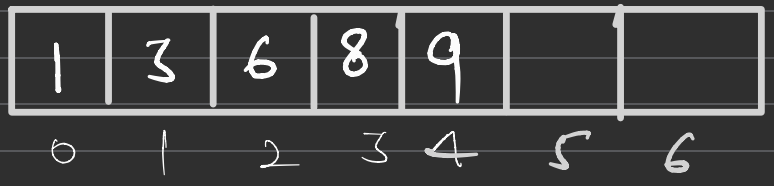
Doing the best at this moment, puts you in the best place for the next moment.

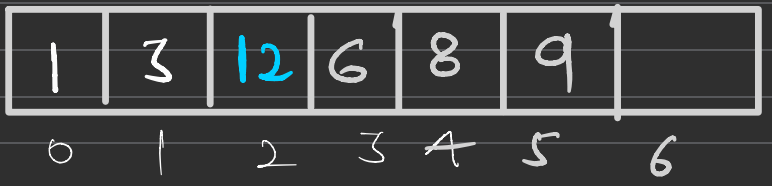
If we have Array, then Why Linked List

dis-advantages

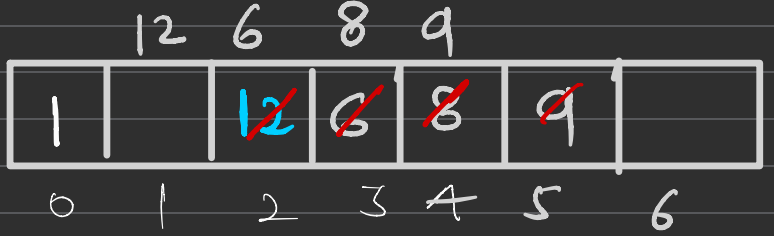


Element = 12

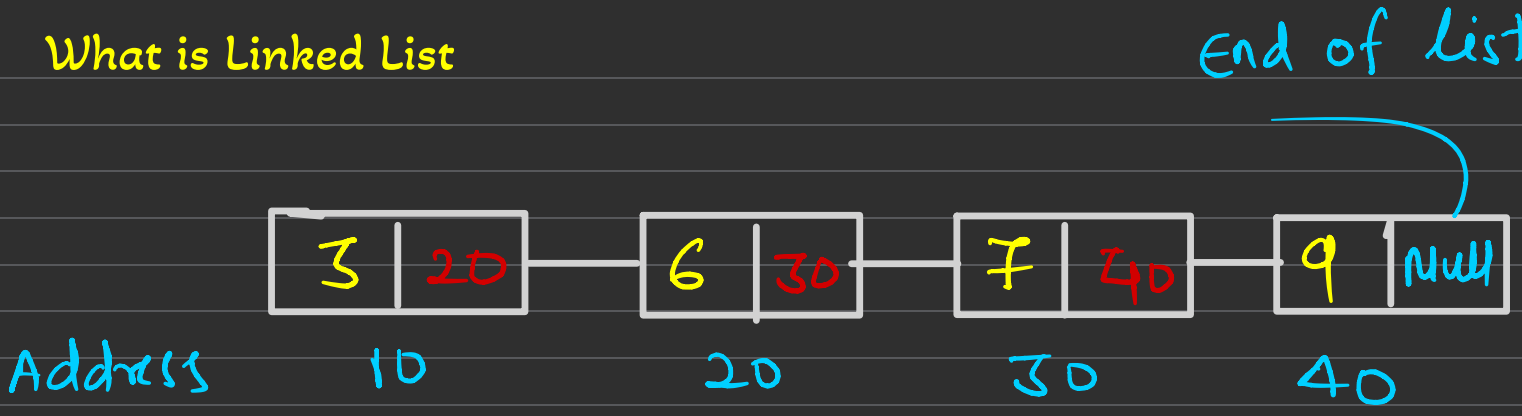
(insert at index 2)



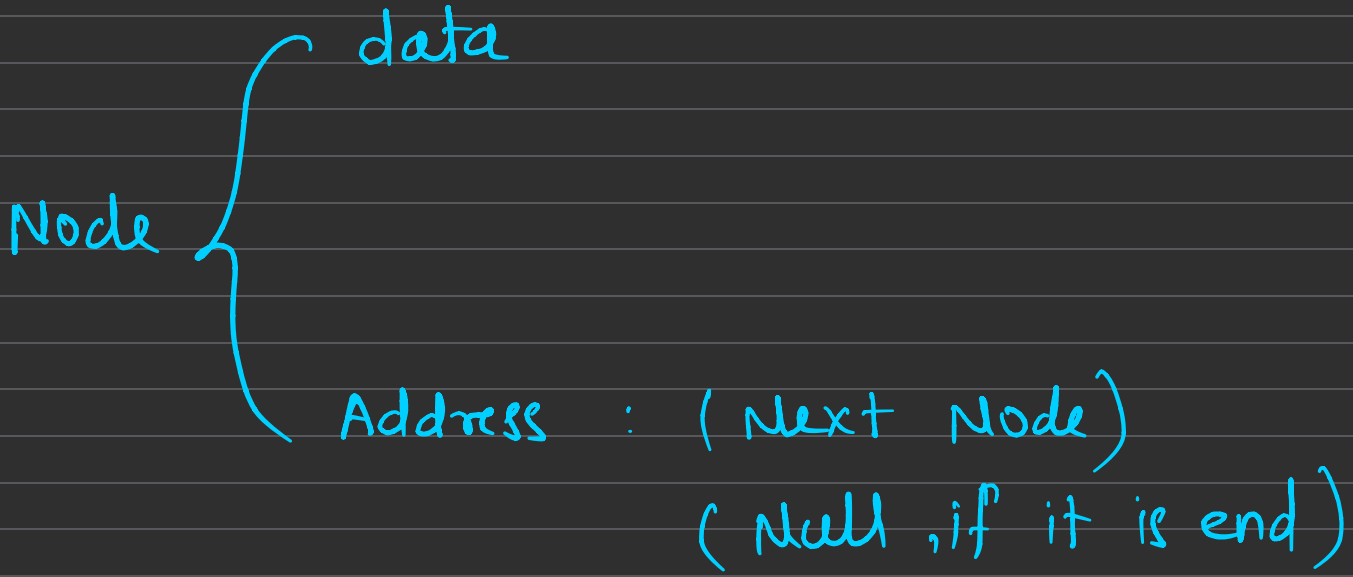
Please delete element from index 1



What is Linked List



Collection of data, which is represented in term of Node.



Head / Start Node

- if head == null → list is Empty
- Not to change head pointer

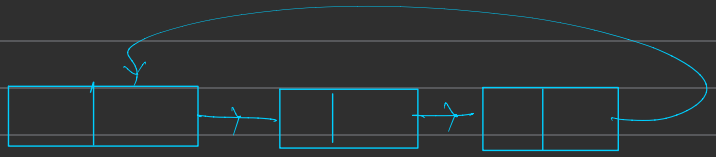
- Head indicates start of list

Types of Linked List

Singly Linked List

Doubly Linked List

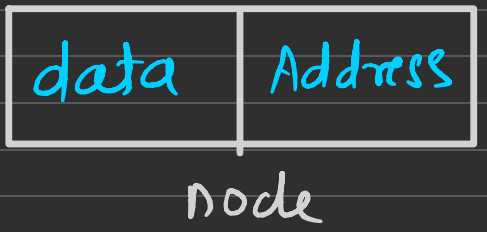
Circular Linked List



- Types of operation:

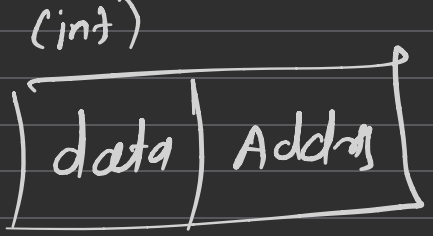
* Insertion
  - start
  - end
  - Position
* Deletion
  - start
  - end
  - Position
* Traversal

Structure of Linked List (SLL)



C++

```
class Node
{
    int data
    Node * next

};
```

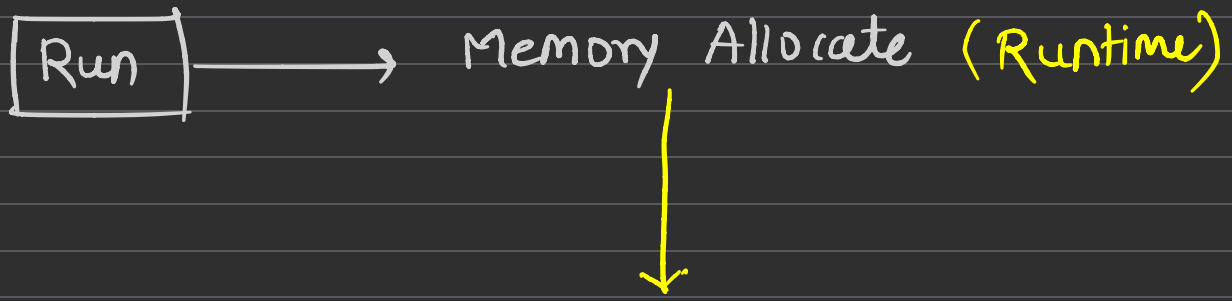


Java

```
class Node
{
    int data
    Node next

}
```

How to create Node

int x



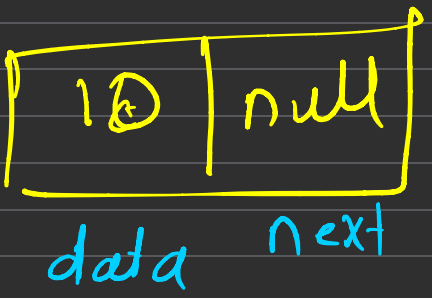
* new keyword allocates memory Dynamic memory, from heap

```
Node *node = new Node()
```

Implementing SLL

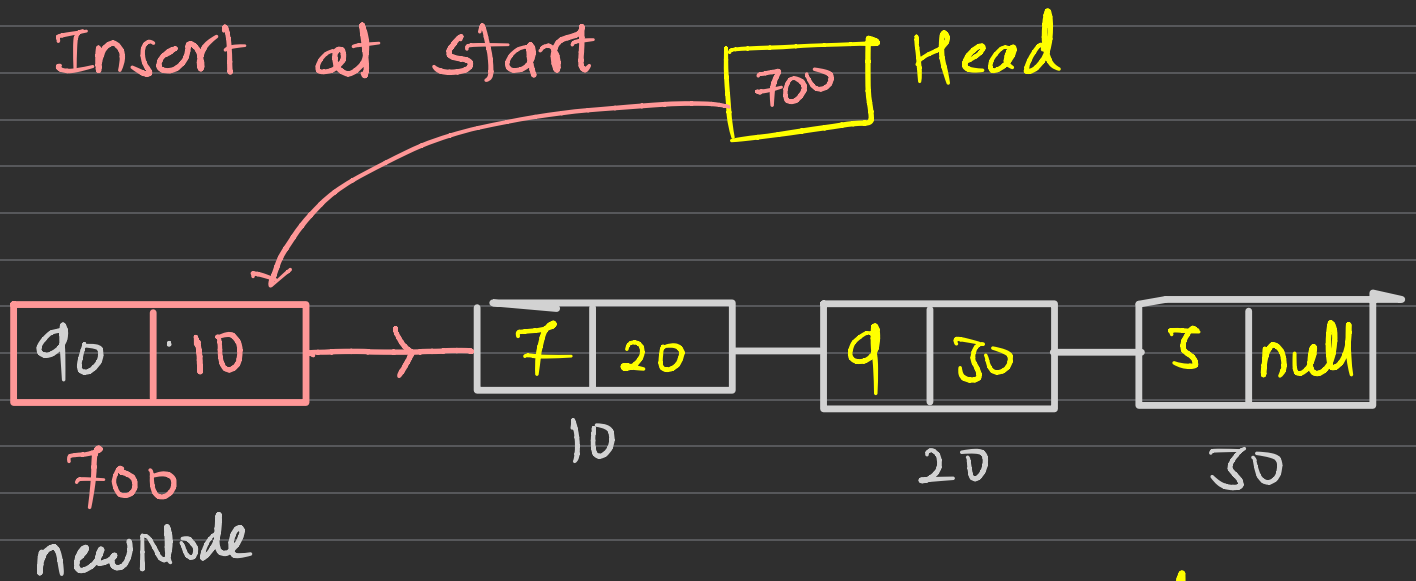
```
class Node
{
    int data
    Node * next

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

```
Node *node = new Node(10)
```



* Insert Operation

① Insert at start



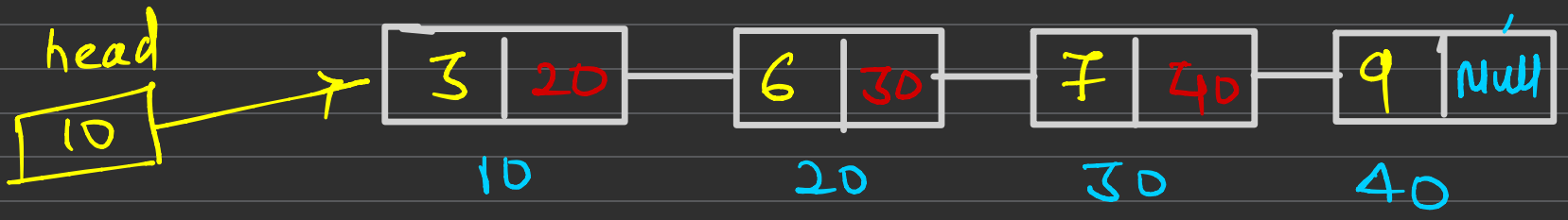
newNode → next = head

head = newNode

head = null (list is Empty)

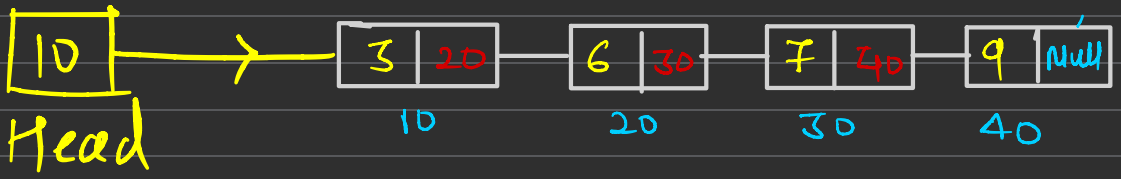
```
if (head == null)
{   head = newNode

}
else
{   newNode -> next = head
    head = newNode
}
```

* Traverse



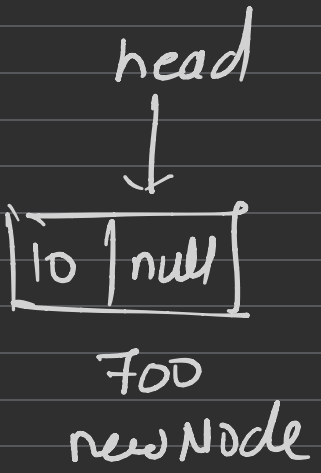

```
Node * temp = head
while( temp != null)
{
    cout << temp -> data
    temp = temp -> next
}
```

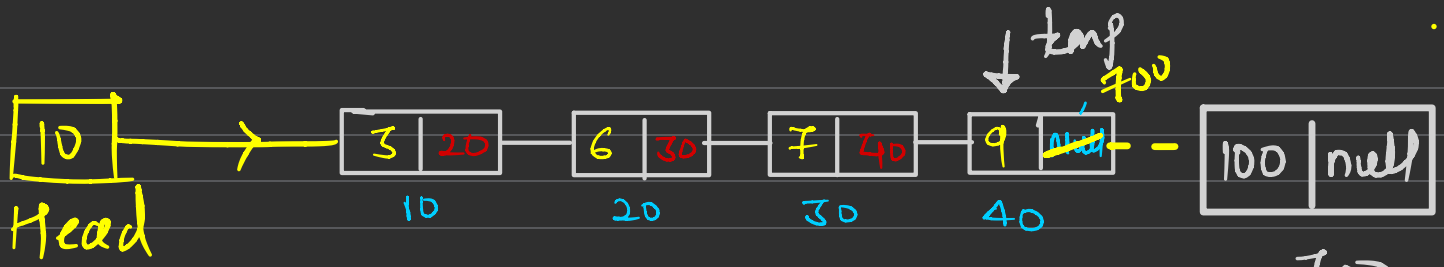
* Insert at End



// head = null

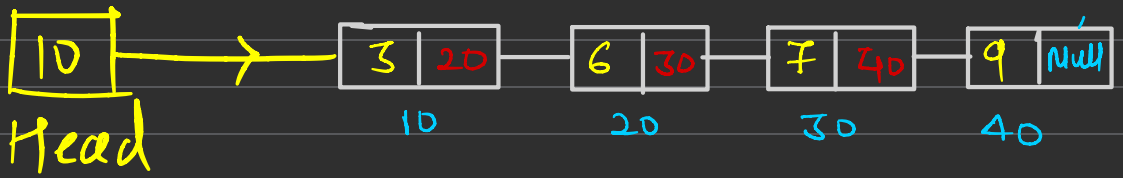
```
if (head == NULL)
  ? head = newNode
```






```
Node * temp = head

while ( temp -> next != Null)
{
    temp = temp -> next
}

temp -> next = newNode
```

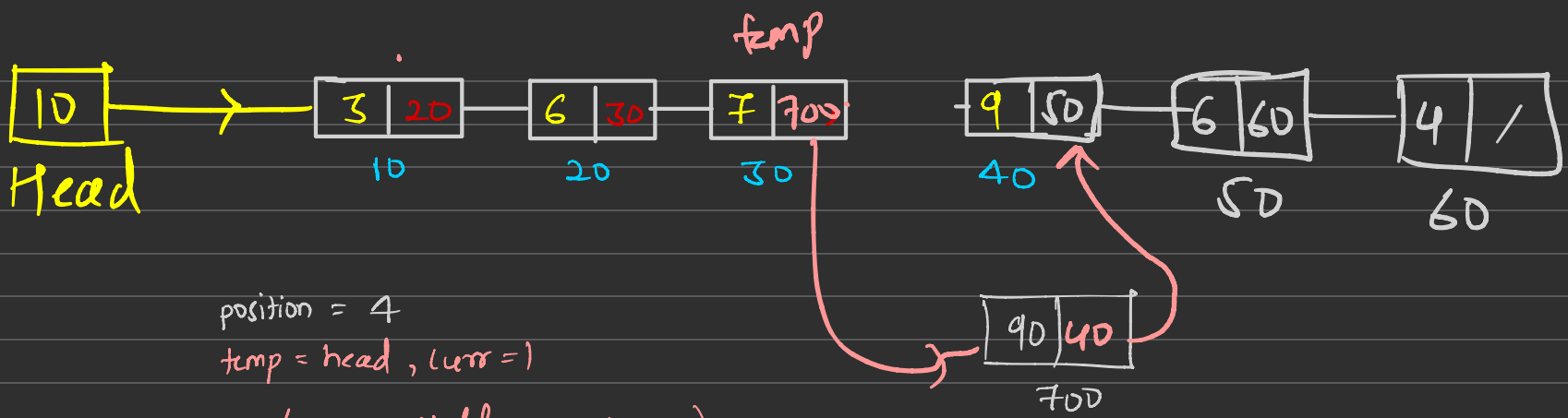
Insert by Position:



① if (position > cnt+1)
invalid

② position = 1 ⟶ insertion at start

③ position == (cnt+1) ⟶ insertion at end



position = 4

temp = head , curr = 1

```
while (temp != null && curr < position - 1)
{
    temp = temp → next
    curr++
}
```

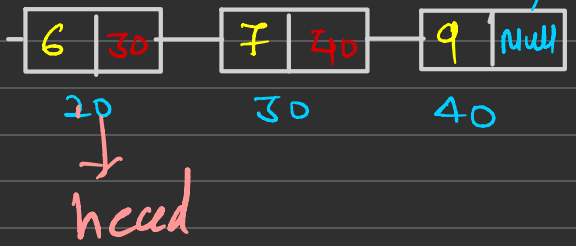
newNode → next = temp → next

temp → next = newNode

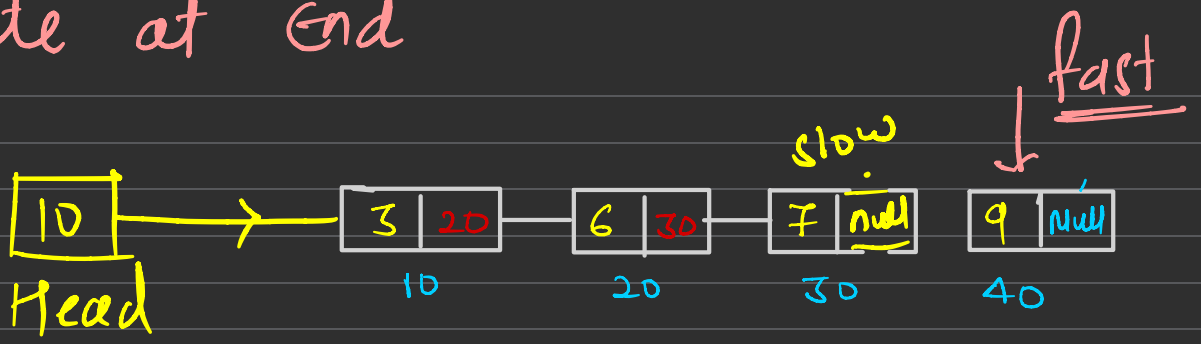
— Delete

⊕ delete at start

```
if(head == Null)
{
    empty
}
else
{
    temp = head
    head = head->next
    delete(temp)
    count--
}
```



– delete at End




```
fast = head, slow = Null
while (fast → next != Null)
{
    slow = fast
    fast = fast → next
}
```

slow → next = null

delete (fast), count--