

September 28

## Linked list

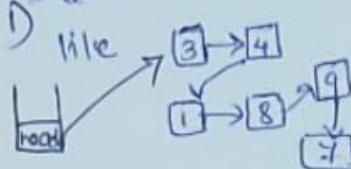
ex:- suppose we have an array 

3	4	1	6	3
---	---	---	---	---

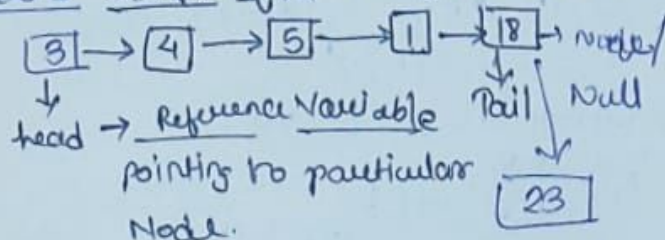
 → 

3	9	1	8	9	7
---	---	---	---	---	---

 we use array list. It is not a continuous memory allocation. It will break the boxes.

like  • These are connected by arrows • stored in a different-different memory. • Not Initially.

Terms example of LL - single linked list



class Node {  
int val;  
Node next;  
} → representation of every single box in linked-list.

\* We can't access directly like we used to do in array.

Ques Add a new item in the last of linked list?

→ 

23
----

 → create a new node

→ Tail.next = new node

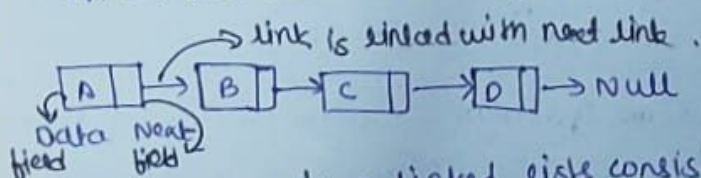
→ Tail = new node

September 30 9F9

## LINKED LIST (SINGLE)

• A linked list is a linear data structure in which elements are not stored at contiguous memory.

→ All elements are linked using pointers.



• In other words a linked list consists of nodes where each node contains a data field & a reference (link) to the next node in the list.

• like array linear data structure.  
• Unlike arrays elements are not stored at contiguous location.

## Why linked list?

- 1) Because array size is fixed
- 2) Insertion / deletion of a missing element in an array is expensive.

ex:- sorted list of array id []  
= [1000, 1010, 1050, 2000, 2040]

If we want to insert 1005 ID then to maintain sorted order we have to move all element excluding (1000) & deletion is also expensive  
eg:- to delete 1010 in id [], everything after 1010 has to be moved.

Advantage 1) Dynamic 2) ease of insert / deletion.

Drawbacks - 1) Random access is not allowed. Access elements sequentially starting from 1st node. [Binary search not possible]  
2) Extra memory space for a pointer  
3) Not cache friendly.



## TYPES

- ① Singly - One can move / traverse the linked list in only one direction.
- ② Doubly - One can move / traverse the list in both the direction
- ③ Circular - The last node of the LL contains the link of the first / head node of LL in its pointer & the first / head node contains the link of the last node of the LL in its prev pointer.

## Representation of linked lists:-

- Represented by a pointer to 1st node, 1st node = head, Empty LL = Null

Each node contains at least two parts:-

- 1) Data Item (int, string, double & anythg)
- 2) Pointer (to next node).

## Simple Java Program

```
class linkedlist {
    Node head;
```

```
class Node {
    int data;
    Node next;
```

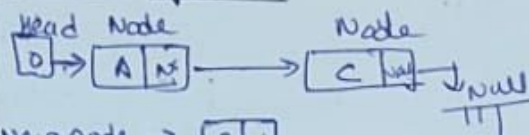
```
Node (int d) {
    data = d;
    next = null;
```

}

Basic Operations that can be performed on linked lists are:-

- 1) Insertion - Adds an element at the beginning
- 2) Deletion - ~~Deletes~~ <sup>Deletes</sup> an element at the beginning
- 3) Display - Displays the complete list
- 4) Search - Searches an element using the given key
- 5) Delete - Deletes an element using given key.

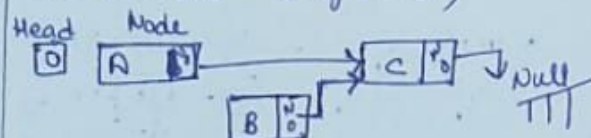
## Insertion operation -



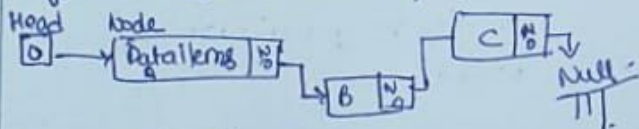
New node → B

∴ A is a left node and C is a right node

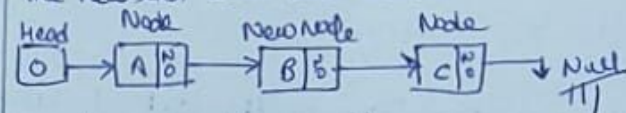
New node.next → Right node;



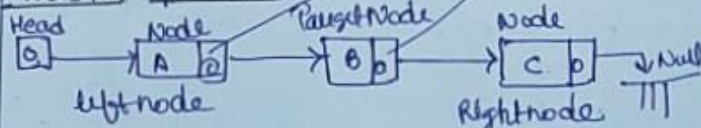
Left node.next → New Node;



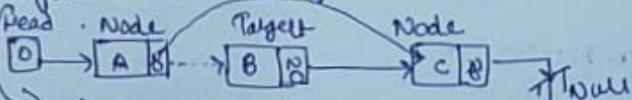
The new list will look like:-



## Deletion operation -

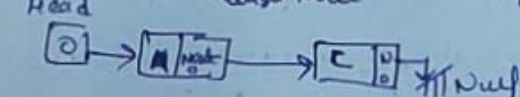
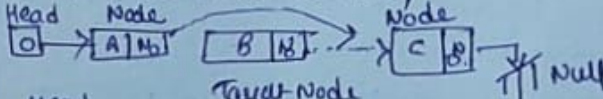


Left node.next → Target Node.next



This will remove the link pointing to the target node.

Target Node.next → Null;



Reverse Operation - We need to make last node to be pointed by head node & reverse whole linked list.