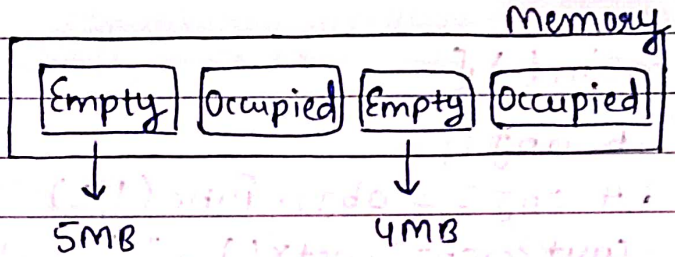
03/11/2023
Friday

* **Limitation of linear Data Structure - Array**

- As arrays stores data in continuous memory blocks. This leads to wastage of memory.

Suppose,

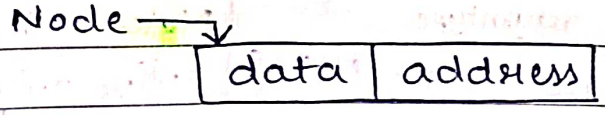


Total empty space = 9MB
But we can't make an array of size 9MB as the space provided is not continuous.

* **LINKED LISTS** - It is a linear data structure.

Linked list is a collection of nodes which stores data in non-continuous memory blocks. This reduces memory wastage.

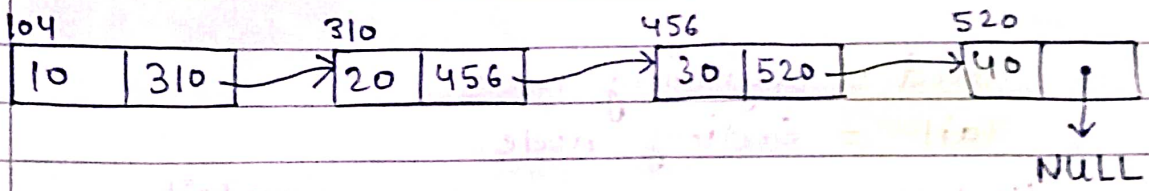
Node - Node consists of two blocks. One that stores data and another stores the address of next node.



- LL mai insert/shift O(1) T.C. mai kr skte hai provided location pta ho jha insert/shift krna hai.
- LL can grow and shrink dynamically.
- LL mai index ka nhi address ka concept aata h.

Note- Kabhi bhi pointer declaration nhi krna hai. Always initialise the pointer either initialise it from NULL.

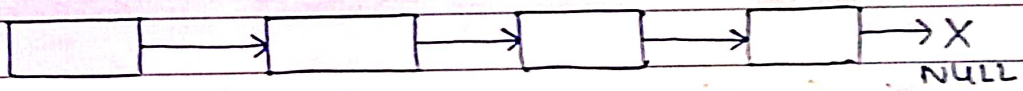
Linked list -



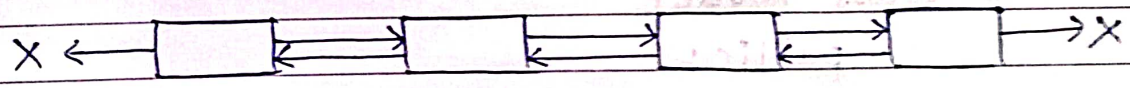
= Linked list is Hindi → Tag line

* Types of linked list →

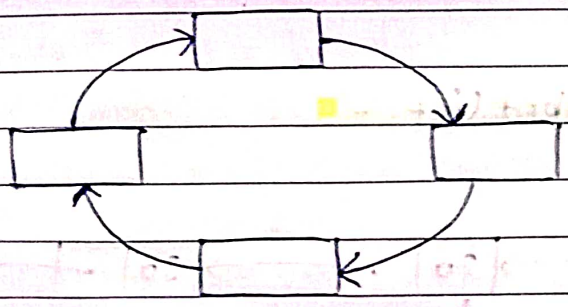
- Singly linked list -



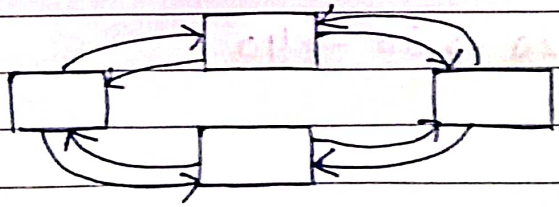
- Doubly linked list -



- Circular singly linked list -

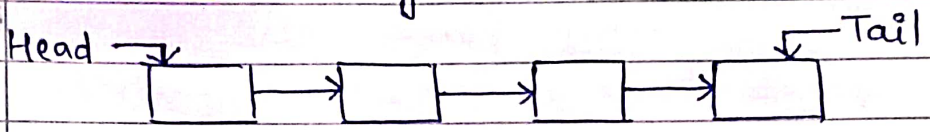


- Circular doubly linked list -



Note - Linked list ke ques mai corner cases banenge. Don't miss them.

- **Head** = Beginning node.
- **Tail** = Ending node.

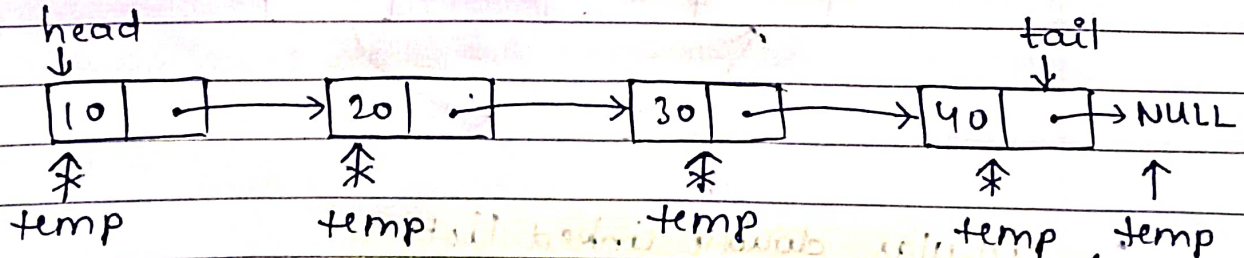


Note - Jb bhi linked list pass krenge function mai using head or tail ka pointer, to linked list ko traverse krne ke liye original pointer ka use nhi krenge rather than us original pointer ko ek nye temp pointer ko assign krva denge.

* **Creating a node -**

```
class Node{
  public:
    int data;
    Node* next;
};
```

* **Print Linked list -**

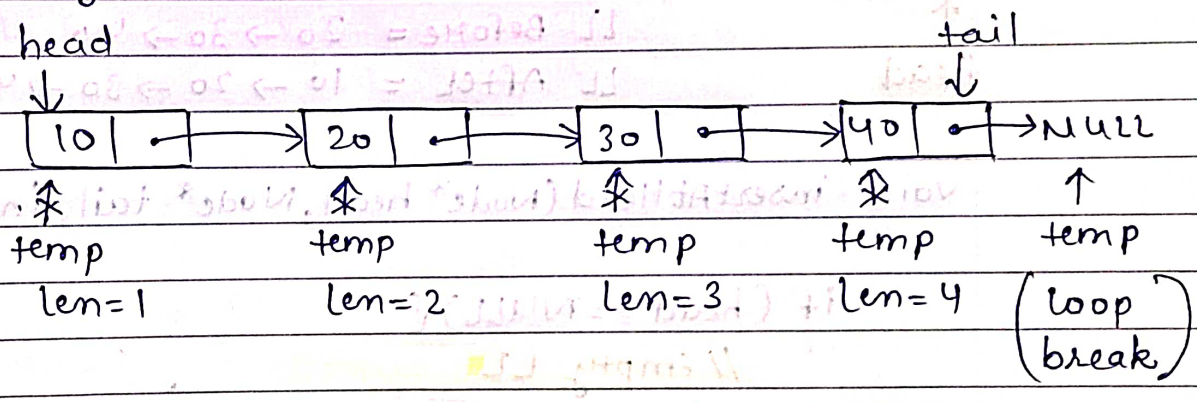


10 → 20 → 30 → 40

```
void print (Node * head){
    Node * temp = head;

    while (temp != NULL){
        cout << temp -> data << "->";
        temp= temp->next;
    }
}
```

* length of linked list -

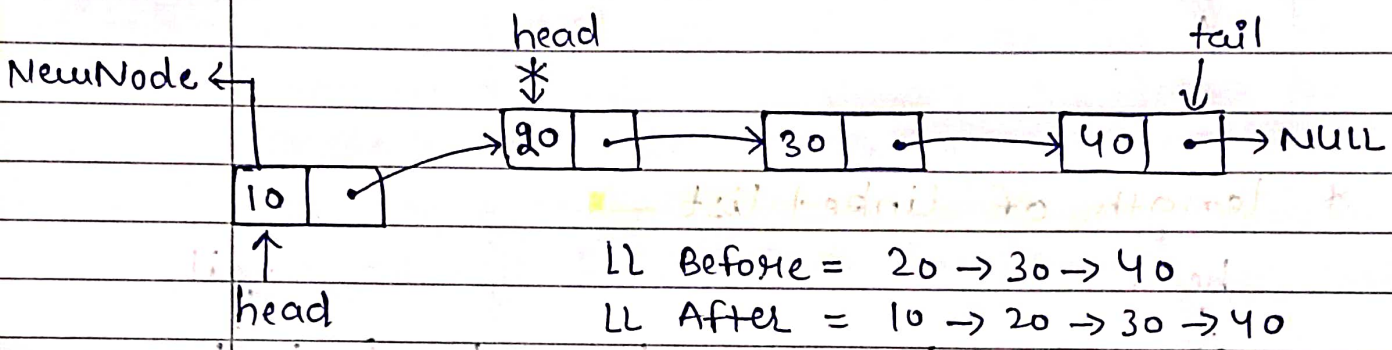



```
int lengthLL (Node* head){
    Node * temp = head;
    int length= 0;
    while (temp != NULL){
        length ++;
        temp = temp -> next;
    }
    return length;
}
```

Note- Always follow the practice - Jb bhi function mai pass krenge head and tail ko, pass by reference krenge.

* **Insert Node at Head -**

Step 1 - Create new node.
Step 2 - attach new node to head node.
Step 3 - Update head. Now, head is new node.



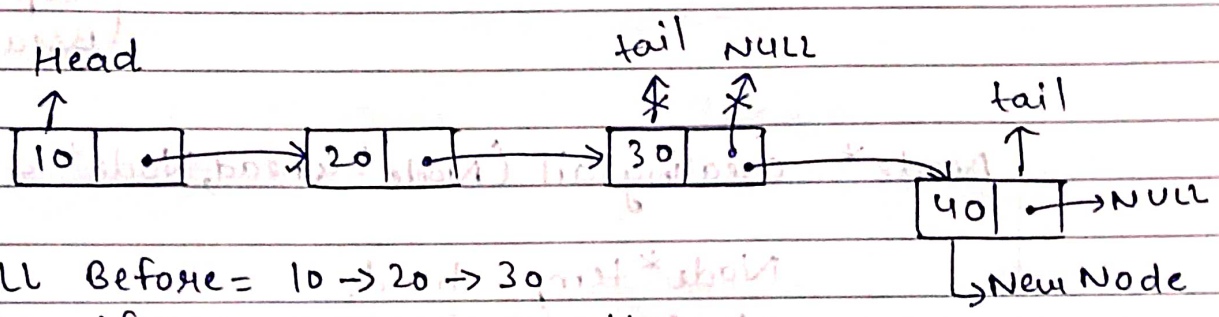
LL Before = 20 → 30 → 40
LL After = 10 → 20 → 30 → 40

```
void insertAtHead(Node* head, Node* tail, int data){
    if(head==NULL){
        // Empty LL
        Node* NewNode = new Node(data);
        head= NewNode;
        tail = NewNode;
    }
    else{ // Non-empty LL
        Node* NewNode = new Node(data);
        NewNode->next = head;
        head = NewNode;
    }
}
```

Note -
- Jb empty linked list hogi, then - head = tail = NULL
- Jb linked list contains single element - head == tail

* **Insert Node at Tail -**

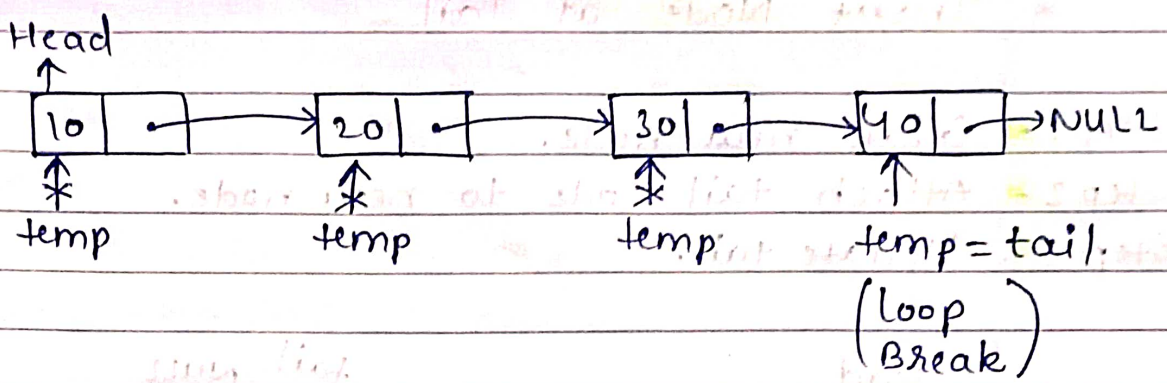
Step 1 - Create new node.
Step 2 - Attach tail node to new node.
Step 3 - Update tail.



LL Before = 10 → 20 → 30
LL After = 10 → 20 → 30 → 40

```
void insertAtTail(Node* head, Node* tail, int data){
    if (tail == NULL){
        //Empty LL
        Node* NewNode = new Node(data);
        head = NewNode;
        tail = NewNode;
    }
    else{
        // non-empty LL
        Node* NewNode = new Node(data);
        tail -> next = NewNode;
        tail = NewNode;
    }
}
```

* **Tail in linked list -**




```
Node* creatingTail (Node* &head, Node* &tail){
    Node* temp = head;
    while (temp -> next != NULL){
        temp = temp -> next;
    }
    tail = temp;
    return tail;
}
```

* **Insert in between linked list -**

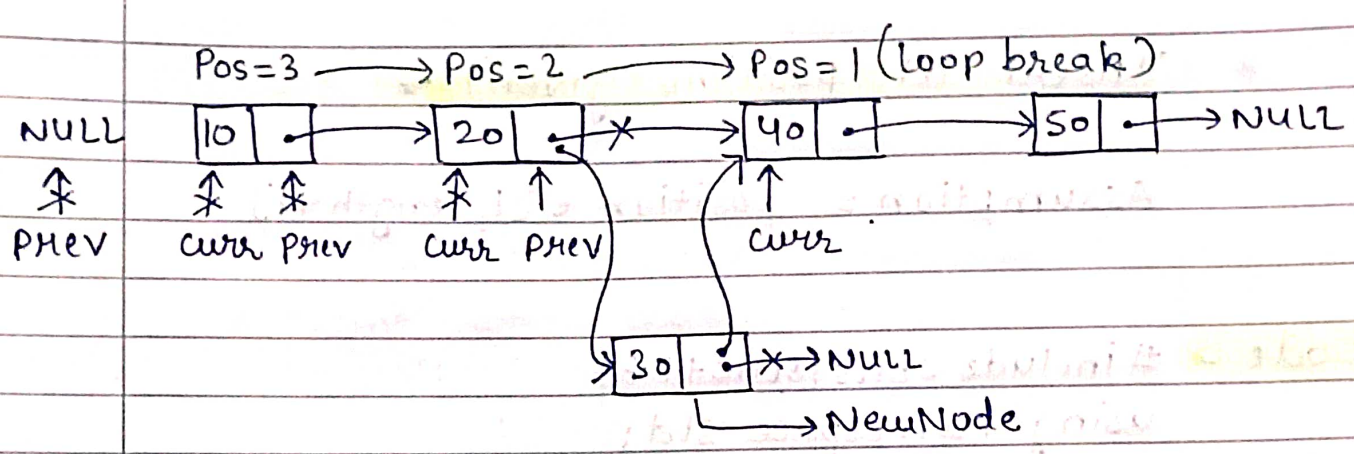
Step 1 - Create new node.

Step 2 - Traverse prev & curr pointers to the required position.

Step 3 - Attach prev node to NewNode.

Step 4 - Attach NewNode to curr node.

Given position = 3



LL Before = 10 → 20 → 40 → 50

LL After = 10 → 20 → 30 → 40 → 50

```
void insertInBwll(Node* head, Node* tail, int data,
                  int position) {

    Node * NewNode = new Node(data);
    Node * prev = NULL;
    Node * curr = head;

    while ( position != 1 ) {
        position--;
        prev = curr;
        curr = curr->next;
    }
    prev->next = NewNode;
    NewNode->next = curr;
}
```

__/__/____

* Insert at position (given) →

Assumption = position ∈ [1, length+1]

Code -

```
#include <bits/stdc++.h>
using namespace std;

class Node {
    public:
        int data;
        Node *next;

    Node (int data){
        this -> data = data;
        this -> next = NULL;
    }
};

// Print function

void print (Node *&head){
    Node * temp = head;
    while (temp != NULL){
        cout << temp -> data << "->";
        temp = temp -> next;
    }
}
```

```
//length of LL

int lengthLL (Node*&head){
    Node * temp =head;
    int length = 0;
    while (temp != NULL){
        length ++;
        temp =temp ->next;
    }
    return length;
}

//insert at head

void insertAtHead (Node*&head, Node* &tail, int data)
{
    if (head == NULL) {
    Node * NewNode = new Node (data);
    head = NewNode;
    tail = NewNode;
    }
    else {
    Node * NewNode = new Node (data);
    NewNode -> next = head;
    head = NewNode;
    }
}
```

// insert at tail

```
void insertAtTail(Node* &head, Node* &tail, int data)
{
    if (tail == NULL){
        Node* NewNode = new Node(data);
        head= NewNode;
        tail = NewNode;
    }
    else{
        Node * NewNode = new Node (data);
        tail -> next= NewNode;
        tail = NewNode;
    }
}
```

// insert in between LL

```
void insertInBwLL(Node* &head, Node* &tail,
                  int data, int position){
    Node * NewNode = new Node(data);
    Node * prev = NULL;
    Node * curr = head;

    while (position != 1) {
        position --;
        prev = curr;
        curr = curr ->next;
    }

    prev ->next = NewNode;
    NewNode ->next = curr;
}
```

```
//insert at given position

void insertAtPosition (Node* &head, Node* &tail,
                       int data, int position) {

 int length= lengthLL (head);

 if (position == 1){
     insertAtHead (head, tail, data);
    }

 else if (position== length+1){
     insertAtTail (head, tail, data);
    }

 else{
     insertInBwLL (head, tail, data, position);
    }
 }
int main (){
      Node* head= NULL;
      Node * tail = NULL;

      insertAtHead (head, tail, 30);
      insertAtTail (head, tail, 40);
      insertAtHead (head, tail, 20);
      insertAtHead (head, tail, 10);
      insertAtTail (head, tail, 50);
      insertAtPosition (head, tail, 60, 4);
      print (head);
    }
```

O/P → 10 -> 20 -> 30 -> 60 -> 40 -> 50 ->