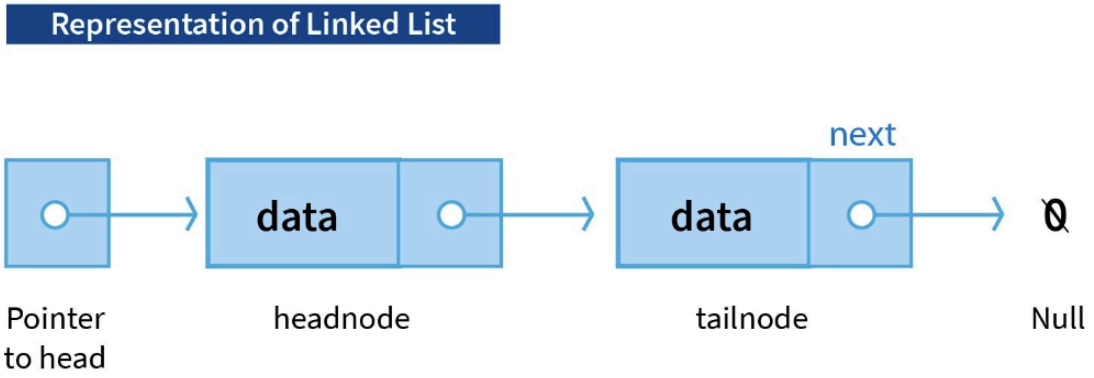
LinkedList

A Linked List is a linear data structure consisting of connected nodes where each node has corresponding data and a pointer to the address of the next node. The first node of a linked list is called the Head, and it acts as an access point. On the other hand, the last node is called the Tail, and it marks the end of a linked list by pointing to a NULL value.



- Structure of the Node in Linked List :

```
Class Node {
    Int data ;
    Node next ; // This is used to store the reference of the next node
}
```

- How to move the reference variable “head” to the next node ?
    - *head = head.next*
      (But never use your head to traverse the single linked list because if you do so you’ll not be able to come back to the head of the linked list if you want to.)

    - Better approach ->

```
temp = head;
temp = temp.next
```

- How to traverse the Linked List until end
    - Keep moving forward to next node until your reference becomes NULL

```
temp = head;
```

```
while(temp != NULL) {
    temp = temp.next;
}
```

- What if we want to stop at the last node ?
  Change the condition in while loop

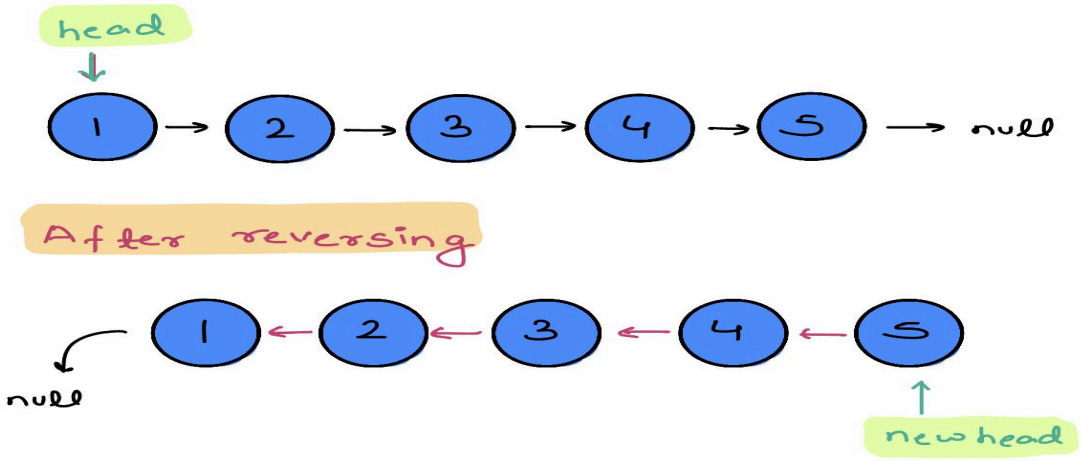
```
temp = head;

while(temp.next != NULL) {
    temp = temp.next;
}
```

  - Is this code completely accurate ? NO
    - This would throw *NULL POINTER EXCEPTION* when the linked list is empty, “temp” will be initialized to NULL and NULL references don't have any property “next”.

  - Correct implementation :

```
temp = head;

while(temp != NULL && temp.next != NULL) {
    temp = temp.next;
}
```

Reverse a LinkedList by modifying pointers

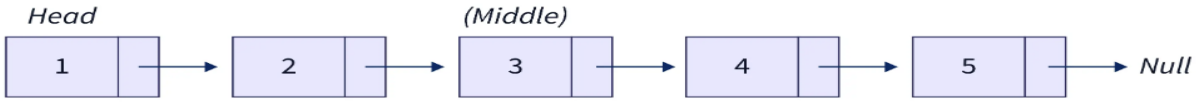


Code to reverse the LinkedList

```
Node reverse ( Node head) {

        Node curr = head;
        Node prev = null;
        Node forw = null;

        while( curr != null ) {
              forw = curr.next;
              curr.next = prev;
              prev = curr;
              curr = forw;
        }
       return prev;
}
```

Middle of LinkedList

Eg 1:-



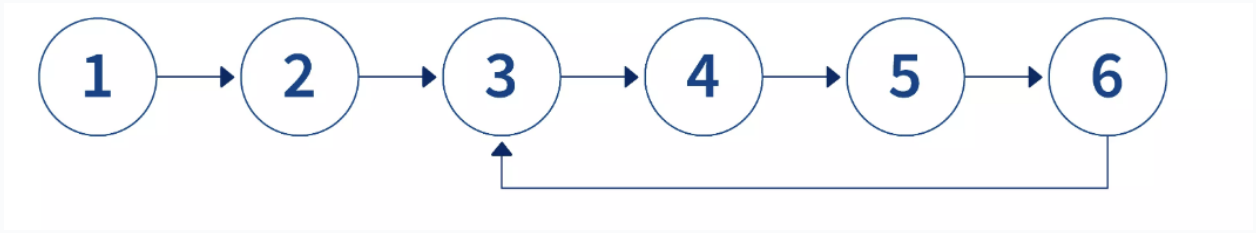
Eg 2:-

Code to find middle of LinkedList using Slow & fast ptr technique

```
void print_middle(Node head) {

        Node slow = head;
        Node fast = head;

        While ( fast != null && fast.next != null ){

                fast = fast.next.next;  // taking a jump of two
                slow = slow.next;   // taking a jump of one
         }

        System.out.println(slow.data);
}
```

Detect Loop in LinkedList

Given a linked list, check whether the linked list is having a loop or not(detect loop in linked list). A cycle exists in a linked list if it contains a node that may be accessed again by following the next pointer.



We can detect the loop in the linked list using **Floyd's Cycle**. This is the fastest method for detecting a loop in a linked list:

- Traverse the linked list using two pointers, a fast pointer, and a slow pointer starting from the first node.
- Now in a loop move the fast pointer by 2 nodes and the slow pointer by 1 node.
- If both the pointers point to a same node then loop is detected and return true, else if fast pointer details the end of the linked list return false

```
boolean hasLoop (Node head) {
	Node slow=fast;
	Node fast=head;

	while( fast !=null && fast.next!=null) {
		slow=slow.next;
		fast=fast.next.next;
		if(slow == fast) break;
	}
if(fast==null) return false; //fast ptr has reached null without meeting slow ptr
else  return true;
```

- Important Interview Problems
  - Remove Loop from Linked List
  - LRU Cache

Revision Video - https://drive.google.com/file/d/1edSzhani6dGv7c8QGBm-Q-MJN6_aoCwJ/view?usp=share_link