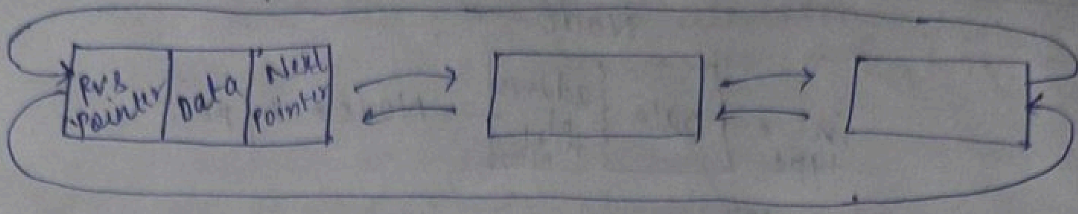
④ circular Doubly Linked List:

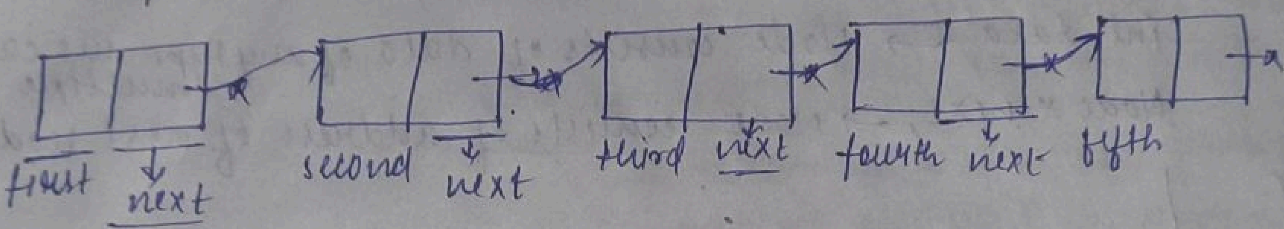


* Magical Line of ou Linked List:

~~[illegible]~~ → cooking maggie steps involve to cook a maggie sim. If you understand, how to cook a maggie & steps involved in it. It is easy.

Linked List in Hindi [Important from understand point of view]

* How to print the Nodes?



which is pointer hold address of next node.

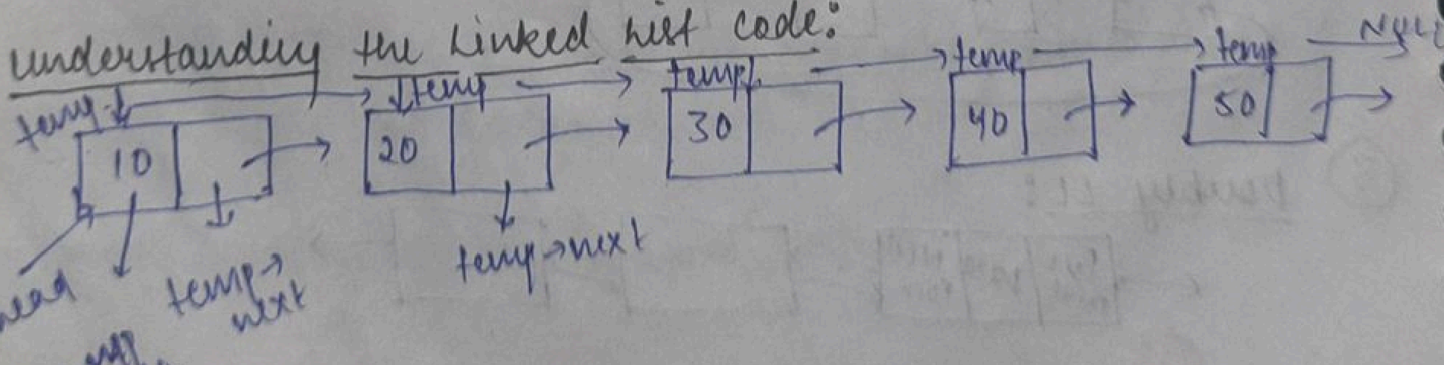
first → next = second;
second → next = third;
third → next = fourth;
fourth → next = fifth;

3 steps involve to print nodes

① print the data ② move forward the pointer

③ stop when pointer become null.

* understanding the Linked list code:



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

Output: 10 20 30 40 50

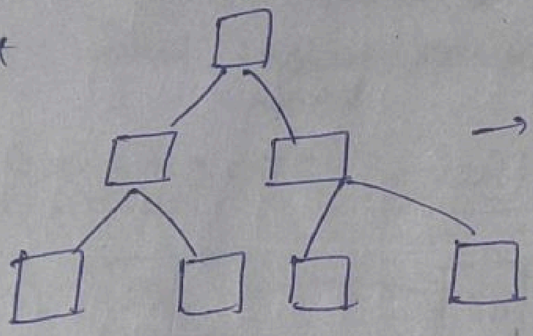
temp

* 5 → 10 → 15 → 20 → 25 → NULL

temp = 1st node
temp → next = 2nd node
temp → next → next = 3rd node
temp → next → next → next = 4th node
temp → next → data = 10

* □ → □ → □ → □ → X → Linear Data Structure

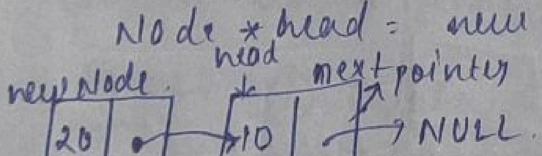
*



→ Non-Linear Data Structure

* Inserting a Node: (At head)

Node * head = new Node(10);



We want to place the next Node left to the current node i.e on the ~~head~~ left to the head node.

insertAtHead(head, 20)
Data

① Create a new Node
② NewNode → next = head;
③ head = NewNode

```
void insertAtHead (Node * &Head, int data){      by reference bcz changes we want in original not in a copy
        Node* newNode = new Node(data);
        newNode → next = head;
        head = newNode;
}

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

~~int~~ It is above all code

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

            Node(){
                this → data = 0;
                this → next = NULL;
            }
            Node(int data){
                this → data = data;
                this → next = NULL;
            }
};

int main(){
        Node *head = NULL;
        insertAtHead(head, 20);
        "     "     ( " , 30);
        "     "     ( " , 40);
        "     "     ( " , 50);
```

print(head);
}

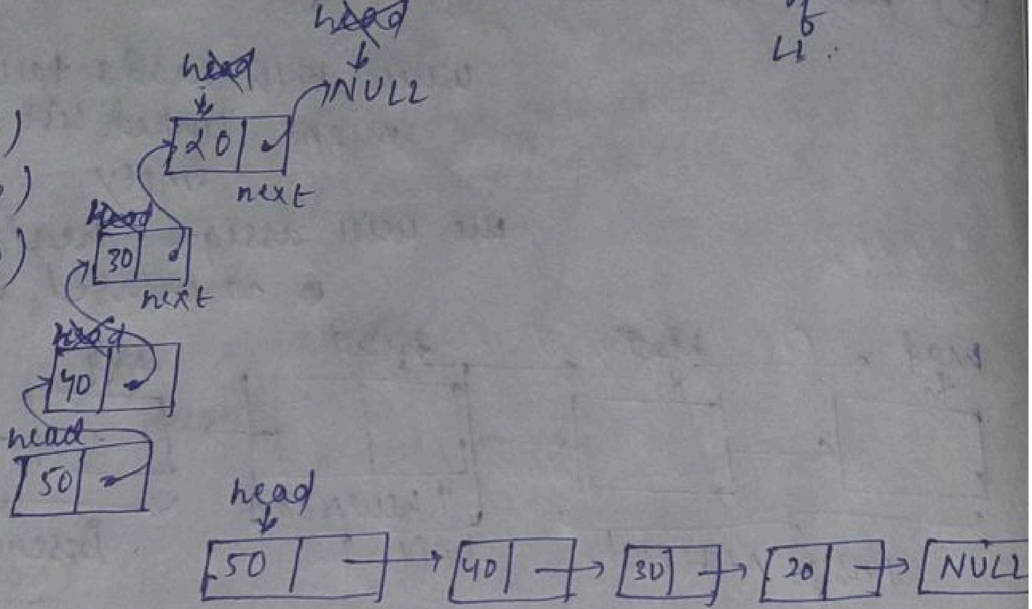
output = 60 50 40 30 20

Q why output in reverse order?

head → starting pointer point of LL.

head → NULL

insertAt(head, 20)
insertAt(head, 30)
insertAt(head, 40)
insertAt(head, 50)

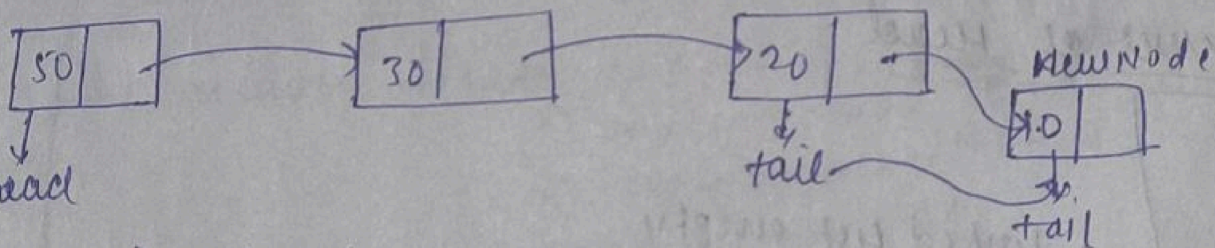


output:

50 40 30 20.

* Note If we insert a Node at Head then the elements will be printed in reverse order.

* Insert a Node: (At tail): We want to create a new Node right to the tail Node



insertAttail(tail, 10)

Steps: ① create a Node.
② tail → next = newNode.
③ tail = newNode

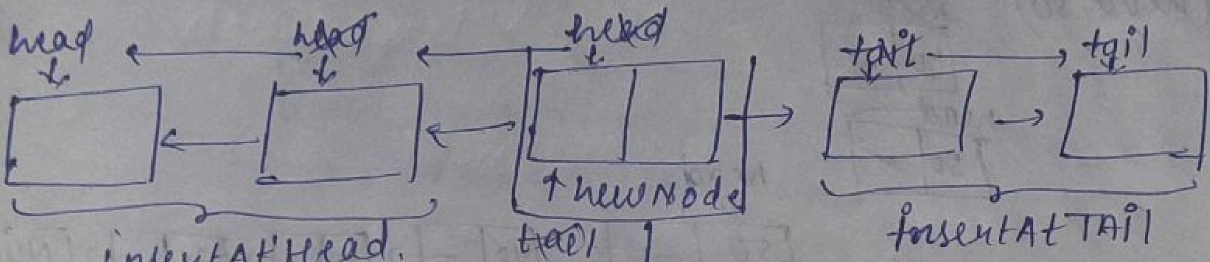
insert At Head

1. create a Node
2. newNode → next = head
3. head = newNode.

Insert At Tail

1. create a Node
2. tail → next = newNode.
3. tail = newNode

when both head & tail are Null
means Linked List is
empty
we will assign head & tail with
to created new Node

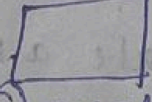


It is basically
first Node which
we are creating when
both head & tail are Null.

left side head will
change with newNode
& in right side tail
will change with
newNode.

* Insert at Head

→ Linked List empty
Head = Tail = NULL



A. create New Node.
B. head = new Node
C. tail = new Node
D. return / if-else

→ Non-empty
A. Create a Node
B. newNode → next = head
C. head = new Node

* Insert At Tail:

- LL is empty → head = tail = NULL
  - (A) create newNode
  - (B) head = newNode
  - (C) tail = newNode
  - (D) return / if-else
- LL non-empty
  - (A) create a node
  - (B) tail → next = newNode
  - (C) tail = newNode.

*. Insert At Position: If you want to place a node at specific position

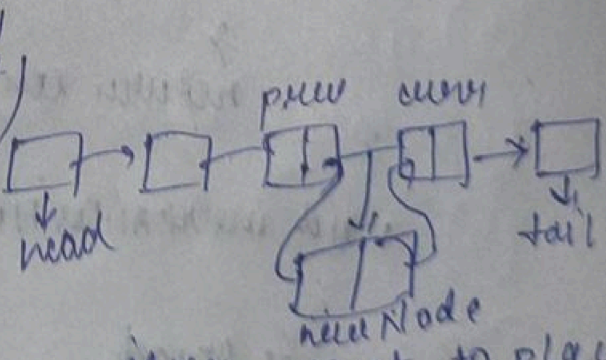
- Place at particular position i.e ex 3^{rd} position
- place after some specific value.
  i.e place a node after value 5.

insertAtPosition (head, tail, data)

- LL is empty → head = tail = NULL
  - (A) create a Node
  - (B) head = newNode
  - (C) tail = newNode
  - (D) return / if-else

  newNode

- Non-empty.
  - (A) find the position (find prev & curr node)
  - (B) create a node
  - (C) newNode → next = curr
  - (D) prev → next = newNode.



if you want to place newNode b/w prev & curr for non-empty insert at position we can't do step (D) before step (C) bcz if we do so we will loose the track of all the nodes right to the newNode if this happen we won't able to find our right most all Node that's why step C will be newNode → next = curr & step D prev → next = newNode.

* Code:

Note: During writing code of LL it is sure there will be error we can't find the edge cases. It's ok.

```
class Animal {

    from previous

}

insertAtHead() {

}
insertAtTail() {

}

//find length of all total nodes present
int findLength(Node* head) {
    int len = 0;
    Node *temp = head;
    while (temp != NULL) {
        temp = temp -> next;
        len++;
    }
    return len;
}

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

    // check empty LL case
```

```
//non-empty LL
if (position==0){
    insertATHead (head, tail, data);
    return;
}

int len = findLength (head);
if (position >= len){
        insertATTail (head, tail, data);
        return;
}

//step1;
int i=1;
Node *prev = head;
 while (i < position){
            prev = prev->next;
            i++;
 }
 Node* curr = prev->next;
Node* newNode = new Node(data);
newNode->next = curr;
prev->next = newNode;

}    //print the Node from previous
int main(){
   insertATposition (position, head, tail, 101);
   print(head);

}
```

output : position: 4

101

* Deletion of Node

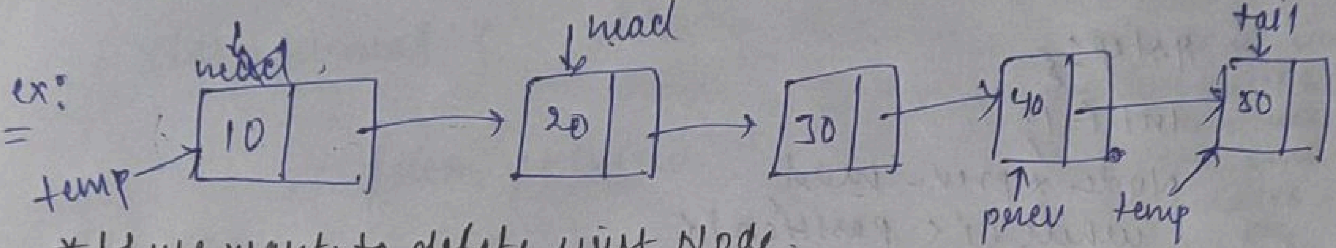
You can delete a node from three different position

- → start
- → middle
- → end.

→ for start/end make sure to update ~~the~~ head & tail.

handle both cases of empty & non-empty linked list.

ex:



* If we want to delete first Node.

(A) shift the head to next node

* create temp Node

(B) temp → next = NULL

· Now you can see Node [10 |] → NULL is single Node. we can delete it by calling destructure. in class.

(C) delete temp // dynamic memory object creation (manually calling destructor with delete keyword)

* If we want to delete last Node

*create temp Node

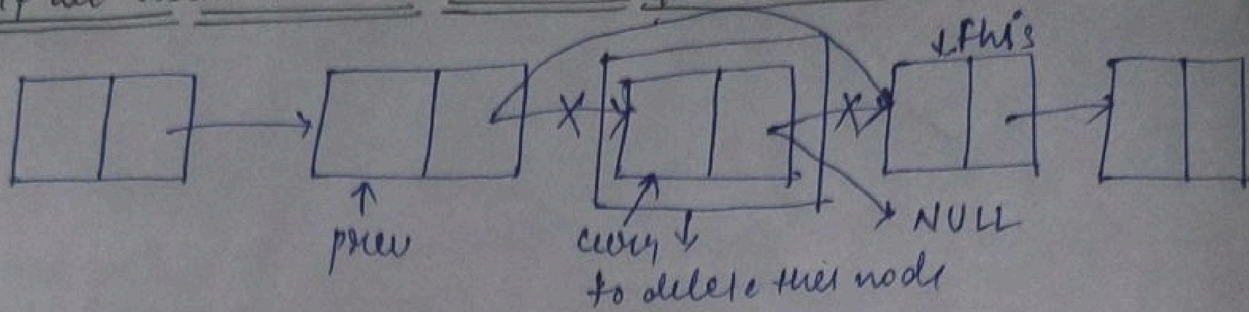
(A) find the prev Node (As shown above)

(B) prev → next = NULL

(C) tail = prev

(D) delete temp (manually

* If we want to delete a node from middle:



(A) find the prev node.

(B) prev→next = curr→next [to join prev pointer to curr pointer ki next se (i.e this node)

(C) curr→next = NULL [curr node ko null kar do taki single node without pointing ban jake.

(D) delete (curr). (delete kar do)