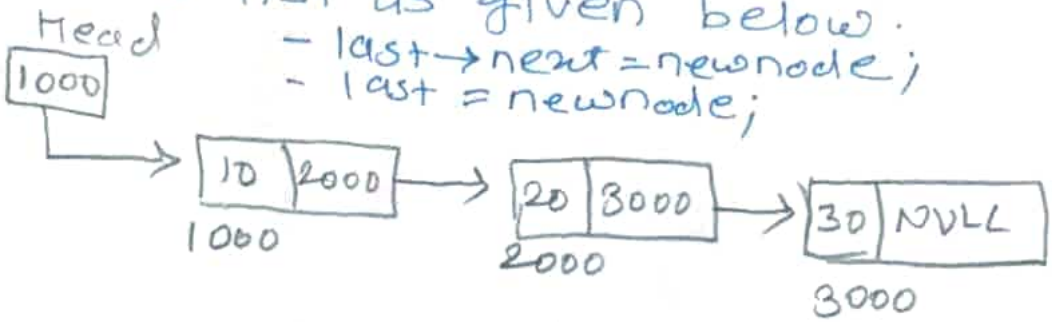
- The dotted line represent link manipulation needed to add node at the end



[before Insertion]

Now, using following steps we will get the list as given below.

- last→next = newnode;
- last = newnode;

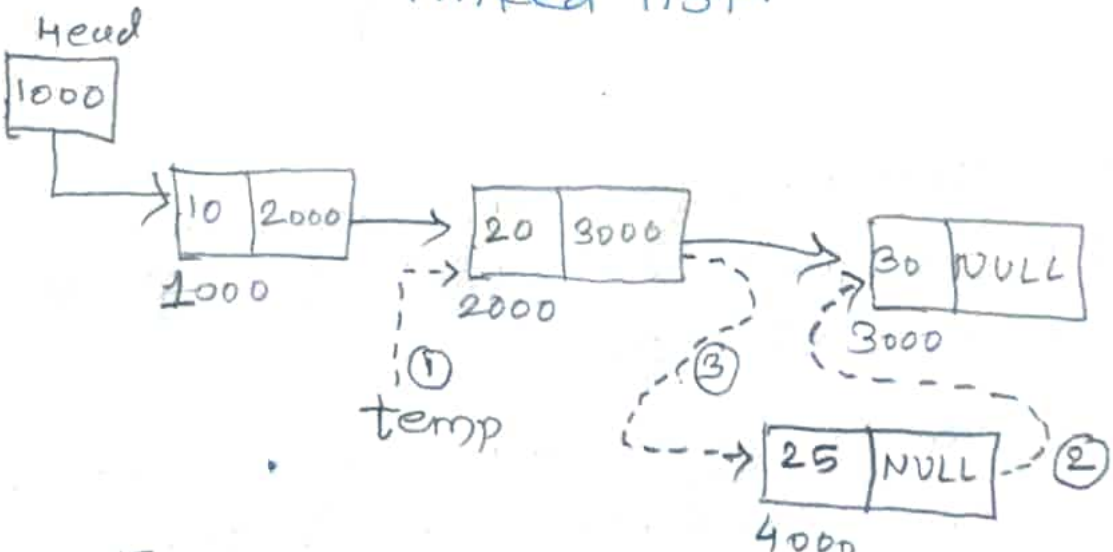


[After Insertion]

3. Insertion of a node at given position.

It involves insertion after the specific node of the linked list.

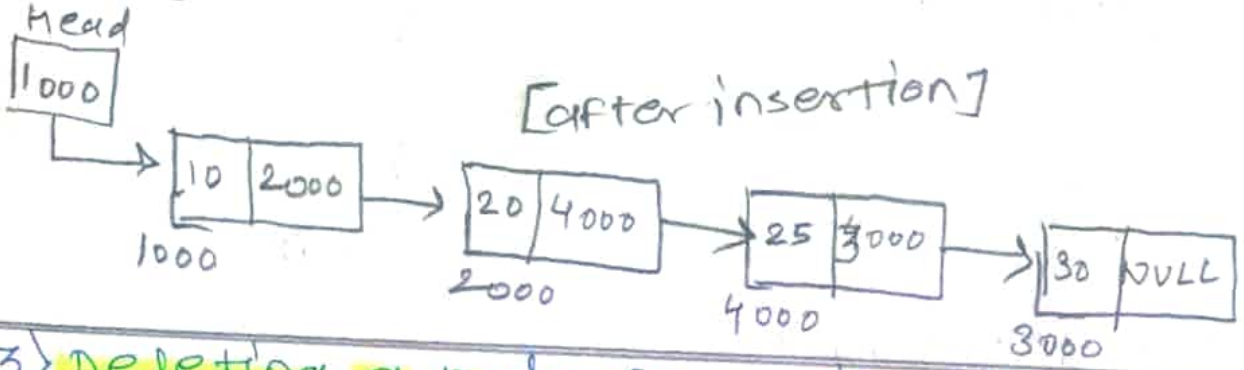
- consider 3 nodes present in the list & we will insert new node after second node in the linked list.



[before insertion] Newnode = 4000

- Here the new node is to be inserted after temp. the node which is successor of temp will now become successor of newnode. (5)

- After insertion we will get list as shown below.



[after insertion]

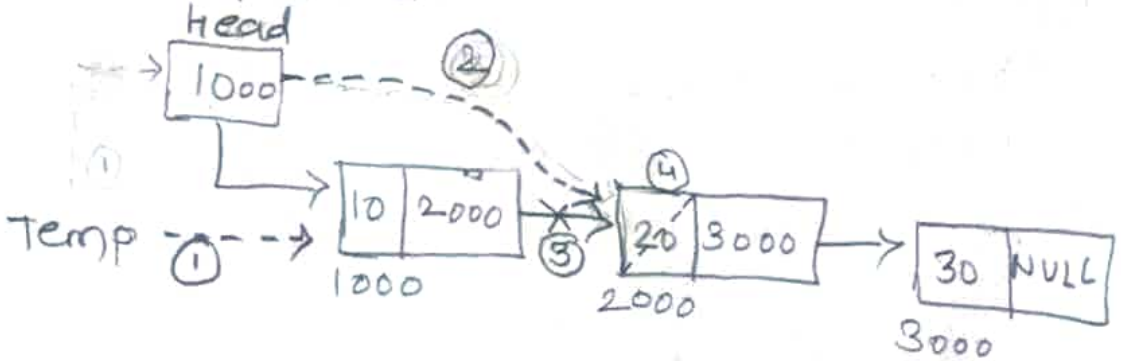
3) Deleting a node from linked list.

- Deletion of a node from singly linked list can be performed at different position.
  1. Deletion at beginning.
  2. Deletion at end.
  3. Deletion ~~after spe~~ at any position.

1. Deletion at beginning.

It involves deletion of a node from the beginning of the list. it just need a few adjustment in the node pointers.

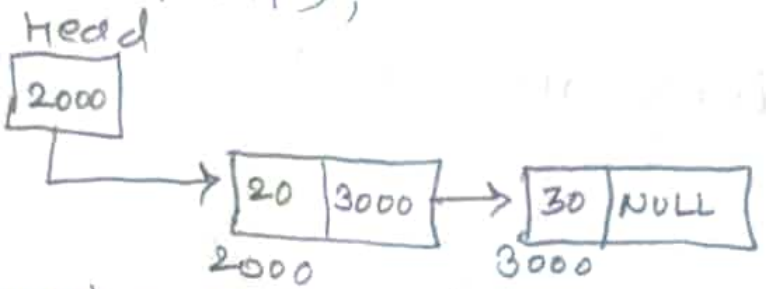
- if the node at first position is to be deleted then we need to modify the head poiwer.



[before deletion]

using following steps, we will get linked list as below.

```
temp = head;
head = head → next;
free (temp);
```

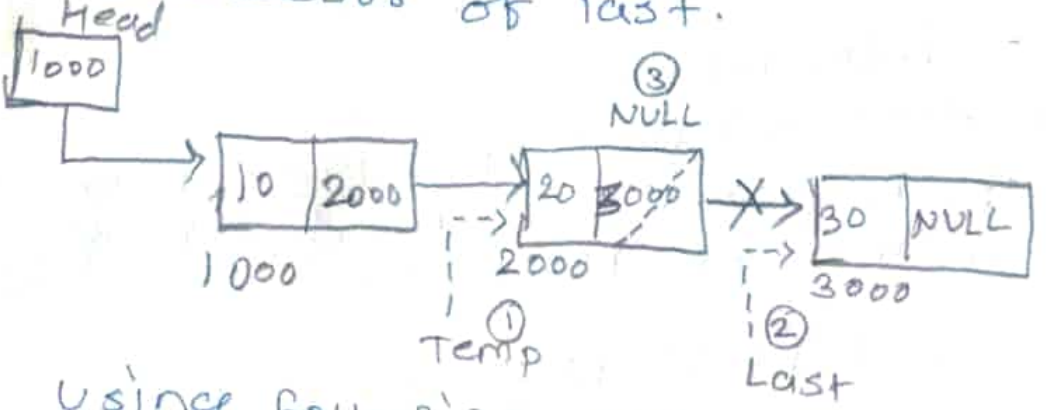


[after deletion]

2. Deletion from end.

- It involved deletion of last node of the linked list.

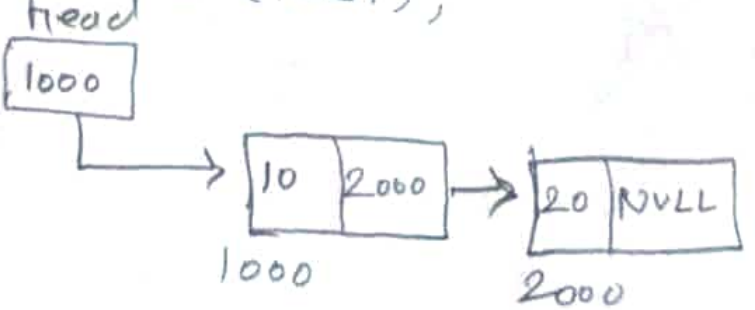
- when we wants to delete last node pointed by ~~ta~~ ~~last~~ 'last' pointer, then take 'temp' pointer to the node which is predecessor of last.



[before deletion]

Using following steps we will get linked list as below

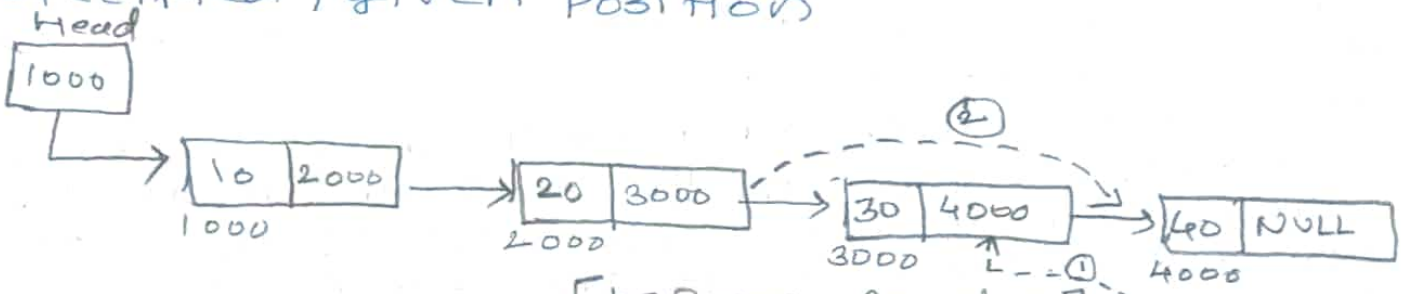
```
temp → next = NULL;
free (last);
```



[After deletion]

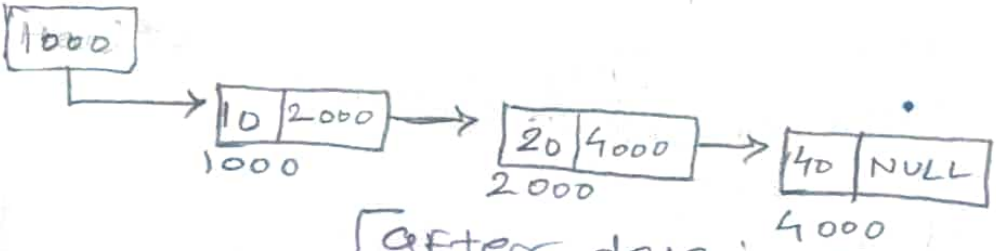
3. deletion from specific position:

It involved deletion of node from any specified / given position



[before deletion]

After deletion of element it will become as below.



[after deletion]

4) Traversing Linked List:-

-Traversing is the most common operation that is performed in almost every scenario of singly linked list.

-Traversing means visiting each node of the list once in order to perform some operation on that.

Algorithm:-

step 1: start
step 2: Get the Address of first Node, say temp.
step 3: if temp is not null repeat step 4 & 5.
step 4: process the data field of 'temp'.
step 5: Move to the next node using 'next' field of temp.
step 6: stop.

5) Searching in Linked List.

- searching operation is performed to find out the location of a particular element in the linked list.
- searching any element in the list needs traversing through the list and make the comparison of every element of the list specified element. if the element is matched with any of the list element then location of the element is returned from the function.

Algo → [Algorithm]

step 1: start
step 2: set ptr = head
step 3: set i = 0
step 4: if ptr = NULL
write List is empty
go to step 9
end of if
step 5: Repeat step 6 to 8 until ptr != NULL
step 6: if ptr → data = item
write i+1
end of if
step 7: i = i+1.
step 8: ptr = ptr → Next
step 9: stop.

---

6) sort the element of linked list.
7) Merge the element of linked list.
8) Reverse Linked List.

[

* Doubly linked list:

- we know that, singly linked list contain a node which has only one pointer variable, through which we can travel only in one direction from first node to last node.
- To overcome this problem we can creat ~~a linked list~~ - Doubly linked list.
- Doubly linked list is a linear collection of elements.
- In Doubly linked list element are stored in the form of node.
- In Doubly link list node is made up ~~of~~ three Parts.

1) first part contain Address of previous node.
2) second part contain actual element.
3) Third part contain Address of next element.

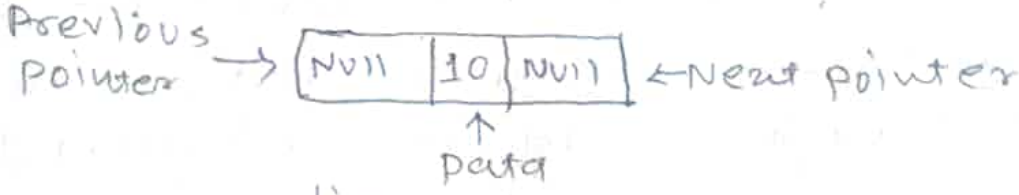


fig. Node

As shown in node, it contain two pointers so, we can travel in both direction, which is not possible in case of ~~link~~ singly linked list.

- structure to create a node

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

Advantages of Doubly linked list:

- In Doubly linked list insertion and deletion operation are simple as compair to other Data structure.
- By using Doubly linked list we can solve most of the problems very easily.
- In doubly linked list we can add or remove elements during the program execution and as per requirement, so it gives efficient utilization of memory.
- we can travel in both direction. so we can access elements easily.

Disadvantages:-

- In Doubly linked list node require more memory as compair to Array.
- each node require two pointers require additional storage for each field.

* Operations on Doubly Linked List:-

1) (create a Doubly linked list)

step 1:

- To create a doubly linked list, we have to create a node one by one and append it to the list.

Initialization head = NULL

step 2: create a new node.

To create a new node we need to follow, following steps.

1. Allocate memory for new node.
2. store data in data field.
3. set previous field to NULL
4. set next ~~Field~~ Field to NULL.

Step 3: Append

Here we will add newly created node at the end of doubly linked list.

case 1: head = null /* if ~~head~~ list is empty */
head = newnode;
last = newnode;

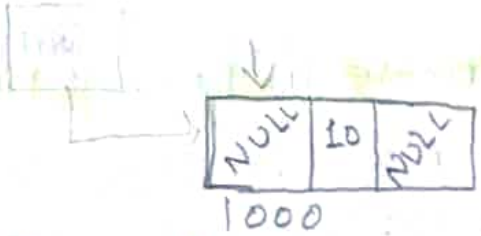
case 2: head != null /* list is not empty */

set last node's next field to newnode
set newnode's prev. field to last
last = newnode.

example: consider {10, 20, 30} data to create a doubly linked list

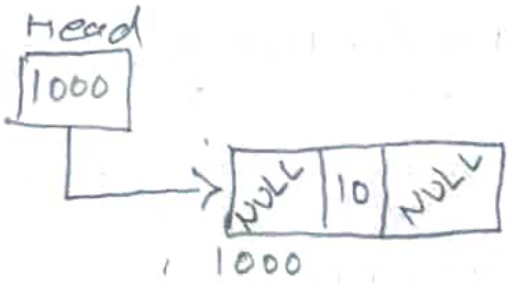
step 1: Initialize head = NULL;

step 2: create new node



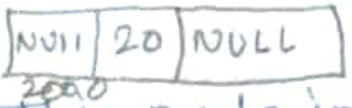
Prev = NULL
Next = NULL
Newnode → data = 10

step 3: Append this node in the linked list



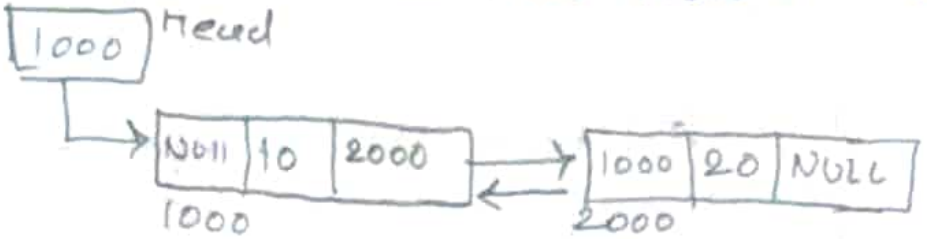
newnode = 1000
last = 1000

step 4: Create a next new node



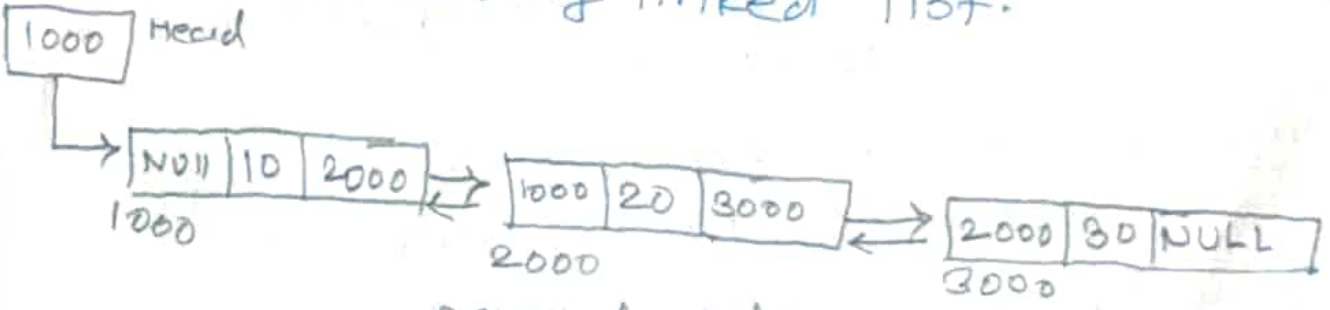
newnode→data = 20

Step 5: Append this node in the linked List.



newnode = 2000
last = 2000

Step 6 & 7: In same way we can create a new node to store element 30 and can append in Doubly linked list.



newnode→data = 30
newnode = 3000
last = 3000

---

2) Insertion of a new node in Doubly linked list.

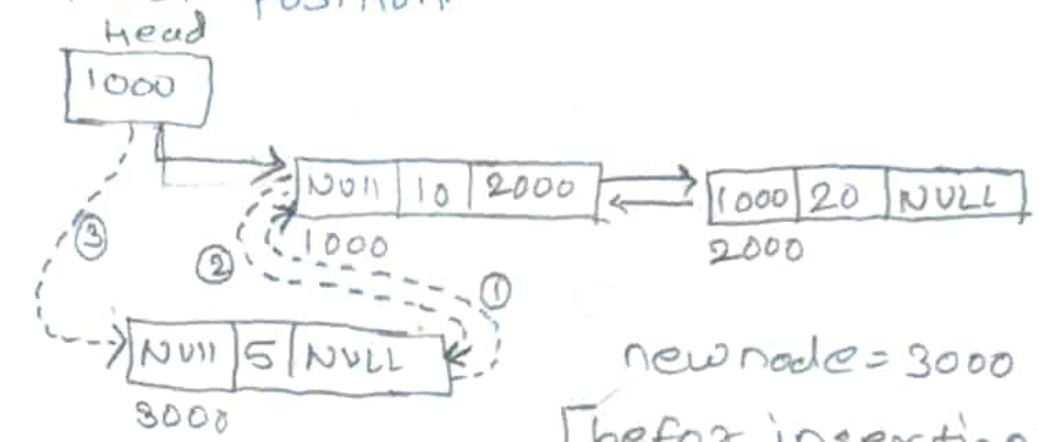
Insertion of a node in to a doubly linked list can be possible at different position.

1) Insertion at Beginning.
2) Insertion at last.
3) Insertion at any given position.

1.
* Insert a node at Beginning:-

It involves insertion of node (element) at the front of the list. We just need a few link adjustment to make new node as the first node.

In below list dotted line represent link manipulation needed to add node at the first position.

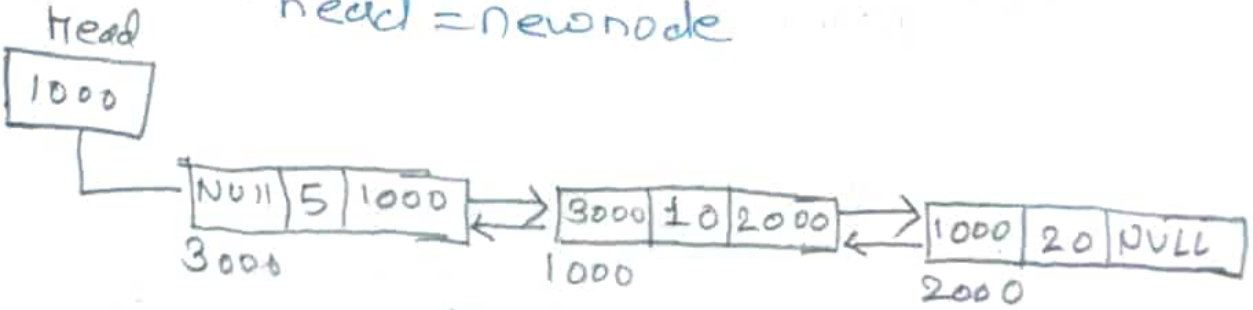


newnode = 3000

[befor insertion]

by using following steps we will get list as shown below.

- newnode → next = head
- head → prev = newnode
- head = newnode



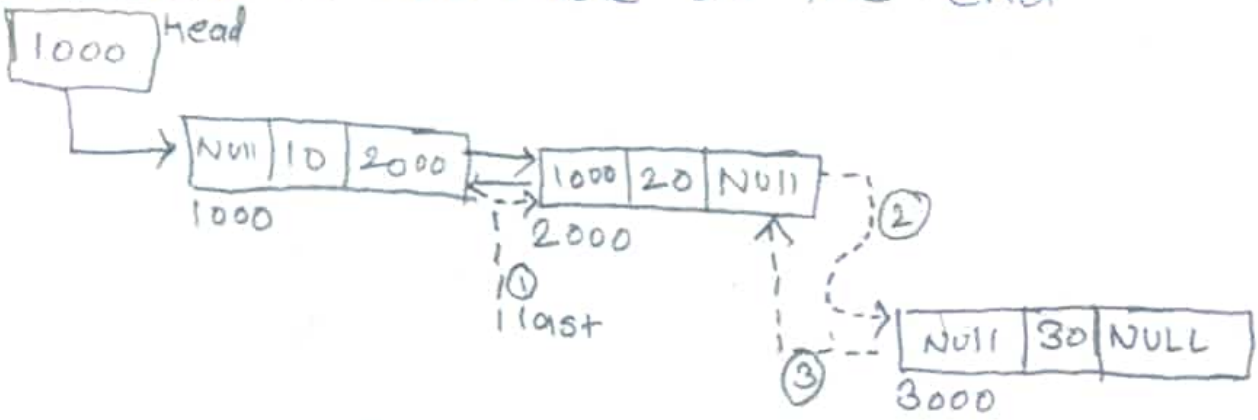
[after Insertion] newnode = 3000

2. Insertion at last: (end)

- It involves insertion of element at the end of list.

- we can insert element at the end of ~~link~~ doubly linked list as shown below.

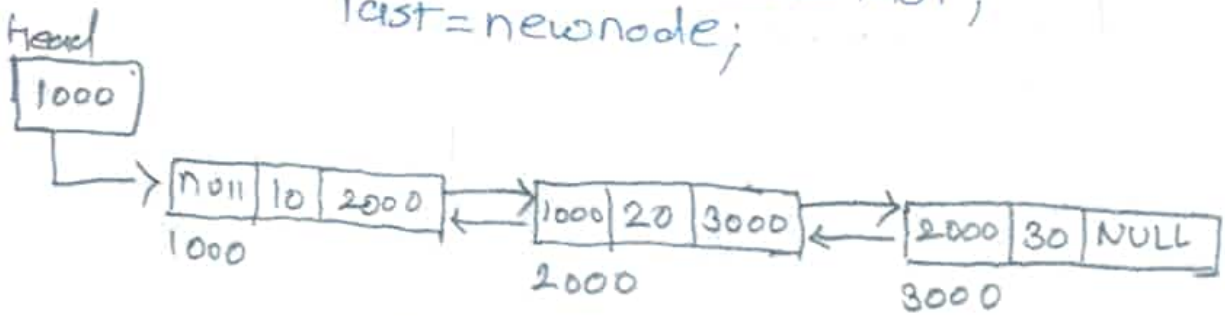
- The dotted line represent link manipulation needed to add node at the end



[before Insertion] newnode = 3000

by performing following steps we will Doubly linked list as below.

```
last->next=newnode;
newnode->prev=last;
last=newnode;
```

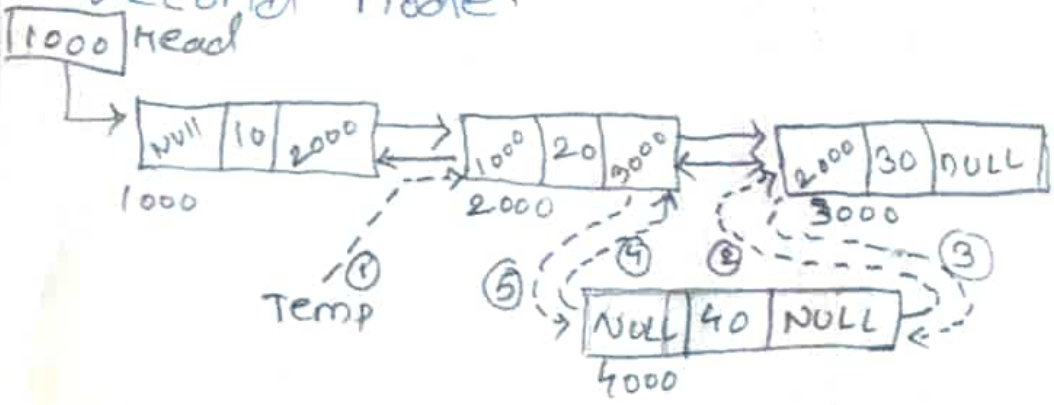


[After Insertion]

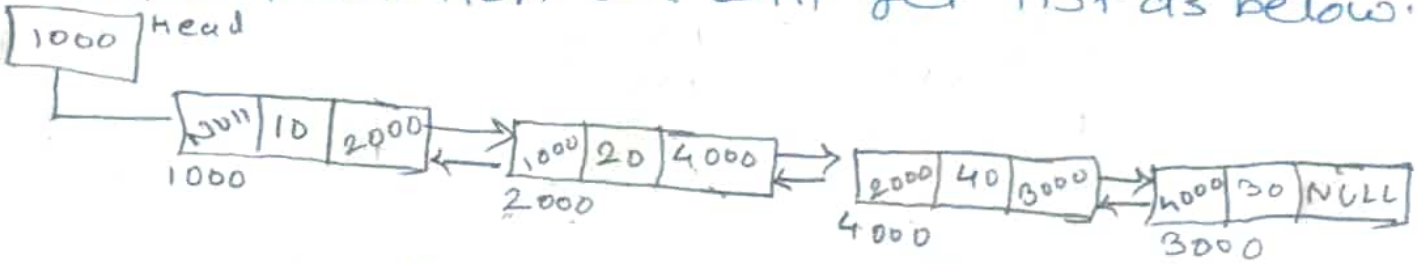
3. Insertion at any given position.

In involves insertion after the specific node of the linked list.

- consider 3 nodes in the linked list and it we will insert new node after second node.



Here new node is to be inserted after temp, The node which is successor of temp now become successor of new node
- After insertion we will get list as below.



[after insertion]

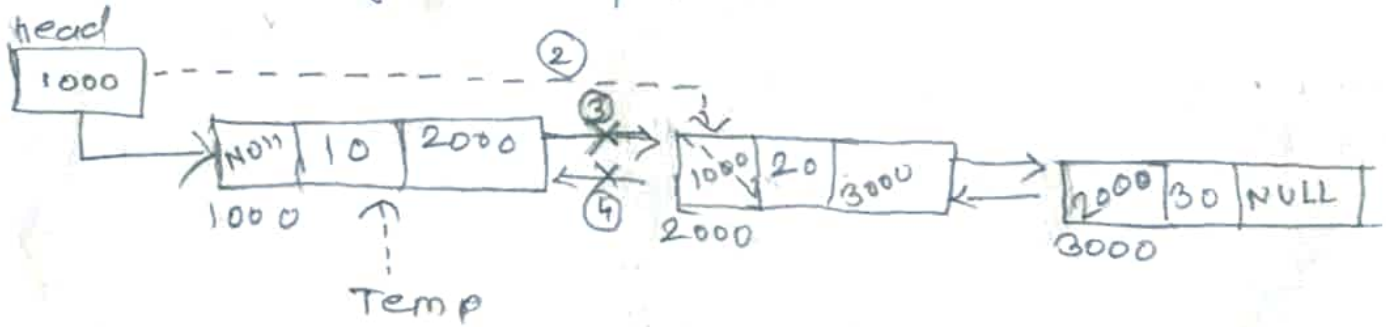
3) Delete a node from Doubly linked list:

Deletion of node from doubly linked List can be performed at different position.

1. Deletion from beginning.
2. Deletion at end
3. Deletion from specific position.

1. Deletion from beginning.

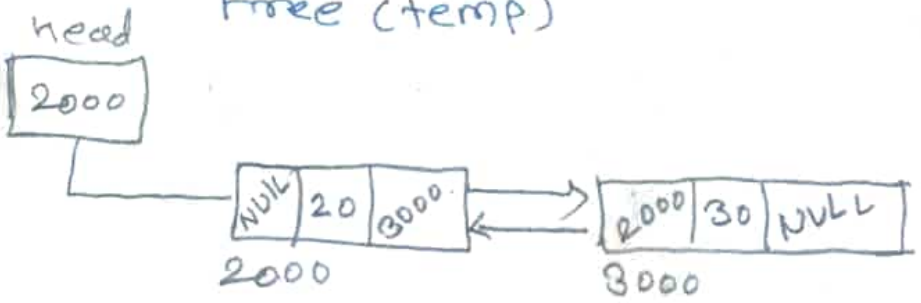
- It involved deletion of node from the beginning of the list. We just need to modify Head pointer.



[before deletion]

using following steps we will get linked list

```
temp = head;
head = head → next;
temp → next = null;
head → prev = null;
free (temp)
```

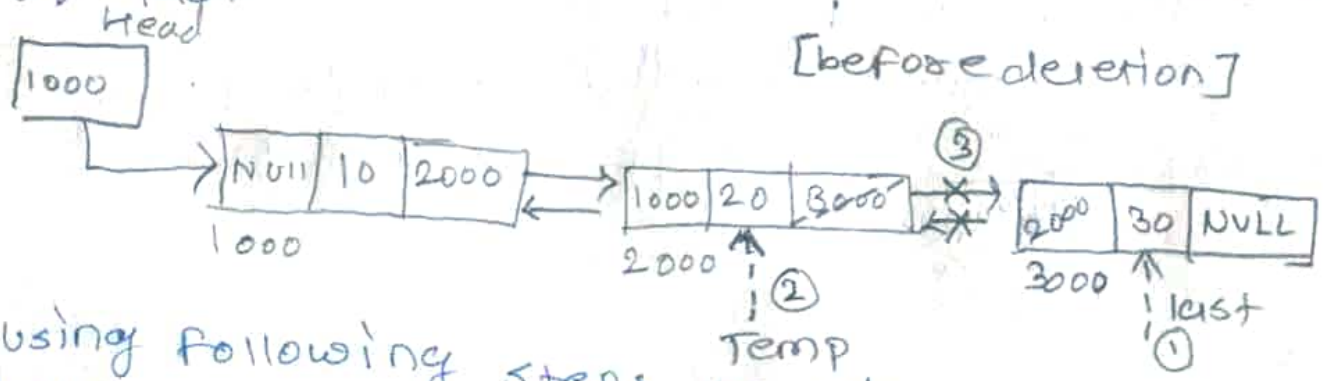


[after deletion]

2. Deletion from end.

It involved deletion of last node of the doubly linked list.

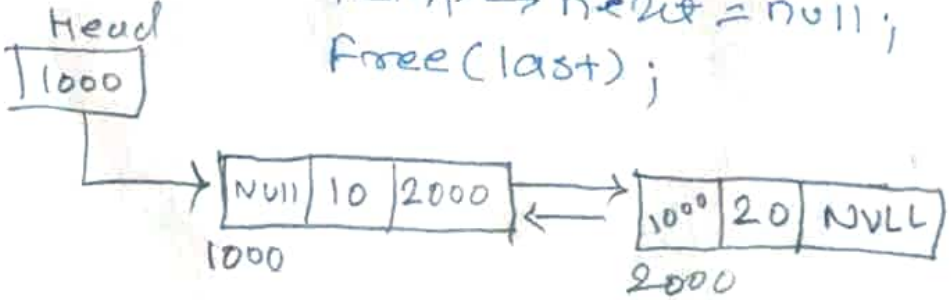
- when we want to delete last node, pointed by 'last' pointer, Then take 'temp' pointer to the node which is predecessor of last.



[before deletion]

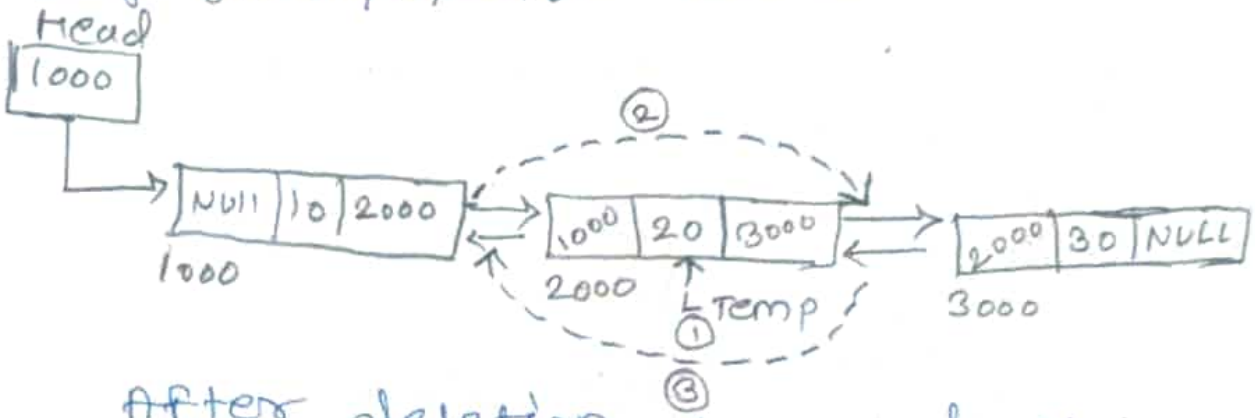
using following steps we will get list ~~etc~~ as below

- Temp → next = null;
- Free (last);

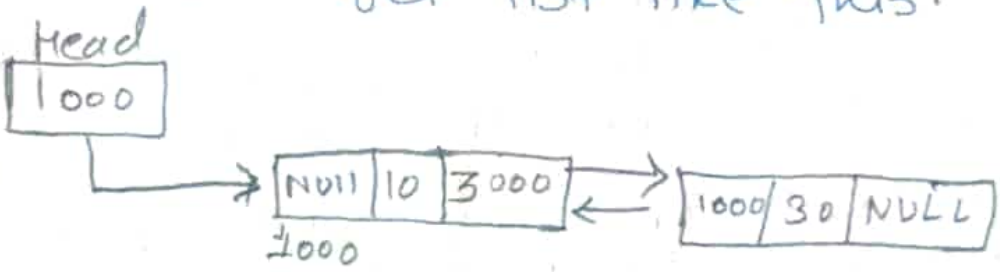


[After Deletion.

3. Deletion from specific position.
- It involved deletion of node from any given / specific position.



After deletion of second element we will get list like this.



[After deletion]

Q) write a algorithm to count number of nodes in linked list.

→ [give short description of linked list then write algorithm]

Step 1: Start

Step 2: Define a method that accept the head of linked list.

Step 3: initialize count = 0;

Step 4: Travel the given linked list till it reaches the last node.

Step 5: increase the value of 'count' variable by 1. in loop.

Step 6: Display value of 'count' as number of nodes.

Step 7: Stop.

Q) ~~Differentiate Between singly Linked List and Doubly linked list~~

Q) Differentiate Between singly & Doubly Linked list.

singly Linked List	Doubly Linked List
1) In SLL node is made up of two parts, first part is data & second is next pointer.	In DLL node is madeup of three parts. first part is previous pointer second is data & 3rd is next pointer
It can be travel only in one direction.	It can be traven in both direction
In this node require less memory compair to DLL	node require more memory as compair to SLL
It is less efficient as compair to DLL	It is more efficient as compair to SLL
It can be implemented on stack	It can be implemented on stack, heap, & binary tree
It require only one pointer variable	It require ~~of~~ two pointer variable.