=> LinkedList :-

=>LinkedList is an implementation class of List interface which is present in java.util package

=>Syntax : public class LinkedList extends AbstractSequentialList implements List, Deque, Cloneable, Serializable { - }

-The underline data structure of LinkedList is Double Linked List or Circular Linked List

-LinkedList was introduced in JDK 1.2 version

-Properties of LinkedList :-

1. LinkedList is an index based Data Structure which means that first element will be inserted at 0 index position
2. LinkedList can store different data types or hetrogeneous elements
3. We can store duplicate elements in the LinkedList
4. We can store any number of null values in the LinkedList

=> Working of LinkedList :-

1. Types of LinkedList (all programming languages)

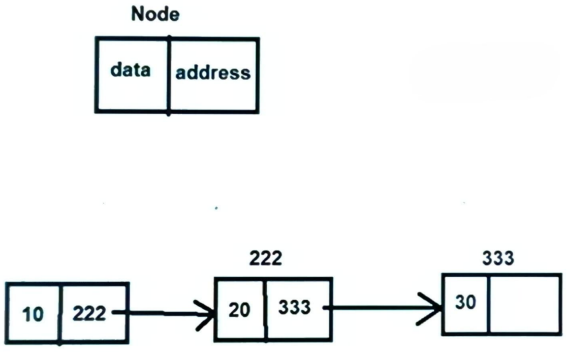
a. Single Linked List

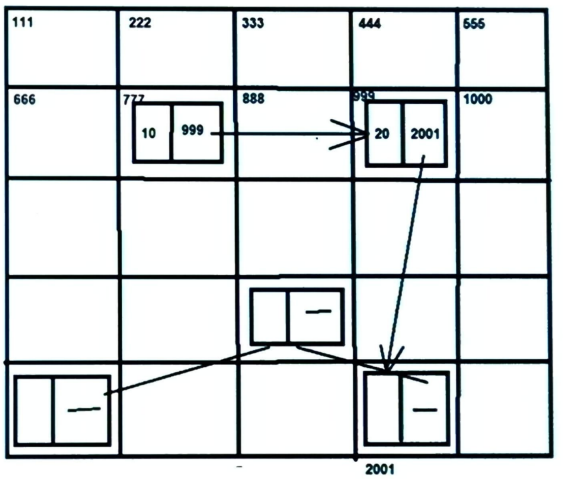
b. Double Linked List

c. Circular Linked List

2. Linked List are linear data structure in whichelements are not stored in contigeous memory locations.

3. There is no capacity concept in LinkedList like ArrayList






```
public class LinkedListDemo {
    public static void main(String[] args) {
        LinkedList ll = new LinkedList() ;
        ll.add("Kumar") ;
        ll.addFirst("Arun");
        ll.addLast("Sharma");

        System.out.println(ll);

        //-------------------------------------

        ArrayList list = new ArrayList() ;
        list.add("is") ;
        list.add("Teaching") ;
        list.add("Java") ;

        ll.add(list) ;
        System.out.println(ll);

        //-------------------------------------

        LinkedList ll2 = new LinkedList(list) ;
        ll2.addFirst("Arun Kumar Sharma");
        System.out.println(ll2);

    }
}
```

-> Constructors :-

1. public LinkedList()
2. public LinkedList(Collection c)

-> Methods of LinkedList :-

1. Methods of Collection Interface
2. Methods of List Interface
3. public void addFirst(Object obj)
4. public void addLast(Object obj)
5. public Object getFirst()
6. public Object getLast()
7. public Object removeFirst()
8. public Object removeLast()

-> When we should use LinkedList ?

= LinkedList is best when we have to use insertion or deletion operations

-> When we should not use LinkedList ?

= LinkedList is worst in case of retrieval or searching operations (as LinkedList does not inherit RandomAccess interface)

-> What is difference between ArrayList & LinkedList ?

1. ArrayList underline data structure is dynamic array or resizable array
   LinkedList underline data structure is double linked list or circular linked list

2. ArrayList stores the elements in contigeous memory locations
   LinkedList does not store the elements in Contigeous memory locations

3. ArrayList acts as List
   LinkedList can acts as List or Deque

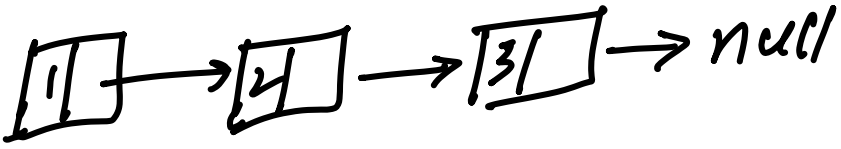
4. ArrayList is good in case of retrieval operations
   LinkedList is good in case of insertion and deletion operations

5. ArrayList is worst in case of insertion or deletion operations
   LinkedList is worst in case of retrieval operations

```
public class LinkedListDemo {
    public static Node head ;
    public static int sizeOfLL(){
        int size = 0 ;
        Node temp = head ;
        while(temp!=null){
            size++ ;
            temp = temp.next ;
        }
        return size ;
    }

    public static void insertNode(int position, int value){
        Node node = new Node(value) ;
        Node temp = head ;
        int size = sizeOfLL() ;
        if(position<1){
            return  ;
        }
        if(position>size+1){
            return ;
        }
        if(position==1){
            node.next=head ;
            head = node ;
        }
        else{
            for(int i=1 ; i<position-1 ; i++){
                temp = temp.next ;
            }
            node.next = temp.next ;
            temp.next = node ;
        }
    }
```

```
    public static void deleteNode(int position){
        int size = sizeOfLL() ;
        if(position<1){
            return;
        }
        if(position>size){
            return;
        }
        if(position==1){
            head = head.next ;
        }
        else{
            Node temp = head ;
            for(int i=1 ; i<position-1 ; i++){
                temp = temp.next ;
            }
            temp.next = temp.next.next ;
        }
    }

    public static void printLL(){
        Node temp = head ;
        while(temp.next!=null){
            System.out.println(temp.value+ " ");
            temp = temp.next ;
        }
        System.out.println(temp.value);
    }
}
```

Q: Create a LL which contains data from 1 to N

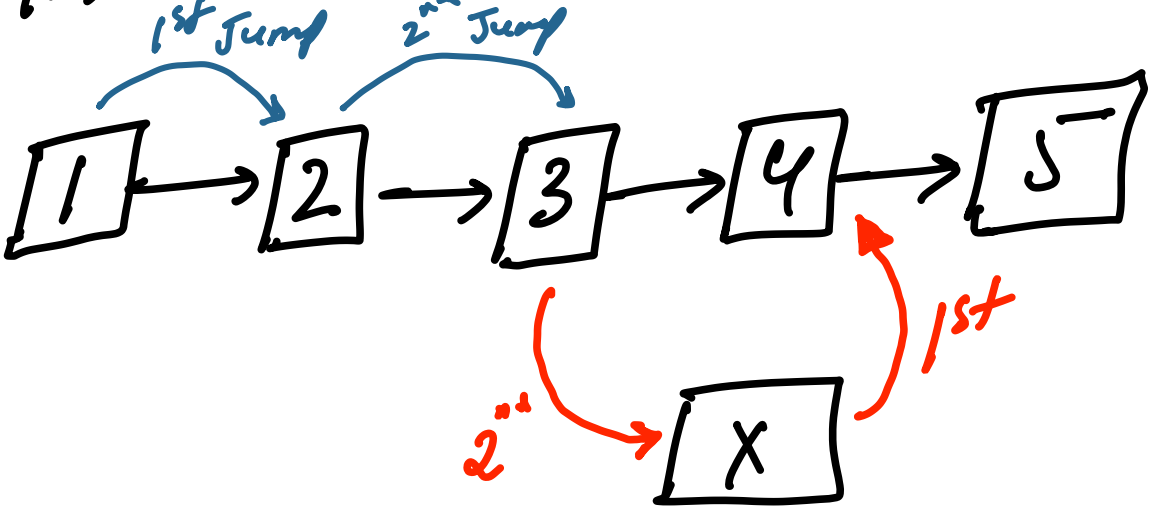



```
public static class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
    }
}
psvm() {
    Node head = new Node(1);
    Node temp = head;
    int i = 2;
    while(i <= N) {
        Node n = new Node(i);
        temp.next = n;
        temp = temp.next;
        i++
    }
    return head;
```

find the size of LL

```
int size = 0
Node temp = head
while (temp != null) {
    size ++
    temp = temp.next
}
}
```

Q: Insert a new node at K^{th} index



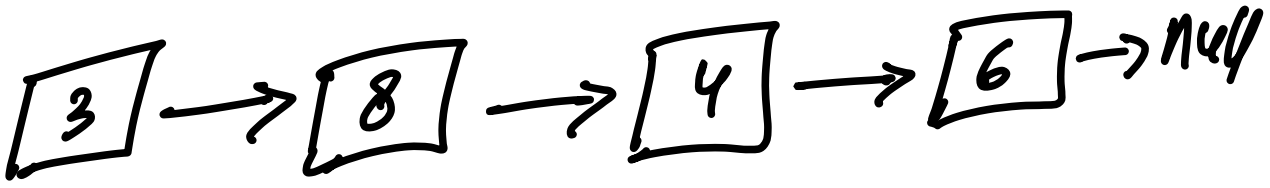
total jumps ⇒ K-1

```
Node n = new Node(x);
int K = 3;
Node temp = head
  i = 1
while (i < K){
    temp = temp.next;
     i++
}
n.next = temp.next
   temp.next = n;
```

Edge case

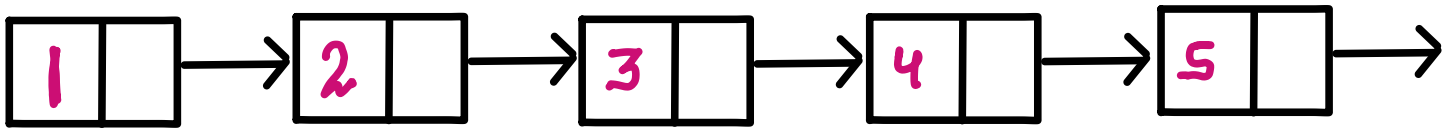
```
if (K == 0){
   n.next = head
     head = n
}
```

Q: Delete node with data x

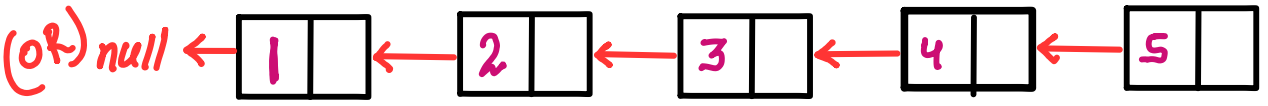


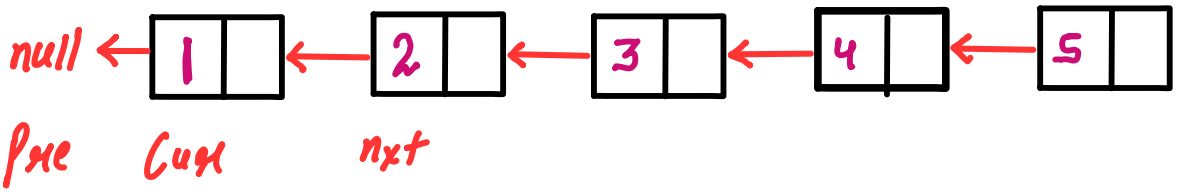

```
if(Head.data == x) {
    Head = Head.next
    return head
}
temp = head
while(temp.next != null){
    if(temp.next.data == x) {
        temp.next = temp.next.next;
        break;
    }
}
```

WAP to Reverse a Linked List:



o/p: 5 → 4 → 3 → 2 → 1 → null






```
Node pre = null;
Node cur = head
  while (cur != null) {
    Node nxt = cur.next;
     cur.next = pre;
      pre = cur;
      cur = nxt;
  }
head = pre;
```

Q: Reverse a LL from B to C position (index)

1 based Index

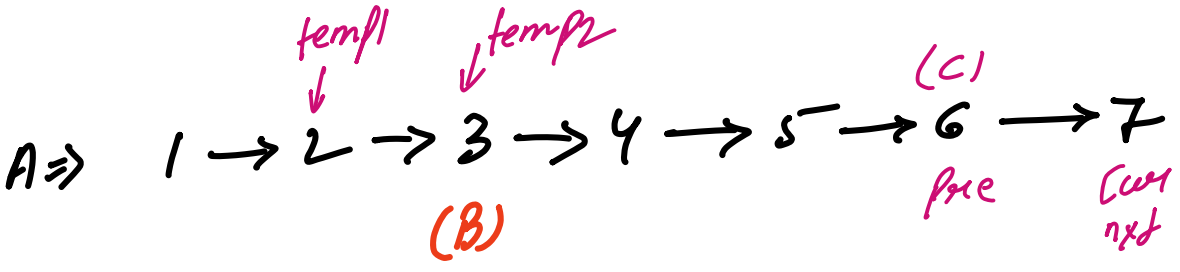
A ⇒ 1 → 2 → 3 → 4 → 5

B = 2
C = 4 ⇒ 1 → 4 → 3 → 2 → 5

B = 1
C = 5 ⇒ 5 → 4 → 3 → 2 → 1

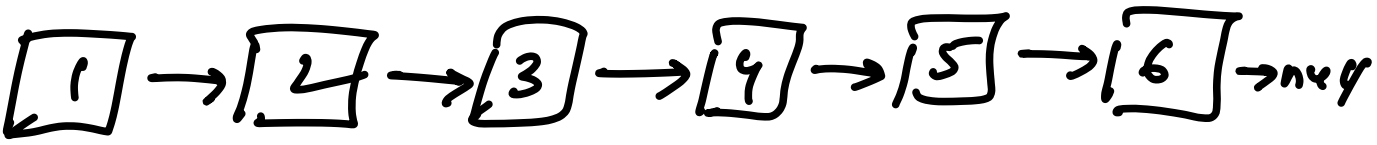
1) Reach till B & also make a note of node before B so that we can connect that with C

2) **Note:** temp1 = null initially, if temp1 remains null till the End that means there is no node before it
   ↳ A = pre



```
Node temp1 = null;  -> Just before B
Node temp2 = A      -> Bth pointed node

int i=1;
while ( i< B) {
    temp1 = temp2
    temp2 = temp2.next
    i++
}
while ( B<= C) {
    Node nxt = cur.next
    cur.next = pre
    pre = cur
    cur = nxt
    B++
}
if(temp1 != null) {   // Some elements are present before B
    temp1.next = pre
}
else {
    A = pre
}
temp2.next = cur ;
return A ;
```

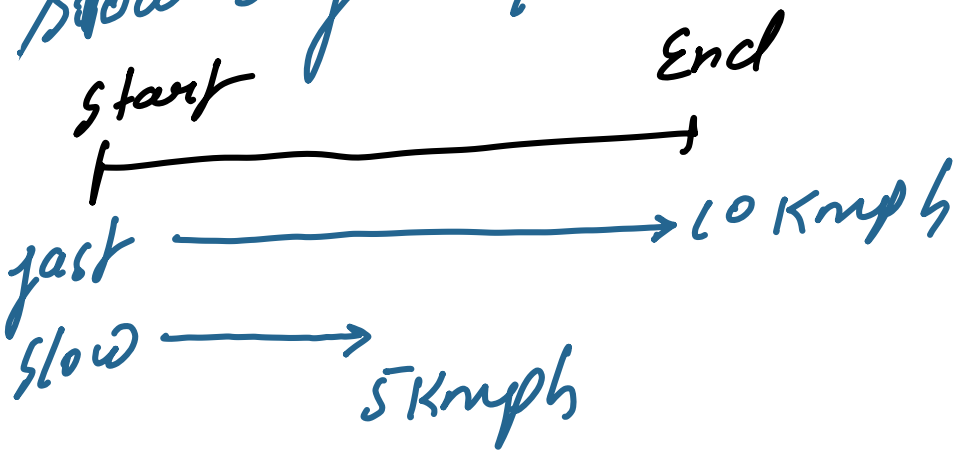
Q: find the middle element of LL



Soln:
```
Node temp = head;
int mid = size/2 + 1;

for (i = 1 to n-1) {
    temp = temp.next;
}
return temp.data;
```

this soln req. 2 loops < 1 for size O(N)
1 for mid O(N)

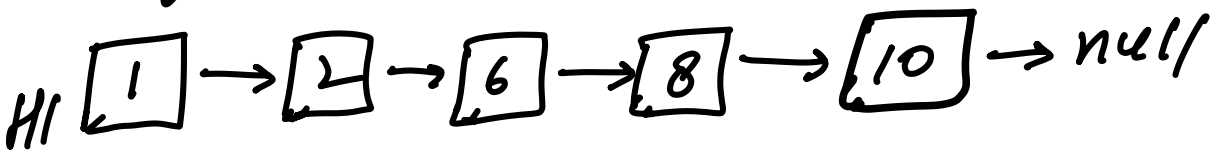
Soln 2: Slow & fast pointer

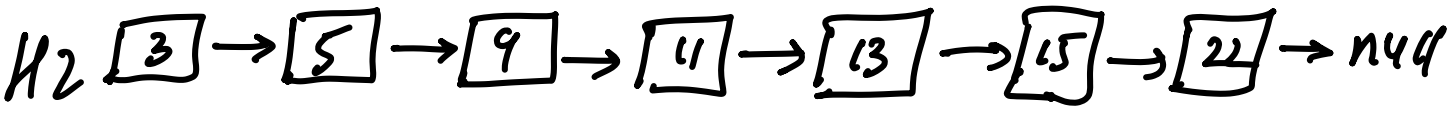


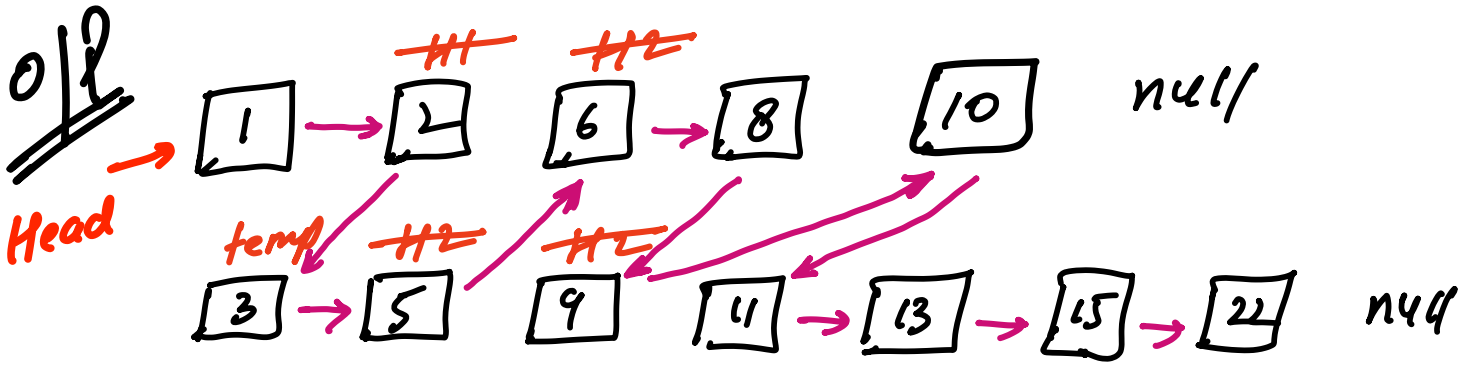
```
Node slow = head;
Node fast = head;

while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
}
return slow;
```

1 loop

Q: merge two sorted list







```
// deciding the head of combined list
if (H1.data < H2.data) {
    Head = H1;
    H1 = H1.next;
}
else {
    Head = H2
    H2 = H2.next
}
// the one which is smaller will move ahead
   & temp will make the link
temp = head;
while (H1 != null && H2 != null) {
    if (H1.data < H2.data) {
        temp.next = H1;
        H1 = H1.next;
    }
    else {
        temp.next = H2
        H2 = H2.next
    }
```

```
        temp = temp.next
    }
if (H1 != null) {
    temp.next = H1;
    }
else {
    temp.next = H2;
}
return head;
```

Q: Given a LL, remove B^{th} node from the end of list & return its head.
if B > size, remove 1st node.

A = 1 → 2 → 3 → 4 → 5

B = 2 1 → 2 → 3 → 5

```
int size = 0
Node temp = A
while(temp != null){        // finding size of LL
    size++
    temp = temp.next
}
if(B >= N){
    A = A.next;
    return A;
}
if (N == 1 && B == 1){
    return null;
}
int i = 1
temp = A
```

```
while( i< (N-3) ){ // reach till Bth node from End.
    temp= temp.next
    i++
}
temp.next = temp.next.next
return A ;
```