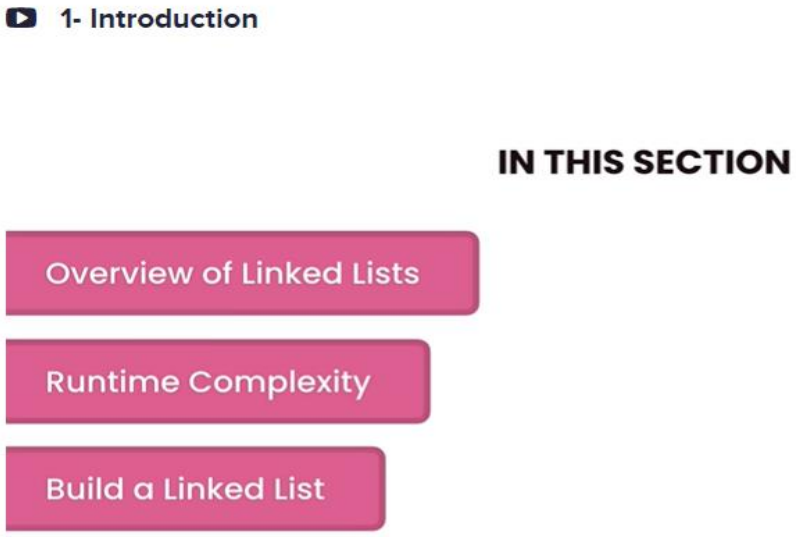
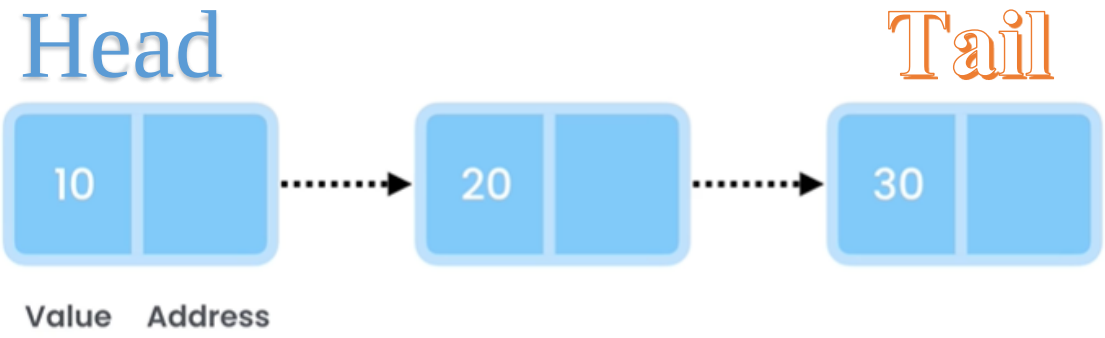
Linked List

V-1) Introduction:-



V-2) What Are Linked List?



Here Data's 10, 20, 30 are linked with one next to each. Where One Node has value and address of next node that's goanna linked the two data's. That's why this structure of dataset is called Linked List. Above List is actually singly Linked List. There are total 8 types of linked lists.

Time Complexities:-

Scenario 1:-

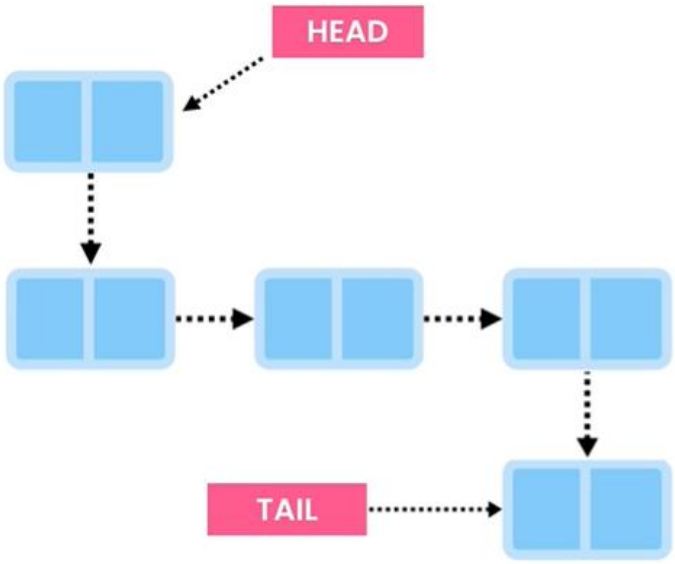
LOOKUP

By Value O(n)

By Index O (n)

For scenario 1 this linked list cannot direct access like indices of Array. Each node has a reference to a node. So at the worst case scenario for both value and index search might be at the end of the list. So , that's why both cases has a complexity of O (n).

2. Insertion



INSERT

At the End	O(1)
At the Beginning	O(1)
In the Middle	O(n)

3. Deletion

DELETE

From the Beginning	O(1)
From the End	O(n)
From the Middle	O(n)

V- 3) Working with Linked_List

```
import java.util.Arrays;
import java.util.LinkedList;

public class Main {
    public static void main(String[] args) {
        // LinkedList <int> = new LinkedList<int>; //this will give error cause it is primitive we have to write the whole class
        LinkedList list = new LinkedList(); // kisu nah khali create holo list
        list.addLast(10);// [ 10 ]
        list.add(20); // [ 10, 20 ]
        list.addLast(10); // [ 10 , 20 , 10 ]
        list.add(30);// [ 10 , 20 , 10 , 30 ]
        list.addFirst(50); // [ 50 , 10 , 20 , 10 , 30 ]
        System.out.println(list);
        System.out.println(list.size()); // return size of the array means 5
        System.out.println(list.indexOf(10));// find the first occurance of the given element which is 1
        System.out.println(list.contains(10)); // return true if found the item in the list , false otherwise
        var array = list.toArray();// eikhane variable er jaigai onno kisu dile kaj korbe nah
        System.out.println(Arrays.toString(array)); // array ta list na niye print korlam
    }

}
```

V-(4-12)-working with LinkedList Descriptively:-

Global Variable in the LinkedList Class:-

```
private Node first;// reference of  head
private Node last;//reference of tail
private int size;// reference of the list items exist in the list
```

Node class which is inside LinkedList class:-

```
private class Node {
   private int value;//Node er value store korbe
   private Node next;//Node er next element er reference rakbe
    private Node(int value){
       this.value = value;
    }
}
```

V-4. Add last Method (add new items at the end of the list)

```
public void addLast(int item){
    // Node node = new Node(); eivabe likhe confused hoar dorkar nai
    // kenona java compiler dan side dekhei buje nibe node kon type er object
    //jodi var use kori
    /*
    var node = new Node();
    node.value =  item;
    */
   var node = new Node(item);
   if(isEmpty())// jodi list empty thake tahole List er first & last node eki
       first=last=node;
   else{
       last.next = node;//existing last node er pore ei notun node boslo
       last = node;//last variable jeta node er tail reference kore seta update korlam
   }
   size++;
}
```

main class:-

```
public class Main2 {
    public static void main(String[] args) {
    var list = new LinkedList();
    list.addLast(10);
    list.addLast(20);
    list.addLast(30);
    }
}
```

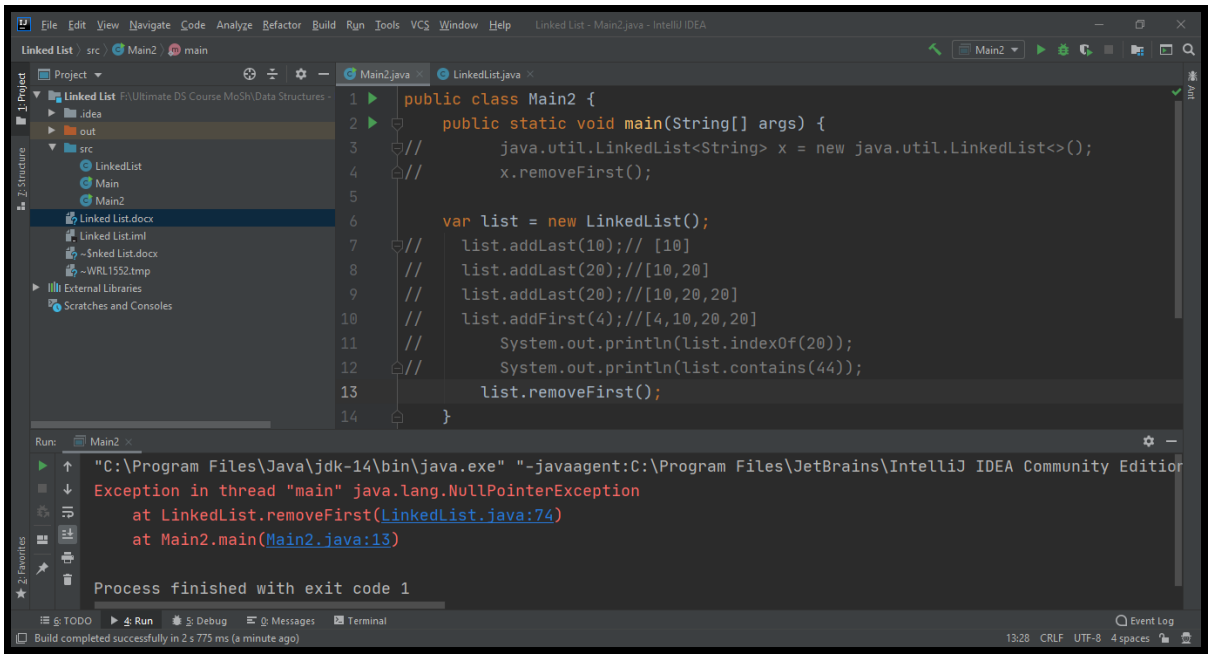
V-5-(addFirst method with listempty checking):-

```
public void addFirst(int item){
   var node = new Node(item);
   if(isEmpty())
       first=last=node; //jodi list empty thake
   else {
       node.next = first;// first e notun node er sathe first ke link
       first = node;// then update the first cause head of the list changes
   }
   size++;
}
```

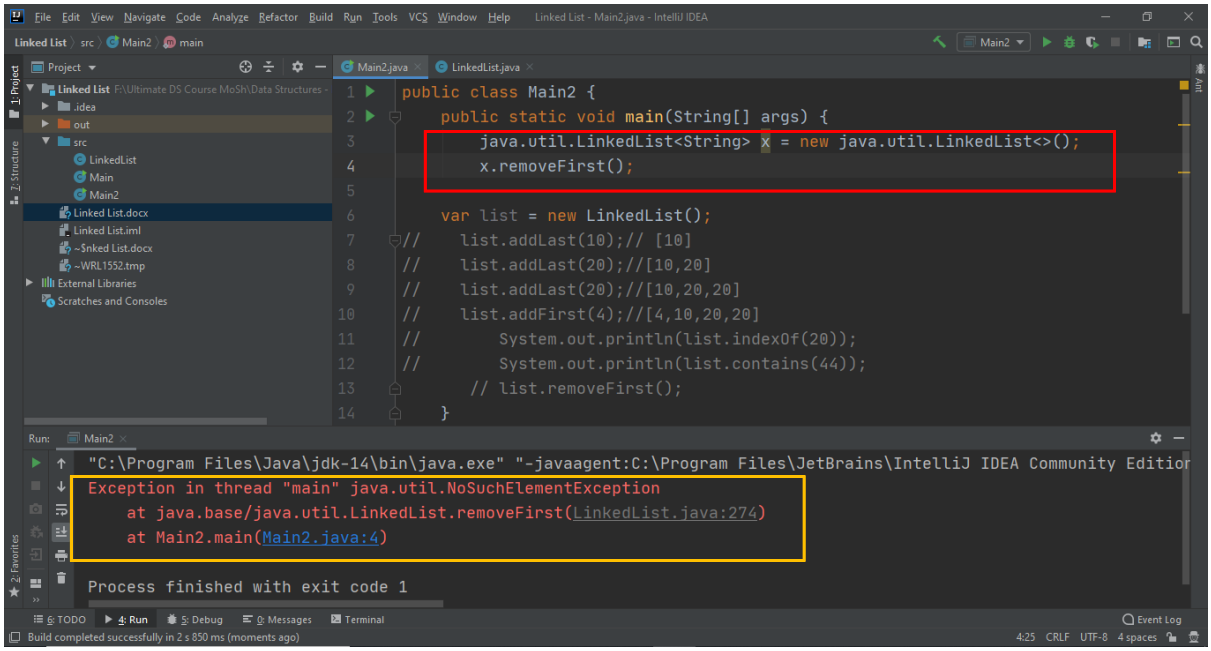
V-6 –(indexof method)

```
public boolean contains(int item){// ei method e amra indexof method ke reuse korlam
    return indexOf(item)!=-1;// eikhane item khuje paile true return korbe otherwise false return korbe
// indexOf method integer return kore
// se jodi uporer contains method er call e -1 return na kore mane list e
// oi item ta ase taile contains method true return korbe
// ar -1 return korle mane item ta list ei nai tai contains method
// false return korbe
}
```

V-7-(RemoveFirst Method[little complicated])



As the list is empty so we got null pointer exception.



In java linkedlist built in class we got NoSuchElementException instead of Nullpointer exception. So, we need to implement our code in the way its built in java packages.

[Built in java linkedlist removeFirst() call:-

```
java.util.LinkedList<String> x = new java.util.LinkedList<>();
x.removeFirst();
```

]

So, our final modified method for removefirst is:-

```
public void removeFirst(){
    if(isEmpty()) throw new NoSuchElementException();
    if(first==last)  //list e jodi ekta element thake
        first = last = null;
    else{
      /*
    ekon amar onek sabdhane korte hobe remove
    ex [10->20->30]
     ami chailei 20 take first banai diye first.next null
    korte parbo nah ete pura list harai jabe
     */
        var second = first.next;//ex [10->20->30] eikhane 20 er reference save raklam
        first = null;// 10 er sathe 20 er bonding remove korlam
        first = second;// ekon first ke 20 e point korlam ekon list hobe [20->30]
    }
    size--;
}
```

V-8-(RemoveLast() method)-[complex]

```
public void removeLast(){
   // list jodi empty hoi tahole remove korar kisui thakbe nah sekehetre exception dekhabo
   if(isEmpty())
      throw new NoSuchElementException();
   //list e only ekta element thakle jevabe remove korbo
   if(first==last)
      first=last=null;
   else {
      // [10,20,30]
      // the logic below written is only working for list with atleast two node
      var previous = getPrevious(last); // previous = 20
      last = previous; // last ekon 30 theke 20 ke point korlo
      last.next = null;// 20 er sathe 30 er bondhon chinno kore dilam (garbage collector)
   }
   size--;
}
```

V-9-(Implementing Size)

```
public int size(){
   return size;
}
```

V-9-(Cheat Sheet Gone Wrong If we memorize specially for Job Interview)

Data Structures

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Indexing	Search	Insertion	Deletion	Indexing	Search	Insertion	Deletion	
Basic Array	O(1)	O(n)	-	-	O(1)	O(n)	-	-	O(n)
Dynamic Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	-	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)

hackerearth.com

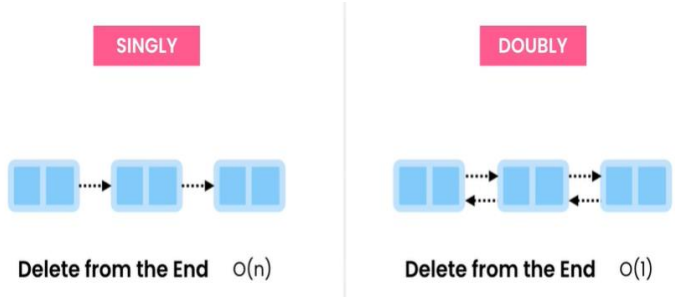
V-10-(Array & LinkedList Comparison)

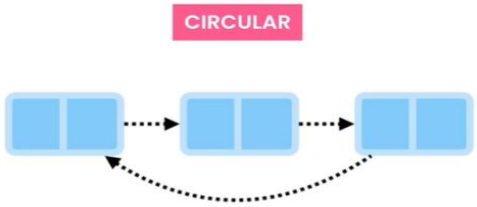
SPACE

- Static arrays have a fixed size
- Dynamic arrays grow by 50-100%
- Linked lists don't waste memory
- Use arrays if you know the number of items to store
- new ArrayList(100)

		ARRAYS	LINKED LISTS (Singly List Only)
Lookup			
	By Index	O(1)	O(n)
	By Value	O(n)	O(n)
Insert			
For Shifting elements	Beginning/End	O(n)	O(1)
	Middle	O(n)	O(n) → For traversing the whole list
Delete			
For Shifting elements	Beginning	O(n)	O(1)
	Middle	O(n)	O(n) → For traversing the whole list
	End	O(n)	O(n) → For traversing the whole list

V-15-(Types of Linked List)

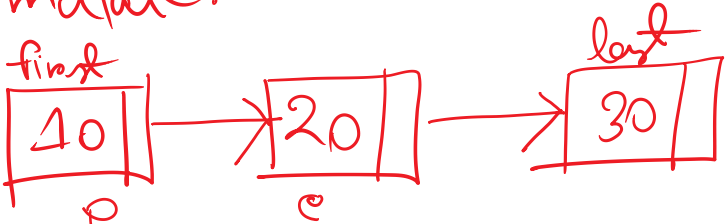




Circular Linked List needed when we build something called music player . Cause at the end of the playlist it needs to start from the beginning .

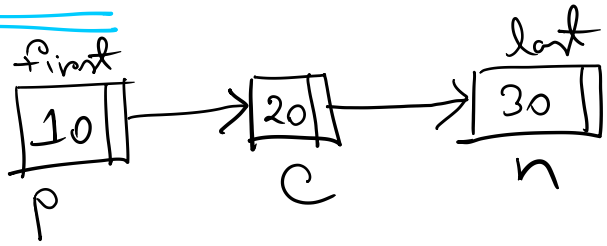
Ex-1 Reversing a singly linked list. (toughest problem for beginners) even senior developers make it wrong sometimes.

lets simulate:-



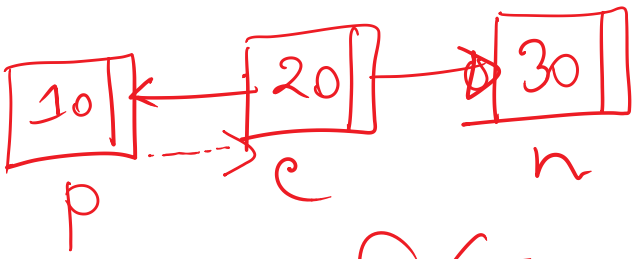
যদি দুটো reference দিয়ে করতে চাই তাহলে তা কোনভাবে সম্ভব না। কেননা এতে তৃতীয় লিংকটি হারিয়ে যাবে।

সঠিক নিয়ম:-



এতে করে singly linked list এর কোন লিংক বা element হারাবে না।

// c.next = p করলে চুঁকবে



অর্থাৎ লিংক পরিবর্তন হলো।

কিন্তু এভাবে remove হবে না প্রথম নিয়ে।

তাই first = first ;

তারপর first.next = null;

p, c, n তিনটি node নিয়ে traversal করবে iterate করে

লিংক change করলেই

সম্ভব হবে সব।

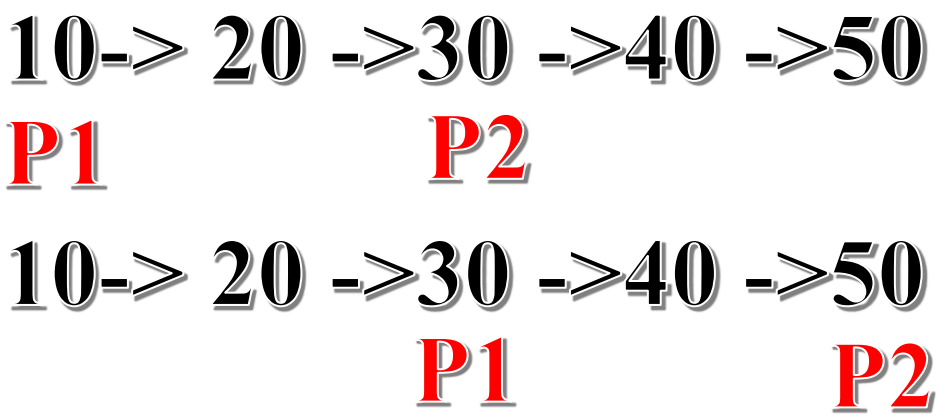
```java
public void reverseList(){
    //List ta inplace reverse korte hobe
    //[10->20->30]
    //hote hobe [30->20->10]
    //onek somossa
    // 10-> 20-> 30
    // p     c    n
    // n = c.next diye reference rakbo
    // link change korbo ekon c.next = p;
    if(isEmpty()) return;
    var previous = first;// p
    var current = first.next;// c
    while (current!=null){
        var next = current.next;// n
        current.next = previous;// link er direction change
        previous = current;
        current = next;
    }
    last = first;
    last.next = null;
    first = previous;
}
```

EX-2) Find kth node from the end. –[Popular interview question about SLL]

If List Has 10-> 20->30->40->50

And if we pass the value 3 for k it should return 30. It should be solved through 1 call . That means we can't iterate through the whole list.

Ans:- We can solve this problem using 2 pointers.



The idea is that when the 2nd pointer reached the end of the list it should find the expected value as in the 1st pointer. So, pointer should be selected properly and also handle the exceptions.

Lets solve this:- [very much complicated]

```
public int getKthfromtheEnd(int k){
    if(isEmpty()) throw new IllegalStateException();
    //setting two pointers
    // 10 -> 20 -> 30 -> 40-> 50
    // a
    // b
    // if k = 3
    var a  = first;
    var b = first;
    //for loop simulation
    // 10 -> 20 -> 30 -> 40-> 50
    // a
    //           b
    // 10 -> 20 -> 30 -> 40-> 50
    // a
    //                 b
    for(int i=0;i<k-1;i++){// this loop will set the distance between two pointers
        b= b.next;
        if(b==null) throw new IllegalArgumentException();
    }
    // while loop simulation
    // 10 -> 20 -> 30 -> 40-> 50
    //           a
    //                        b
    // 10 -> 20 -> 30 -> 40-> 50
    //                 a
    //                                b
    while (b!=last){
        a=a.next;
        b=b.next;
    }
    return a.value;
}
```

LINKED LISTS

- Second most used data structures
- Grow and shrink automatically
- Take a bit more memory

RUNTIME COMPLEXITIES

Lookup

By Index	O(n)
By Value	O(n)

Insert

Beginning/End	O(1)
Middle	O(n)

Delete

Beginning	O(1)
Middle	O(n)
End	O(n) / O(1)

Ex-3-exclusive) **Find the middle of a linked list in one pass.-**

```
public void printMiddle() {
    if (isEmpty())
        throw new IllegalStateException();
    var a = first;
    var b = first;
    while (b != last && b.next != last) {
        b = b.next.next;
        a = a.next;
    }
    if (b == last)
        System.out.println(a.value);
    else
        System.out.println(a.value + ", " + a.next.value);
}
```

Ex-4) if There is any loop or not in a SLL?

```
public boolean hasLoop() {
    var slow = first;
    var fast = first;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
        if (slow == fast)
            return true;
    }
    return false;
}
```

Exercise 3 and 4 explanation is given below:-

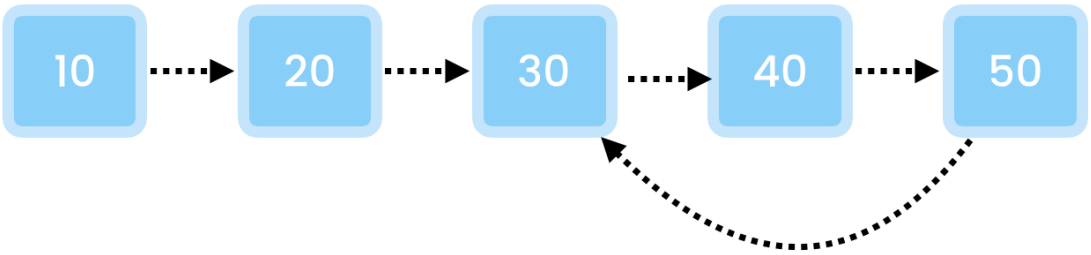
Linked Lists

Exercises

1- Find the middle of a linked list in one pass. If the list has an even number of nodes, there would be two middle nodes. (Note: Assume that you don't know the size of the list ahead of time.)

Solution: LinkedList.printMiddle() - You can read my analysis of the solution in the next page.

2- Check to see if a linked list has a loop.



Hint: use two pointers (slow and fast) to traverse the list. Move the slow pointer one step forward and the fast pointer two steps forward. If there's a loop, at some point, the fast pointer will meet the slow pointer and overtake it. Draw this on a paper and see it for yourself. This algorithm is called *Floyd's Cycle-finding Algorithm*.

Solution: LinkedList.hasLoop() - You can read my analysis later in this document.

Solutions

1- Find the middle of a linked list in one pass.

As always, we start by simplifying and narrowing the problem. Let's imagine the list has an odd number of nodes.

Since we have to find the middle node in one pass, we need two pointers. The tricky part here is that we should figure out how many nodes should the first and second pointers be apart. Let's throw a few numbers to find the relationship between these pointers.

Number of Nodes	Middle Node
1	1
3	2
5	3
7	4
9	5
11	6

Do you see a pattern here? In every row, the number of nodes is increasing by two where as the position of the middle node is increasing by one. Agreed?

So, we can define two pointers that reference the first node initially. Then, we use a loop to move these pointers forward. In every iteration, we move the first pointer one step and the second pointer two steps forward. The moment the second pointer hits the tail node, the first node is pointing the middle node.

Now, let's expand our problem. What if the list has an even number of nodes?

Number of Nodes	Middle Node
2	1,2
4	2,3
6	3,4
8	4,5
10	5,6

We see the same pattern. In every step, the number of nodes increases by two whereas the position of the middle node is increasing by one. The only difference is that here we need to return two middle nodes. That's easy. Once we find the left middle node, we'll also return the node next to it.

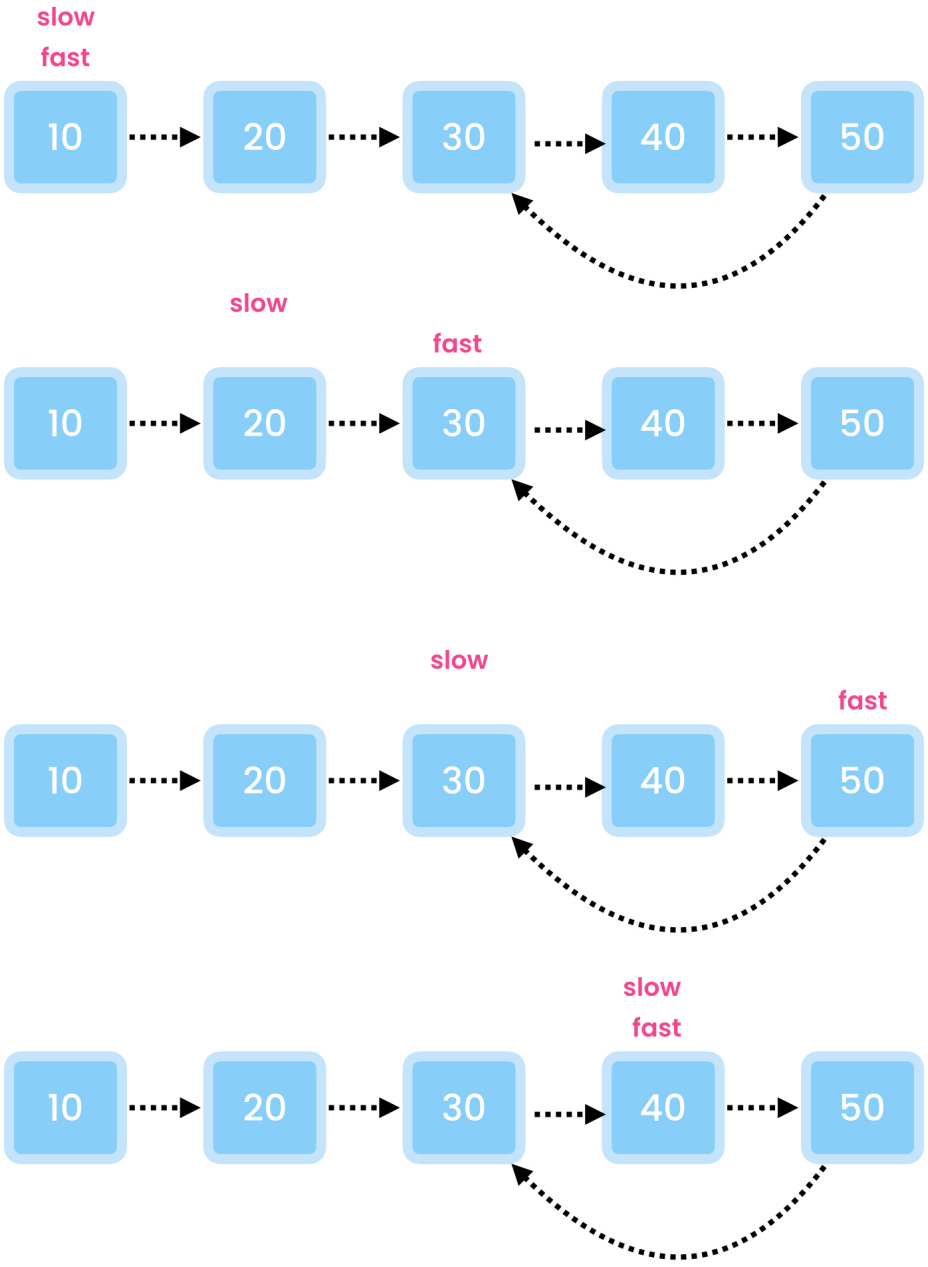
How do we know if the list has an even or odd number of items? We can declare a count variable and increment it in each step. But we don't really need this. If the list has an even number of nodes, at the end of the last iteration, the second pointer will reference the tail node; otherwise, it'll be null. (Try a few examples and you'll see this yourself.)

Here's how we can implement this algorithm:

```
1:    var a = first;
2:    var b = first;
3:    while (b != last && b.next != last) {
4:       b = b.next.next;
5:       a = a.next;
6:   }
7:
8:   if (b == last)
9:      System.out.println(a.value);
10:   else
11:      System.out.println(a.value + ", " + a.next.value);
```

This algorithm works for any lists with at least one element. For an empty list, we need to throw an **IllegalStateException**. You can see the complete solution in **LinkedList.printMiddle().**

2- Check to see if a linked list has a loop.



You can find the implementation of this algorithm in **LinkedList.hasLoop().** To test this:

```
var list = LinkedList.createWithLoop();
System.out.println(list.hasLoop());
```

Academic Notes CSE 220

Linked List (Singly)

1/ public class Node {

Object element; // value to store

Node next; // link to next node

public Node (Object e, Node n) {

element = e;

next = n;

}

2/ creating a list :- (Singly)

(type-1)

Node head = null;

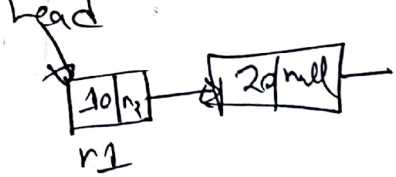
Node n1 = new Node("10", null);

Node n2 = new Node("20", null);

n1.next = n2;

~~n1 = head~~

head = n1;



type-3 (Singly list creation) [লজিক, লিংক create করার আগে দেখতে হবে node create হয়েছে কি-না?]

Node head = null;

// গিফ্ট কিনে যেভাবে last node থেকে create করতে পারব।

Node n4 = new Node("40", null);

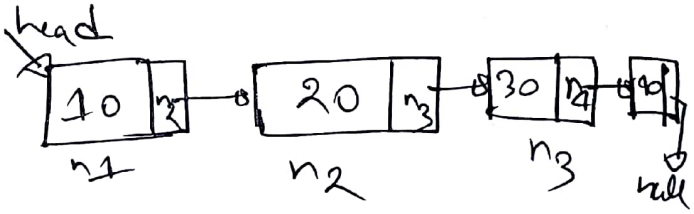
Node n3 = new Node("30", n4);

Node n2 = new Node("20", n3);

Node n1 = new Node("10", n2);

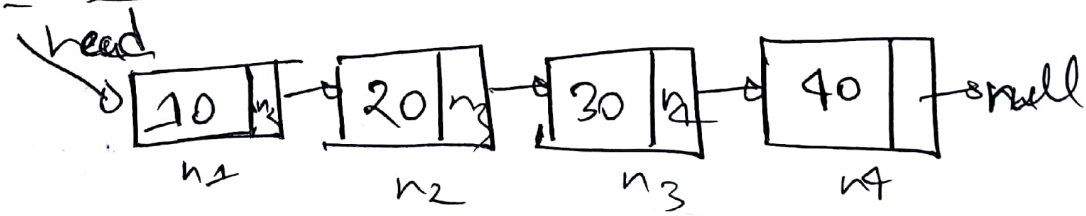
// Assigning the head reference

head = n1;



Manipulation :-

a) List iteration :- (Singly)



for(Node n = head; n != null; n = n.next) {

// do whatever we want to do

}

৪/ countNode :- (Singly list)

```
int count=0;
for(Node n=head; n!=null; n=n.next){
    count++;
}
return count;
```

৫/ Getting a specific Node :- (Singly list)

— # কোন specific index এর node এর value return করবো।

```
public Object get(Node head, int index){
// যেহেতু random access list এ নেই। তাই আমাদেরকে
// iterator use করে করে যেতে হবে। [// index valid ধরে নিচ্ছি]
    int c=0;
    for(Node n=head; n!=null; n=n.next){
        if(c==index){ // index এর সাথে মিল হলে সেই node এর value return করবো //
            return n.element;
        }
        c++;
    }
    return -1; // কোন অসঠিক index দিলে -1 return করে দিবো।
}
```

৬// NodeAt(); // কোন একটি Node কোথায় আছে তা খুঁজে বের করা।

// random access use a iteration.

public Node NodeAt(Node head, int size, int index){

//index এর validition check করব।

if(index<0 || index>=size){

System.out.println("invalid index");

return null;

}

int c=0;

for(Node n=head; n!=null; n=n.next){

if(c==index){

return n; //এভাবে iterate করে খুঁজে বের করা

c++;

}

return null; // যদি index খুঁজে null return করবে।

}

⑥

7// set function(): (singly)

কোন একটি specific node এ user থেকে নেয়া element insert করব।

```
public void setNewElement(Node head, int index, Object elem){
  //validation of index
  if (index<0 || index >= size){
      System.out.println("Invalid index");
  }
  int c=0;
  for( Node n=head; n!=null; n=n.next){
        if (c==index){
            n.element= elem;   // element টা
            break;             // insert করে দিলাম
        }
        c++;
  }
}
```

b// search function (Singly) কোন একটা element খুঁজে বের করা list থেকে।

// কত নম্বর index এ element টা আছে সেটা বের করব

```
public int indexOf(Node head, Object elem){

    int c = 0;

for(Node n=head; n!=null; n=n.next){
        if (n.element.equals(elem){
            return c; // যেই node এই node এ element
                      // আছে তা return করবে
        }
        c++;
    }

    return -1;
}
```

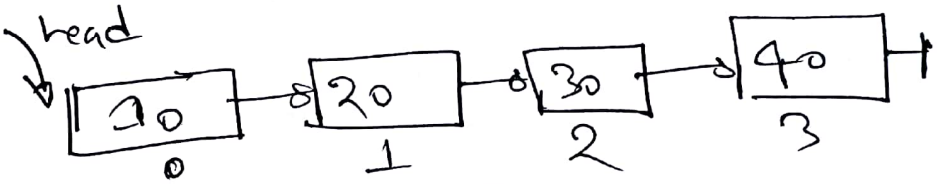
// কোন একটা node List এ আছে কি-না দেখব।

```
public boolean contains(Node head, Object elem){
    boolean contain = false;
    for(Node n = head; n!=null; n=n.next){
        if (n.element.equals(elem)){
            contain = true;
            break;          // খুঁজে গেলে
        }
    }
    return contain; // true বা false পরিস্থিতি
                    // অনুযায়ী return করবে
}
```

09// insertion in Linked List :- (Singly)

(1) List এর প্রথমে element insert করা।

(2) List এর middle/end এ " " "

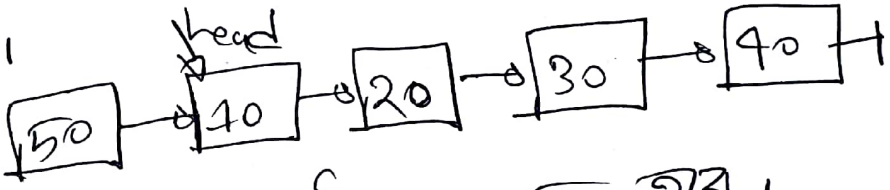


(1) insert
(2) Print
simulation

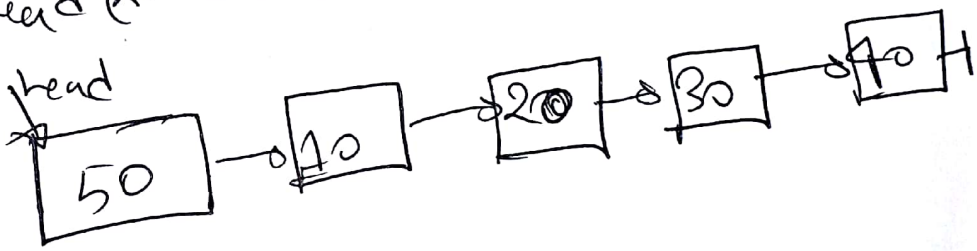
insert 50 at 0th index

(I) create node 50

(II) এই নতুন Node এর next এ head কে দিব।



(III) head কে আগাতে করতে হবে।



code logic :-

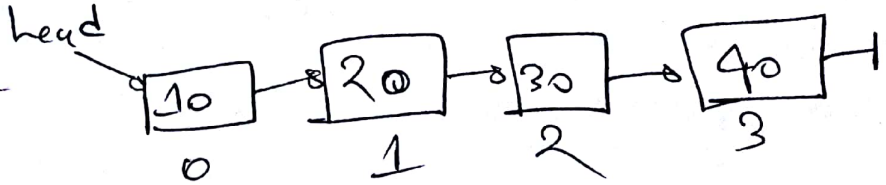
~~Node n = new Node();~~

P.T.O.

2.

insert node in the middle or end of the list

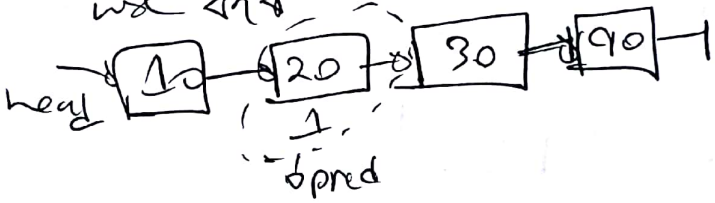
Simulation:



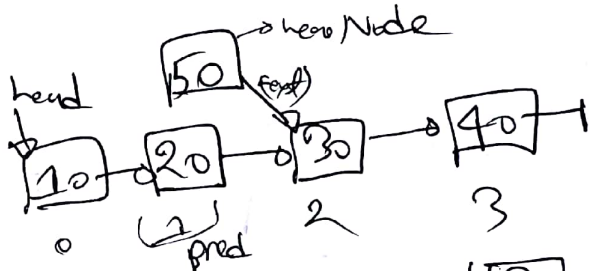
insert 50 @ 2 :-

(I) create node 50

(II) আগের Node গুলো যেয়ে একটা nodeAt নাম করব

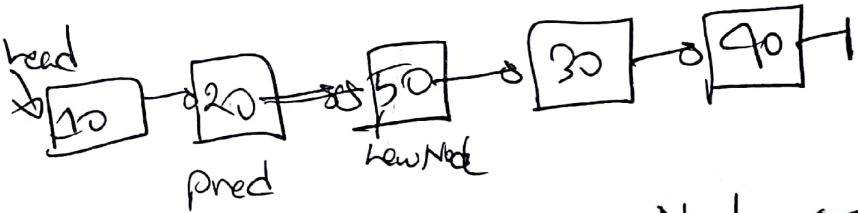


(III)



50 এর next কে pred 20 এর next করলাম। এতে list হারালো না.

(IV)



pred Node এর next কে newNode করে দিলাম কাজ শেষ।

P.T.O

code of inserting new nodes in last (singly)

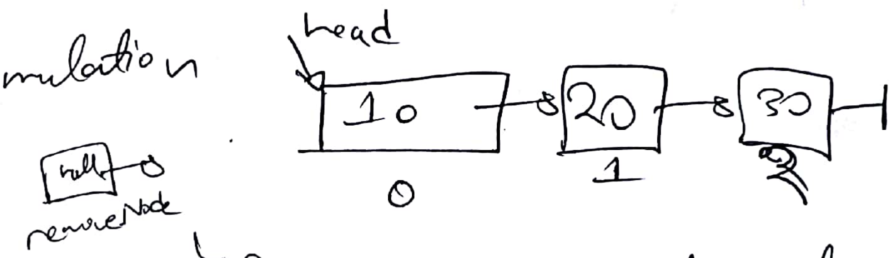
```
public Node insert(Node head, int size; Object elem, int index)

    if (index < 0 || index >= size){
        S.O.P ("Invalid index");
    }
        Node newNode = new Node(elem, null);

    if (index == 0) {
        newNode.next = head;
        head = newNode;
    }else{
            Node pred = NodeAt(index-1);
            newNode.next = pred.next;
            pred.next = newNode;
        }

        return head;
}
```

১১// Removing Nodes :— (Singly)

(i) 0 index থেকে remove মানে start থেকে remove

(ii) মাঝে বা শেষ থেকে remove।

(i) simulation

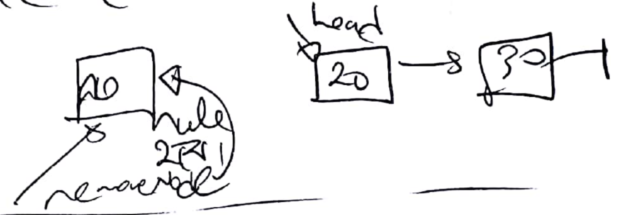


(১) একটি removeNode = null create করে রাখা

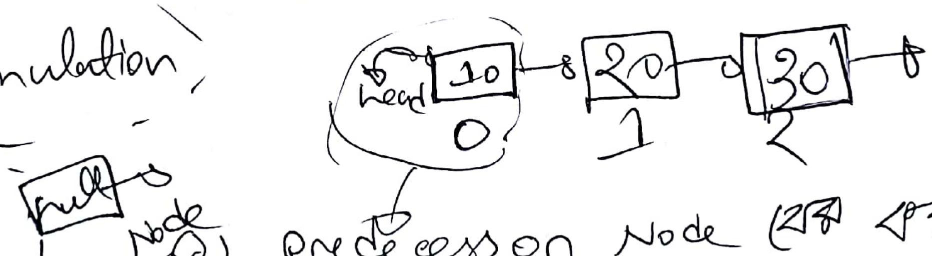
প্রথম ২টি

(২) removeNode এ head Node store করবো।

(৩) head কে আগানো হবে।

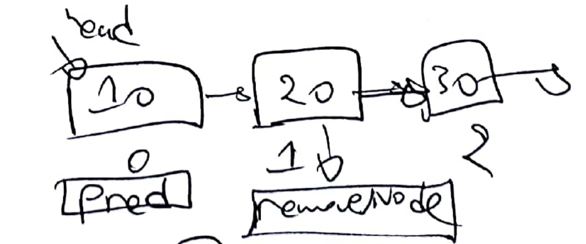


(ii) simulation



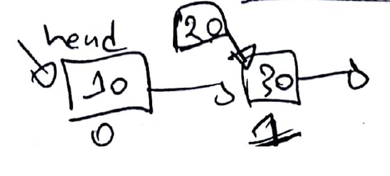
(১) predecessor Node কে একবার Node এ নিতে

(২)



[removeNode যেটা null ছিলো সেটাতে এখন নাও]

(৩)



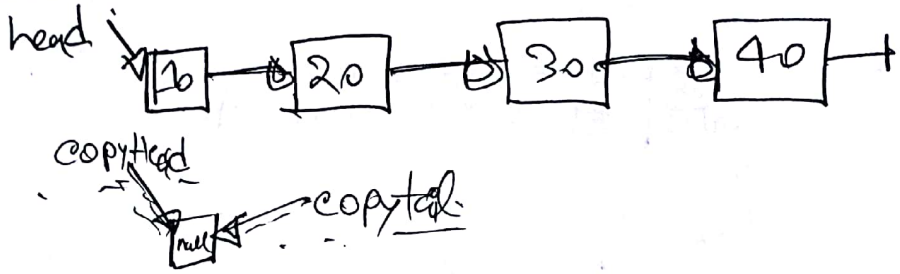
[এগুলো garbage collection

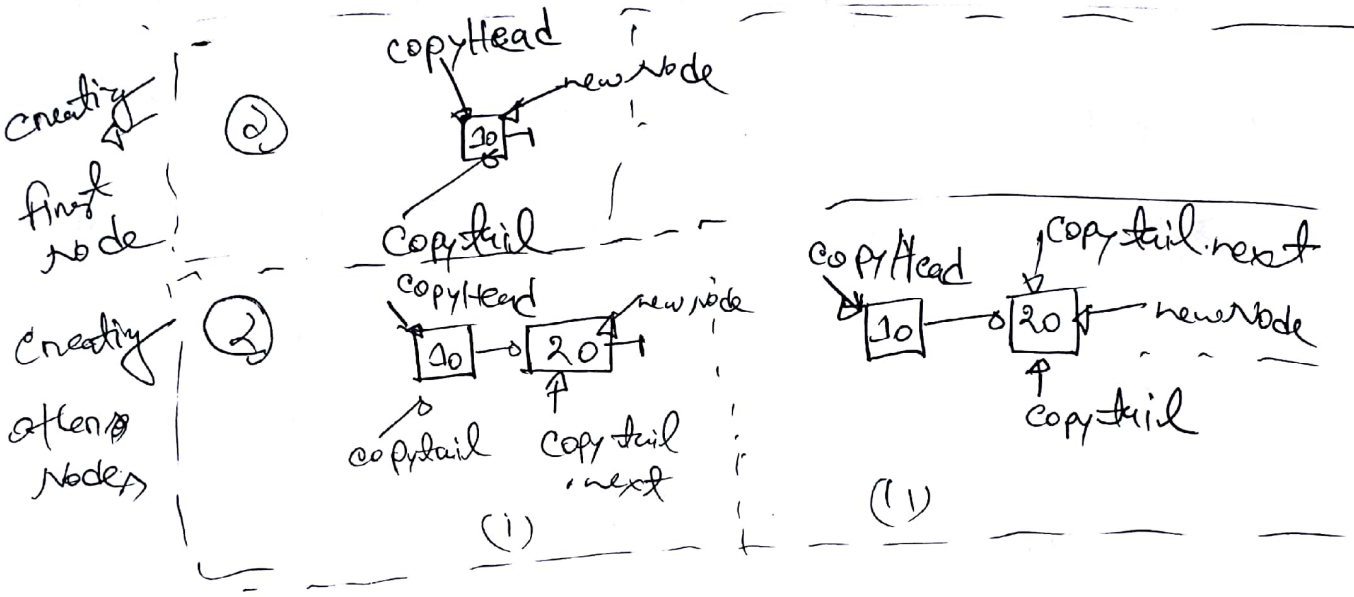
p.T.O

Code of deletion of nodes (singly) :-

```
public Node Remove (Node head, int size, int index){
    // index validation check
    if (index<0 || index>=size){
        S.O.P ("Invalid index");
        return null;
    }
    Node removeNode = null;
    if (index==0){
        removeNode = head;
        head = head.next;
    }
    else{
        Node pred = nodeAt(index-1);
        removeNode = pred.next;
        pred.next = removeNode.next;
    }
    removeNode.next = null;
    removeNode.element = null;
    return head;
}
```

22// Copying a list

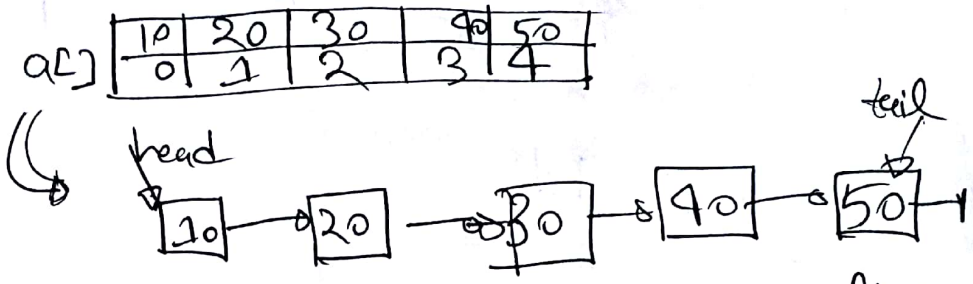





```
public Node copyList(Node head){
    Node copyHead = null;
    Node copyTail = null;
    for(Node n = head; n!=null; n= n.next){
        Node newNode = new Node(n.element, null);
        if (copyHead == null){
            copyHead= newNode;
            copyTail = newNode;
        }else{ copyTail.next = newNode;
               copyTail = newNode;
        }
    }
```

20/1 Creating a singly List from an array :-

(copyList এর মতো)



(1) 0th index থেকে শুরু করে List এর creation শুরু হয় এবং কোন index এ গেলে copy করা বন্ধ হয়।

```
public Node createList(int[] a){

    Node head = null;
    Node tail = null;

    for(int i=0; i<a.length; i++){
        Node newNode = new Node(a[i], null);

        if(head == null){
            head = newNode;
            tail = newNode;
        }else{
            tail.next = newNode;
            tail = newNode;
        }
    }
    return head;
}
```

28# Reversing a List :-

↳ (I) inplace → given list use to reverse

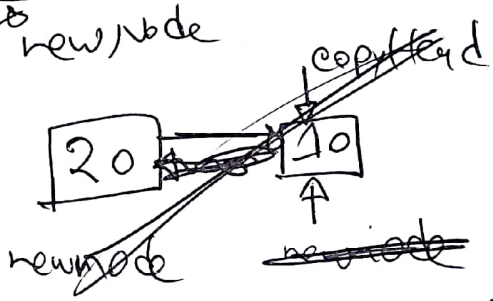
(II) outplace → creating a new list in reverse. ~~and reverse~~

(II) Outplace :- (নতুন করে reverse way তে list create করা।)

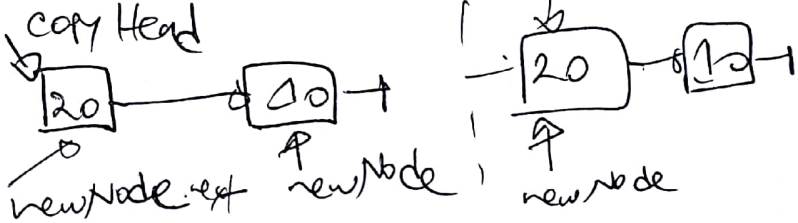
(I)

↳ reverse

(II)



(II)



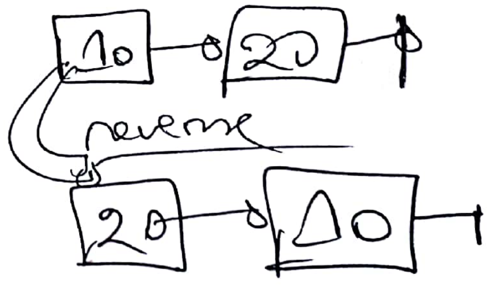
(1) (2)

```
Public Node reverseList (Node head){
    Node copyHead = null;
    for(Node n=head; n!=null; n=n.next){
```

P. 70

Code (reversing a list (outplace) :- (simply) ②

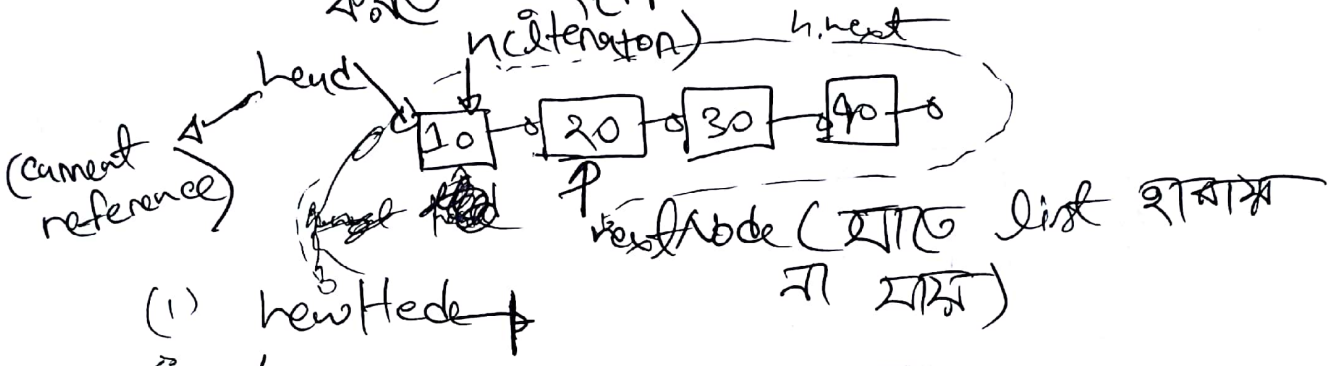
```java
public Node reverseList(Node head){
    Node copyHead = null;

for(Node n=head; n!=null; n=n.next){

    Node newNode = new Node(n.element, null);
    if (copyHead == null){
            copyHead = newNode;
    }
    else{ // নতুন Node এর next এ copyHead দিলাম
        newNode.next = copyHead;
        copyHead = newNode; //copyHead
                              update করলাম
    }

    return copyHead;

}
```



(1) Reversing a List (Singly)

In place approach :-

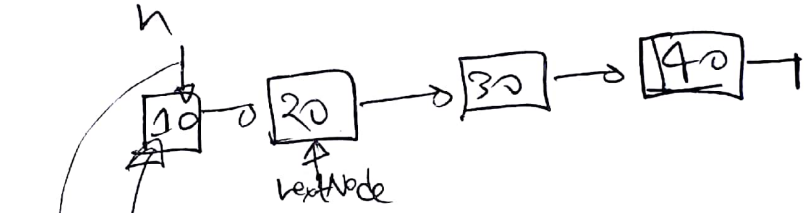
যে list দেওয়া আছে তার মাঝেই reverse করতে হবে।



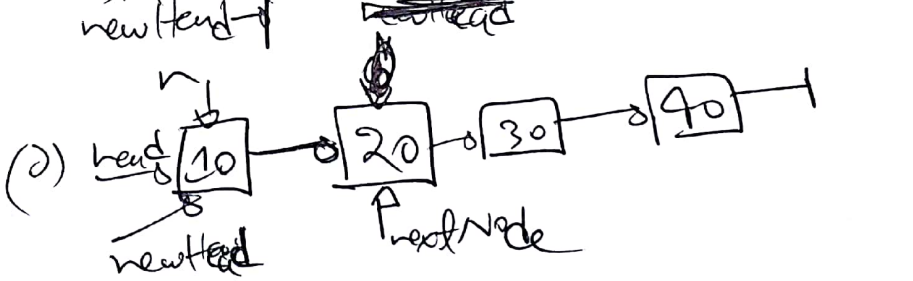
(1) newHead → list এর যেখান থেকে শুরু হবে তার reference রাখবে।

simulation

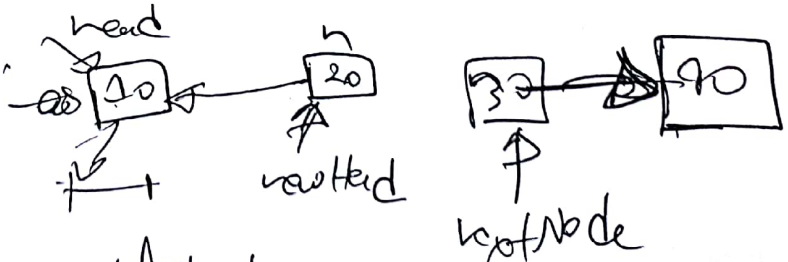
(11)



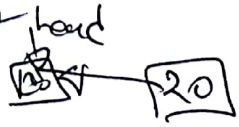
(2)



(2).



(3) n.next head



#code-1

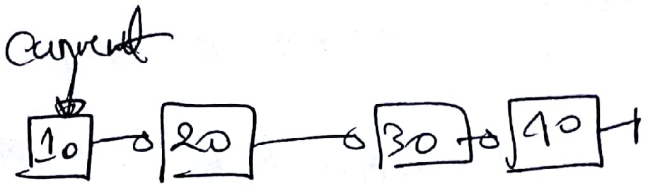
```
Public Node reverseList (Node head) {

    Node newHead = ~~new~~ null;

    Node n = head;

    while(n != null) {
        // এখানে আমার কাছে nextnode এর reference
           লাগবে।
        Node nextnode = n.next;
        // ~~newHead~~ newHead টিকে n এর সাথে
           সংযোগ ঘটাবো।
        n.next = newHead; // n.[10] → newHead

        newHead = n;      // n[10] ← newHead

        n = nextNode; // n কে সামনে iterate
                         করে নিয়ে যাচ্ছি
              └→ n iterate করবো
                 [n.next না দিয়ে
                   nextnode দিয়ে
    }
    return newHead;
}
```

#Code-3 (Reverse a singly list) :-

Alternative (In-place Approach)

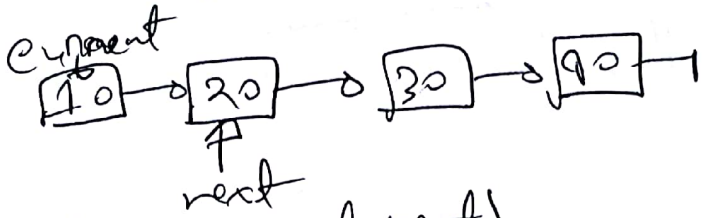
```
Public Node reverseList (Node head){

    Node current = head;
    Node previous = null;
    Node next = null;

    while (current != null){

        next = current.next;
        current.next = previous;
        previous = current;
        current = next;

    }

    return previous;

}
```

Simulation



previous = null, next = null,

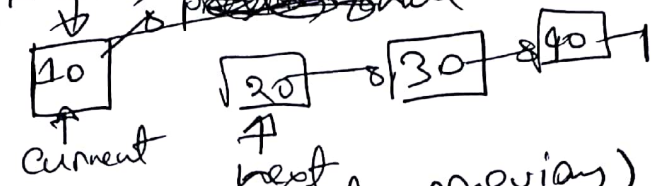
(I)



(next = current.next)

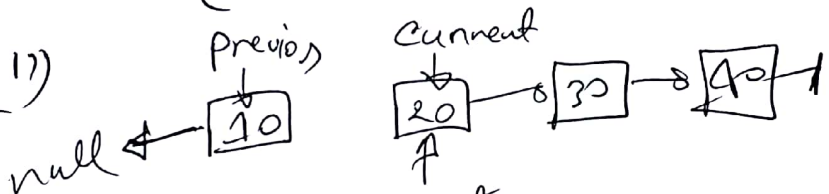
loop

(II)



(current.next = previous) & (previous = current)

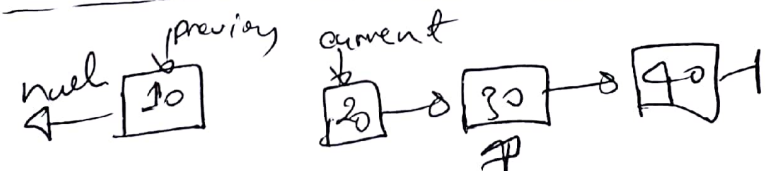
(III)



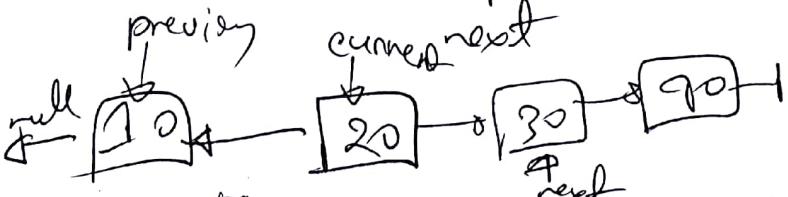
---

2 번째 loop

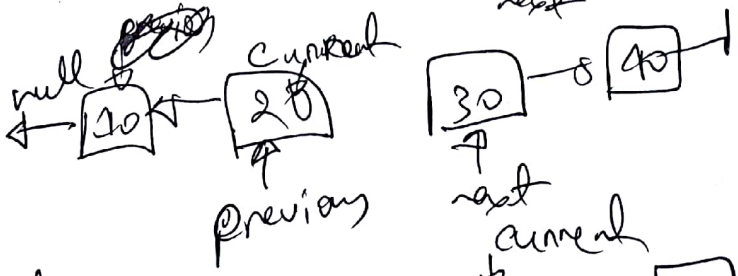
(I)



(II)



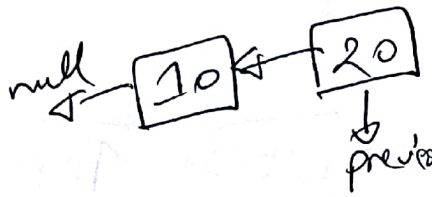
(III)

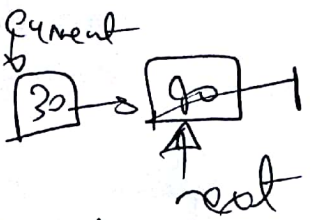


(IV)

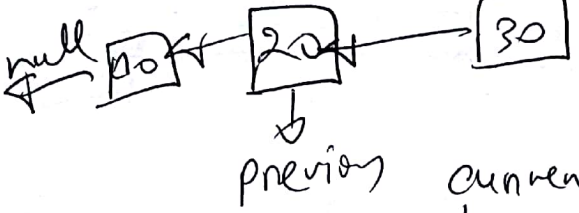
৩য় ধাপ

(I)



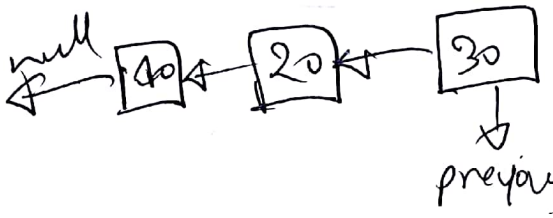


(II)



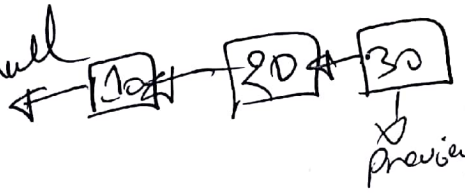
(III)

(IV)

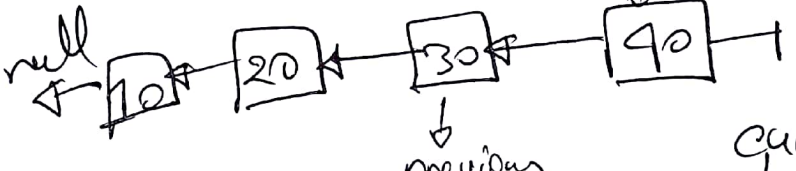


২য় ধাপ

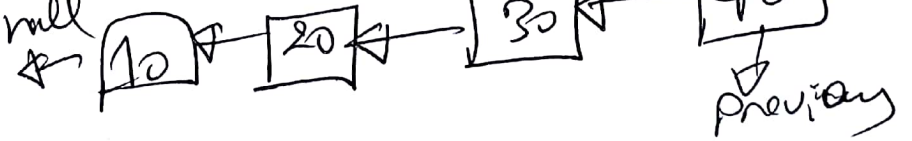
(I)



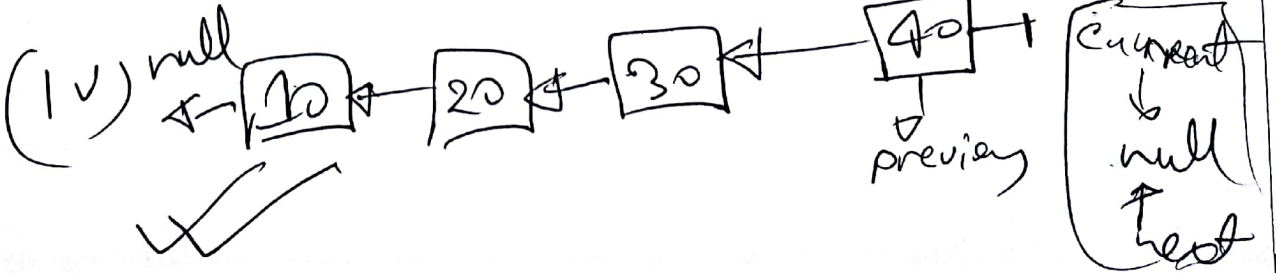
(II)



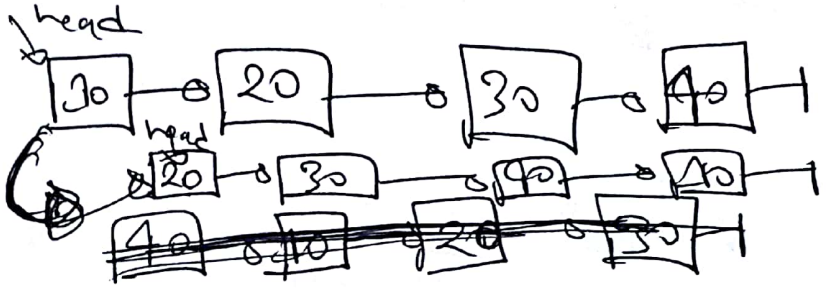
(III)



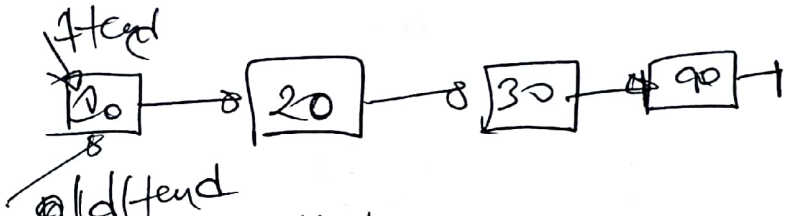
(IV)



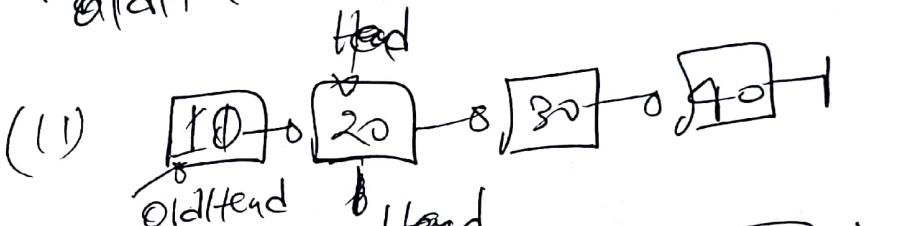
2C notatleft (singly) :-



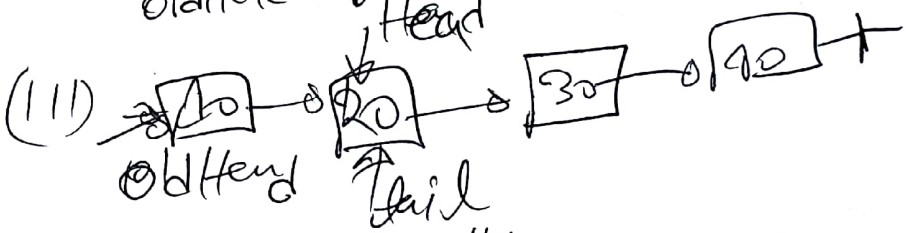
(I)



(II)

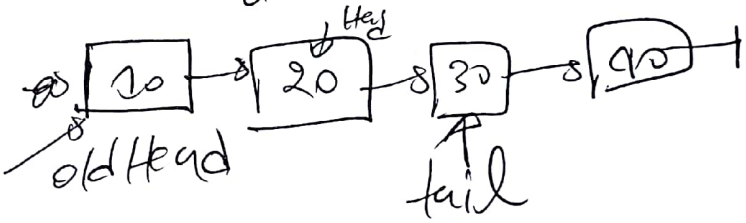


(III)

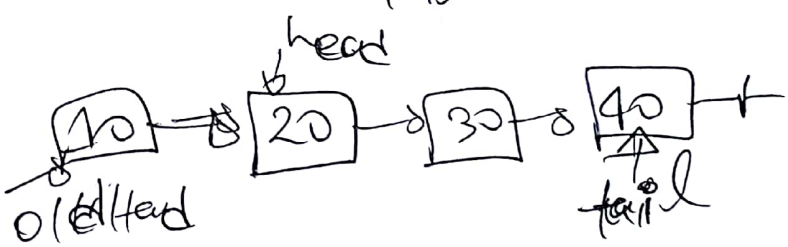


while loop

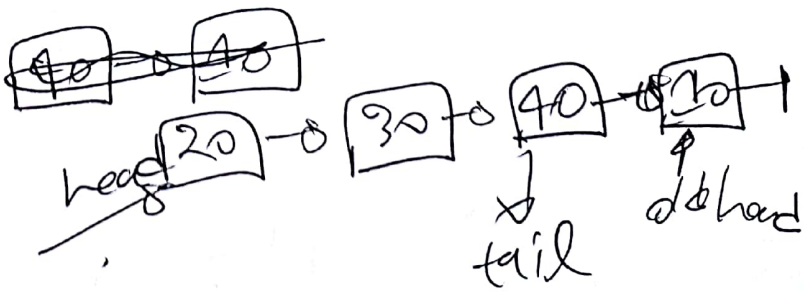
(IV)



(V)



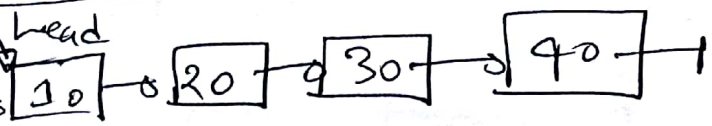
(VI)



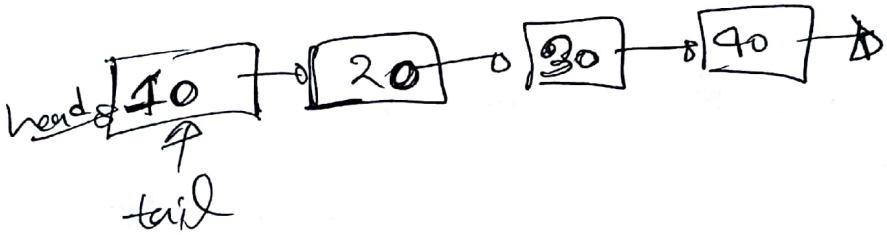
```
public Node rotateLeft(Node head, int size){
if (size == 0){
    S.O.P ("Nothing do. Node");
    return null;
}

Node oldhead = head;

head = head.next;
Node tail = head;
while (tail.next != null){
    tail = tail.next;
}

tail.next = oldHead;

oldHead.next = null;

return head;

}
```

2) moveLeft (Singly) :-

pnull a

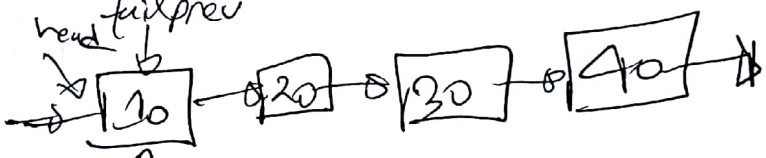


(I)

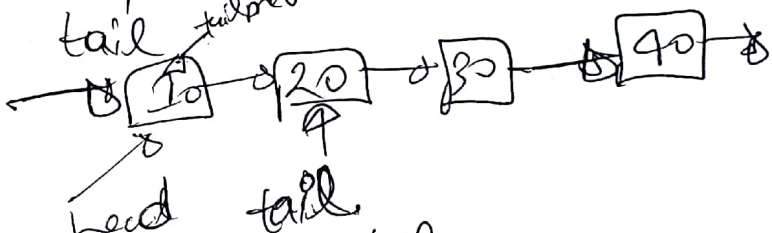


tailprev = null

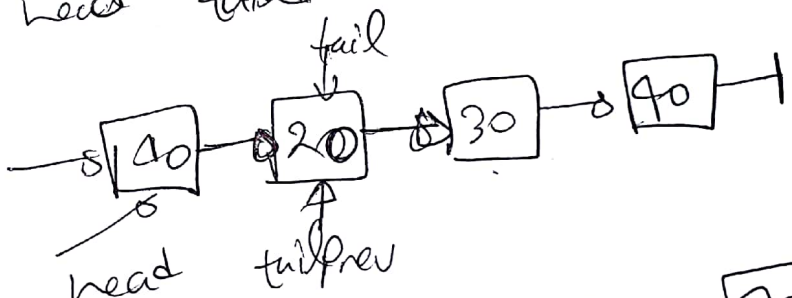
(II)



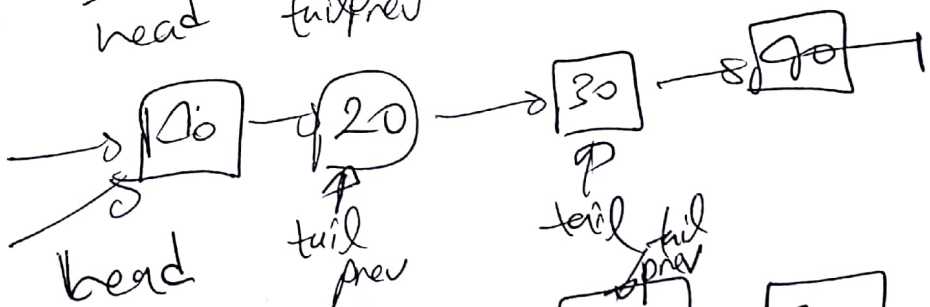
(III)



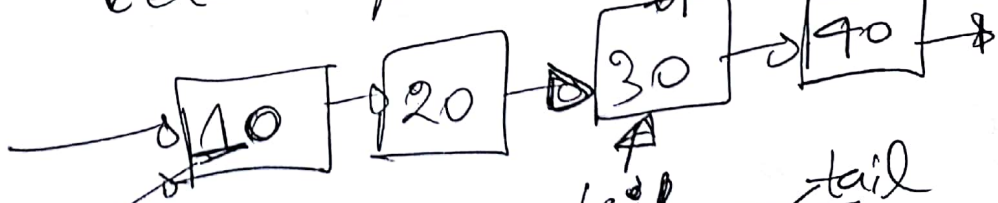
(IV)



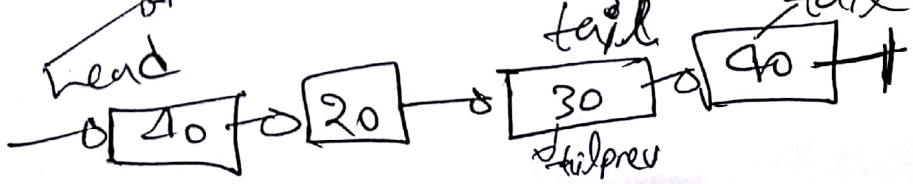
(V)

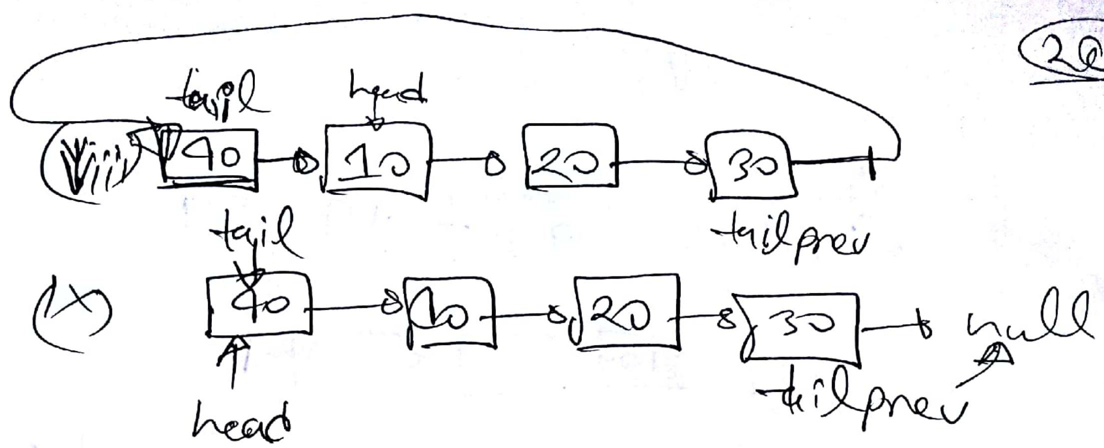


(VI)



(VII)





code RotateRight :-

```
public Node RotateRight(Node head){
    Node tail = head;
    Node tailprev = null;
    while(tail.next != null){
        tailprev = tail;
        tail = tail.next;
    }

    tail.next = head;
    head = tail; //tail কে head হলো।
    tailprev.next = null; // তারপর 30 এর next band করে null করে দিলাম
    return head;
}
```

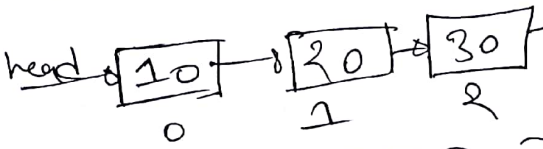
Linked Lists – (2)

Various types

total types (eight)

Singly

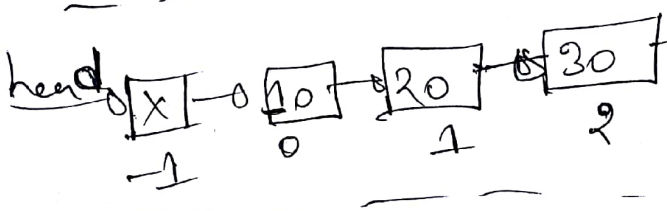
1// Singly Linked List :–



iteration

```
for (Node n=head; n!=null; n=n.next)
{
}
```

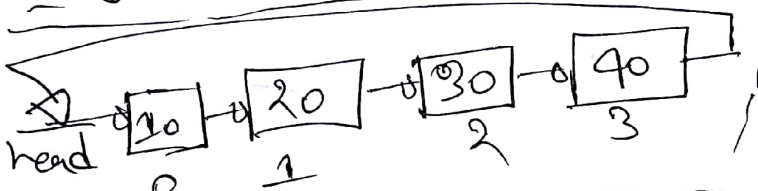
2// Dummy Headed Singly List :–



iteration :–

```
for(Node n=head.next; n!=null;
    n=n.next)
{
}
```

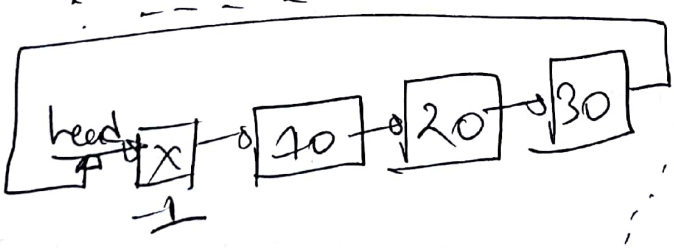
3// Singly Circular Linked List :–



iteration

```
for(Node n=head; n.next!=head;
    n=n.next)
{
}
```

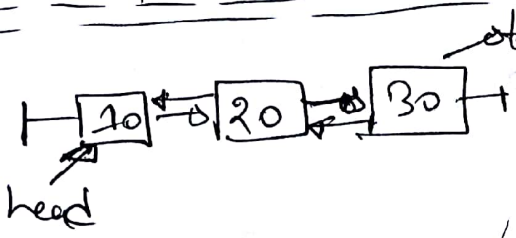
4// Dummy Headed Circular Singly Linked List :–



iteration :–

```
for(Node n=head.next; n!=head; n=n.next)
{
}
```

a/ Doubly Linked List :-



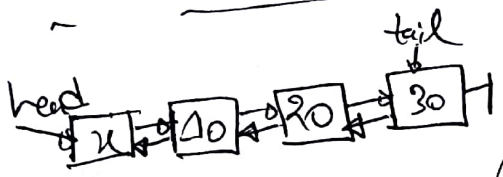
(i) Forward iteration:

```
for(Node n = head; n!=null; n=n.next){
}
```

(ii) Backward iteration :-

```
for(Node n=tail; n!=null; n=n.prev){
}
```

b/ Dummy Headed Doubly Linked List



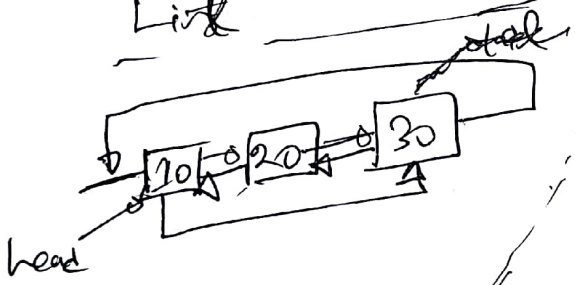
(i) Forward iteration :-

```
for(Node n=head.next; n!=null; n=n.next){
}
```

ii) Backward iteration :-

```
for(Node n=tail; n.prev!=null; n=n.prev){
}
```

c/ Doubly Circular Linked List :-



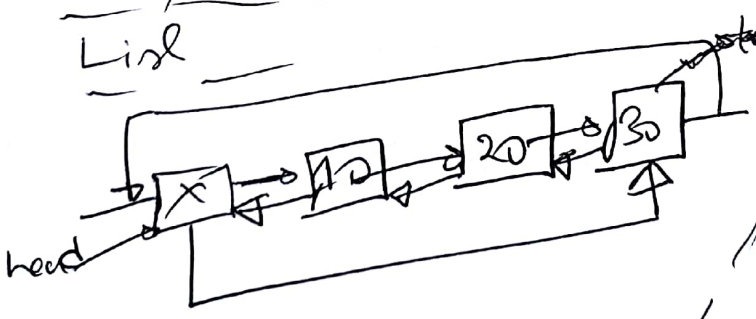
(i) Forward iteration :-

```
for(Node n=head; n.next!=head; n=n.next){
}
```

(ii) Backward iteration :-

```
for(Node n=head.prev; n!=head; n=n.prev){
}
```

d/ Dummy Headed Doubly Circular Linked List



(i) Forward iteration :-

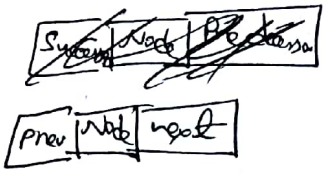
```
for(Node n=head.next; n!=head; n=n.next){
}
```

(ii) Backward iteration :-

```
for(Node n=head.prev; n!=head; n=n.prev){
}
```

Experimenting with Dummy Headed Doubly circular Linked List:

```
public class Node {
    public Object element;
    public Node next;
    public Node prev;

    public Node(Object e, Node n, Node p) {
        element = e;
        next = n;
        prev = p;
    }
}
```



// Creating a list

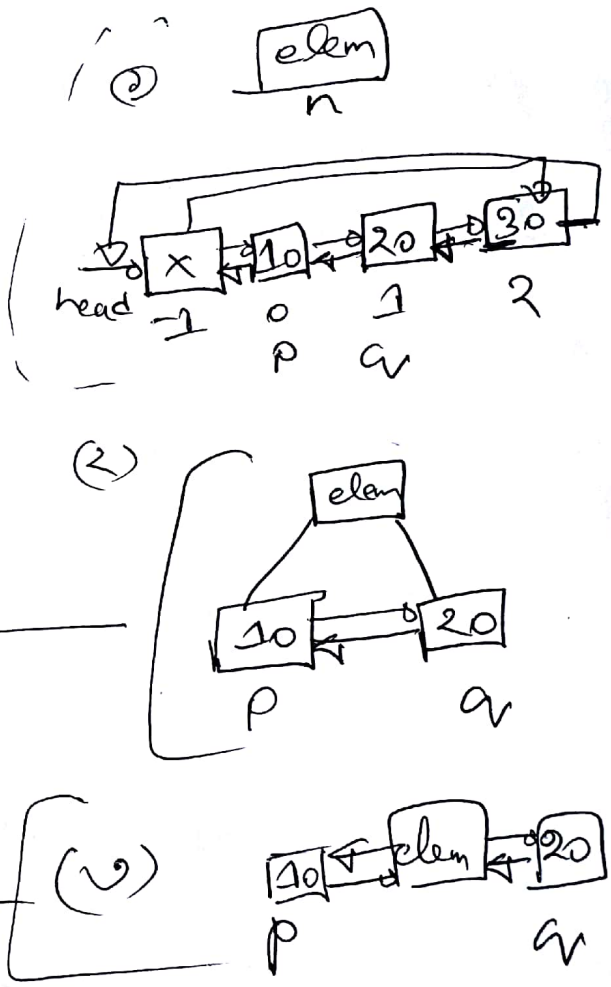
```
Node head = new Node(null, null, null);
head.next = head.prev = head;
```

// Adding first Node

```
Node n = new Node("elem", null, null);
n.next = head.next;
n.prev = head;
head.next = n;
head.prev = n;
```

(2) node insertion :- (2)

```
public Node insertAfter(Node p, Object elem)
{
    Node n = new Node(elem, null, null);
    Node q = p.next;
    n.next = q;
    n.prev = p;
    p.next = n;
    q.prev = n;
    return n;
}
```



(1) node insertion:- // specific element ... ©

// ...

public ~~Node~~ Void insertBefore(Node head, Object elem, Object newElement){

Node p = new node(newElmnt, null, null);

Node q = null;

~~for(Node n=head; n!~~

for (Node n=head.next; n!=head; n=n.next){

if (n.element.equals(elem)){

q=n;

break;

}

}

q.prev = p;

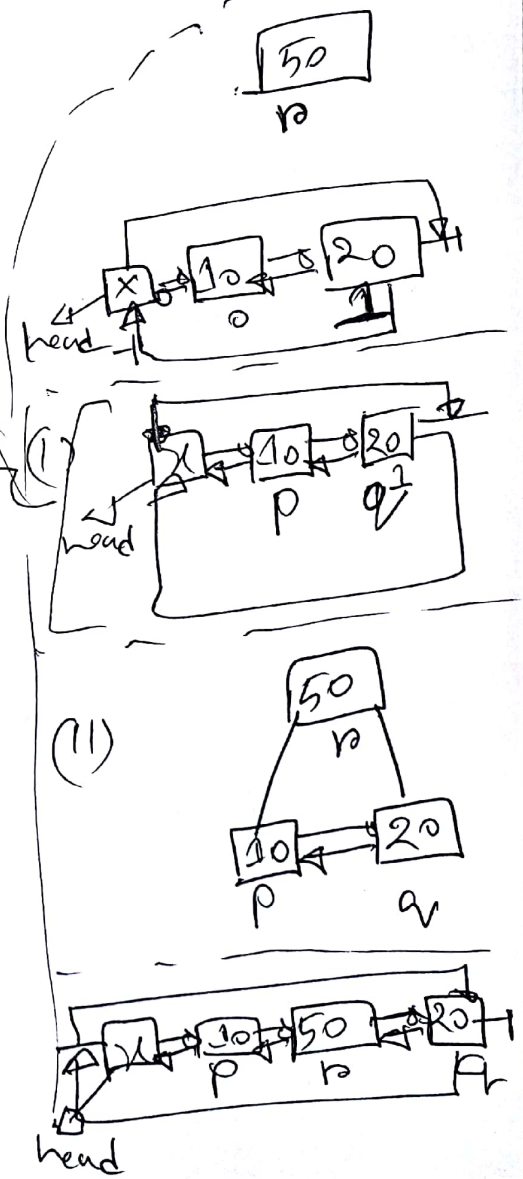
p.next = q;

p.prev = p;

p.next = p;

p.prev = p;

}



Remove :-

```
public static void removeNode(Node n){
    Node p = n.prev;
    Node q = n.next;

    p.next = q;
    q.prev = p

    n.next = n.prev = null;
    n.element = null;
}
```