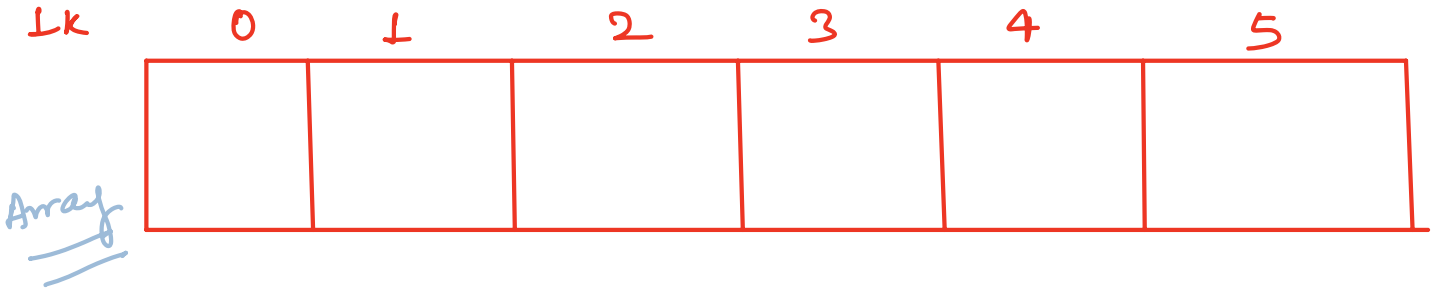
LinkedList

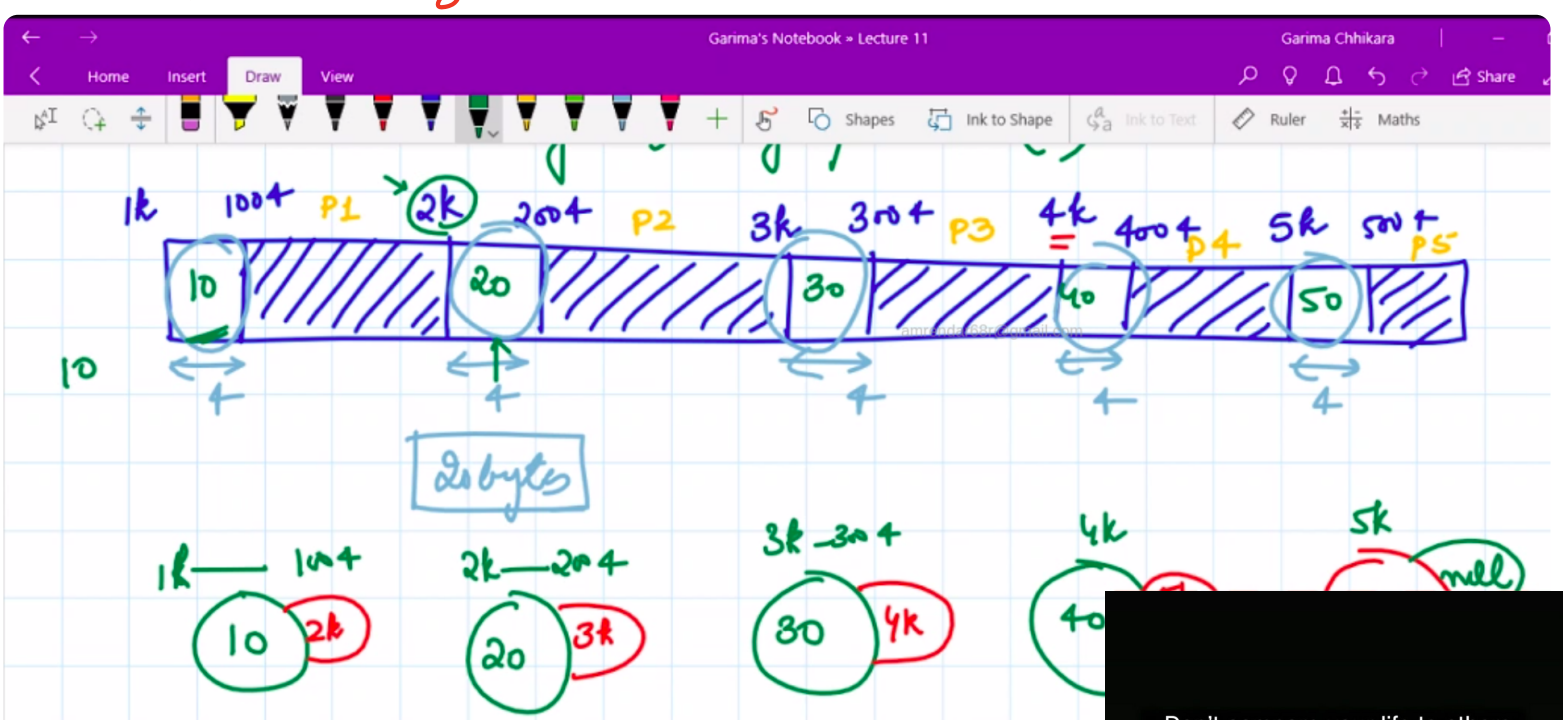
1 int → 4 bytes.

5 int → 20 bytes



Advantages of contiguous memory allocation is we can get a value or set a value in constant time.

once i know the base address we can get any index.

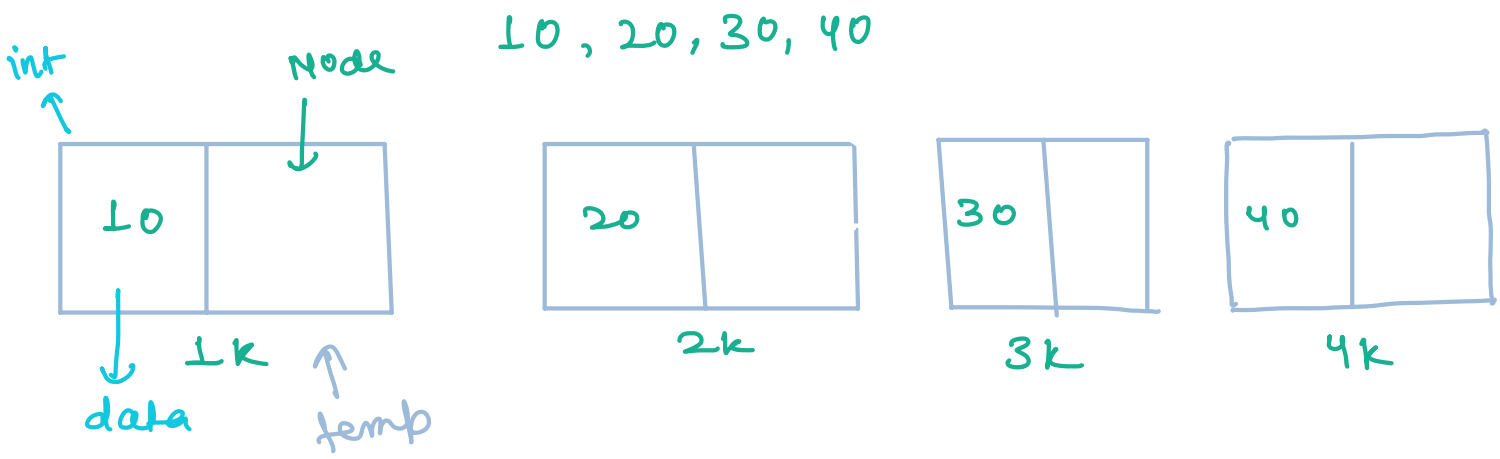


⇒ overview of storing data in the linked list

CODING BLOCKS
Code Your Way To Success

Don't compare your life to others. The sun and the moon both shine when it's their time.

Implementation of linked list :-

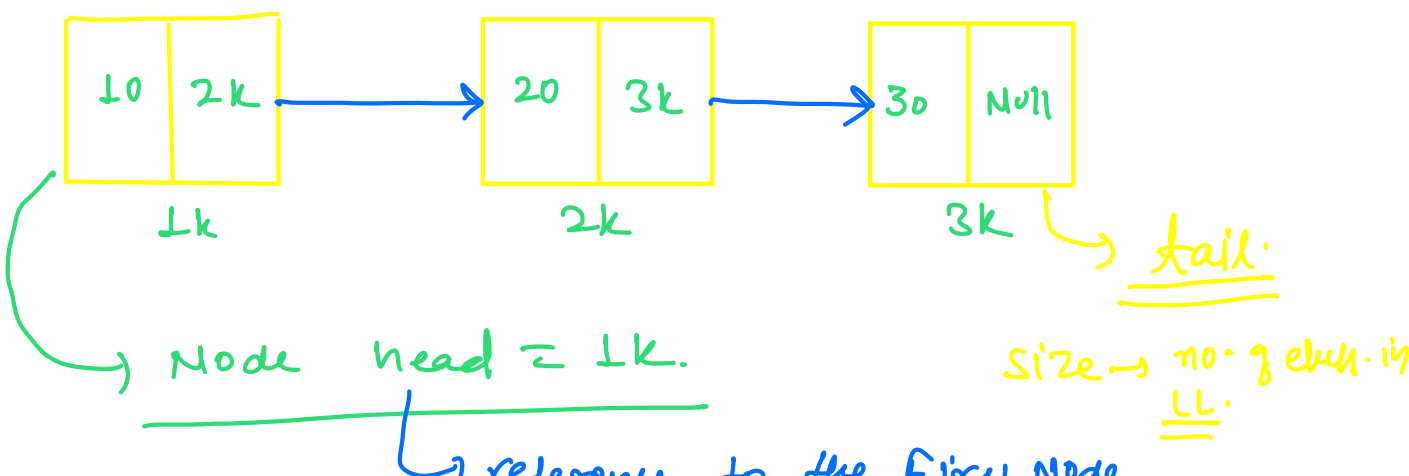



```
Node {
    int data;
    Node next;
}
```

temp.data = 10

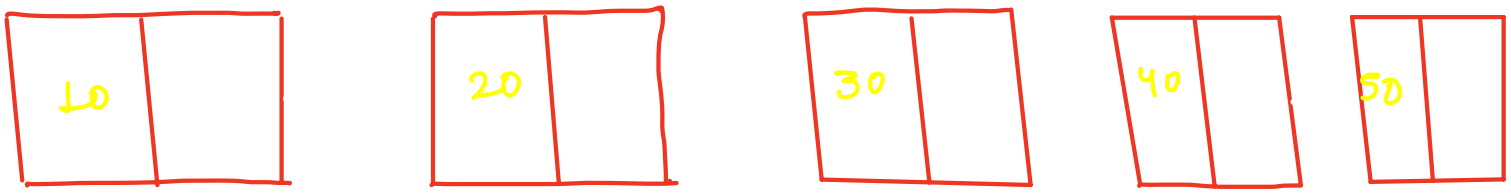
temp.next = 2k

temp = 1k



```
Public class linkedlist {

    Private class Node {
        int data;
        Node next;
    }

    Private Node head;
    Private Node tail;
    Private Node size;

}
```

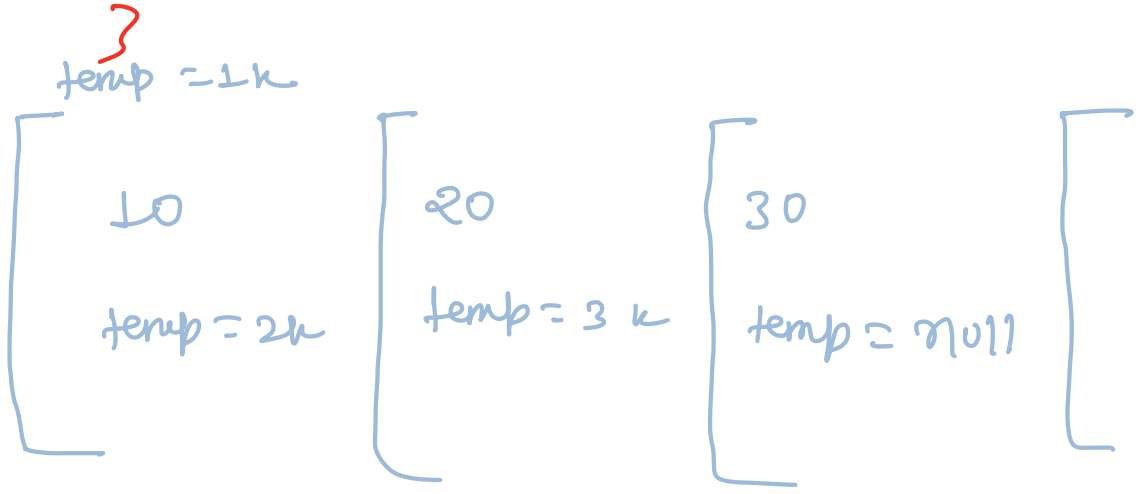
Traversing the linked list :-



[

] i++

```
Node  temp = head ;
while ( tem != null ) {

    SYSO ( temp.data ) ;
    temp = temp.next ;
}
```



10 , 20 , 30.

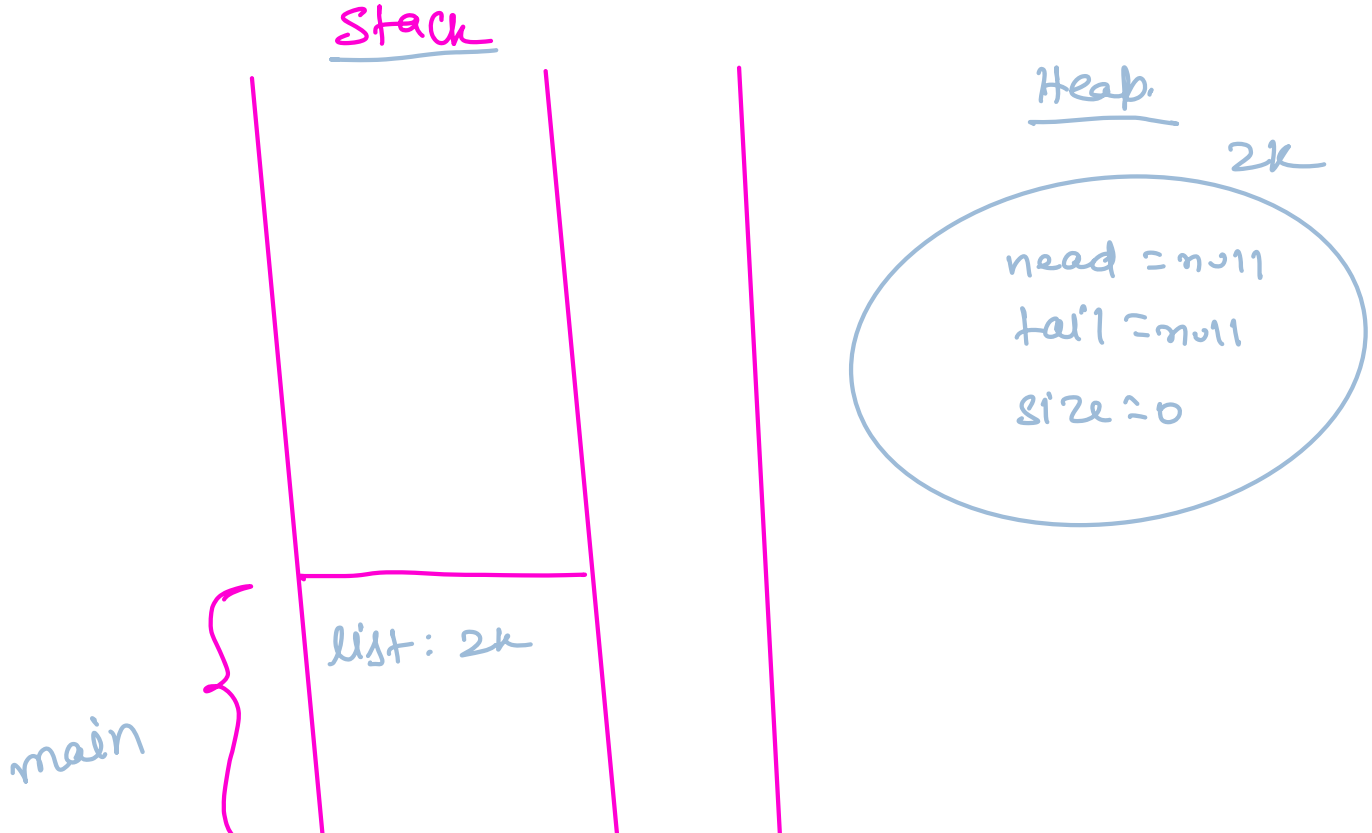
```
Public void display ( ) {

    Node  temp = this.head ;

    while ( tem != null ) {

        SYSO ( temp.data + " " ) ;
```

```
            temp = temp.next;
        }
    }
```

Addlast.

If we new any classes then the space is allocated to the data member of class.

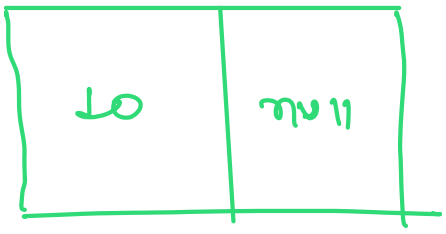
```
LinkedList list = new LinkedList();
```



Summary object

head = null
tail = null
size = 0

addlast (10)



// new Node Creation
Node nn = new Node();

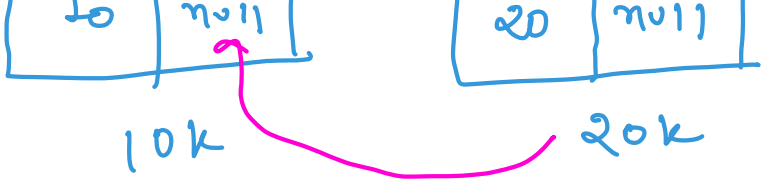
nn.data = 10;

if (size == 0) {
head = nn;
tail = nn;
size ++;
}

summary object

head = 10k
tail = 10k
size = 1

addlast (20);




```
Node nn = new Node();
```

```
tail.next = nn;
```

Summary object

```
if(size >= 1){ tail = nn;
               size++;
}
```

```
// O(1) Public void addLast(int item){
    // Create a new node
    Node nn = new Node();
    nn.data = item;
    nn.next = null;
    // attach
```

Add First

```
Public void addFirst(int item) {   // O(1)

    Node
```

```
nn.next = head;
```

head

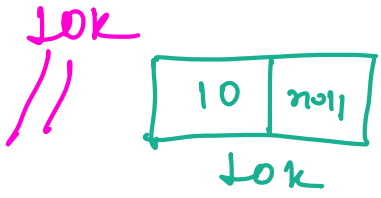
first // O(1)

addFirst(10)

head = null

tail = null

size = 0



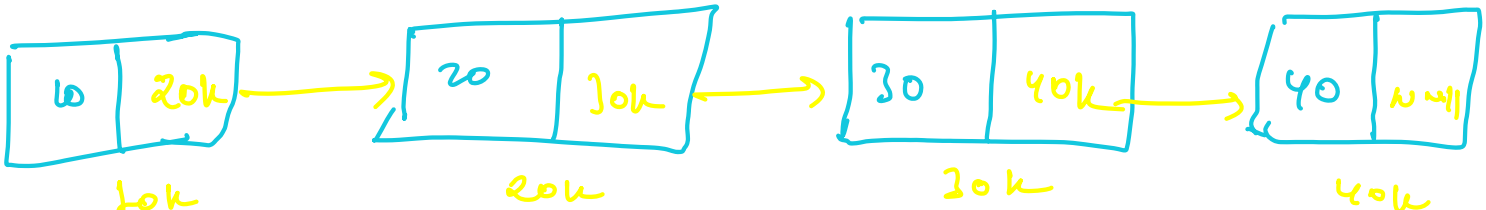
1. Create a new Node.
2. attach the node with existing Node (size>, 1)

```
{
    nn.next = head;
```

3. Summary object

```
if( size >, 1 ) {
this. head = nn;
} else {
    head = nn;
    tail = nn;
    size++;
}
```

// Get First.



head

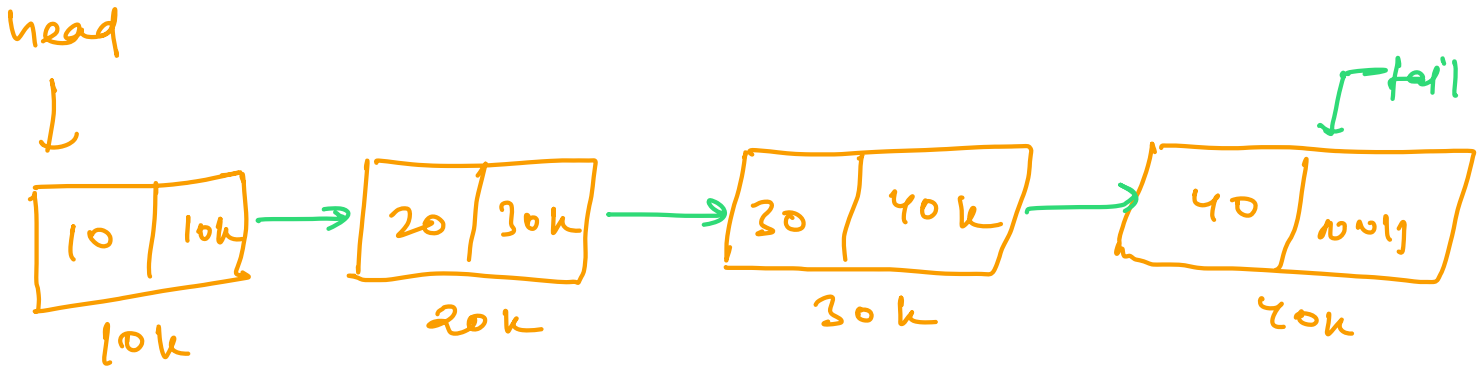
tail.

// O(1)

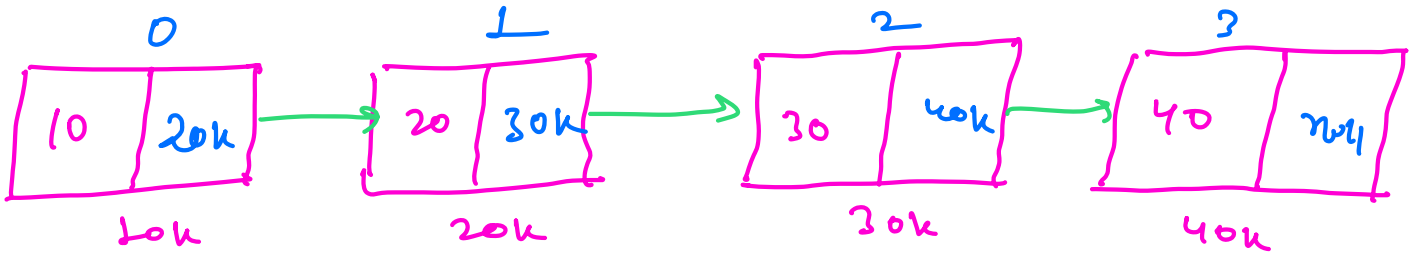
```
Public int getFirst() throws Exception {
    if (this.size == 0) {
        throw new Exception("LL is Empty");
    }
    return this.head.data;
}
```

Get Last

// O(1)


head
tail
10
10k
20
30k
30
40k
40
null
10k
20k
30k
40k

GetAt

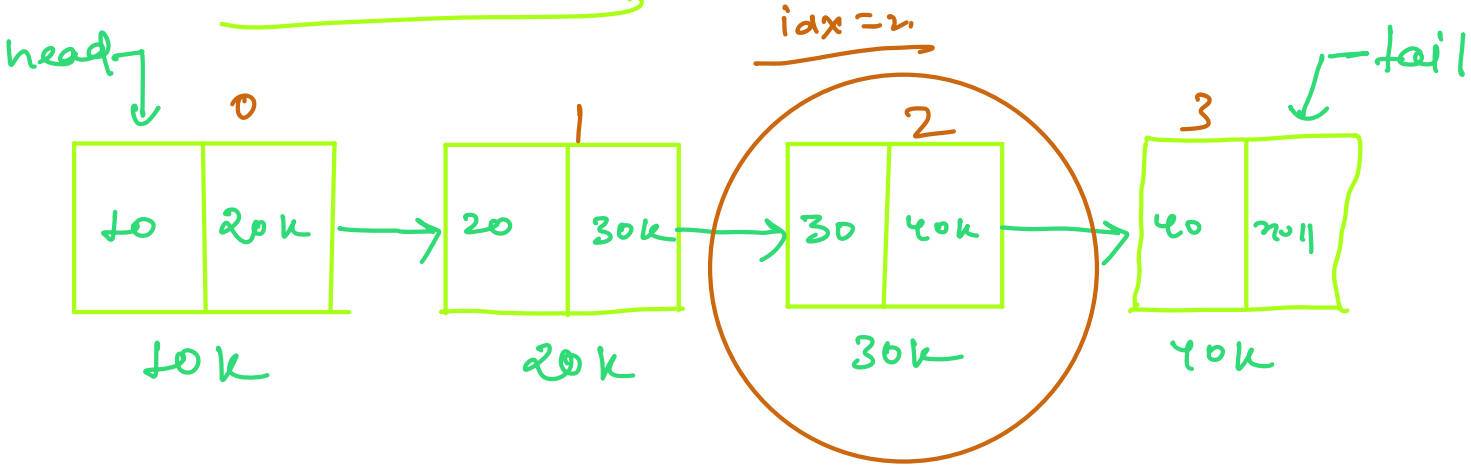

0
1
2
3
10
20k
20
30k
30
40k
40
null
10k
20k
30k
40k

// O(n)

throws Exception

```
Node temp = head;
for(int i=0; i < idx ; i++) {
    temp = temp.next;
}
return temp.data;
}
}
```

GetNodeAt.



private | public

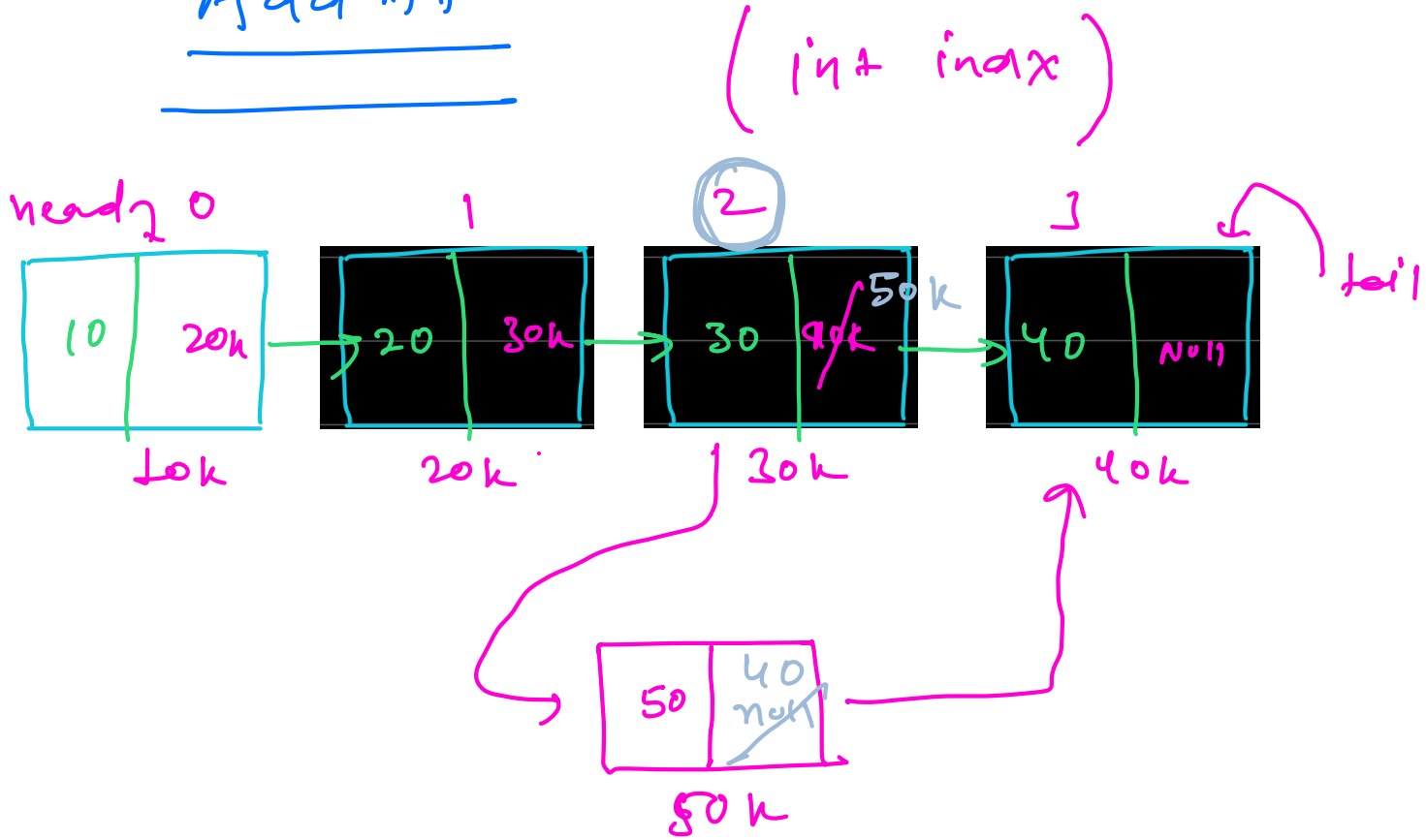
Node type return.

|| O(n)

```
Private Node getNodeAt(int idx) throws Exception {

    if ( this.size == 0) {
    throw new Exception("LL is Empty");
    }

if [ idx < 0 || idx >= this.size) {
    throw new Exception("Invalid index");
}

    Node temp = this.head;
    for( int i=1; i<=idx; i++) {

        temp = temp.next;

    }
```

```
        return temp;
    }
```

Add At.



→ n minus 1

```
nm1 = getNodeAt (idx-1);
```

```
np1 = nm1.next;   ( also getNode(idx) )
```

↳ n plus 1.

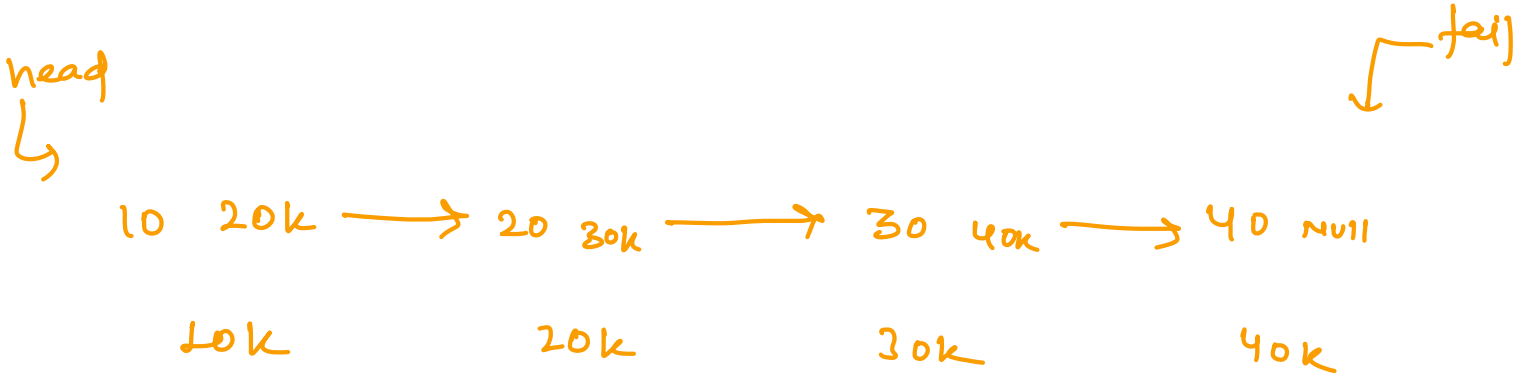
```
nm1.next = nn;
nn.next = np1;
```

//

throws Exception

```
Public void addAt (int item, int idx) {

  if ( idx < 0 || idx > size) {

    throw new Exception ("Invalid Index");

  }

  if ( idx == 0 ) {

    addFirst (item);

  } else if ( idx == this.size) {

     addLast ( item);

  } else {

    //

    Node nn = new Node ();

     nn.data = item;

      nn.next = null;


    //

                  ↳ Node minus one

    Node nm1 = getNodeAt ( idx - 1);

              ↳ node plus one

     np1 = nm1.next;

     nm1.next = nn;
```

```
nn.next = npL;

// summary object

size++;

}
```




```
int rv = head.data;        → 10

head = head.next;
head = 20k

return rv;
```

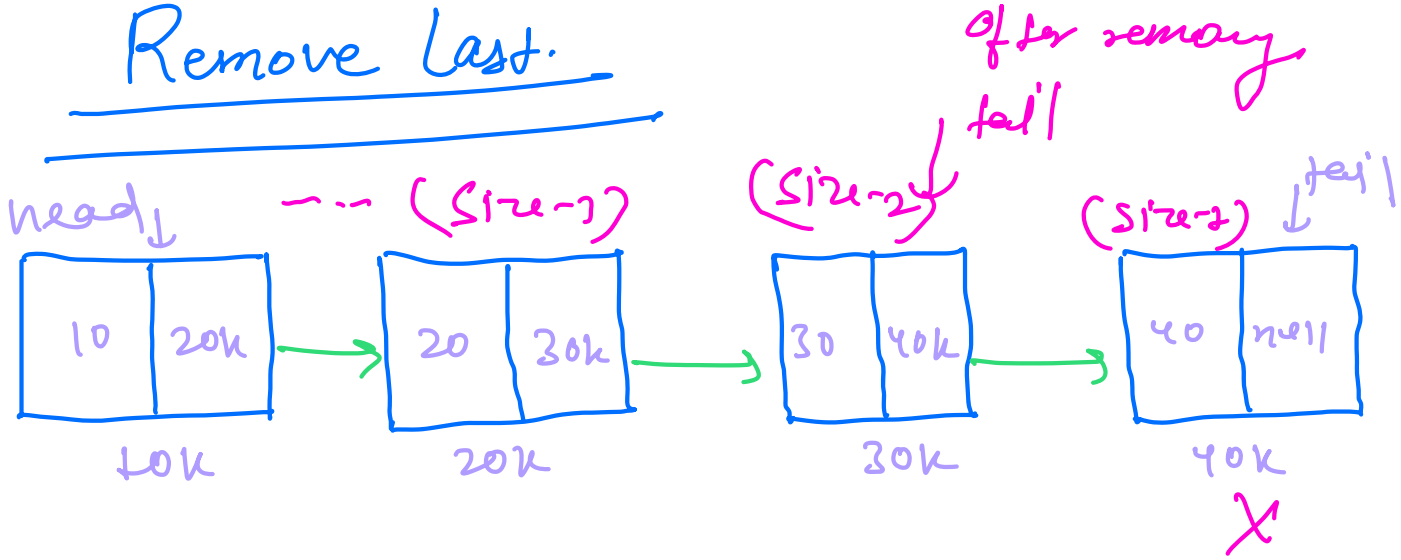
Special Case

// O(1)

```
if (size == 0) {
throw new Exception ("LL is Empty");
```

```
3
    int rv = this.head.data;

    if (this.size == 1) {

        this.head = null;
        this.tail = null;
        this.size = 0;

    } else {

        head = this.head.next;
        this.size--;
    }
    return rv;

}
```

Remove Last.



```
rv = tail.data;
```

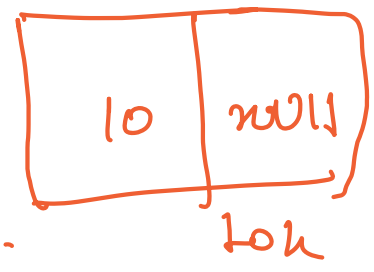
```
Node lm1 = getNodeAt(size-2);
```

↳ last minus one.

```
tail = lm1;
tail.next = null;
```

```
if (size == 1) {
```

```
head = 10k
tail = 10k
size = 0.
```

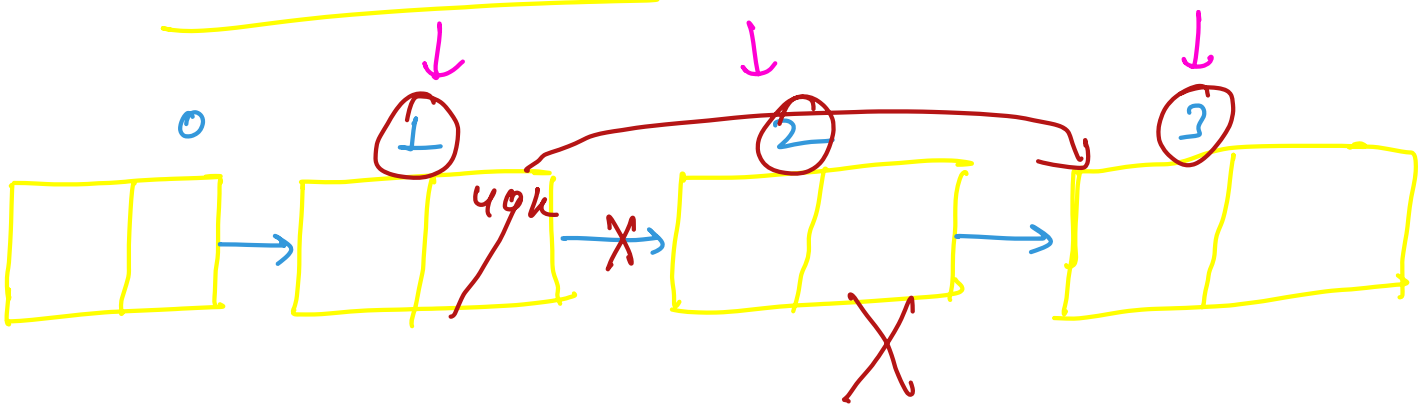



```
Public int removeLast() throws Exception {

if (this.size == 0) {

  throw new Exception ("LL is empty");

  }

  int rv = this.tail.data;
```

```
if ( this.size == 1 ) {
    this.head = null;
    this.tail = null;
    this.size = 0;
} else {
    Node sizeM2 = getNodeAt(size-2);
    this.tail = sizeM2;
    this.tail.next = null;
    this.size--;
}
return rv;
}
```

size minus 2

Remove At



int rv =

nm1.next = np1

size--;

// O(n)

Public int removeAt(int indx) throws exception {

if (this.size == 0) {

throw new Exception("LL is Empty");

}

if(idx < 0 || idx >= this.size) {

throws Exception (" Invalid index ");

}

if(idx == 0){

return removeFirst();

} else if (idx == this.size -1) {

return removeLast();

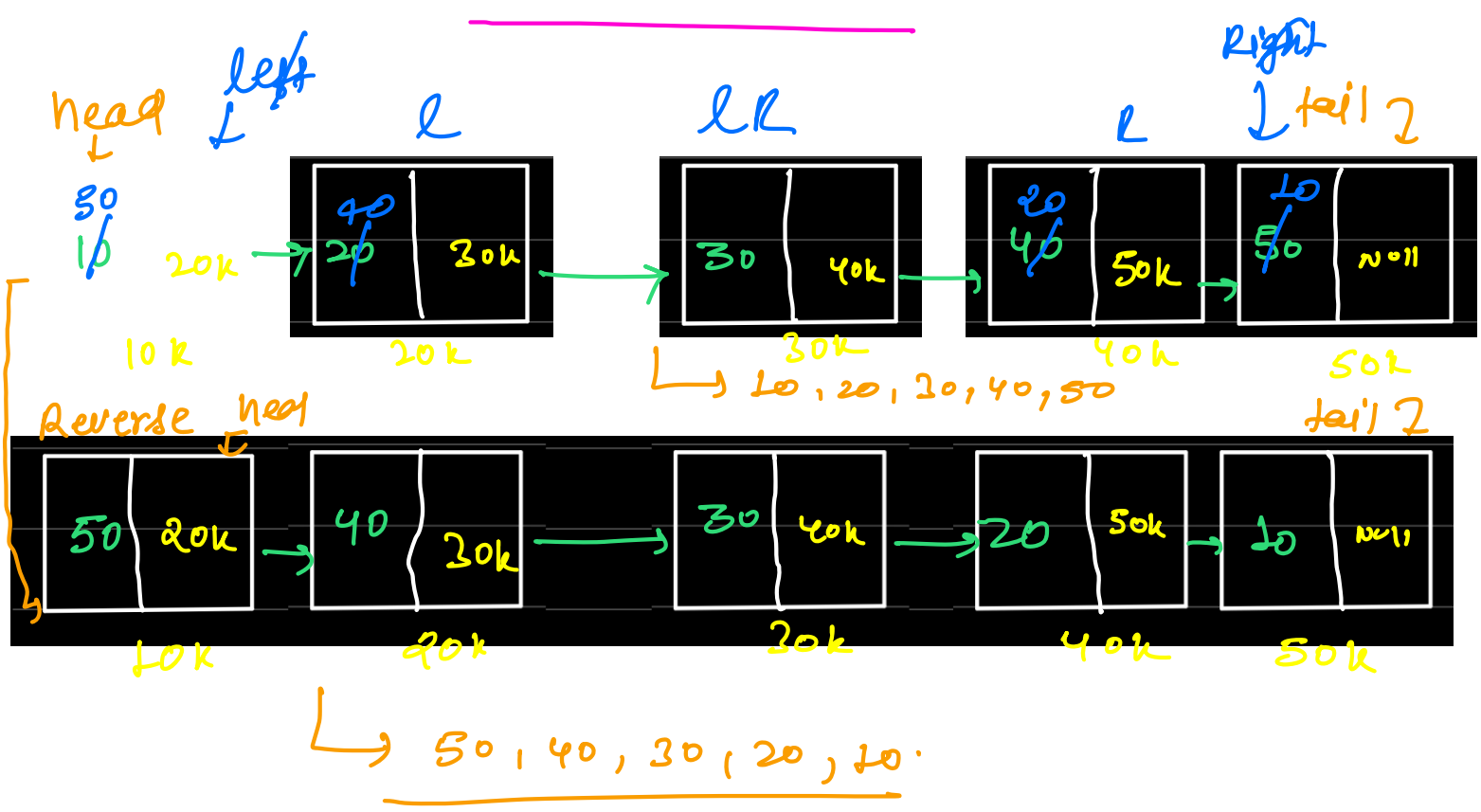
} else{

Node nm1 = getNodeAt (idx-1);

```
        Node n = nm1.next;
        Node np1 = n.next;

        nm1.next = np1;
        this.size--;

        return n.data;
      }
    }
  }
```

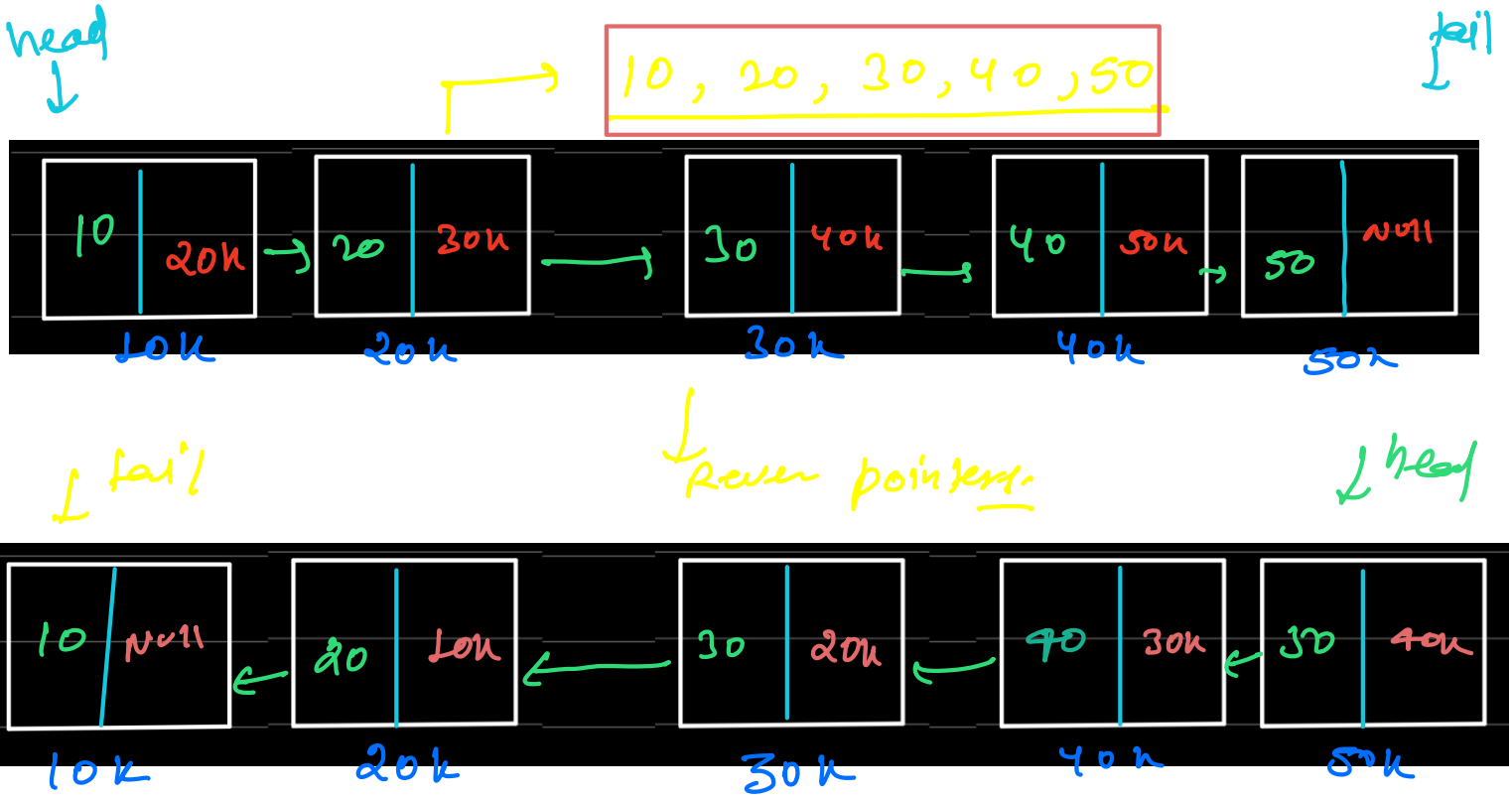
Reversing Linkedlist.

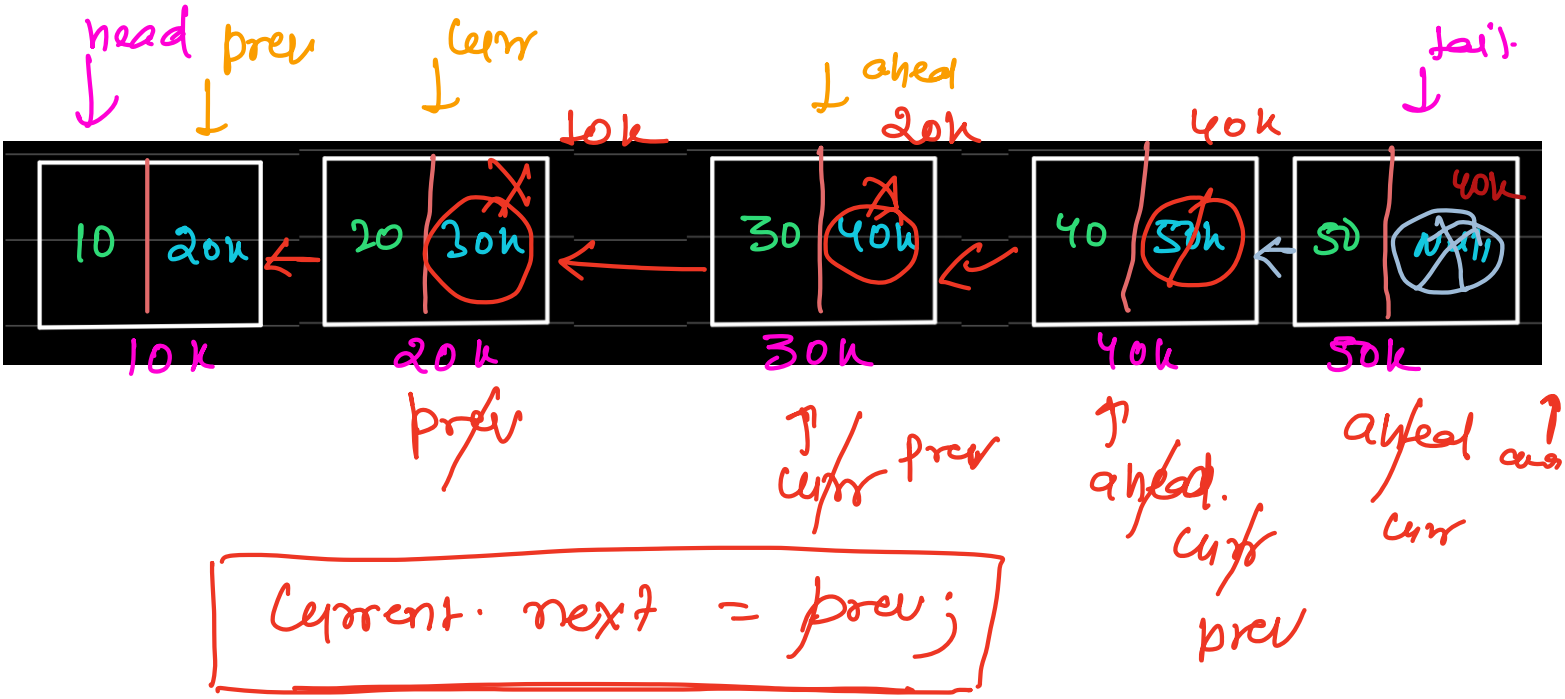
Reverse Data:




```
Public void reverseData ( ) throws Exception {

    int left = 0;
    int right = this.size - 1;

    while (left < right) {

        Node ln = getNodeAt ( left );

        Node rn = getNodeAt (Right);

        int temp = ln.data;
```

```
ln.data = rn.data;
rn.data = temp;

left++;
right--;

}

}
```



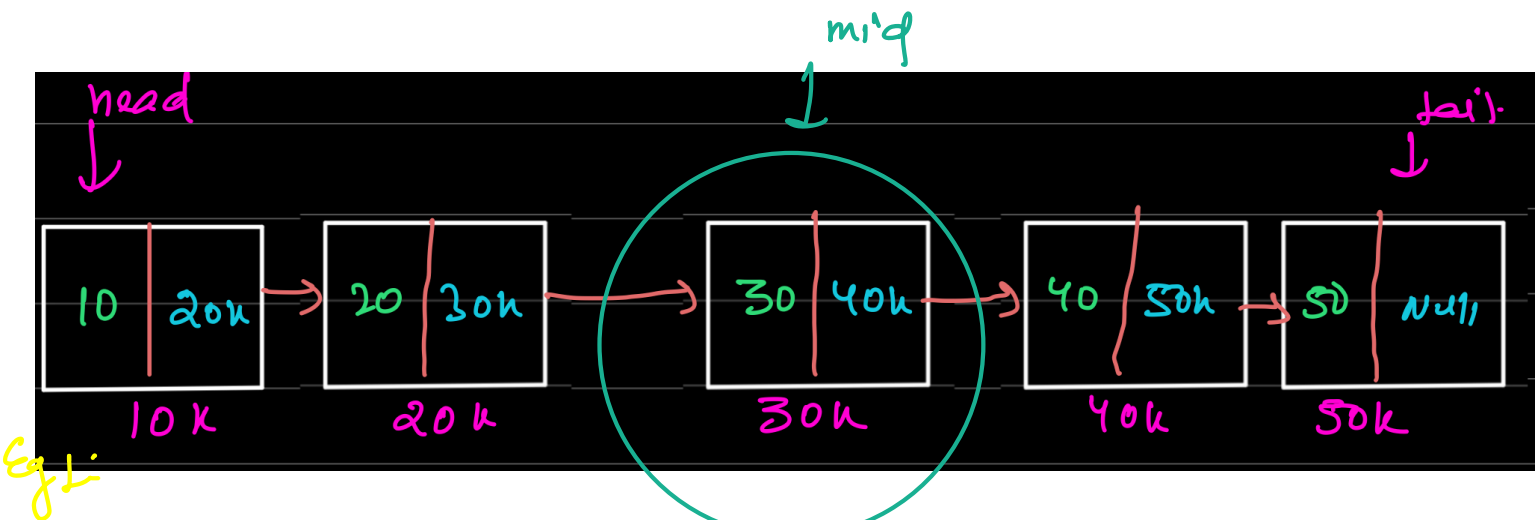


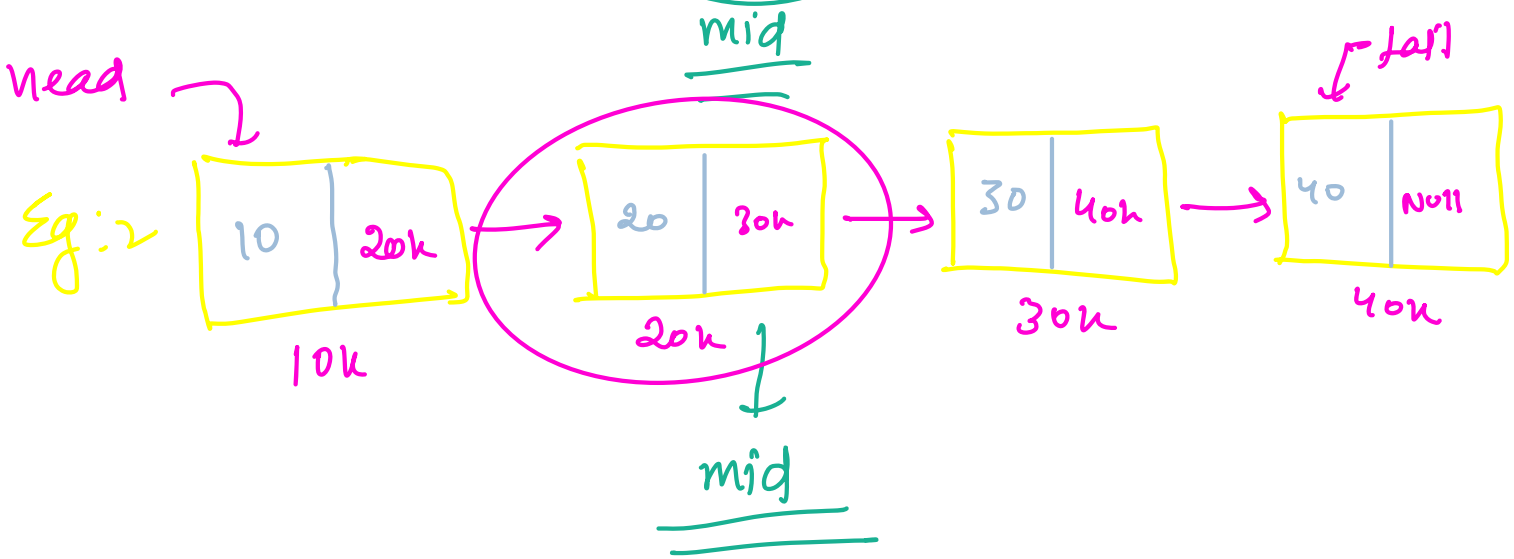

```
Current.next = prev;
```

```
Public void reversePointers ( ) {

    Node prev = this.head;
    Node curr = prev.next;

    While ( curr != null ) {

        Node ahead = curr.next;
        curr.next = prev;
        prev = curr;
        curr = ahead;

    }

    // swap
```

```
Node t = this.head;
this.tail = this.head;
this.head = t;

this.tail.next = null;
}
```

Mid point of linked list :-





(1) Just simply call the getAt function

this function returns me the mid value of linkedlist.

(ii) Without using size we have to find mid node data:

odd length → (fast.next = now)

we gonna stop.

even length: (fast.next.next == null)

stop here

```
Public int mid () {

Node slow = this.head;
Node fast = this.head;

while ( fast.next != null && fast.next.next != null){
```

```
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow.data;
}
```