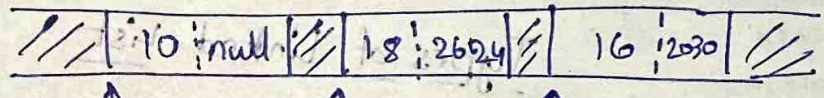
DATA STRUCTURES & ALGORITHMS

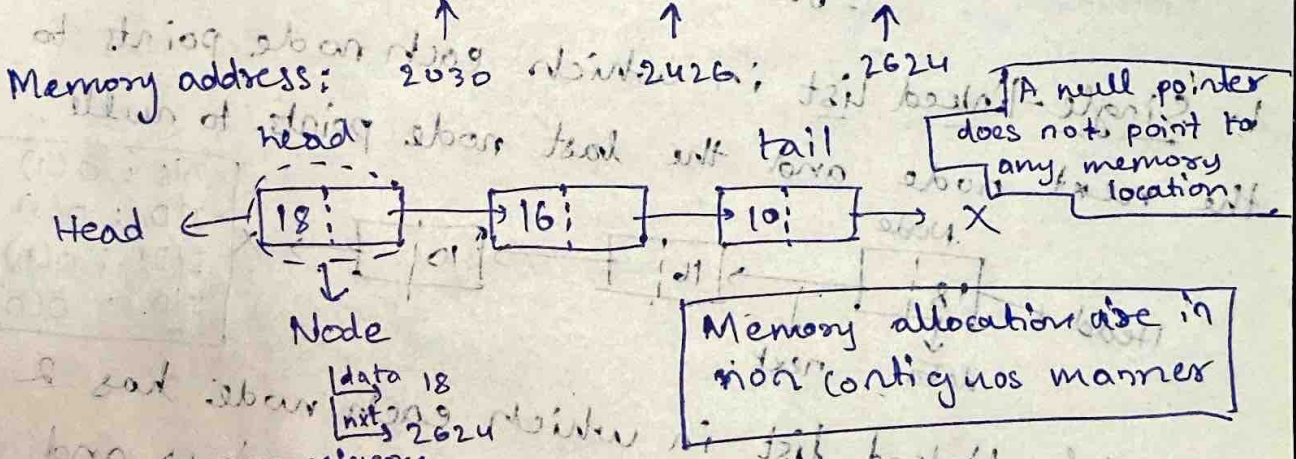
Linked List

Linked list is an linear data structure, which consists of a group of nodes in a sequence (or) Linked List ~~is which we~~ uses nodes to store data in linear. and not in contiguous memory allocation.

Unlike Array where we define the size first, linked list is dynamic, we don't have to define it's size.

In linked list we can add elements as many as we want, But, in array size is fixed.





Linked list is a collection of nodes where each node contains data as well as the memory allocation of the next node in the list.

Advantages

1. Dynamic Nature
2. Optimal insertion & deletion
3. Stack's & queues can be easily implemented
4. No memory wastage

Disadvantages

1. More memory usage due to address pointer
2. Slow traversal compared to arrays.
3. No reverse traversal in singly linked list.
4. No random access.

A stack is a linear data structure that follows last In, First Out principle, widely used in backtracking algorithms, and managing function calls and returns.

Queue is a linear data structure that follows FIFO principle, used in job scheduling, BFS and handling tasks

Real-life applications:

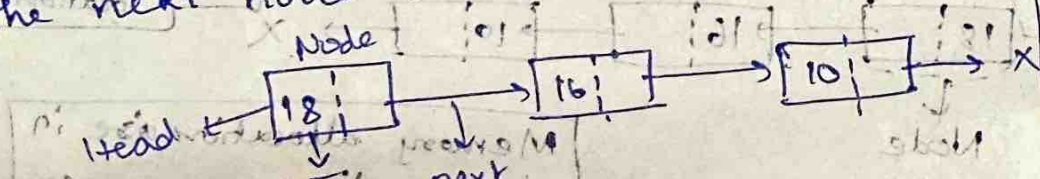
→ Previous - next page browser.

→ Image Viewer.

→ Music Player.

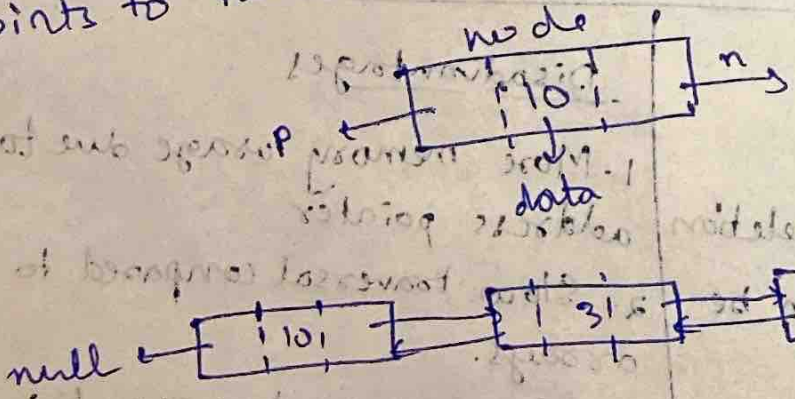
Types of Linked List

1. Single-linked list in which each node points to the next node and the last node points to null.



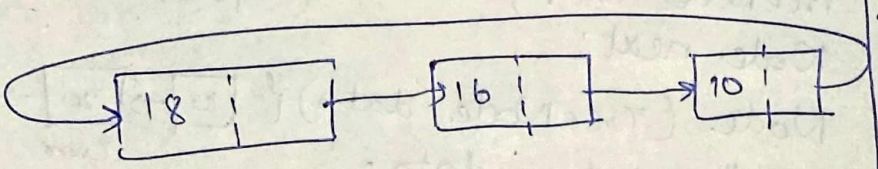
A/S : O(1)
I/D : O(1)
I/D : O(n)
I/D : O(n)
m

2. Double-linked list in which each node has 2 pointers, p and n, such that p points to previous and n points to next node, the last node's n pointer points to null.



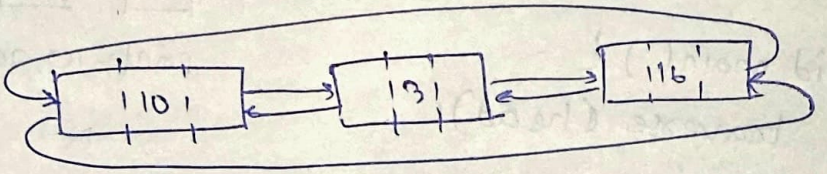
A/S : O(n)
start I/D : O(1)
end I/D : O(1)
middle I/D : O(n)

3. Circular-linked list: in which each node points to the next node and the last node points back to the first node



A/S : O(n)
start I/D : O(1)
end I/D : O(n)
mid I/D : O(n)

4. Circular-Doubly-linked list:



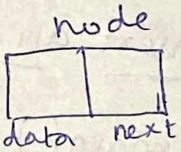
Time complexity:

Access: O(n)
Search: O(n)
Insert: O(1)
Remove: O(1)

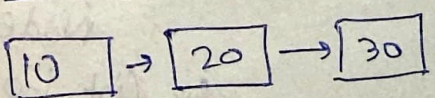
/* Time complexity is a measure of the amount of time an algorithm takes to complete as a function of the input size */

How to create a linked list?

```
class Node{
    int data;
    int next;
    node (int data){
      this.data =data;
    }
}
void main () {
    Node n1 = new Node (10);
    Node n2 = new Node (20);
    Node n3 = new Node (30);
    Node head = n1;
    Node.next =
      n2.next = n3;
      n3.next = null;
}
```



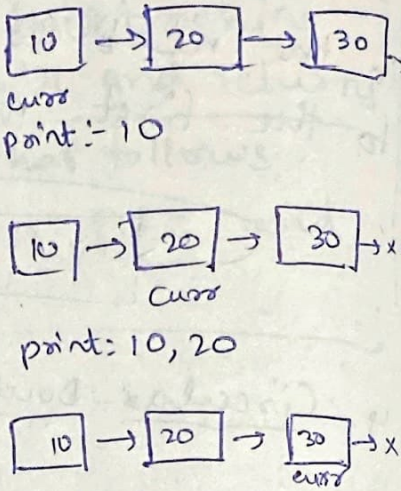
Output:



how we traverse in linked list?

Code:

```
Class Node <TreeNode> {
    TreeNode data;
    Node next;
    Node (TreeNode data) {
        this.data = data;
    }
}
void main() {
    traverse (head);
}
void traverse (Node head) {
    Node curr = head;
    while (curr != null) {
        print (curr.data);
        curr = curr.next;
    }
}
```



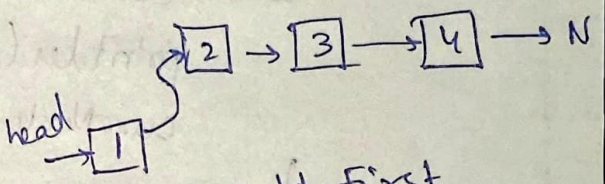
how to insert an node in linked list?

Code:

```
void main() {
    insert (30, head, 3);
}
void insert (int data, Node head, int pos) {
    Node toAdd = new node (data)
    if (pos == 0) {
        toAdd.next = head;
        head = toAd;
        return;
    }
    Node prev = head;
for (int i=0; i<pos-1; i++) {
    prev = prev.next;
}
toAdd.next = prev.next;
prev.next = toAdd;
```

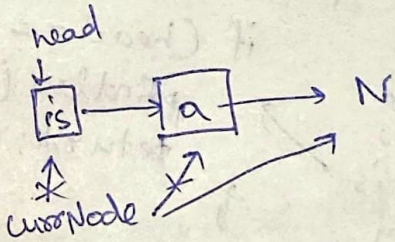
** Don't update head ** instead update currNode **

```
class LL {
    Node head;
    class Node {
        String data;
        Node next;
        Node (String data) {
            this.data = data;
            this.next = null;
        }
    }
```

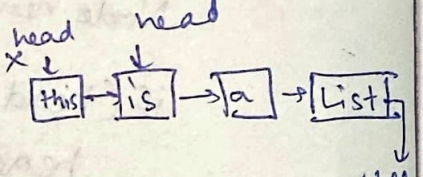


Add-first

```
    // addFirst
    public void addFirst (String data){ /* Initially, new node is pointing
                                           towards null, change it to point
        Node newNode = new Node (data); towards the node where head is
                                        present. Now, change head pos to
        if (head == null){              1st node */
            head = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public void addLast(String data){
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node currNode = head;
        while (currNode.next != null) {
            currNode = currNode.next;
        }
        currNode.next = newNode;
    }
}
```



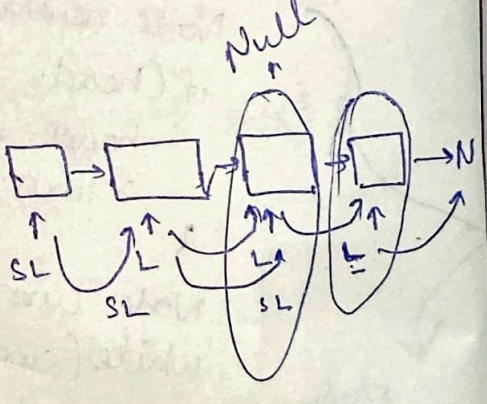
```
public void printList() {
    if (head == null) {
        println ("List is empty");
        return;
    }
    Node currNode = head;
    while (currNode != null) {
        println (currNode.data + " → ");
        currNode = currNode.next;
    }
    println ("Null");
}

public void deleteFirst() {
    if (head == null) {
        println("List is empty");
        return;
    }
    head = head.next;
}
```



To delete first node, make the head.next as a head.

```
public void deleteLast() {
    if (head == null) {
        println ("List is empty");
        return;
    }
    if (head.next == null) {
        head = null;
        return;
    }
    Node secondLast = head;
    Node lastNode = head.next;
    while (lastNode.next != null) {
        lastNode = lastNode.next;
        secondLast = secondLast.next;
    }
    secondLast.next = null;
}
```

Corner Case: if List is empty & if only one node present



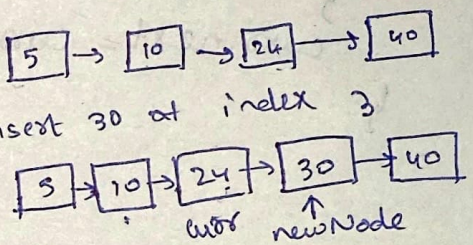
LinkedList methods:

list.add(data);
list.addLast(data);
List.addFirst(data);
list.add(index, data);
	list.remove(index);
	list.removeFirst();
	list.removeLast();

list.get(index); list.size();	

How to insert in the middle of a LinkedList (at index)?

```
public void addInMiddle(int index, String data){
    if(index > size || index < 0){
        println("Invalid");
        return;
    }
    size++
    Node newNode = new Node(data);
// if list is empty or index is '0', insert at start
    if( head==null || index==0){
        newNode.next = head;
        head = newNode;
        return;
    }
// Traverse the list to find the node at specified index
    Node currNode = head;
    for(int i=1; i<size; i++){
        if(i==index){
            // Insert newNode b/w currNode & nextNode
            Node nextNode = currNode.next;
            currNode.next = newNode;
            newNode.next = nextNode;
            break;
        }
        currNode = currNode.next;
    }
```



Delete node at specified index:

```
void main() {
  delete(head, 3);
}
void delete(Node head, int index){
  // base condition
  if (index==0){
      head = head.next;
      return;
  }
  Node curr = head;
  for (int i=0; i< index-1 ; i++) {
       curr= curr.next;
  }
  curr.next= curr.next.next;
}
```