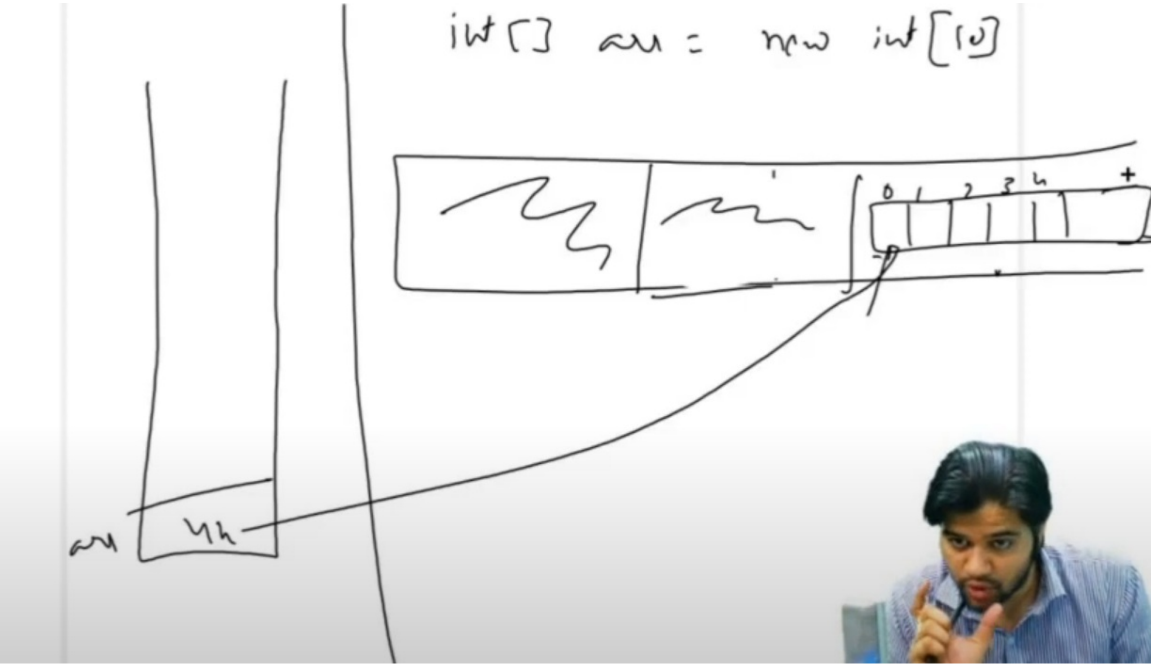
LinkedList

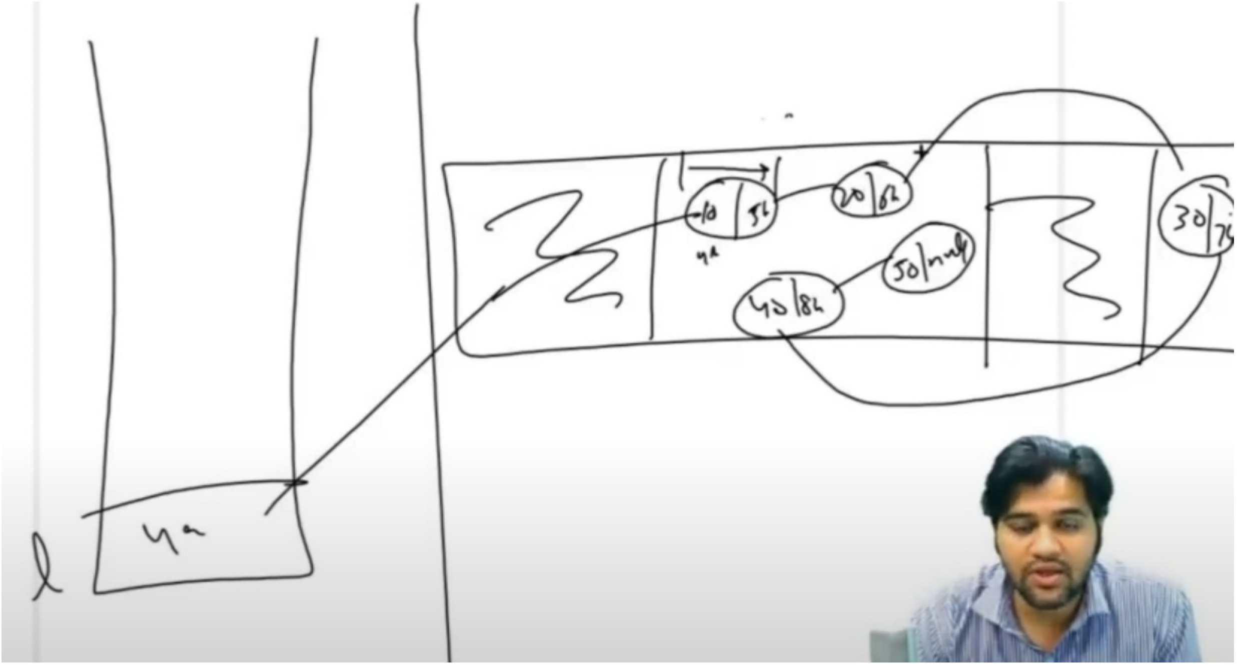
INTRO

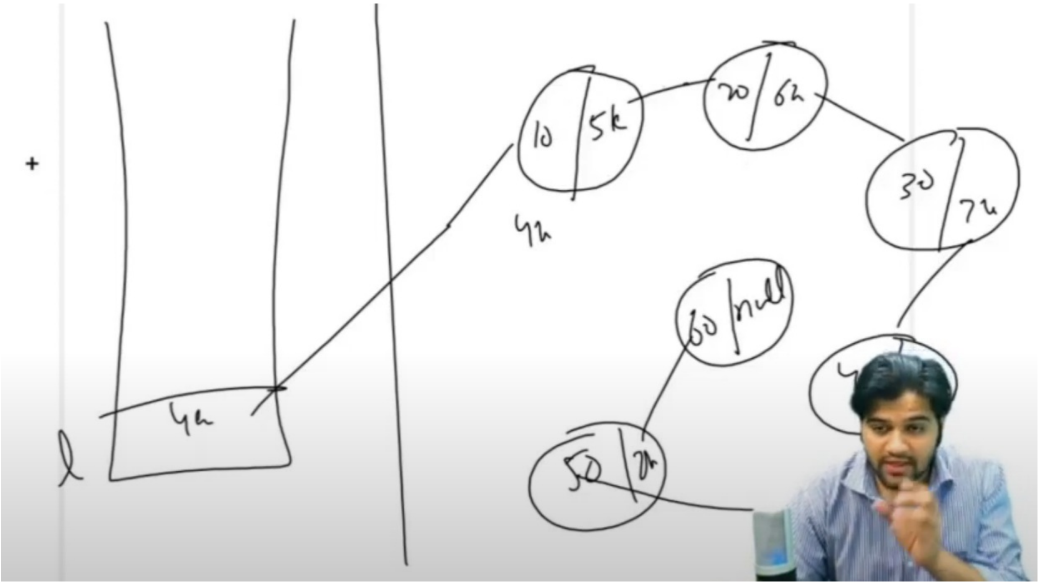
array me continuous data store hota he (4byte int data)



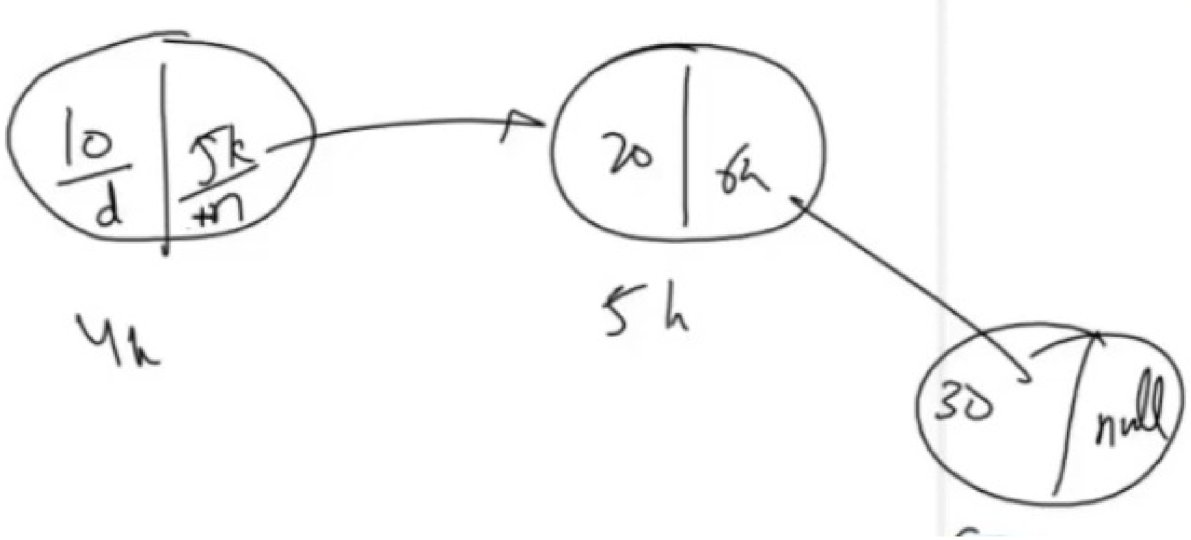
linklist me continuous data store nhi hota he

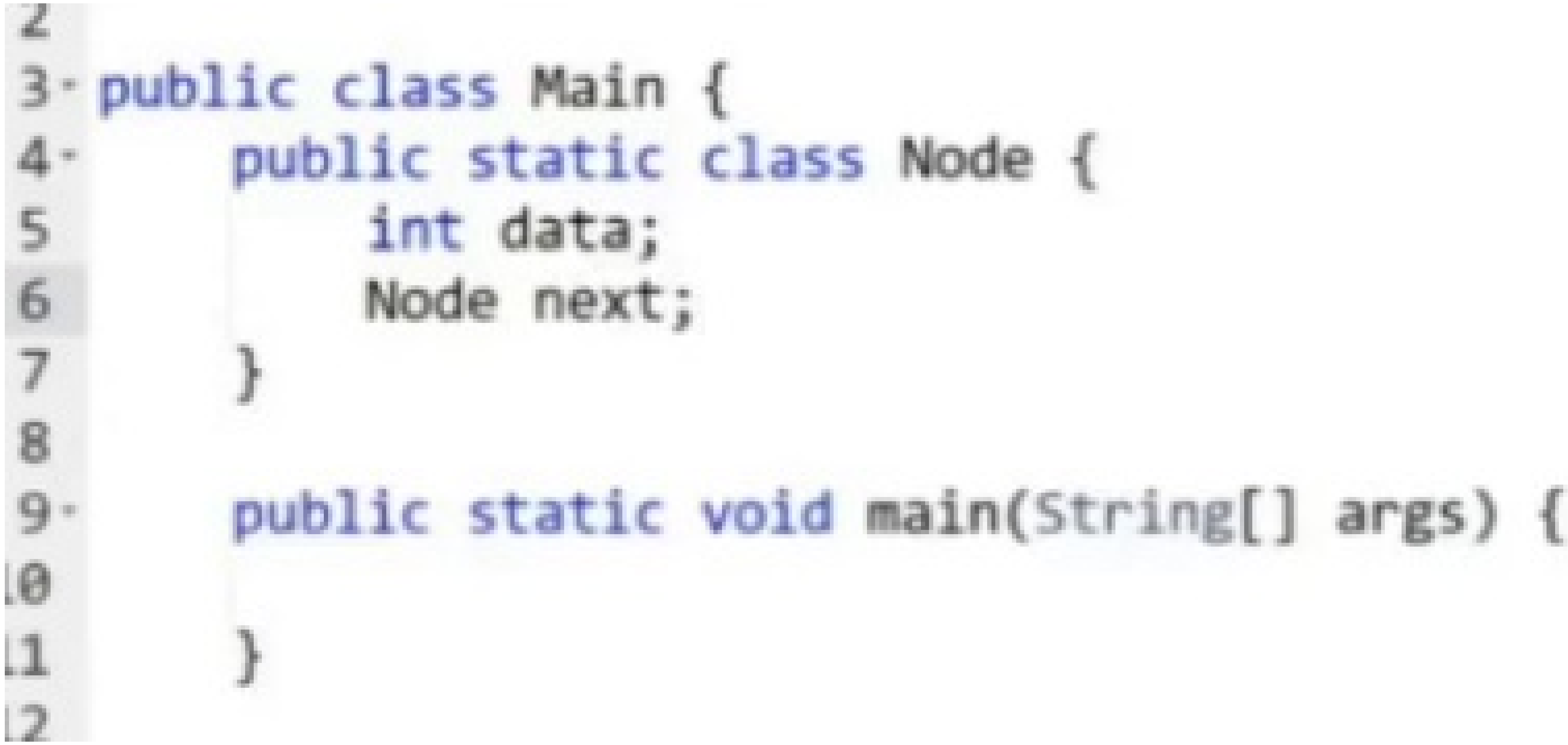
it has data and next address
(4byte int and 4byte address)
and last elemnet have address null





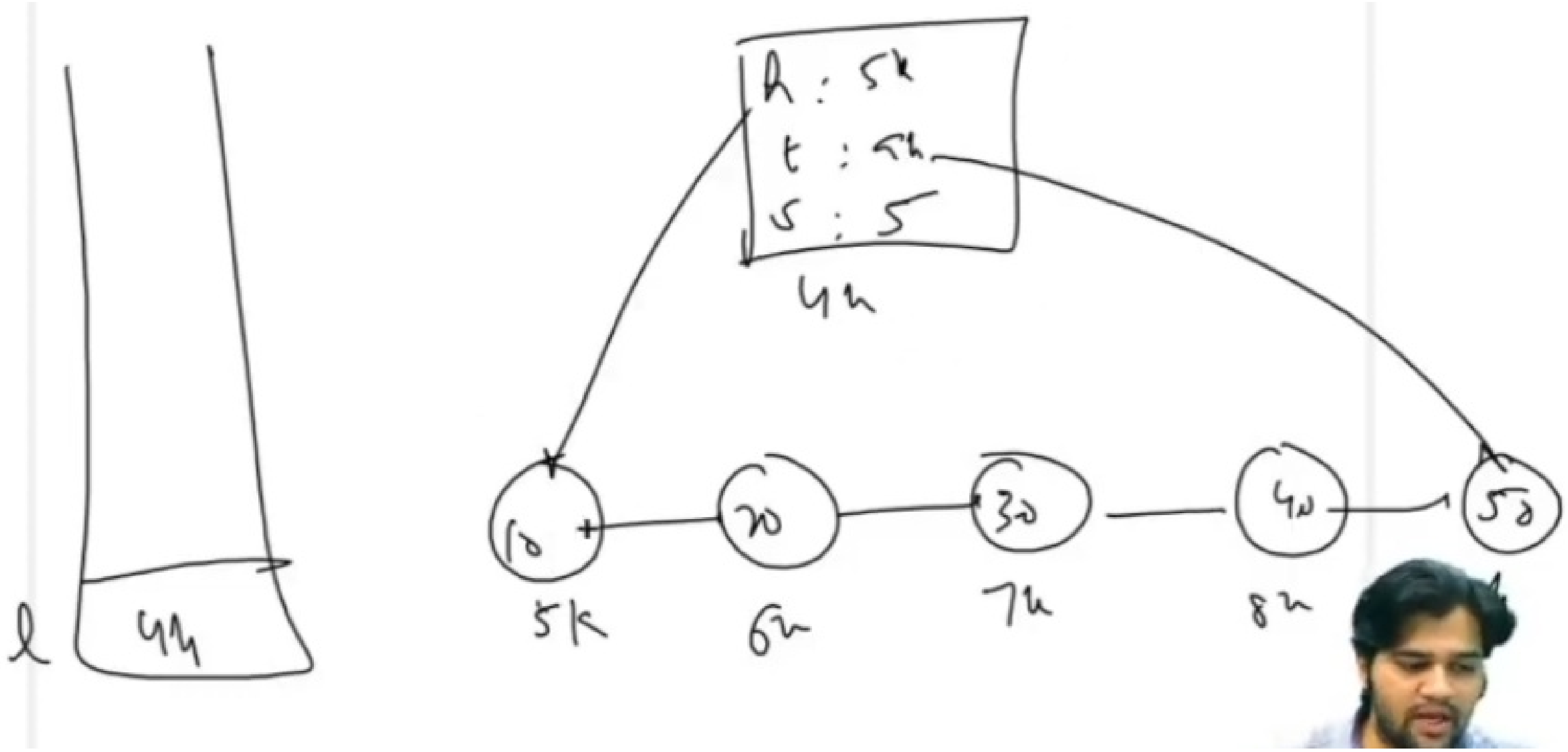
Data Members of a Linked List

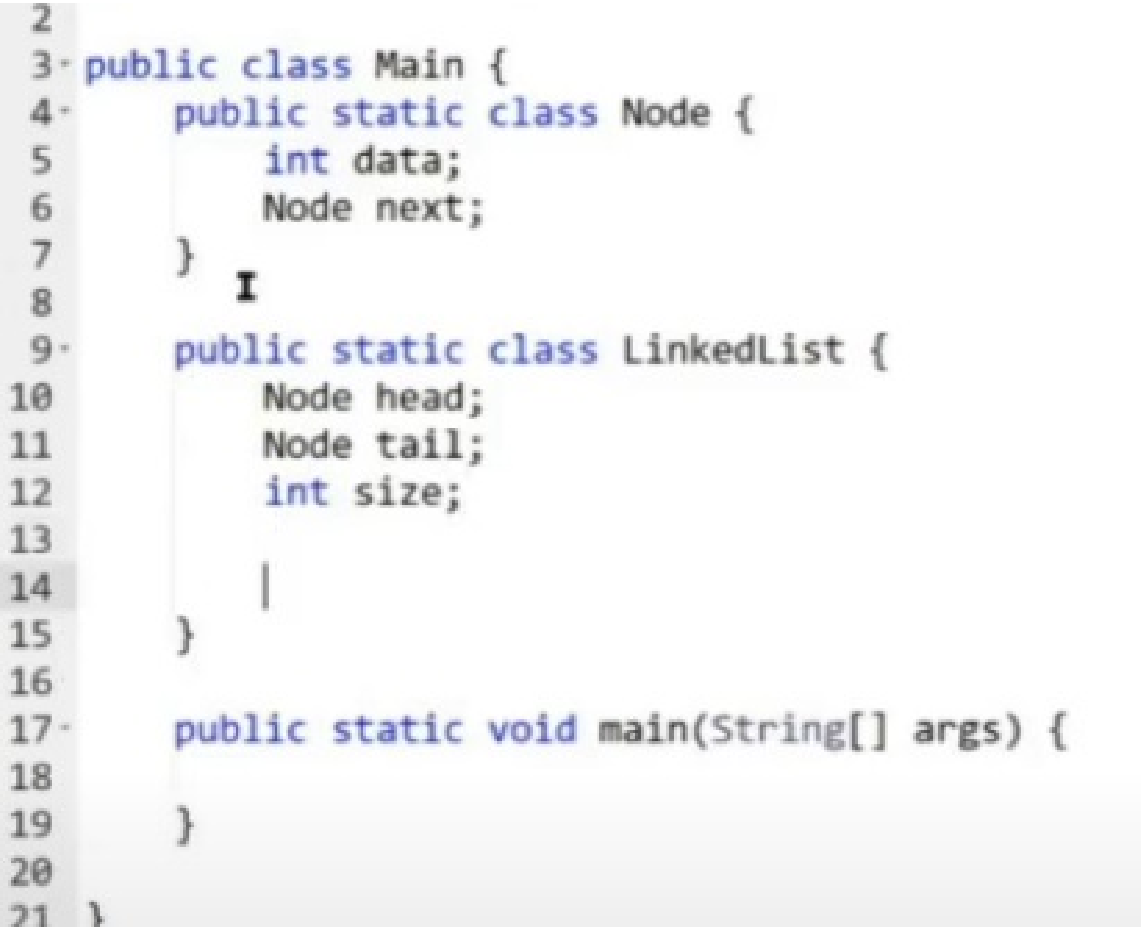




```
public class Main {
    public static class Node {
        int data;
        Node next;
    }

    public static void main(String[] args) {

    }
```

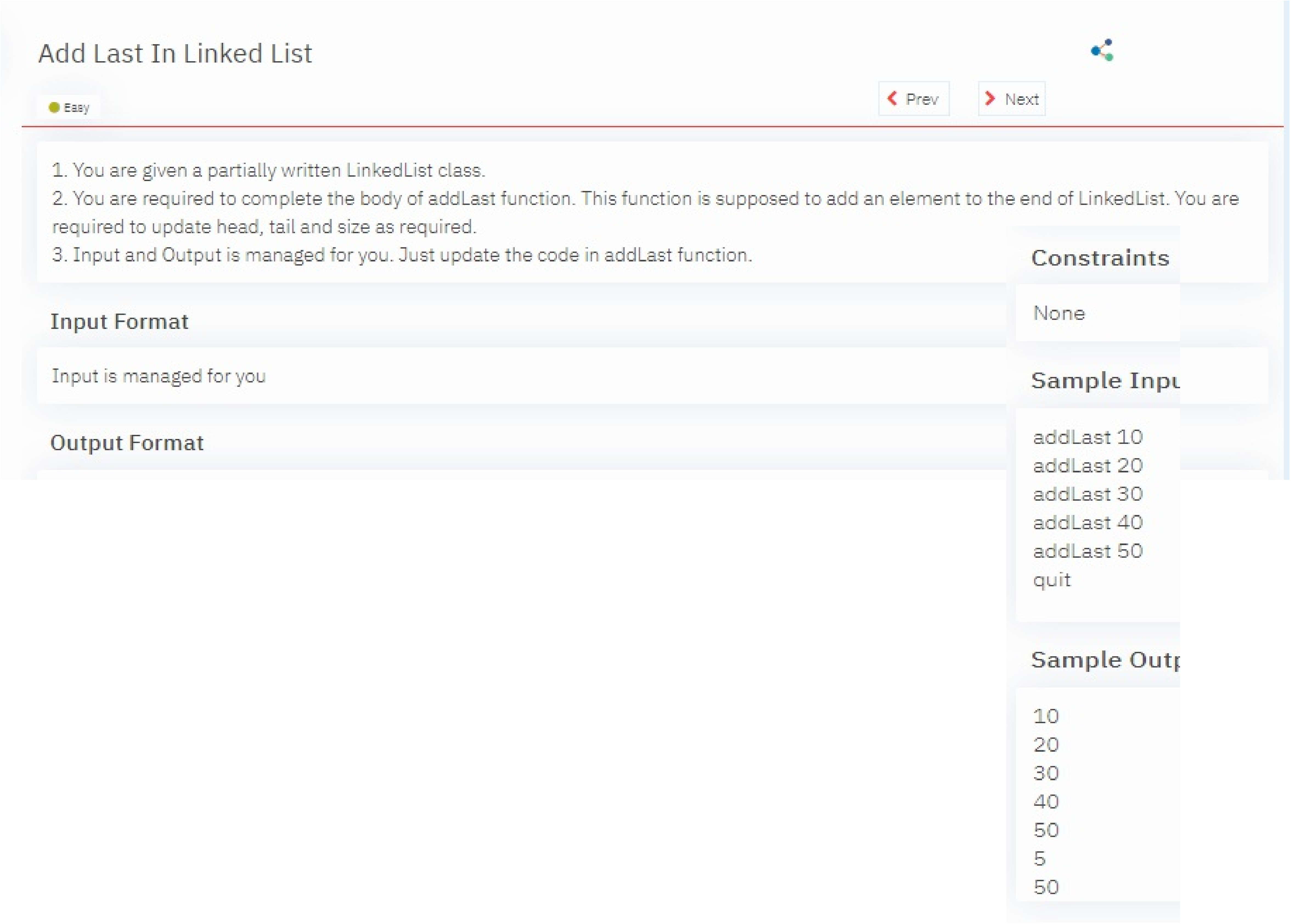
Linklist





```
public class Main {
    public static class Node {
        int data;
        Node next;
    }

    public static class LinkedList {
        Node head;
        Node tail;
        int size;


    }

    public static void main(String[] args) {

    }

}
```

head- 1st node
tail- last node
size- total nodes present



Add Last In Linked List

1. You are given a partially written LinkedList class.
2. You are required to complete the body of addLast function. This function is supposed to add an element to the end of LinkedList. You are required to update head, tail and size as required.
3. Input and Output is managed for you. Just update the code in addLast function.

Input Format

Input is managed for you

Output Format

Constraints

None

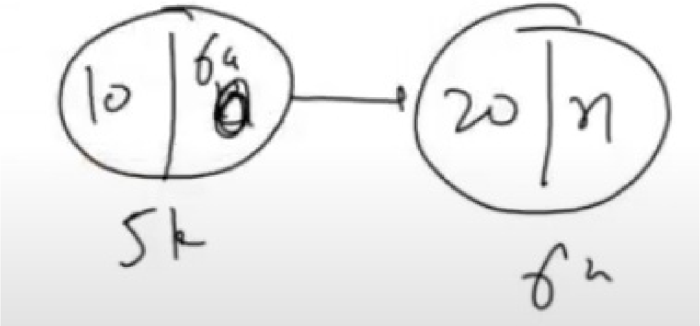
Sample Inpu

```
addLast 10
addLast 20
addLast 30
addLast 40
addLast 50
quit
```

Sample Outp

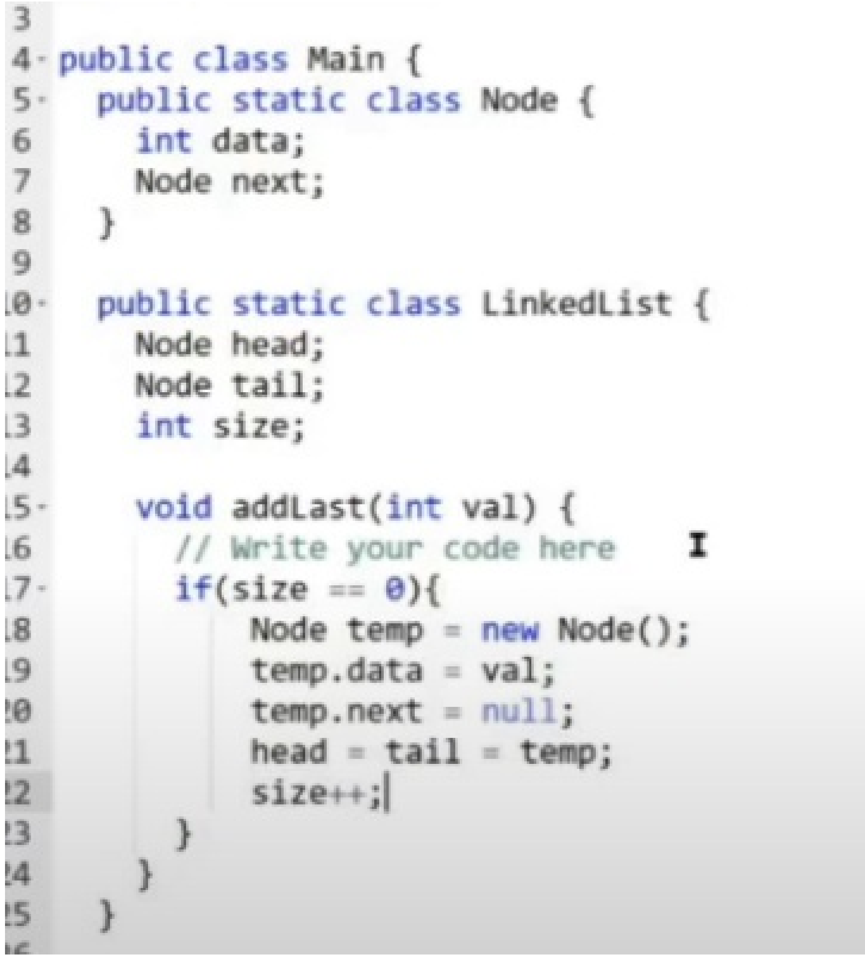
```
10
20
30
40
50
5
50
```

Steps to add node at last



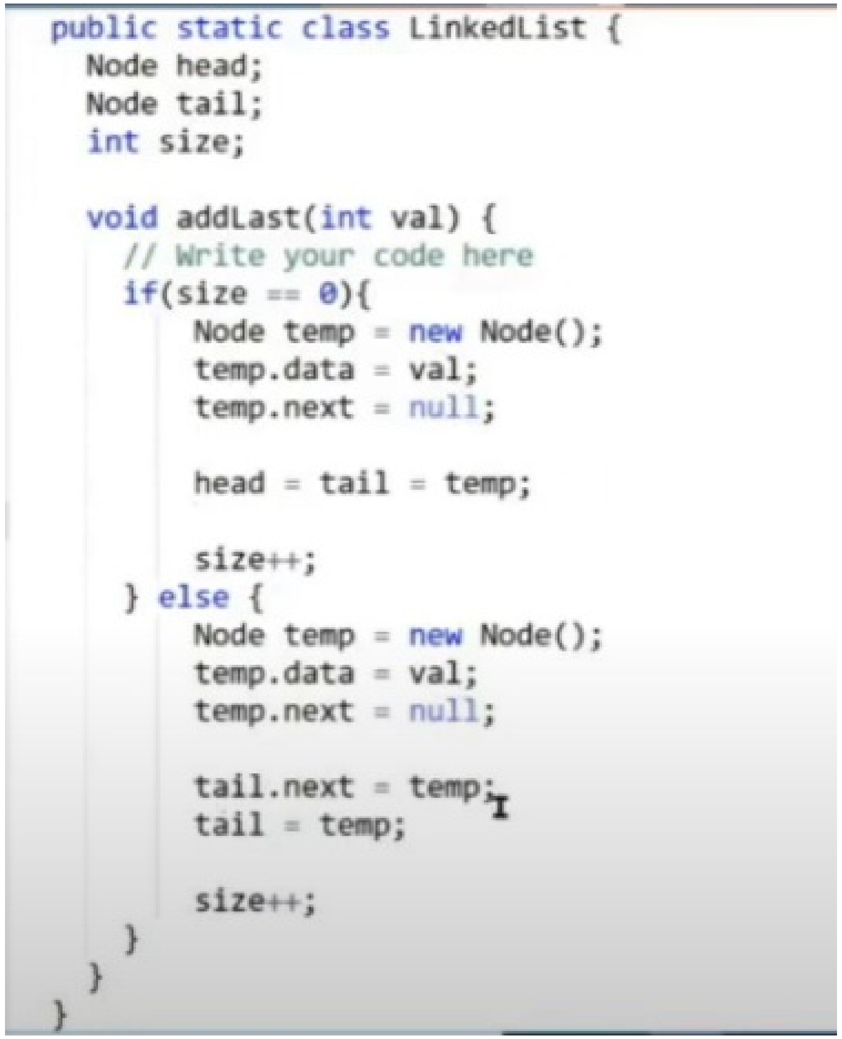
1.make new node
2. node->data = val
3. node->next -null
4. goto last node of linklist and update the node->next with address of new node
5.set current tail->next of linklist as address of above newnode
6. update size of linklist

For 1st node

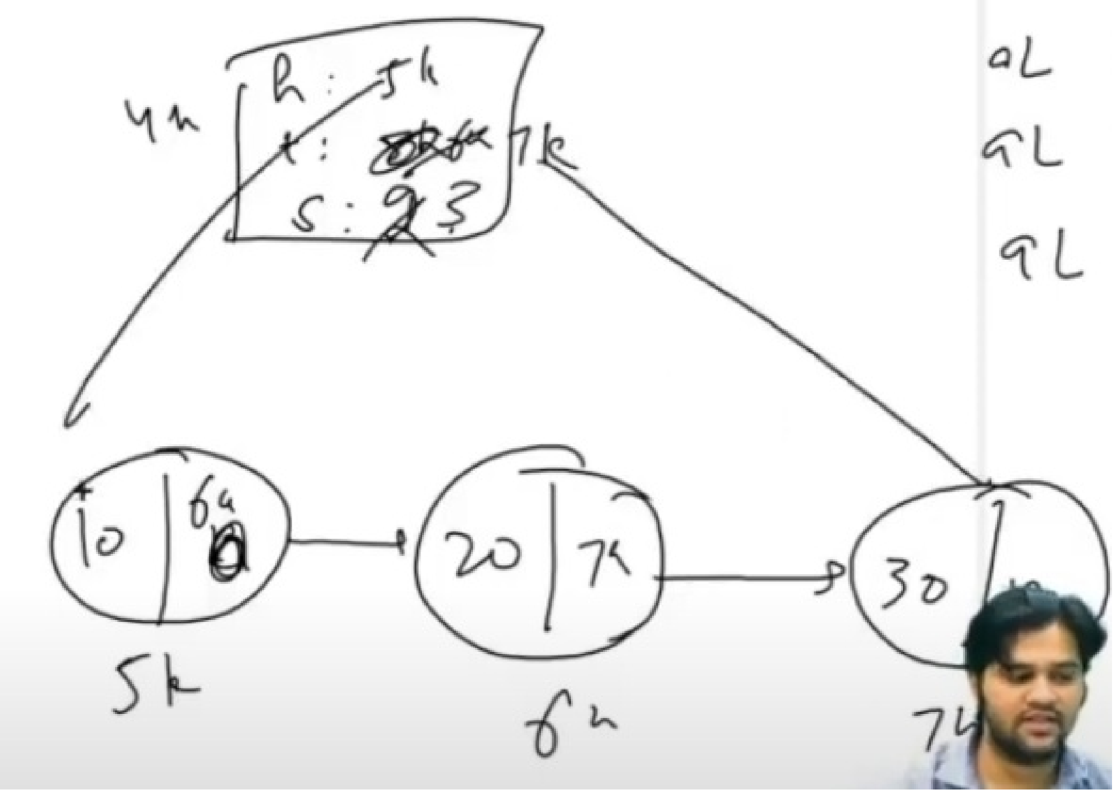


```
public class Main {
  public static class Node {
    int data;
    Node next;
  }

  public static class LinkedList {
    Node head;
    Node tail;
    int size;

    void addLast(int val) {
      // Write your code here
      if(size == 0){
          Node temp = new Node();
          temp.data = val;
          temp.next = null;
          head = tail = temp;
          size++;
      }
    }
  }
```

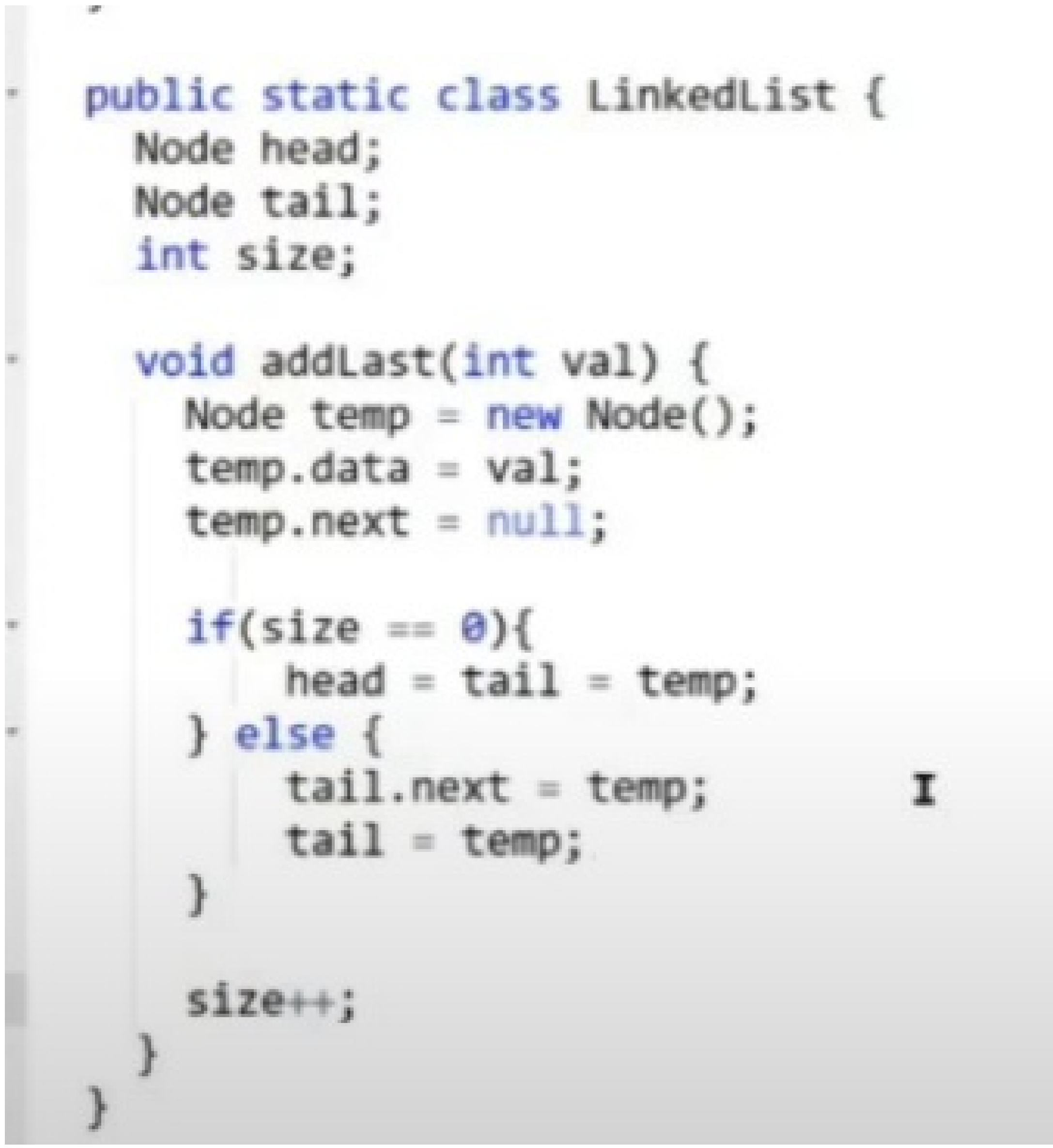
for other node, else part



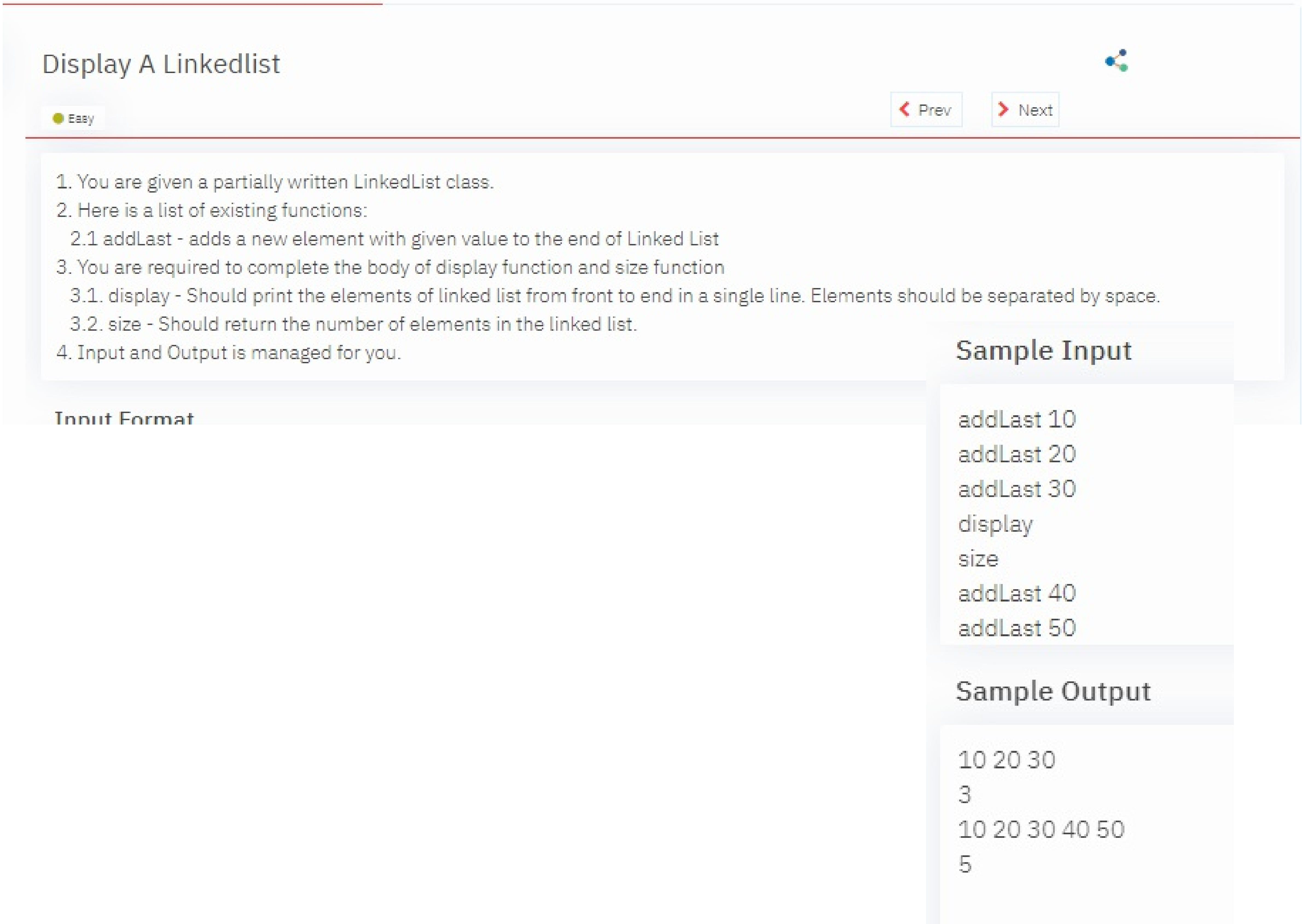
```
public static class LinkedList {
  Node head;
  Node tail;
  int size;

  void addLast(int val) {
    // Write your code here
    if(size == 0){
        Node temp = new Node();
        temp.data = val;
        temp.next = null;

        head = tail = temp;

        size++;
    } else {
        Node temp = new Node();
        temp.data = val;
        temp.next = null;

        tail.next = temp;
        tail = temp;

        size++;
    }
  }
}
```



Clean node



```
public static class LinkedList {
  Node head;
  Node tail;
  int size;

  void addLast(int val) {
    Node temp = new Node();
    temp.data = val;
    temp.next = null;

    if(size == 0){
        head = tail = temp;
    } else {
        tail.next = temp;
        tail = temp;
    }

    size++;
  }
}
```



Display A Linkedlist

1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
  2.1 addLast - adds a new element with given value to the end of Linked List
3. You are required to complete the body of display function and size function
  3.1. display - Should print the elements of linked list from front to end in a single line. Elements should be separated by space.
  3.2. size - Should return the number of elements in the linked list.
4. Input and Output is managed for you.

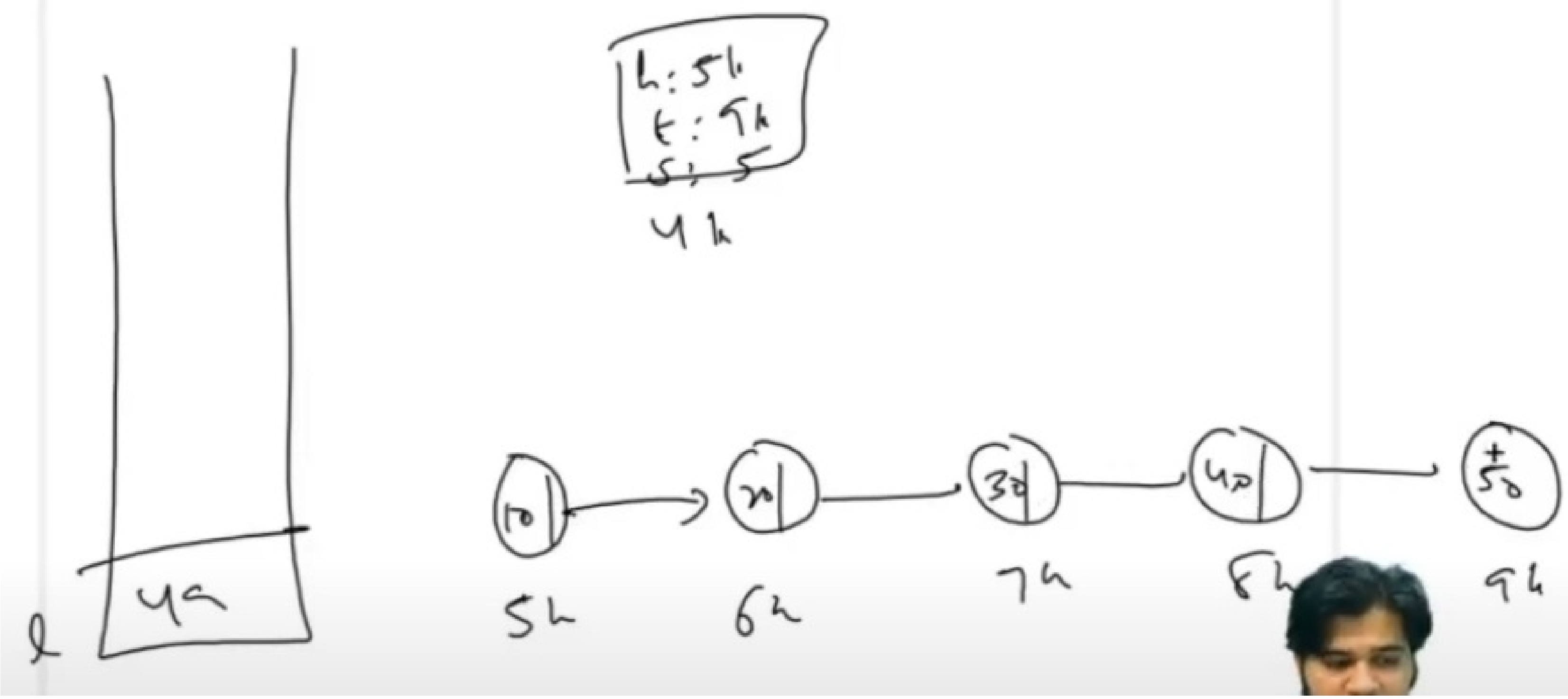
Input Format

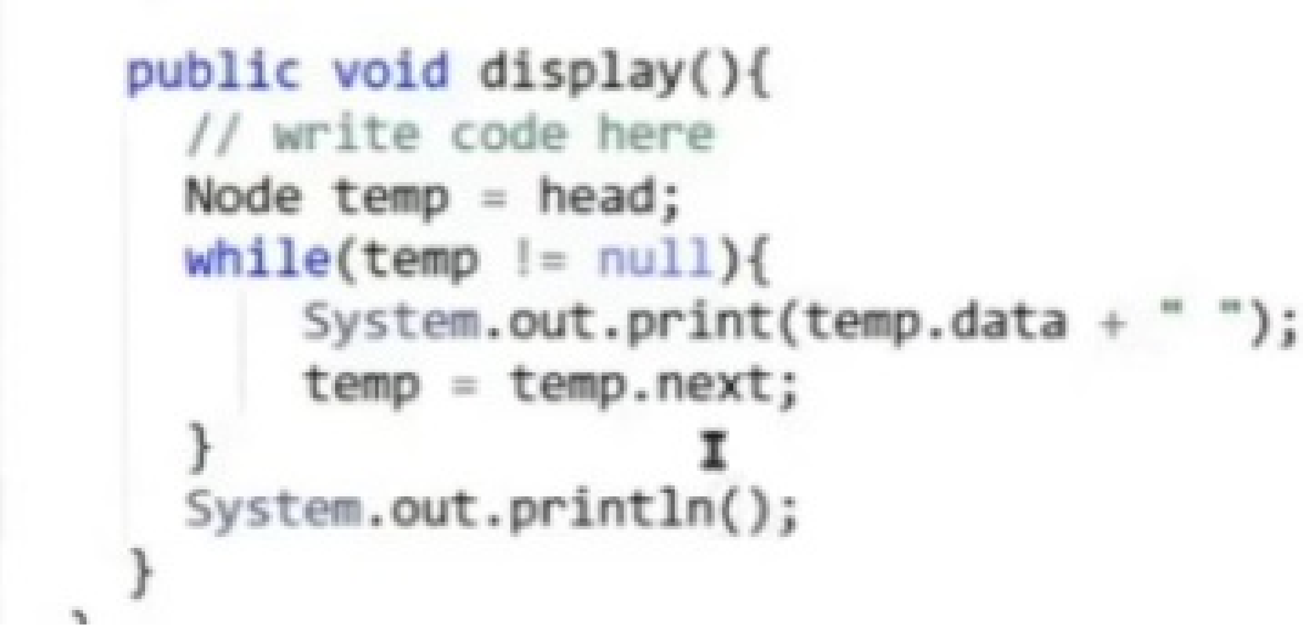
Sample Input

```
addLast 10
addLast 20
addLast 30
display
size
addLast 40
addLast 50
```

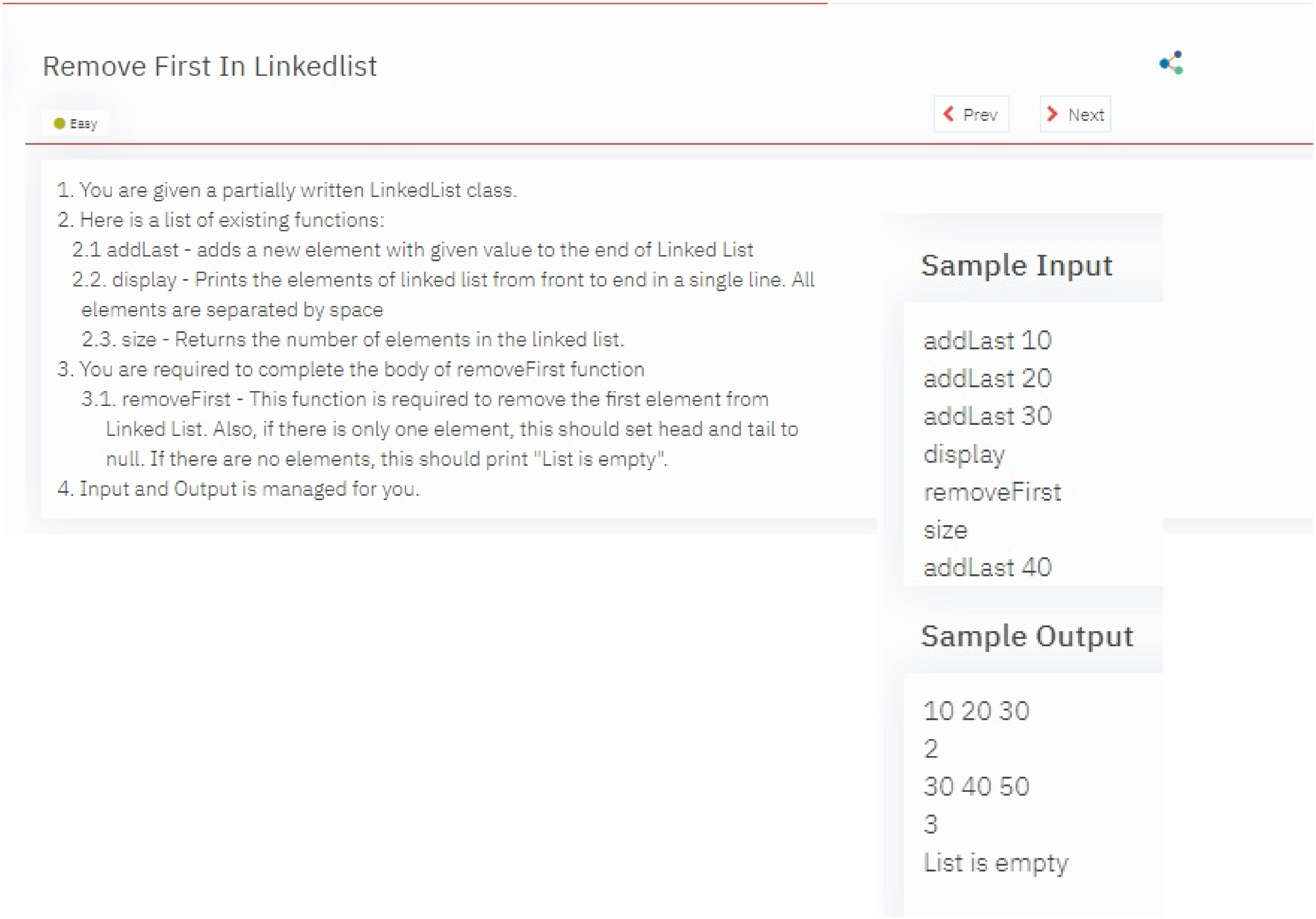
Sample Output

```
10 20 30
3
10 20 30 40 50
5
```





```
public void display(){
  // write code here
  Node temp = head;
  while(temp != null){
      System.out.print(temp.data + " ");
      temp = temp.next;
  }
  System.out.println();
}
```



Remove First In Linkedlist

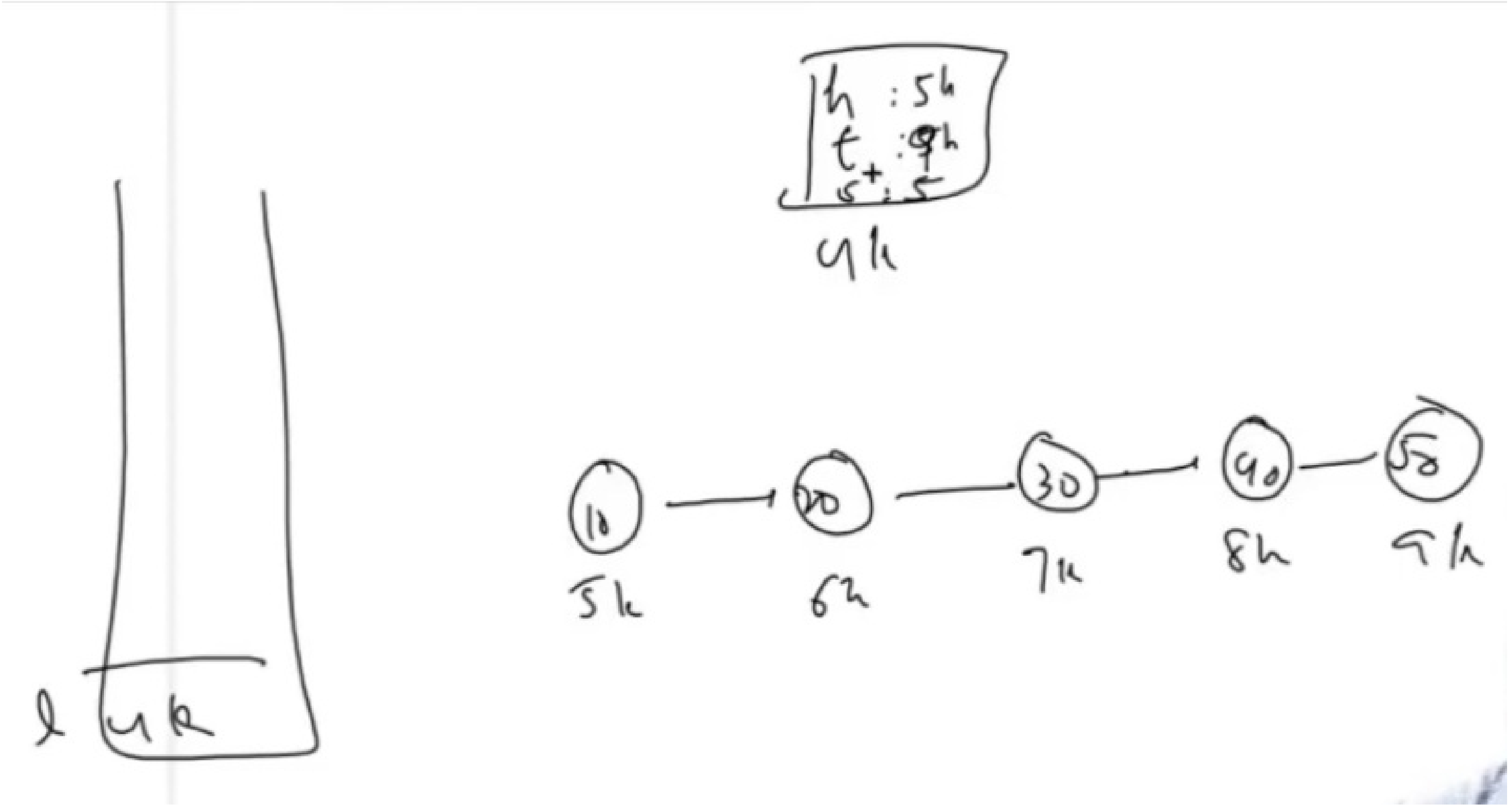
1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
   2.1 addLast - adds a new element with given value to the end of Linked List
   2.2. display - Prints the elements of linked list from front to end in a single line. All elements are separated by space
   2.3. size - Returns the number of elements in the linked list.
3. You are required to complete the body of removeFirst function
   3.1. removeFirst - This function is required to remove the first element from Linked List. Also, if there is only one element, this should set head and tail to null. If there are no elements, this should print "List is empty".
4. Input and Output is managed for you.

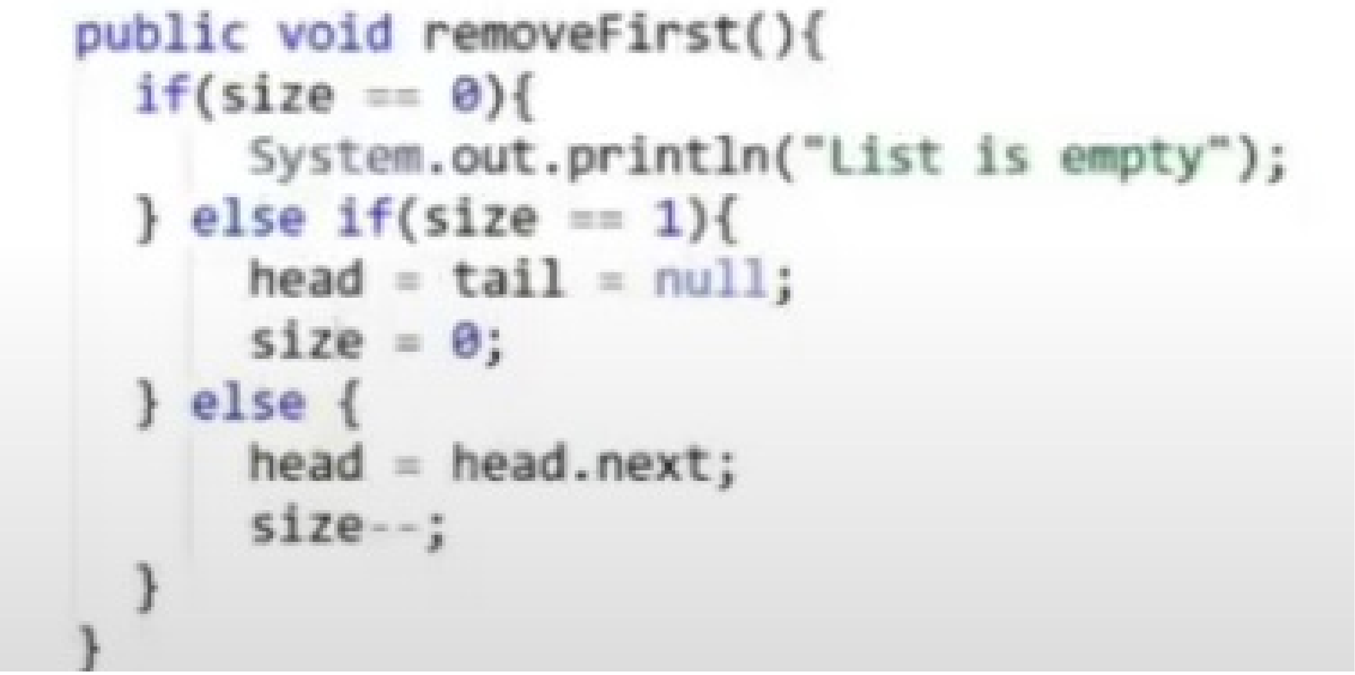
Sample Input

```
addLast 10
addLast 20
addLast 30
display
removeFirst
size
addLast 40
```

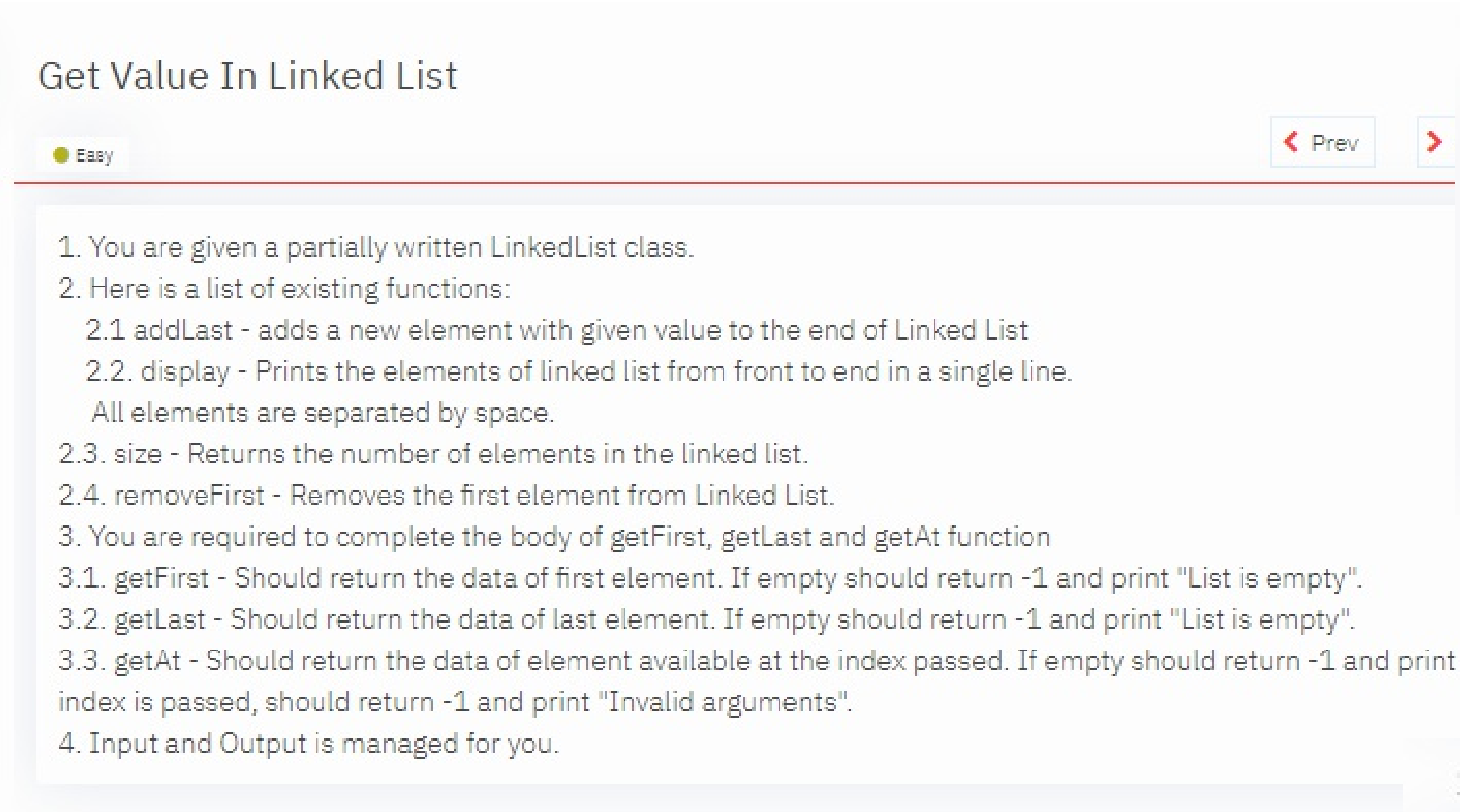
Sample Output

```
10 20 30
2
30 40 50
3
List is empty
```



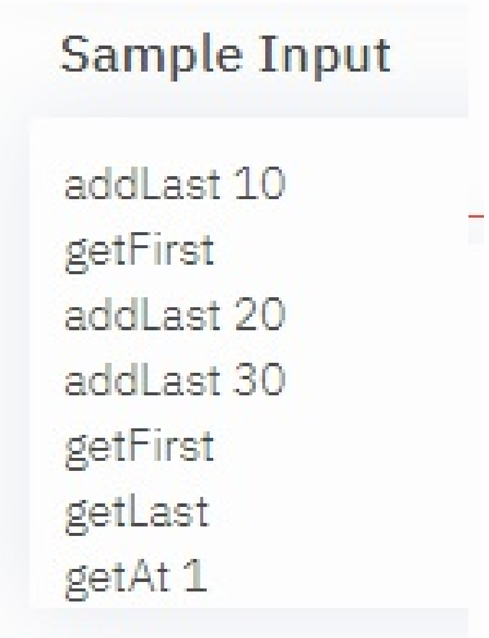


```
public void removeFirst(){
  if(size == 0){
      System.out.println("List is empty");
  } else if(size == 1){
      head = tail = null;
      size = 0;
  } else {
      head = head.next;
      size--;
  }
}
```



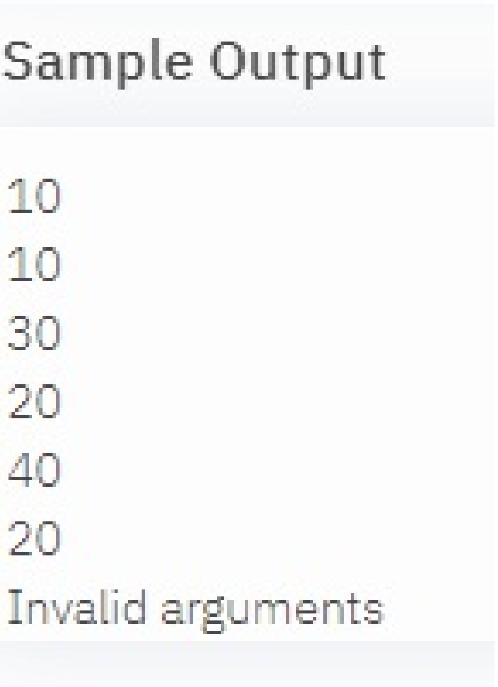
Get Value In Linked List

1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
   2.1 addLast - adds a new element with given value to the end of Linked List
   2.2. display - Prints the elements of linked list from front to end in a single line.
    All elements are separated by space.
2.3. size - Returns the number of elements in the linked list.
2.4. removeFirst - Removes the first element from Linked List.
3. You are required to complete the body of getFirst, getLast and getAt function
3.1. getFirst - Should return the data of first element. If empty should return -1 and print "List is empty".
3.2. getLast - Should return the data of last element. If empty should return -1 and print "List is empty".
3.3. getAt - Should return the data of element available at the index passed. If empty should return -1 and print "List is empty". If invalid index is passed, should return -1 and print "Invalid arguments".
4. Input and Output is managed for you.



Sample Input

```
addLast 10
getFirst
addLast 20
addLast 30
getFirst
getLast
getAt 1
```

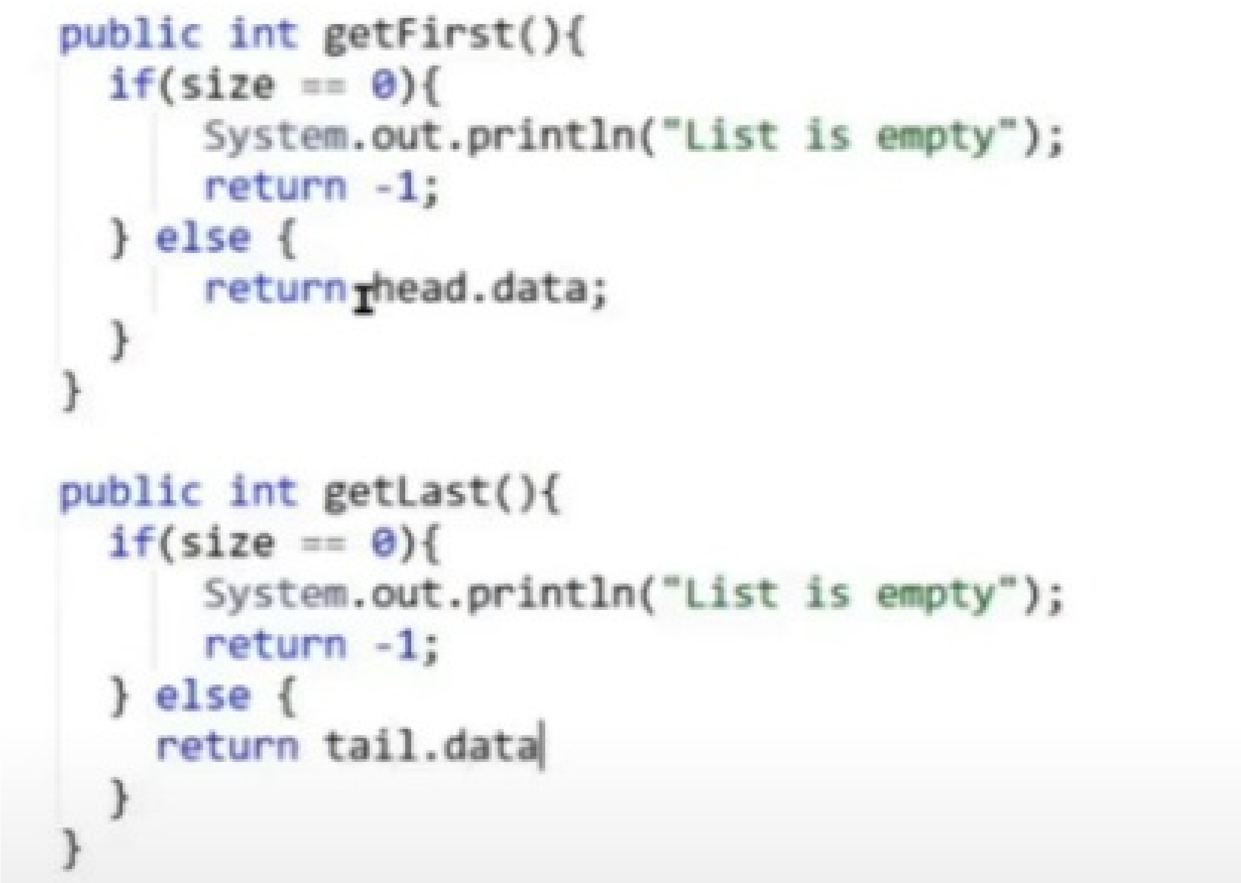


Sample Output

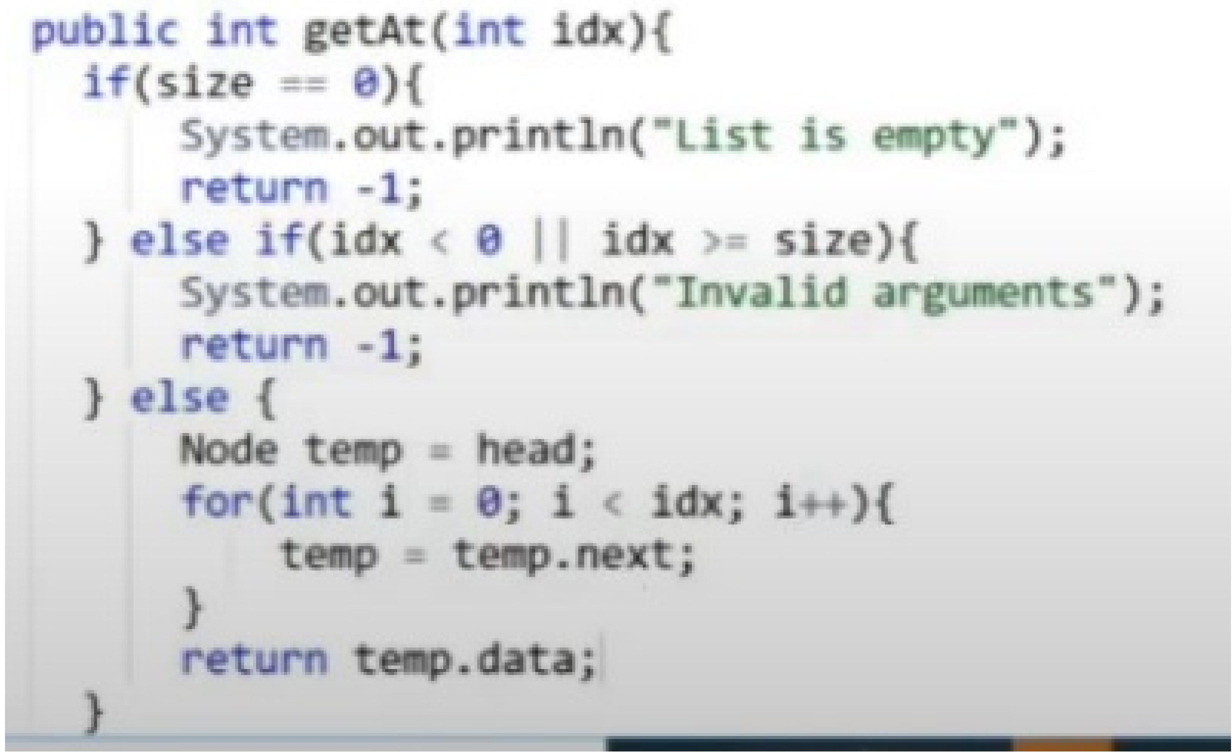
```
10
10
30
20
40
20
Invalid arguments
```

you need to write code for

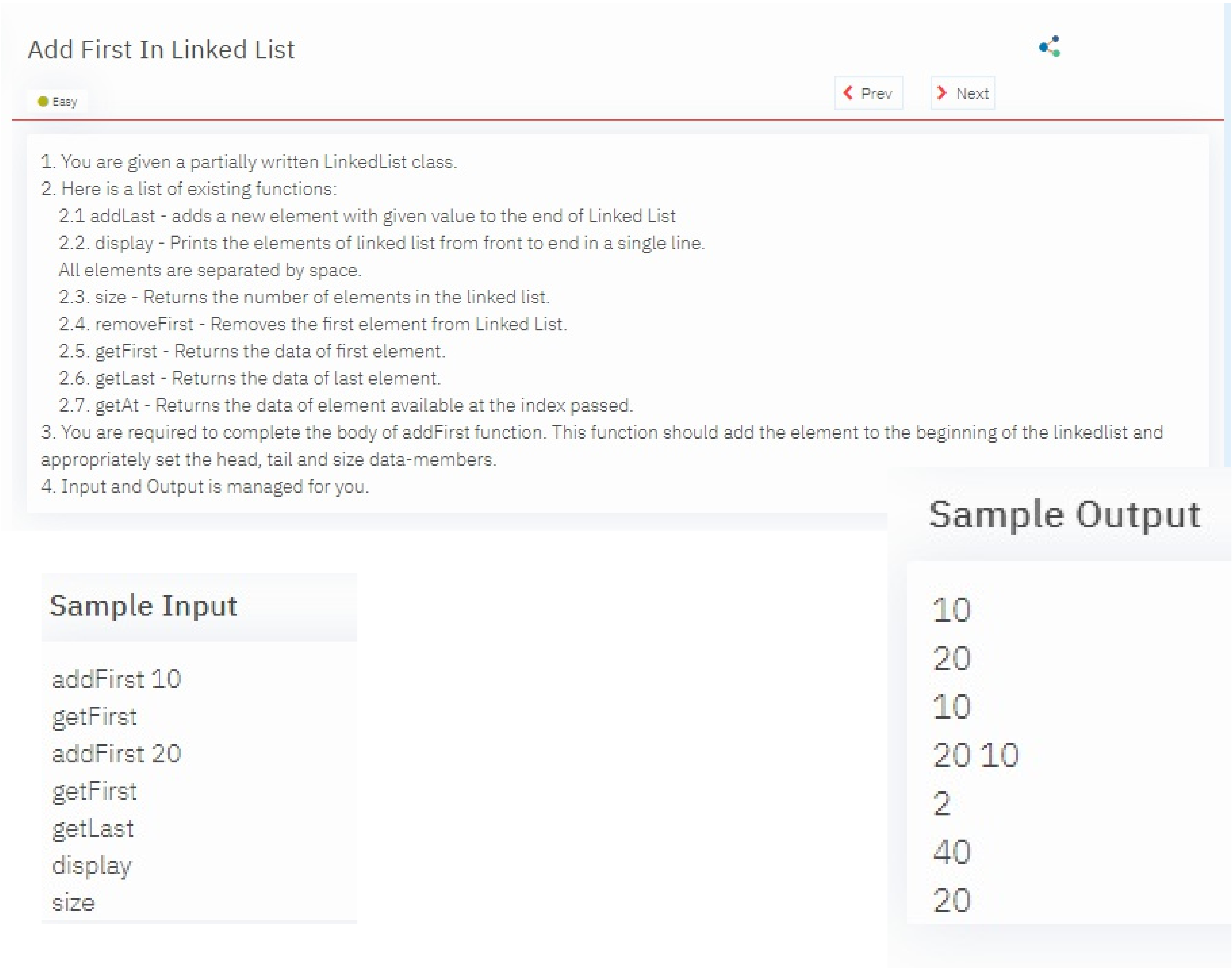
getfirst
getlast
getatindex



```
public int getFirst(){
 if(size == 0){
   System.out.println("List is empty");
   return -1;
 } else {
   return head.data;
 }
}

public int getLast(){
 if(size == 0){
   System.out.println("List is empty");
   return -1;
 } else {
  return tail.data
 }
}
```



```
public int getAt(int idx){
 if(size == 0){
   System.out.println("List is empty");
   return -1;
 } else if(idx < 0 || idx >= size){
   System.out.println("Invalid arguments");
   return -1;
 } else {
   Node temp = head;
   for(int i = 0; i < idx; i++){
     temp = temp.next;
   }
   return temp.data;
 }
```



Add First In Linked List

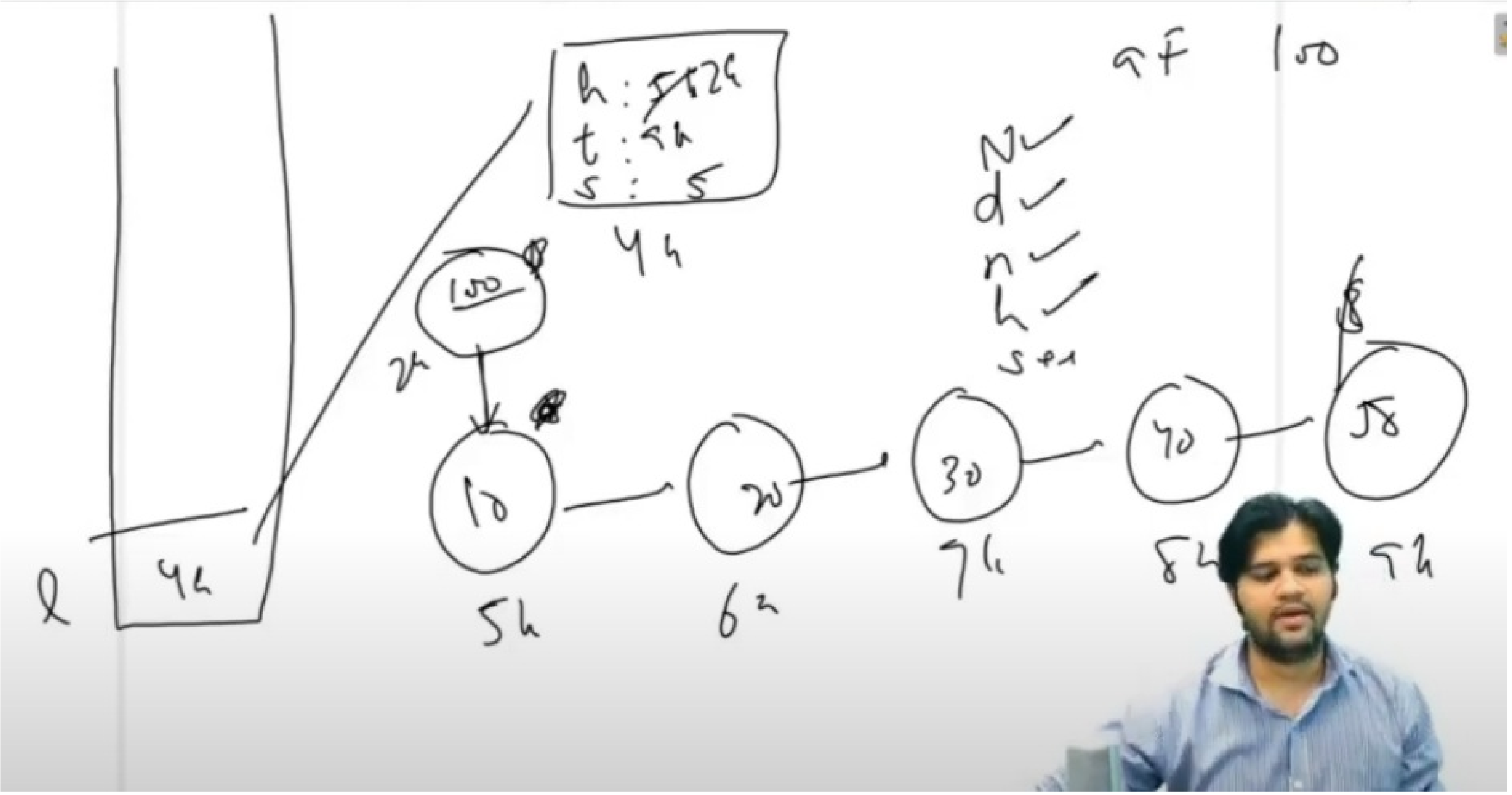
1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
   2.1 addLast - adds a new element with given value to the end of Linked List
   2.2. display - Prints the elements of linked list from front to end in a single line.
   All elements are separated by space.
   2.3. size - Returns the number of elements in the linked list.
   2.4. removeFirst - Removes the first element from Linked List.
   2.5. getFirst - Returns the data of first element.
   2.6. getLast - Returns the data of last element.
   2.7. getAt - Returns the data of element available at the index passed.
3. You are required to complete the body of addFirst function. This function should add the element to the beginning of the linkedlist and appropriately set the head, tail and size data-members.
4. Input and Output is managed for you.

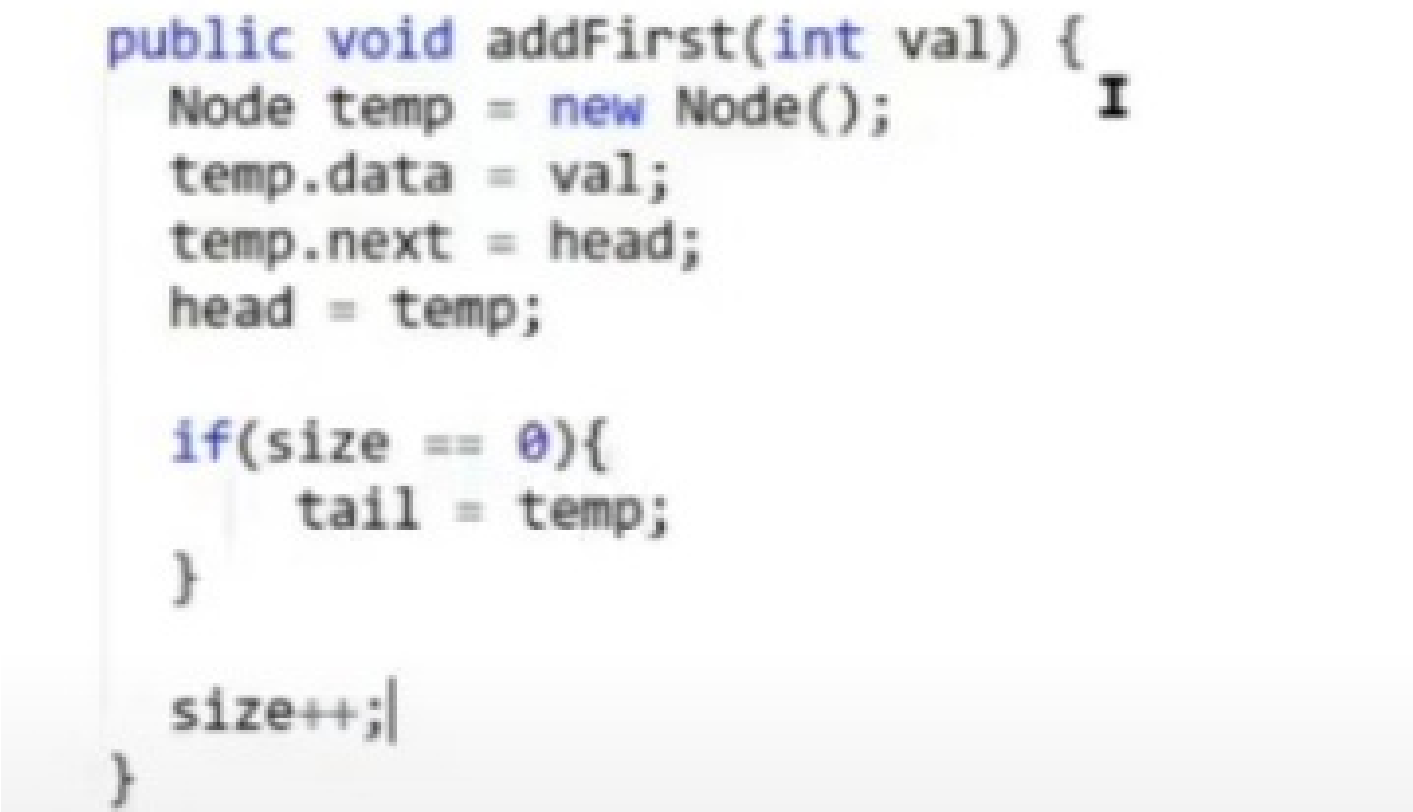
Sample Input

```
addFirst 10
getFirst
addFirst 20
getFirst
getLast
display
size
```

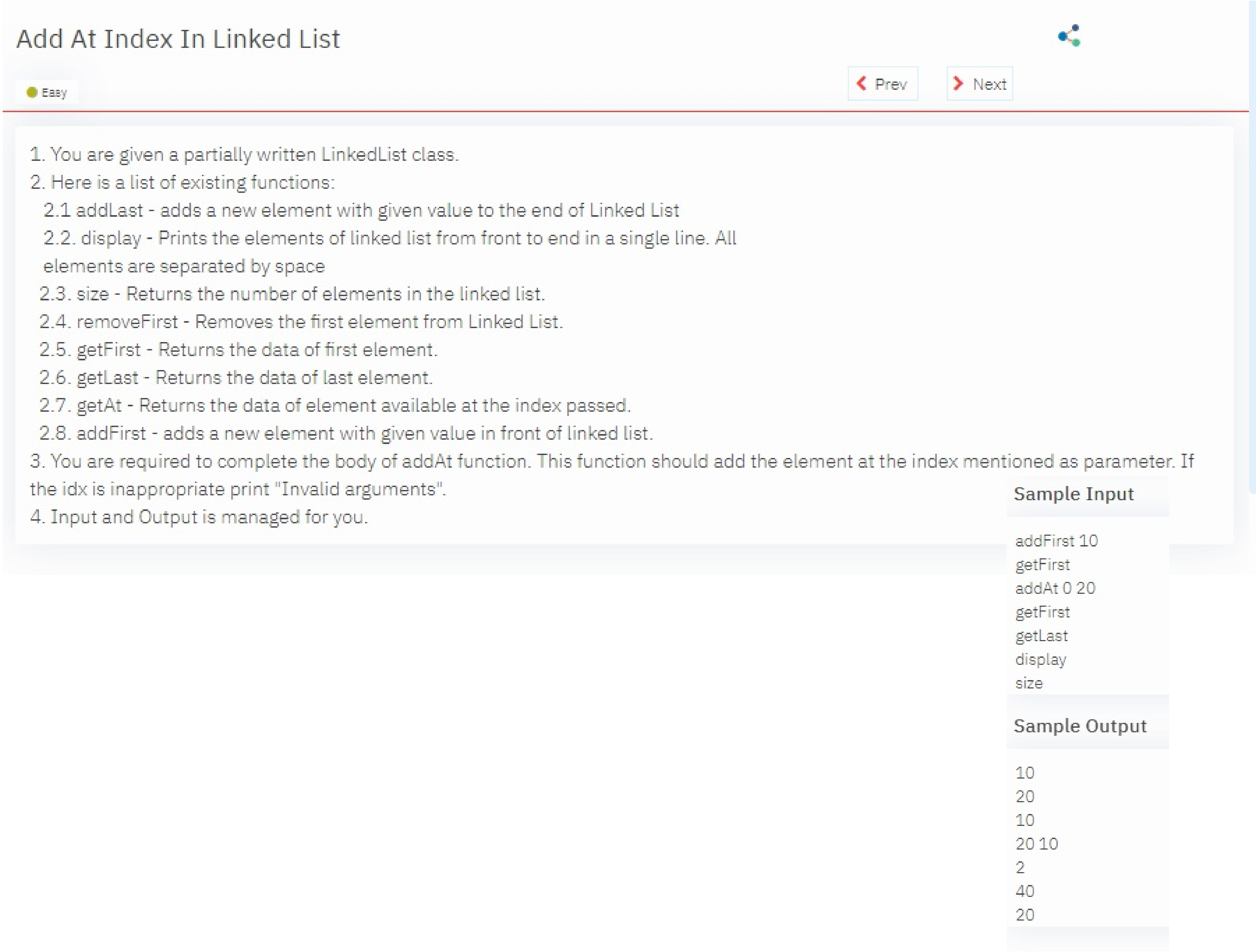
Sample Output

```
10
20
10
20 10
2
40
20
```





```
public void addFirst(int val) {
  Node temp = new Node();
  temp.data = val;
  temp.next = head;
  head = temp;

  if(size == 0){
      tail = temp;
  }

  size++;
}
```



Add At Index In Linked List

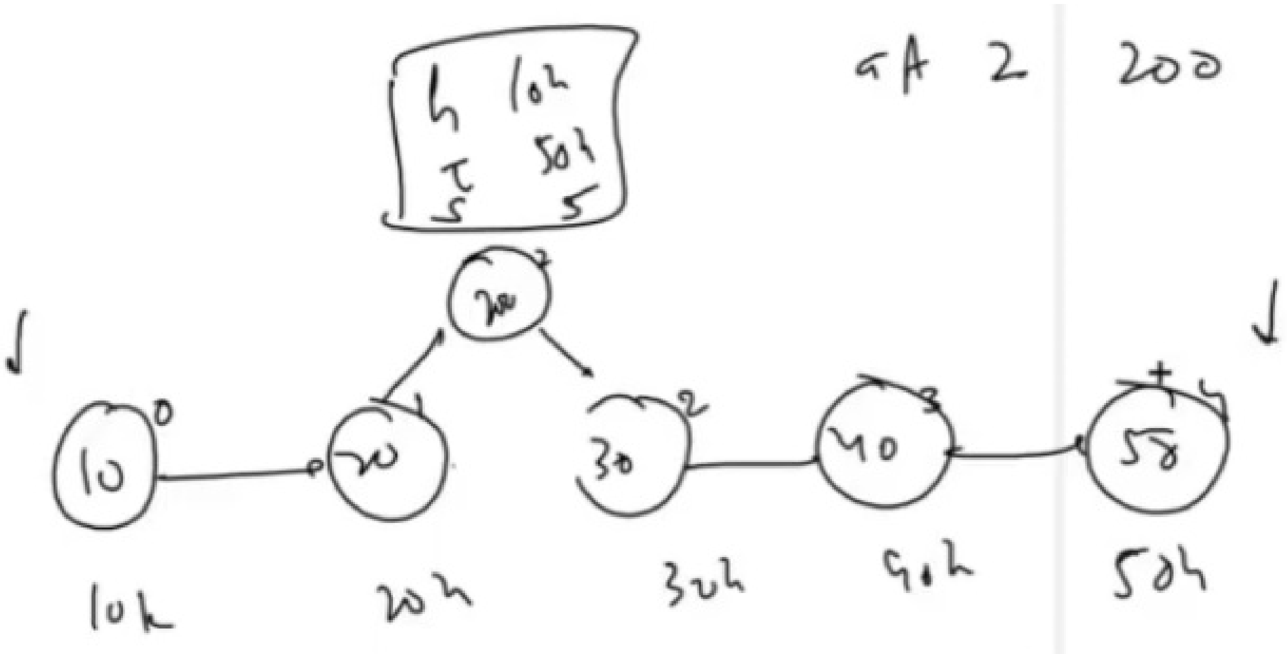
1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
   2.1 addLast - adds a new element with given value to the end of Linked List
   2.2. display - Prints the elements of linked list from front to end in a single line. All elements are separated by space
   2.3. size - Returns the number of elements in the linked list.
   2.4. removeFirst - Removes the first element from Linked List.
   2.5. getFirst - Returns the data of first element.
   2.6. getLast - Returns the data of last element.
   2.7. getAt - Returns the data of element available at the index passed.
   2.8. addFirst - adds a new element with given value in front of linked list.
3. You are required to complete the body of addAt function. This function should add the element at the index mentioned as parameter. If the idx is inappropriate print "Invalid arguments".
4. Input and Output is managed for you.

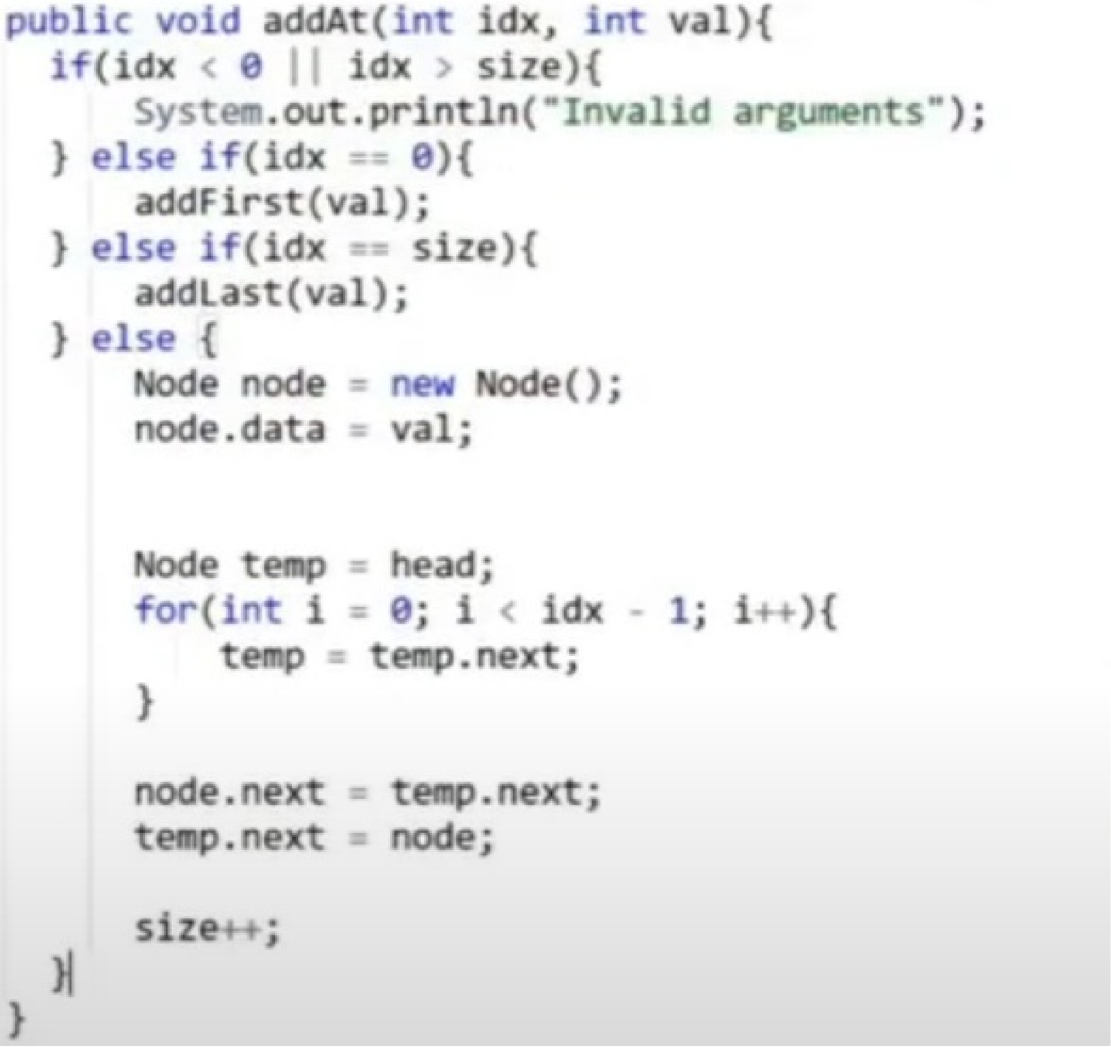
Sample Input

```
addFirst 10
getFirst
addAt 0 20
getFirst
getLast
display
size
```

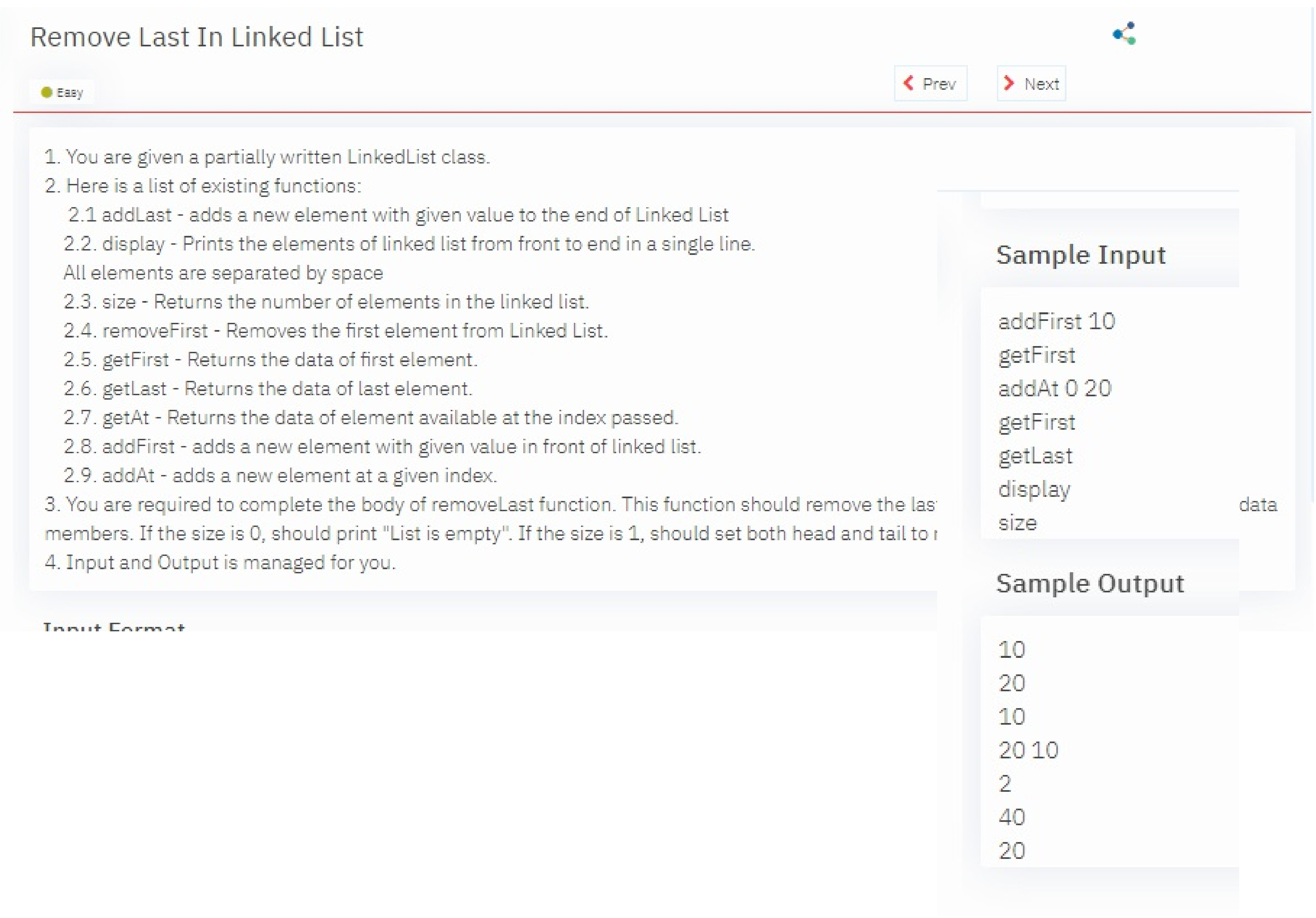
Sample Output

```
10
20
10
20 10
2
40
20
```





```
public void addAt(int idx, int val){
  if(idx < 0 || idx > size){
      System.out.println("Invalid arguments");
  } else if(idx == 0){
      addFirst(val);
  } else if(idx == size){
      addLast(val);
  } else {
      Node node = new Node();
      node.data = val;


      Node temp = head;
      for(int i = 0; i < idx - 1; i++){
          temp = temp.next;
      }

      node.next = temp.next;
      temp.next = node;

      size++;
  }
}
```



Remove Last In Linked List

1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
   2.1 addLast - adds a new element with given value to the end of Linked List
   2.2. display - Prints the elements of linked list from front to end in a single line.
   All elements are separated by space
   2.3. size - Returns the number of elements in the linked list.
   2.4. removeFirst - Removes the first element from Linked List.
   2.5. getFirst - Returns the data of first element.
   2.6. getLast - Returns the data of last element.
   2.7. getAt - Returns the data of element available at the index passed.
   2.8. addFirst - adds a new element with given value in front of linked list.
   2.9. addAt - adds a new element at a given index.
3. You are required to complete the body of removeLast function. This function should remove the las data members. If the size is 0, should print "List is empty". If the size is 1, should set both head and tail to r
4. Input and Output is managed for you.

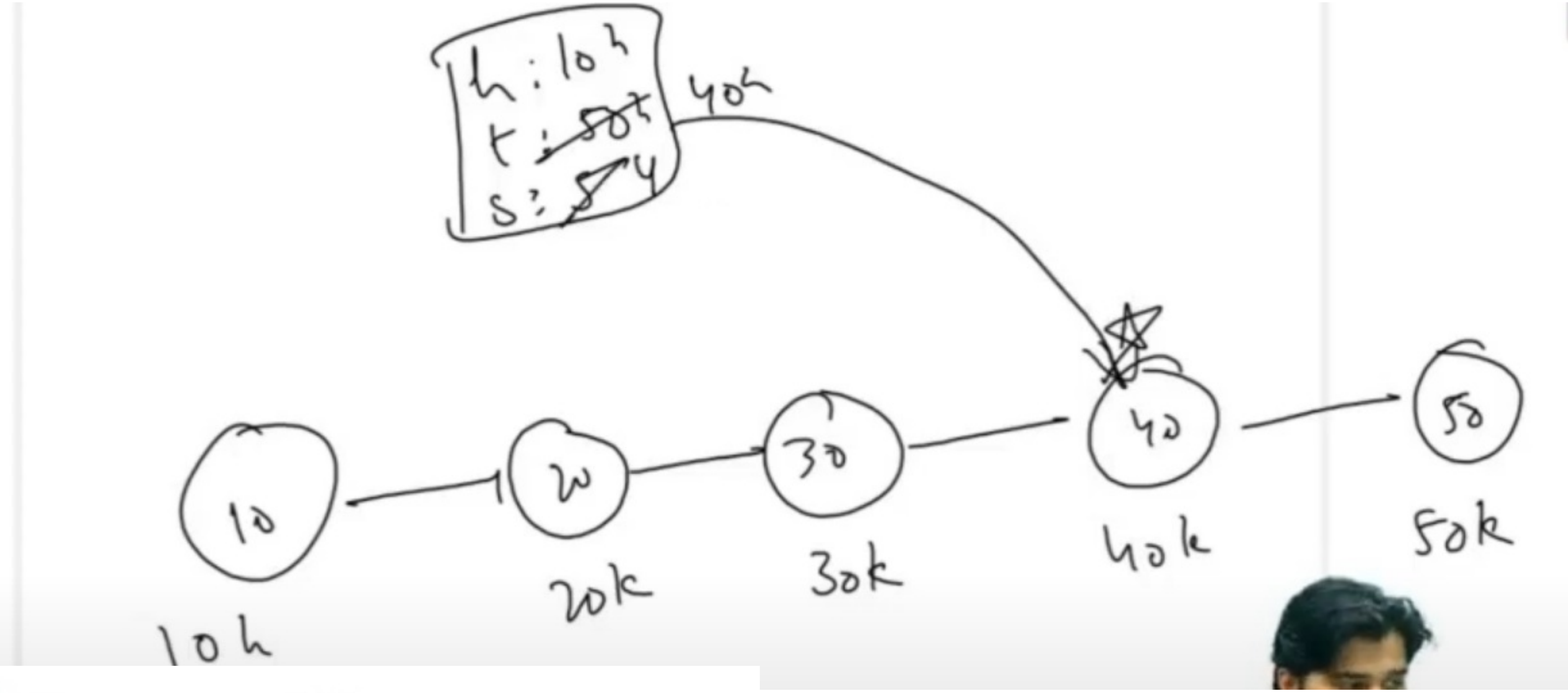
Sample Input

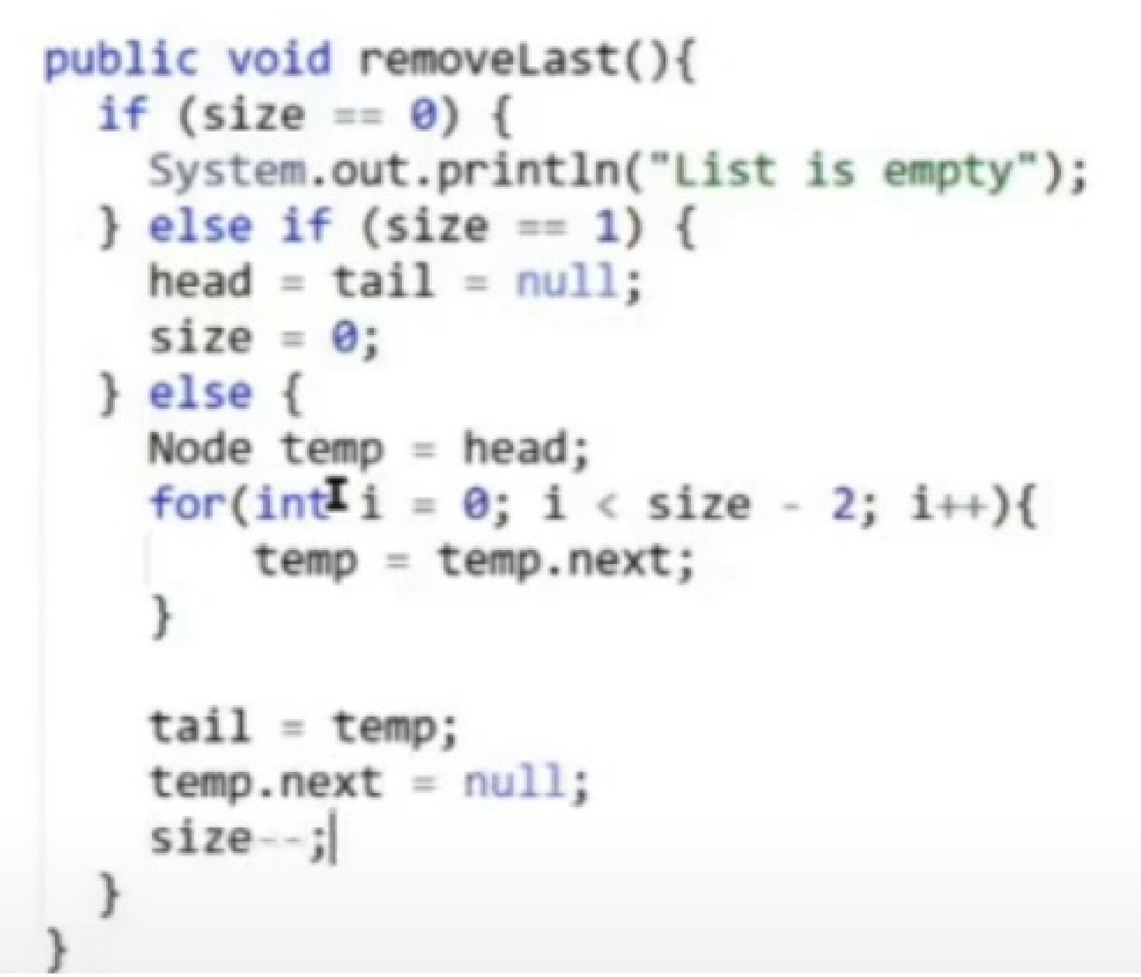
```
addFirst 10
getFirst
addAt 0 20
getFirst
getLast
display
size
```

Sample Output

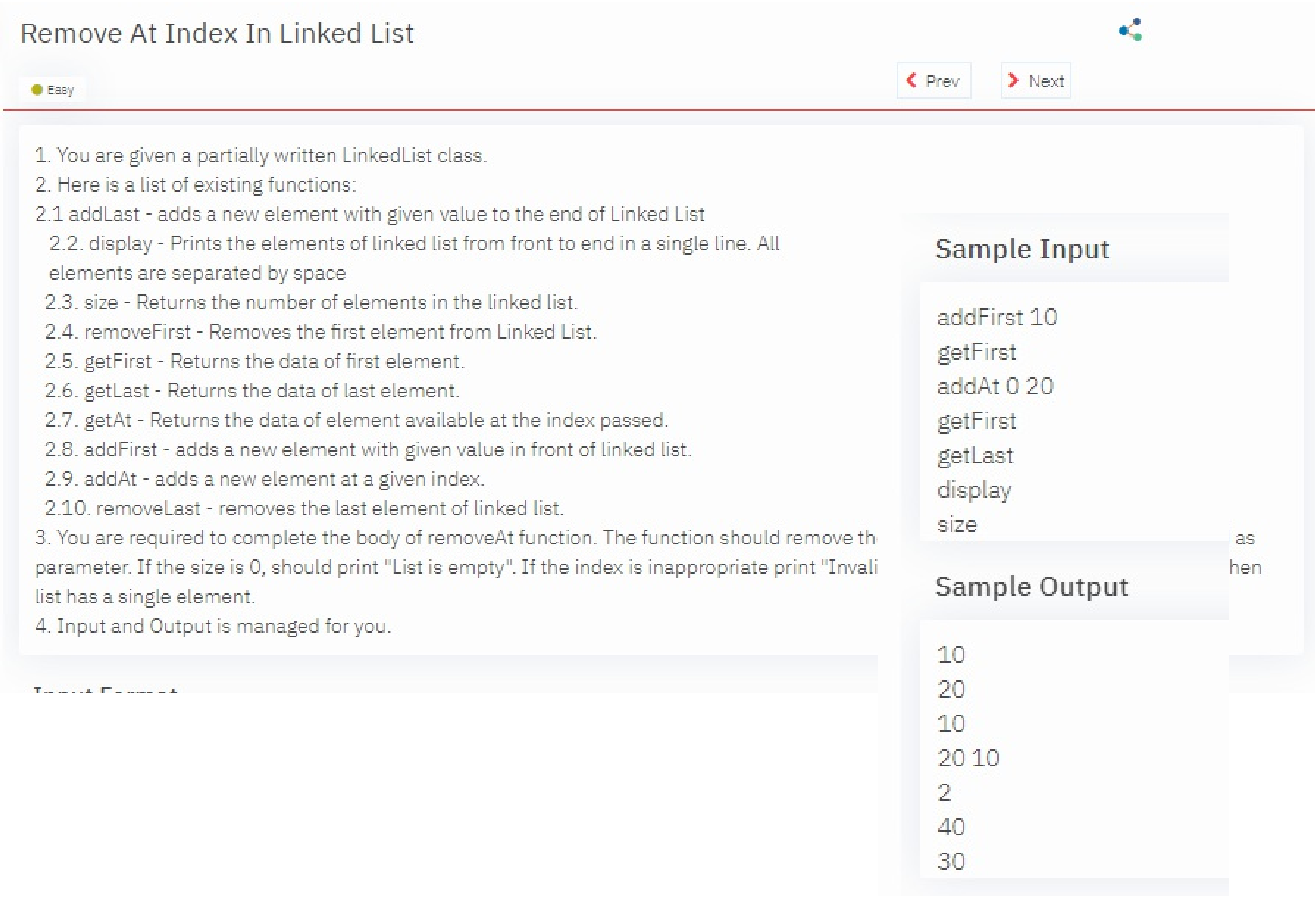
```
10
20
10
20 10
2
40
20
```

Input Format





```
public void removeLast(){
  if (size == 0) {
    System.out.println("List is empty");
  } else if (size == 1) {
    head = tail = null;
    size = 0;
  } else {
    Node temp = head;
    for(int i = 0; i < size - 2; i++){
        temp = temp.next;
    }

    tail = temp;
    temp.next = null;
    size--;
  }
}
```



Remove At Index In Linked List

1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
2.1 addLast - adds a new element with given value to the end of Linked List
2.2. display - Prints the elements of linked list from front to end in a single line. All elements are separated by space
2.3. size - Returns the number of elements in the linked list.
2.4. removeFirst - Removes the first element from Linked List.
2.5. getFirst - Returns the data of first element.
2.6. getLast - Returns the data of last element.
2.7. getAt - Returns the data of element available at the index passed.
2.8. addFirst - adds a new element with given value in front of linked list.
2.9. addAt - adds a new element at a given index.
2.10. removeLast - removes the last element of linked list.
3. You are required to complete the body of removeAt function. The function should remove th as
parameter. If the size is 0, should print "List is empty". If the index is inappropriate print "Invali hen
list has a single element.
4. Input and Output is managed for you.

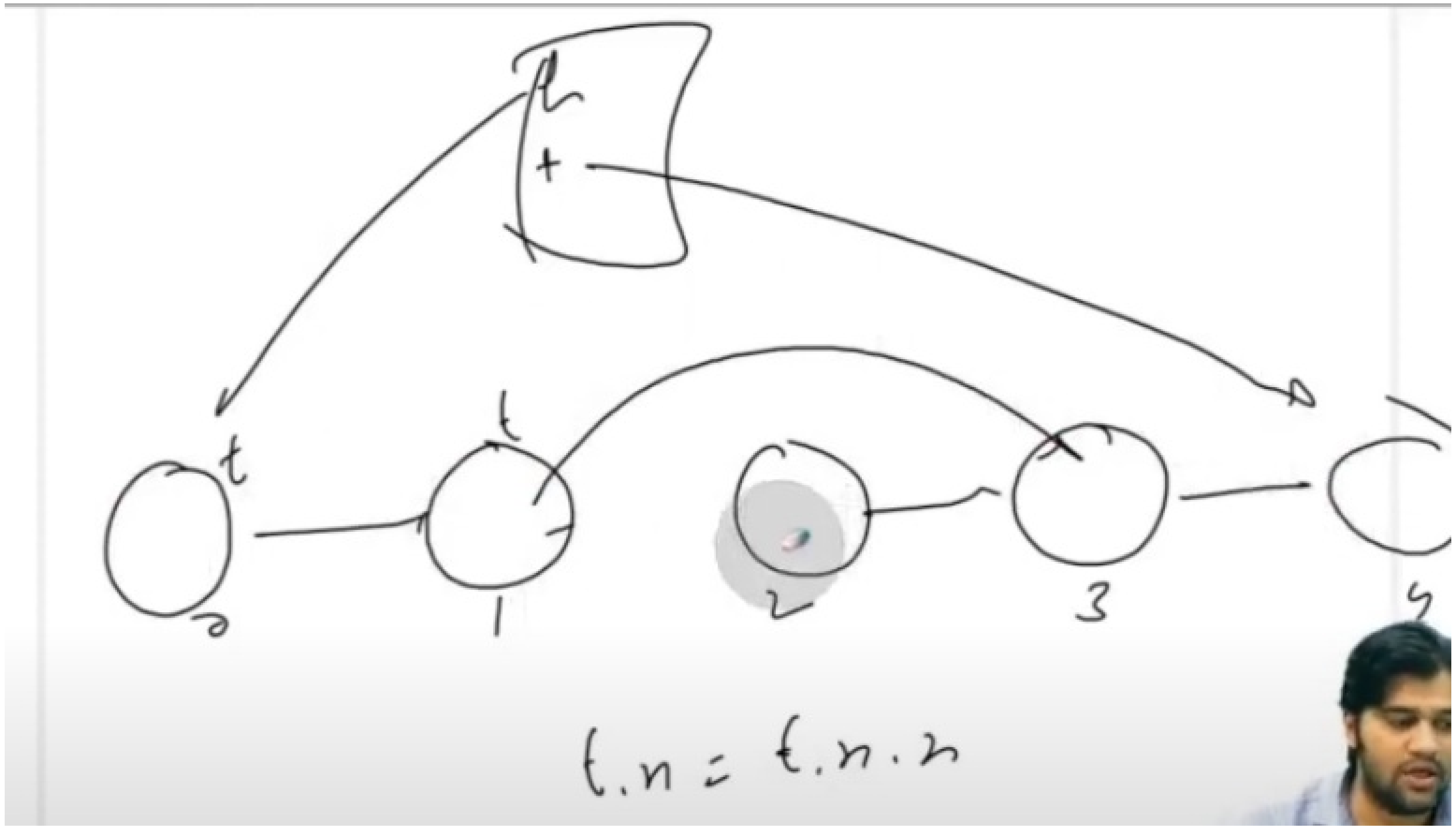
Sample Input

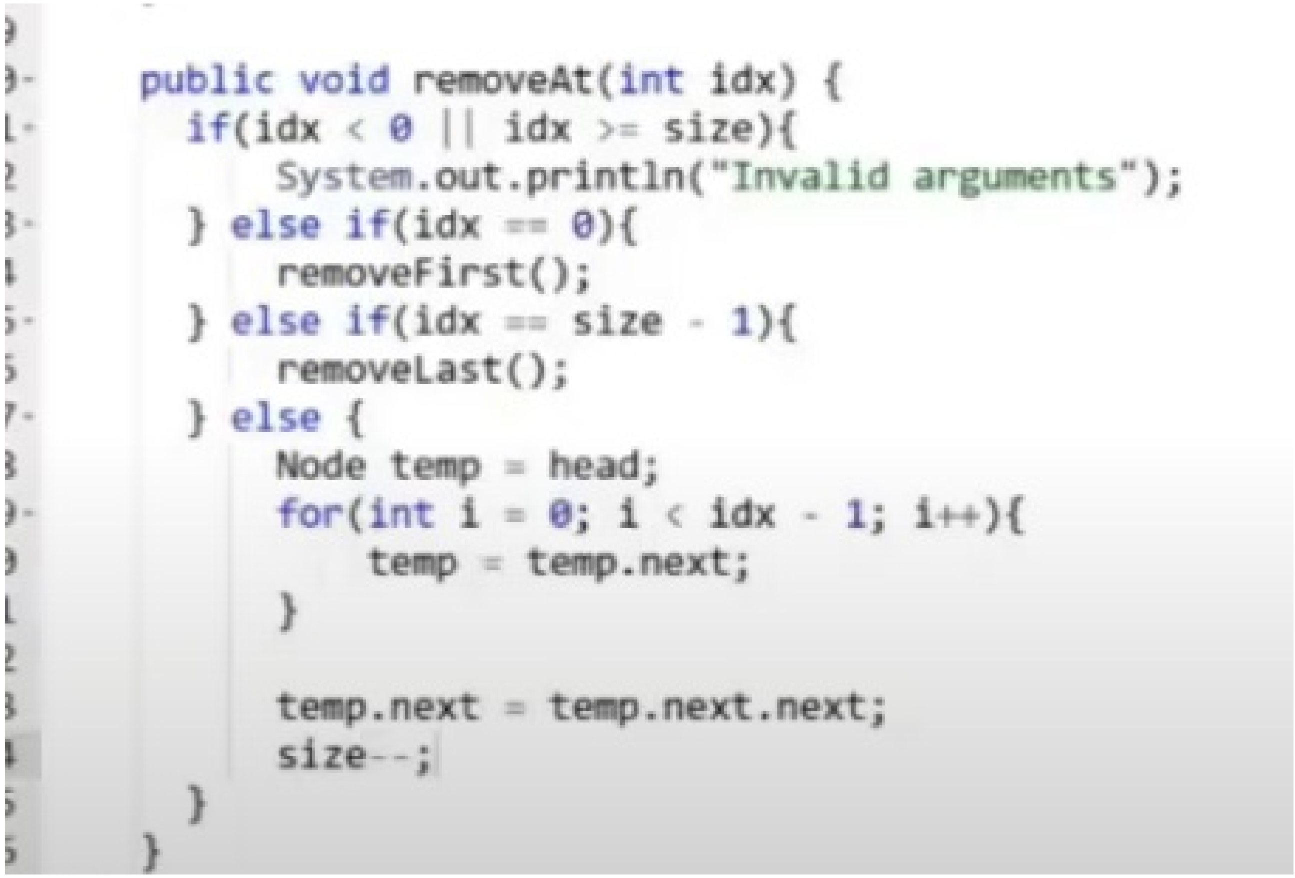
```
addFirst 10
getFirst
addAt 0 20
getFirst
getLast
display
size
```

Sample Output

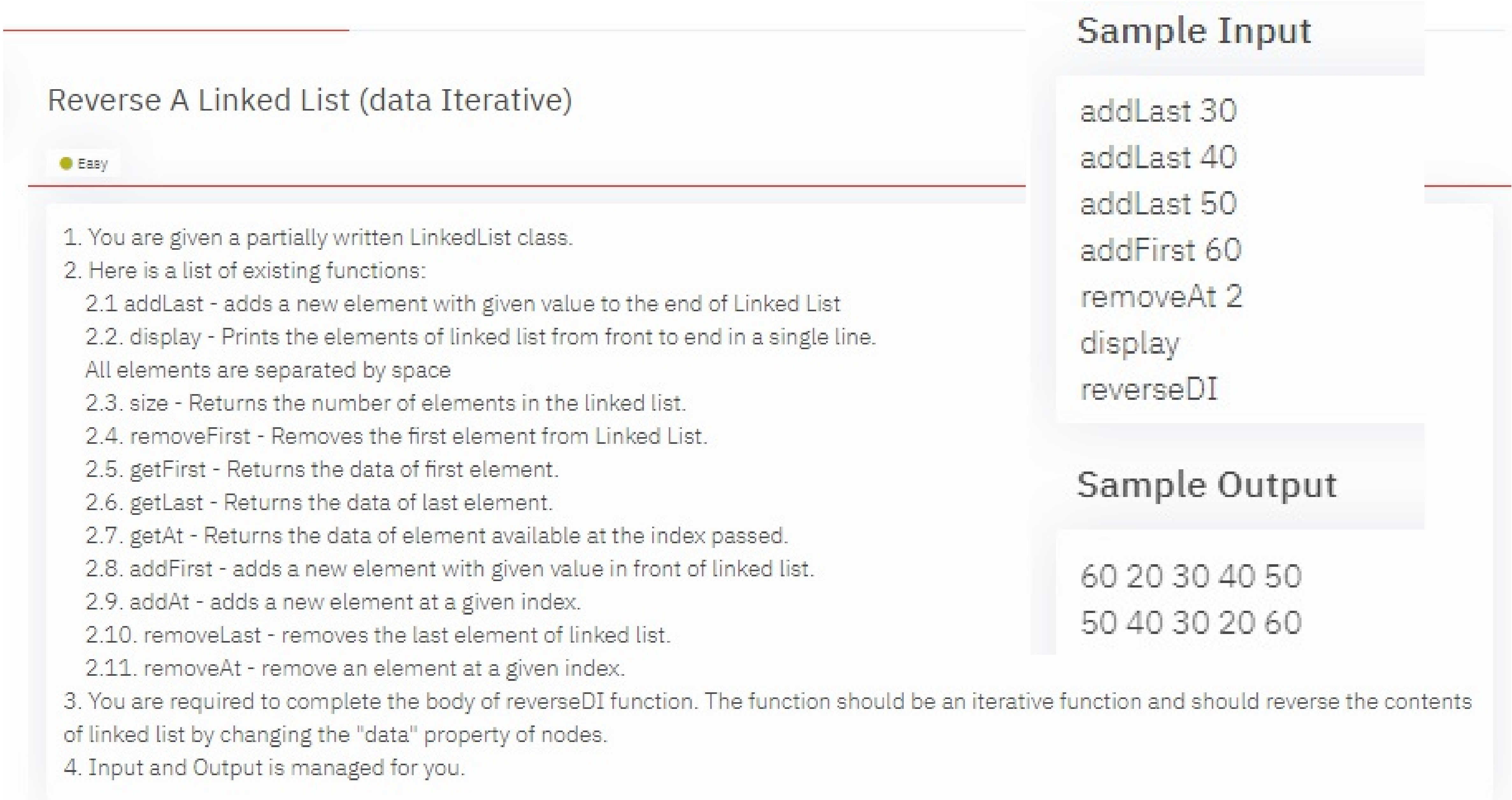
```
10
20
10
20 10
2
40
30
```

Input Format





```
    public void removeAt(int idx) {
      if(idx < 0 || idx >= size){
          System.out.println("Invalid arguments");
      } else if(idx == 0){
          removeFirst();
      } else if(idx == size - 1){
          removeLast();
      } else {
          Node temp = head;
          for(int i = 0; i < idx - 1; i++){
              temp = temp.next;
          }

          temp.next = temp.next.next;
          size--;
      }
    }
```



Reverse A Linked List (data Iterative)

Easy

1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
   2.1 addLast - adds a new element with given value to the end of Linked List
   2.2. display - Prints the elements of linked list from front to end in a single line.
   All elements are separated by space
   2.3. size - Returns the number of elements in the linked list.
   2.4. removeFirst - Removes the first element from Linked List.
   2.5. getFirst - Returns the data of first element.
   2.6. getLast - Returns the data of last element.
   2.7. getAt - Returns the data of element available at the index passed.
   2.8. addFirst - adds a new element with given value in front of linked list.
   2.9. addAt - adds a new element at a given index.
   2.10. removeLast - removes the last element of linked list.
   2.11. removeAt - remove an element at a given index.
3. You are required to complete the body of reverseDI function. The function should be an iterative function and should reverse the contents of linked list by changing the "data" property of nodes.
4. Input and Output is managed for you.

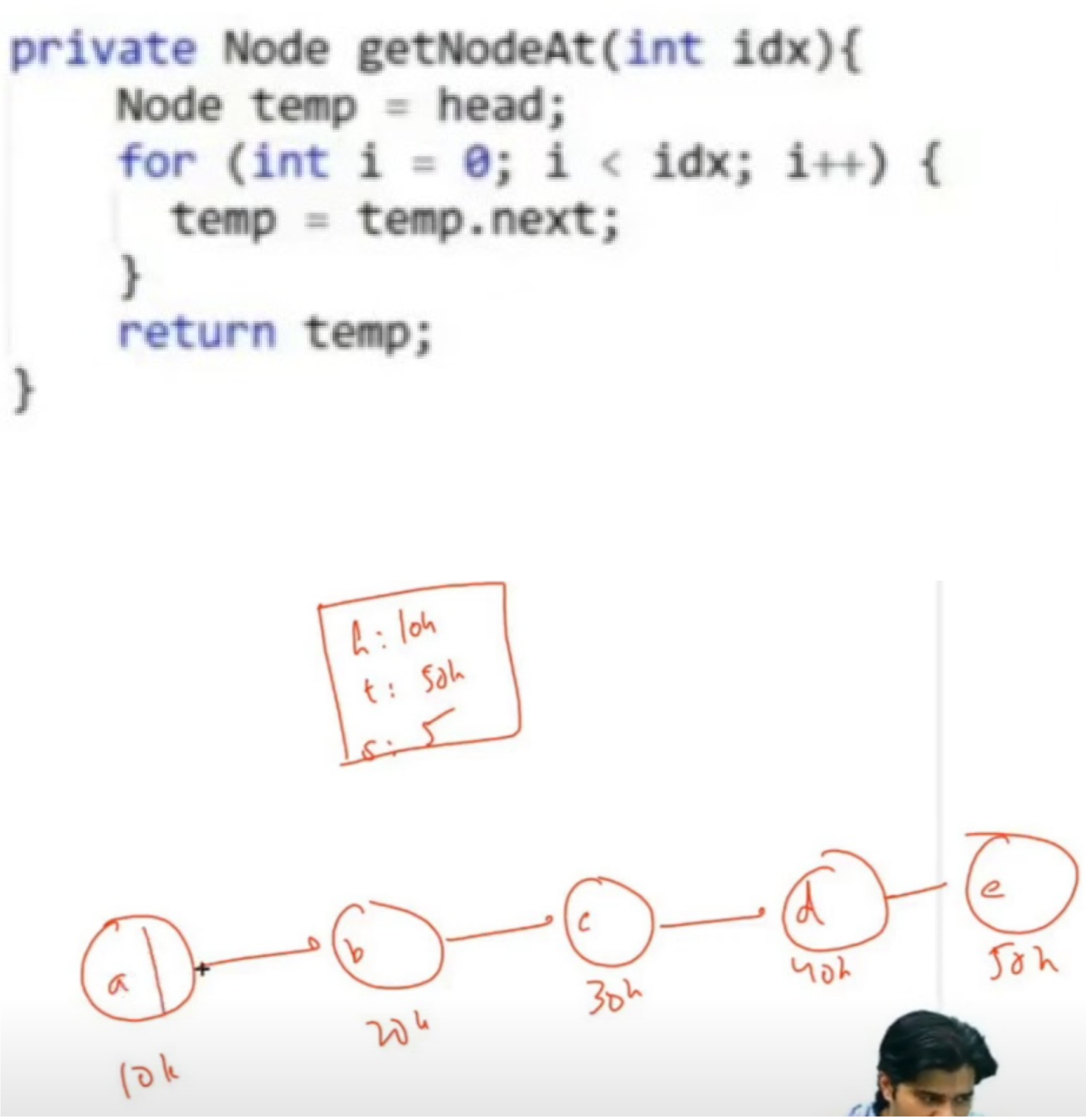
Sample Input

```
addLast 30
addLast 40
addLast 50
addFirst 60
removeAt 2
display
reverseDI
```

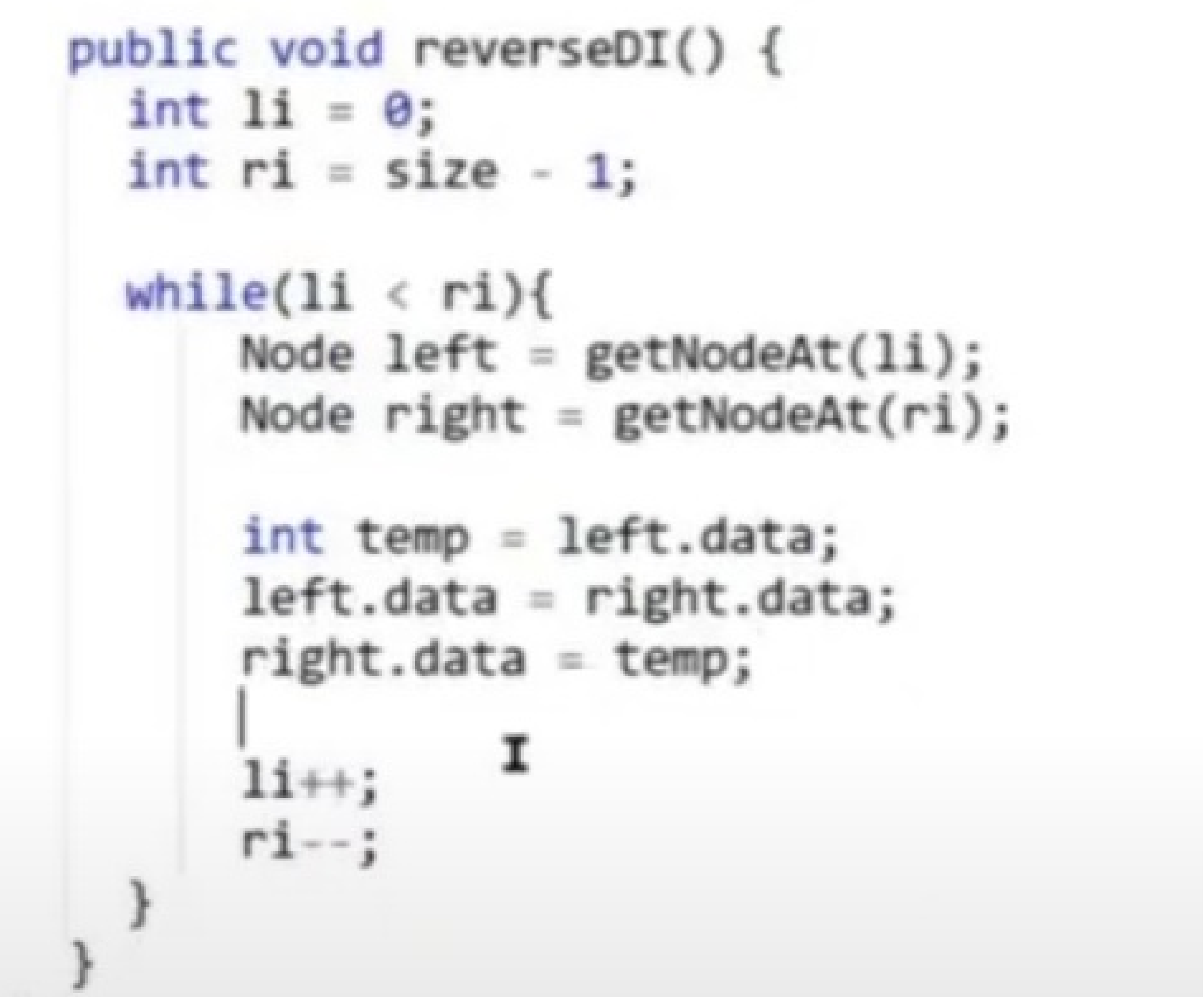
Sample Output

```
60 20 30 40 50
50 40 30 20 60
```

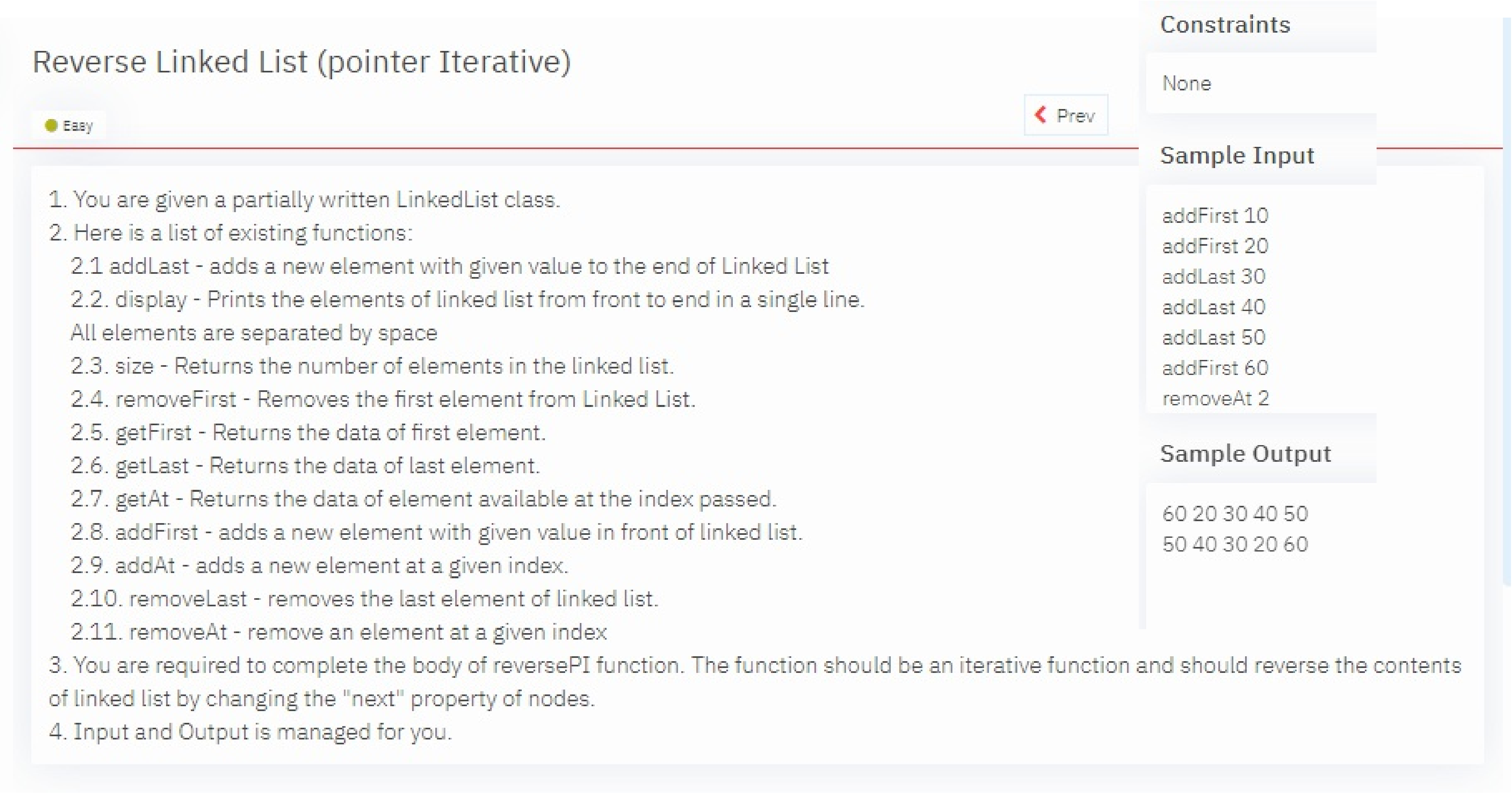
you can touch data property only.
complexity can be o(n^2)



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



```
public void reverseDI() {
  int li = 0;
  int ri = size - 1;

  while(li < ri){
     Node left = getNodeAt(li);
     Node right = getNodeAt(ri);

     int temp = left.data;
     left.data = right.data;
     right.data = temp;

     li++;
     ri--;
  }
}
```



Reverse Linked List (pointer Iterative)

● Easy

< Prev

1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
   2.1 addLast - adds a new element with given value to the end of Linked List
   2.2. display - Prints the elements of linked list from front to end in a single line.
   All elements are separated by space
   2.3. size - Returns the number of elements in the linked list.
   2.4. removeFirst - Removes the first element from Linked List.
   2.5. getFirst - Returns the data of first element.
   2.6. getLast - Returns the data of last element.
   2.7. getAt - Returns the data of element available at the index passed.
   2.8. addFirst - adds a new element with given value in front of linked list.
   2.9. addAt - adds a new element at a given index.
   2.10. removeLast - removes the last element of linked list.
   2.11. removeAt - remove an element at a given index
3. You are required to complete the body of reversePI function. The function should be an iterative function and should reverse the contents of linked list by changing the "next" property of nodes.
4. Input and Output is managed for you.

Constraints

None

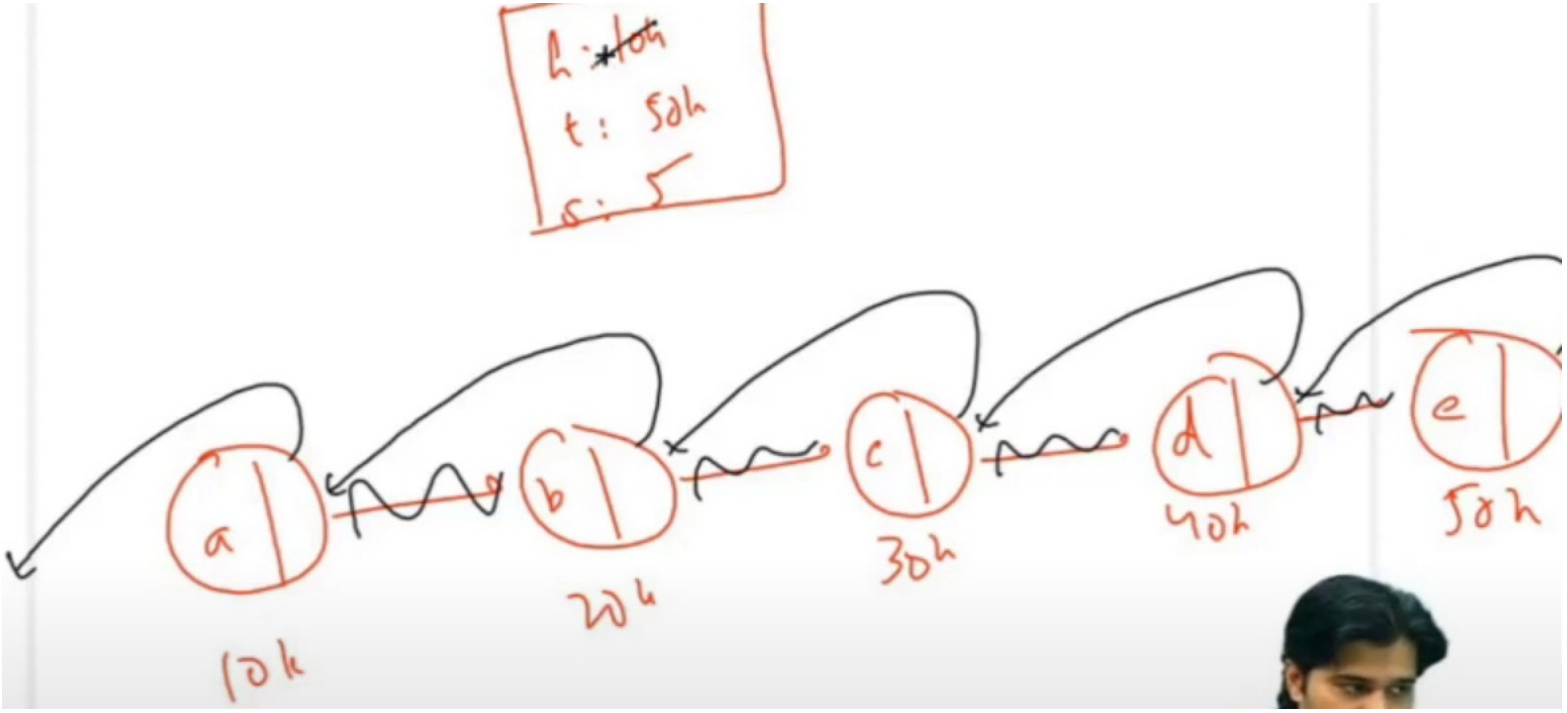
Sample Input

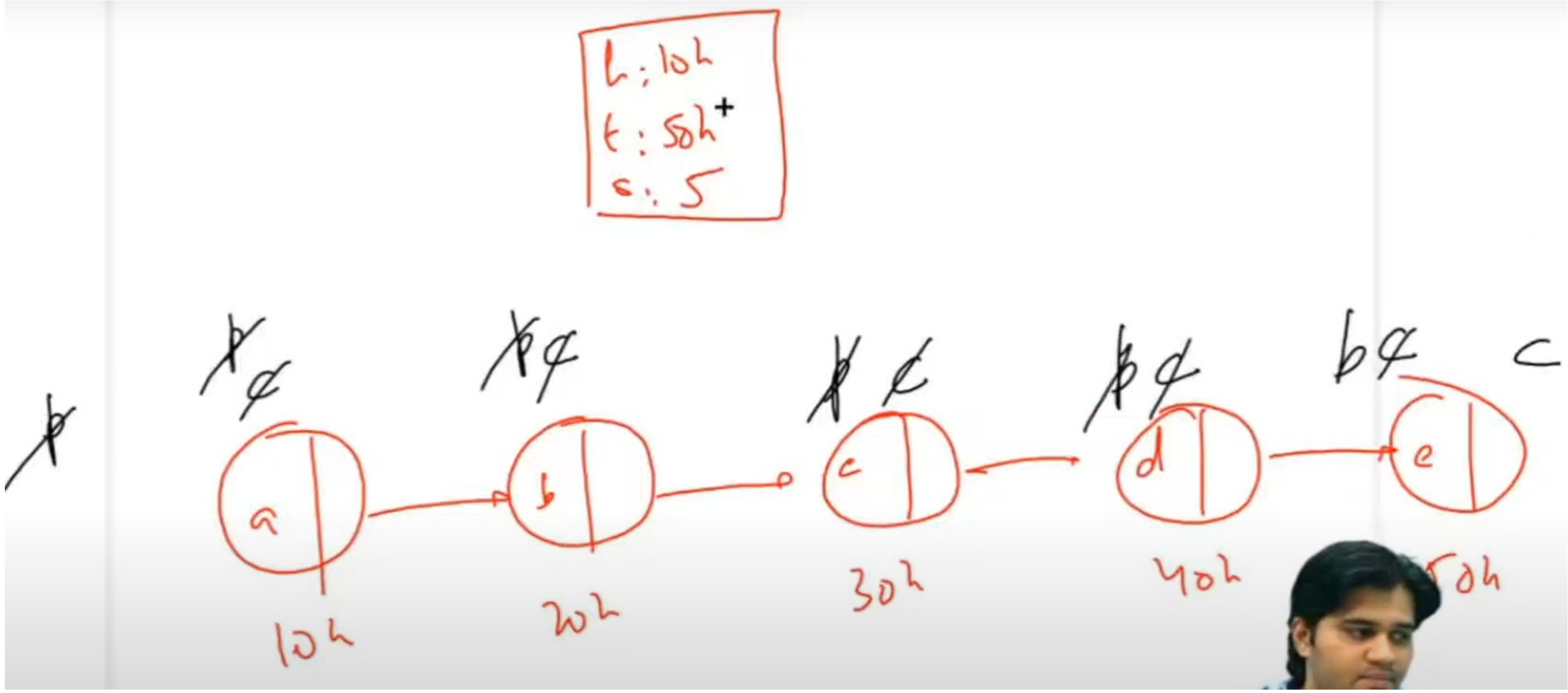
```
addFirst 10
addFirst 20
addLast 30
addLast 40
addLast 50
addFirst 60
removeAt 2
```

Sample Output

```
60 20 30 40 50
50 40 30 20 60
```

using pointer you need to reverse

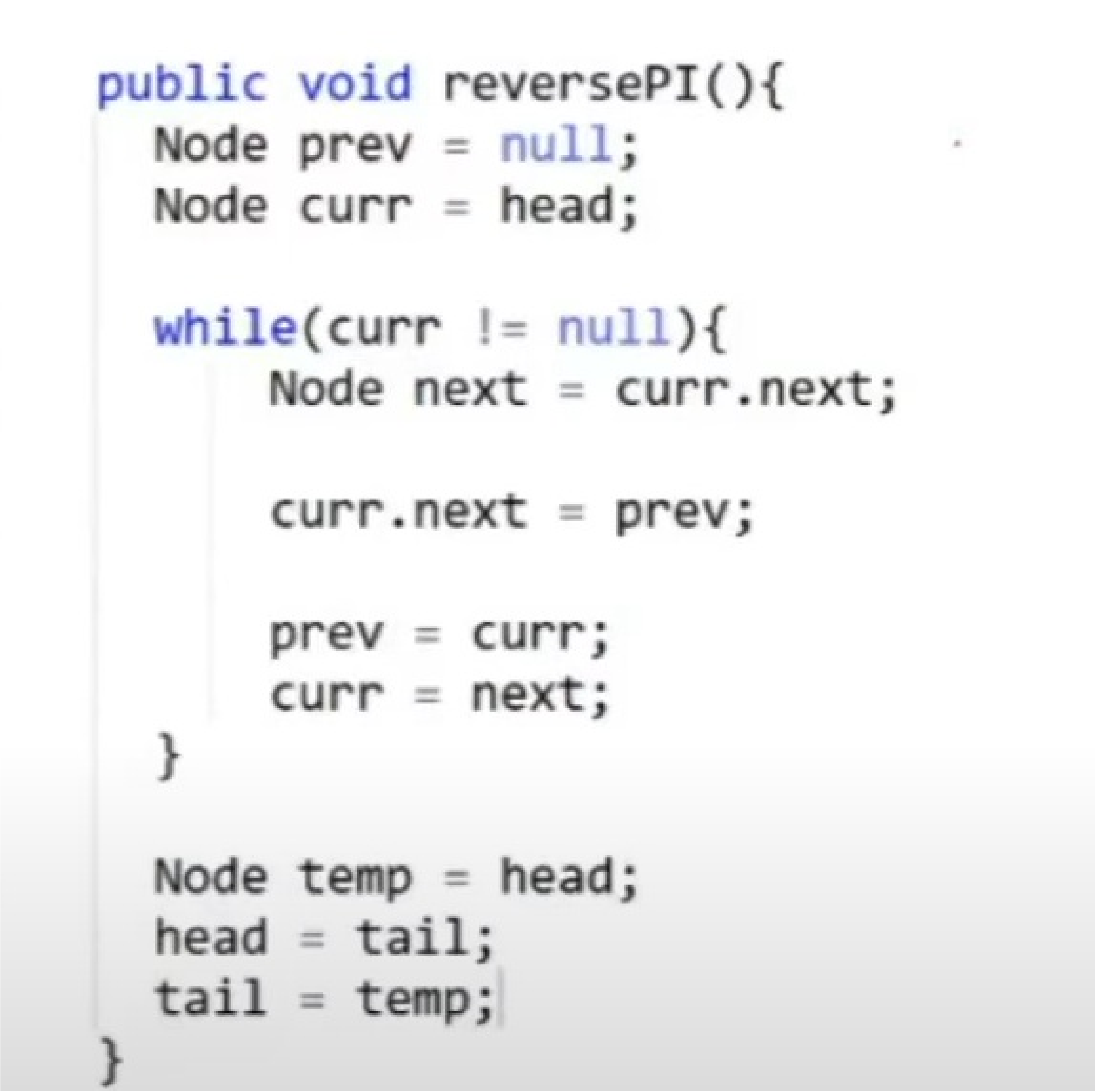




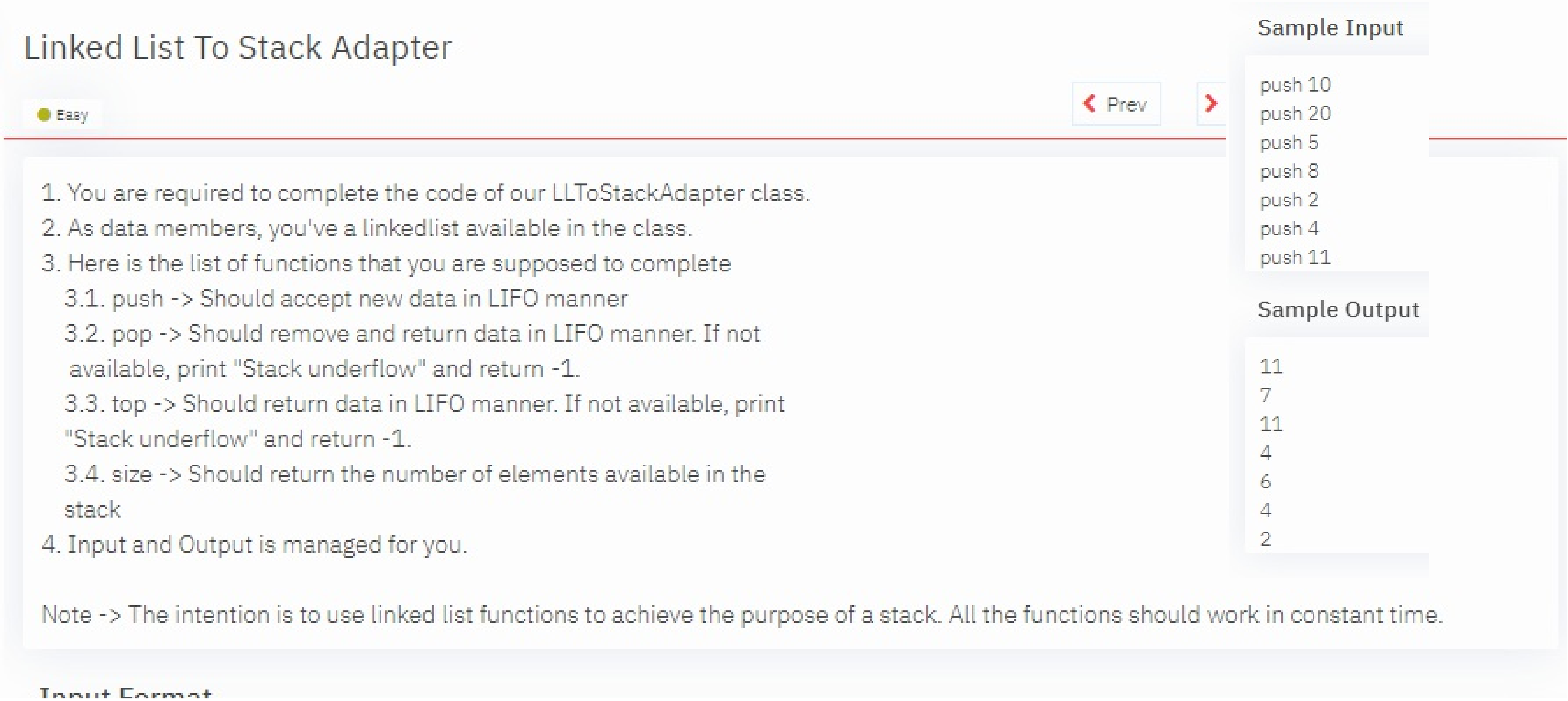
we will use 2 pointer from left to right

prev pointer =null
curr pointer= head

then at the end swap head and tail



```java
public void reversePI(){
  Node prev = null;
  Node curr = head;

  while(curr != null){
      Node next = curr.next;

      curr.next = prev;

      prev = curr;
      curr = next;
  }

  Node temp = head;
  head = tail;
  tail = temp;
}
```



Linked List To Stack Adapter

1. You are required to complete the code of our LLToStackAdapter class.
2. As data members, you've a linkedlist available in the class.
3. Here is the list of functions that you are supposed to complete
   3.1. push -> Should accept new data in LIFO manner
   3.2. pop -> Should remove and return data in LIFO manner. If not available, print "Stack underflow" and return -1.
   3.3. top -> Should return data in LIFO manner. If not available, print "Stack underflow" and return -1.
   3.4. size -> Should return the number of elements available in the stack
4. Input and Output is managed for you.

Note -> The intention is to use linked list functions to achieve the purpose of a stack. All the functions should work in constant time.

Sample Input

```
push 10
push 20
push 5
push 8
push 2
push 4
push 11
```

Sample Output

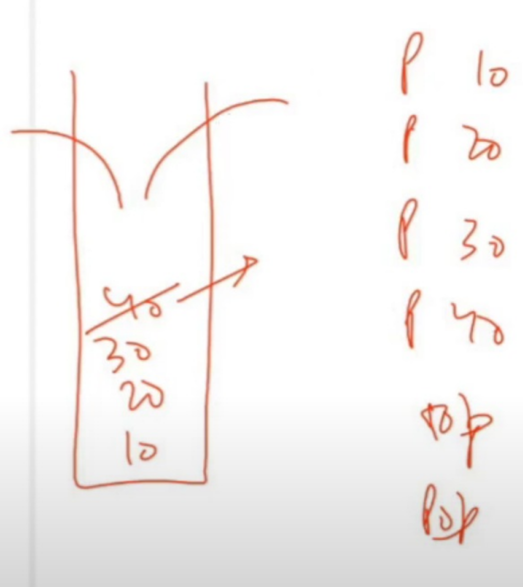
```
11
7
11
4
6
4
2
```

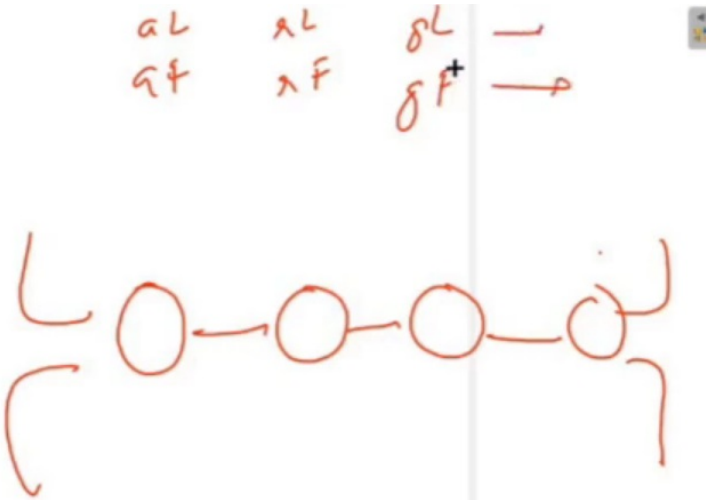
Input Format

IN stack

we need
behaviour of
stack

In Linklist





add last
add first

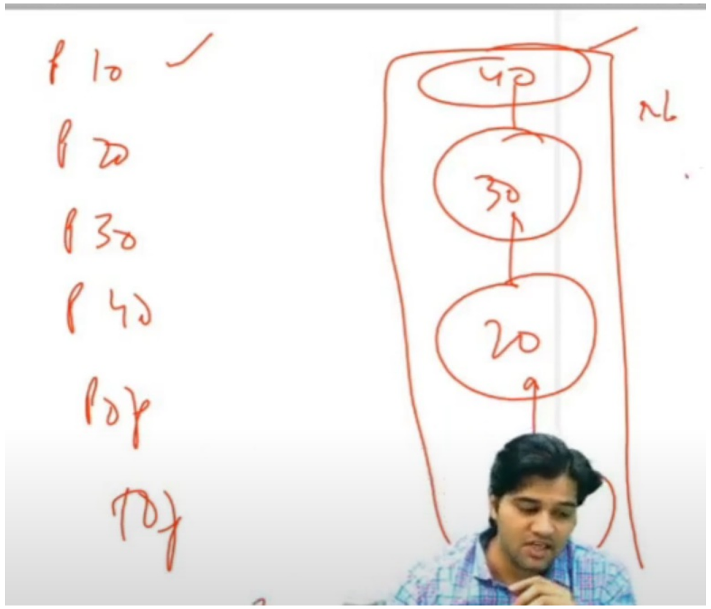
remove last
remove first

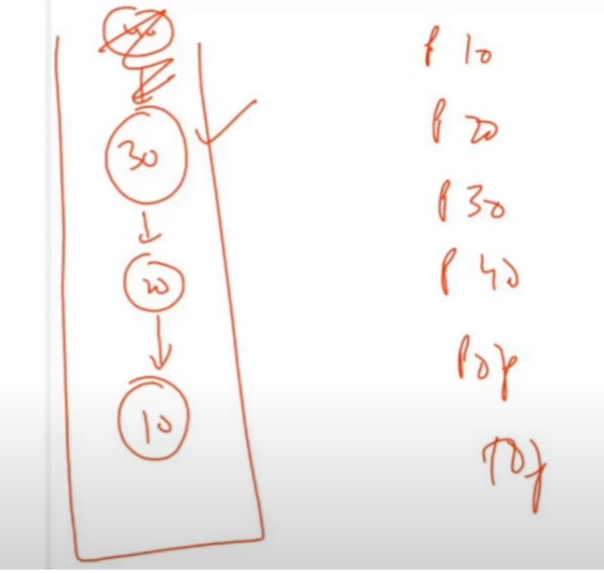
getlast
getfirst

other approach
is al rl gl

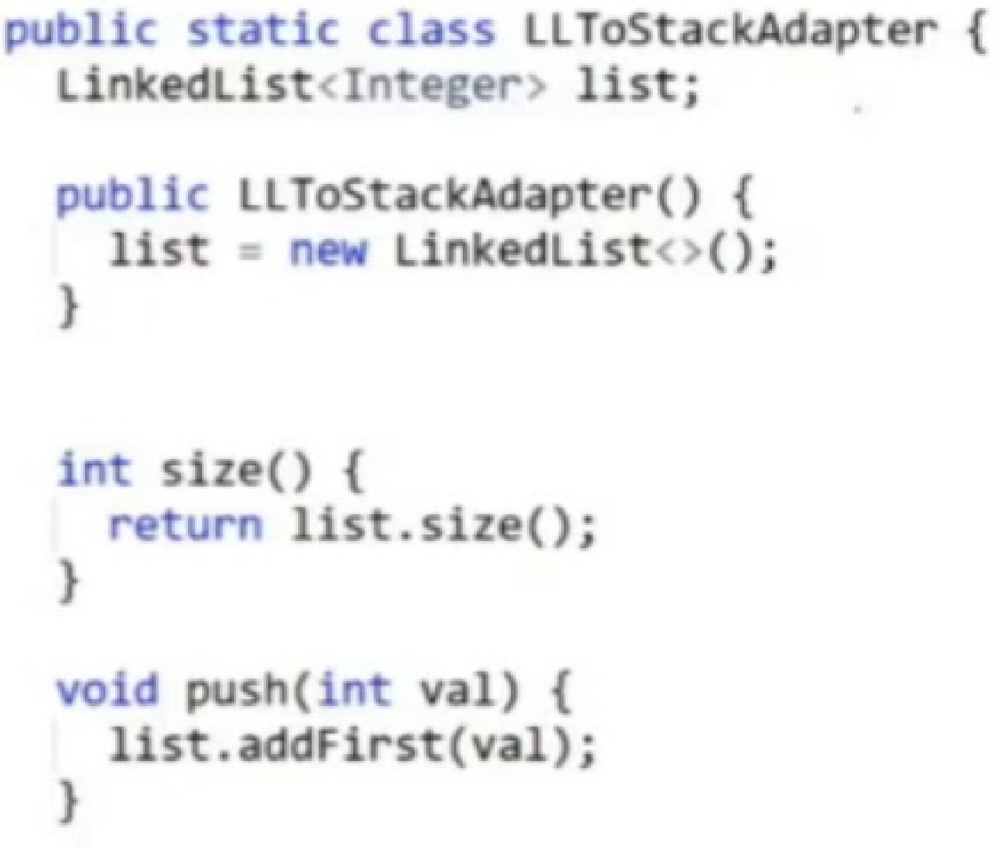
one apporach is
af rf gf

--> this is better appraoch to achieve behaviour of stack

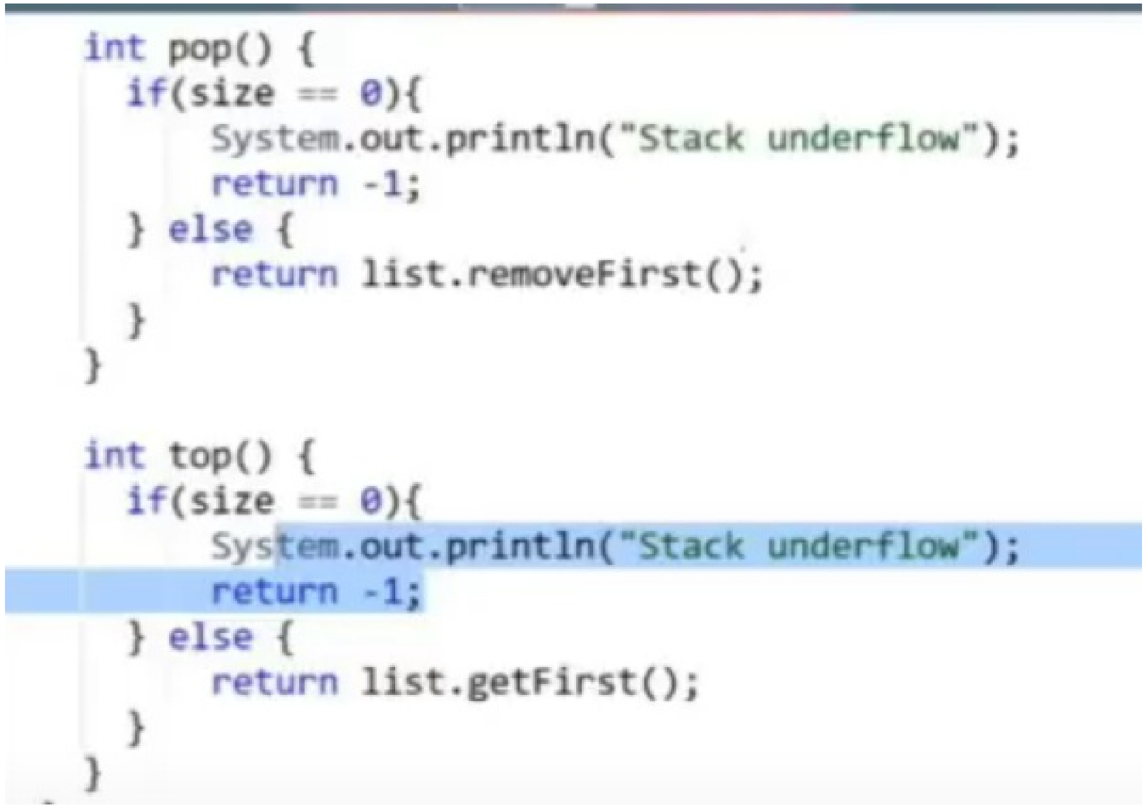




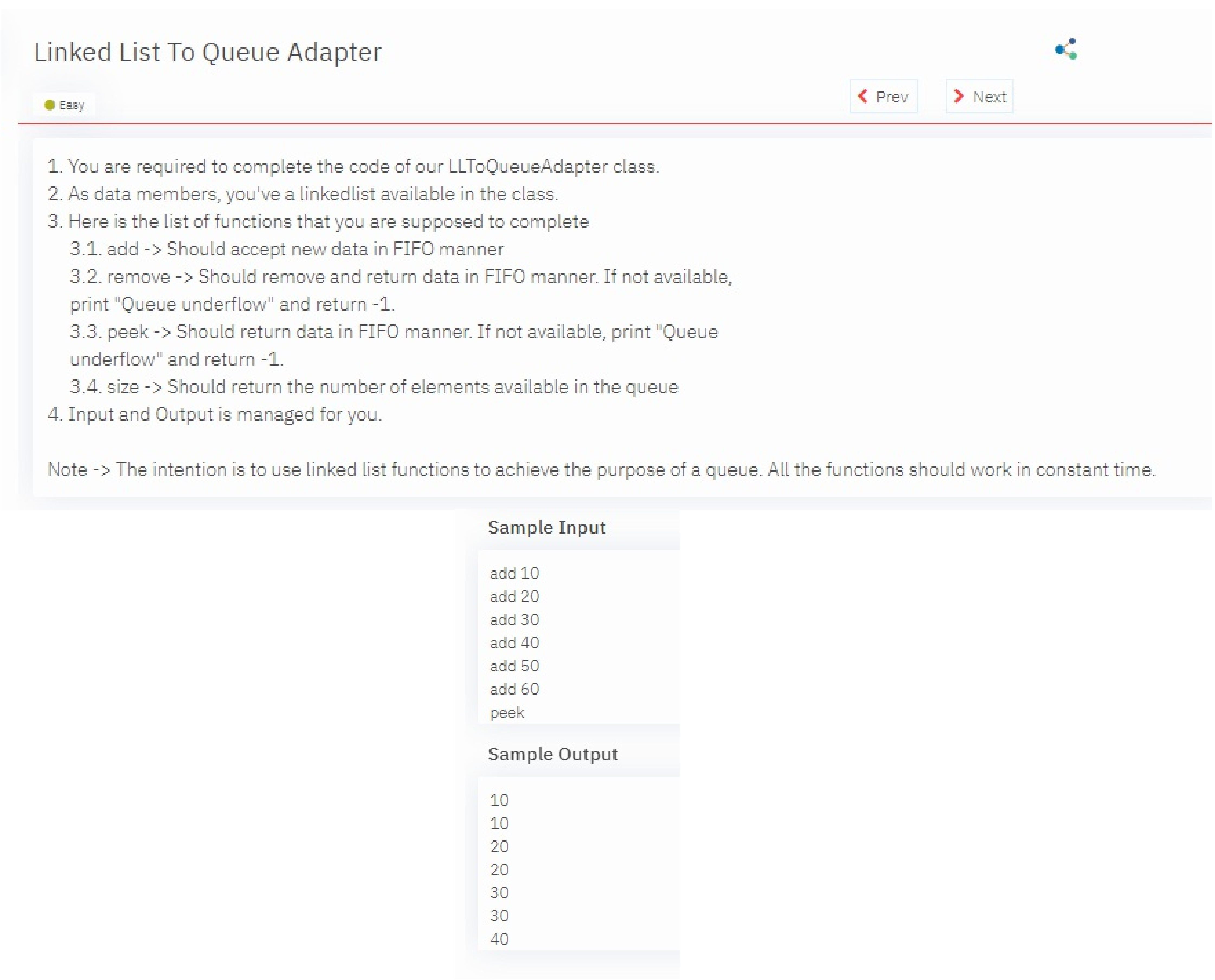
one apporach is
af rf gf --> this is better appraoch to achieve behaviour of stack



```
public static class LLToStackAdapter {
  LinkedList<Integer> list;

  public LLToStackAdapter() {
    list = new LinkedList<>();
  }


  int size() {
    return list.size();
  }

  void push(int val) {
    list.addFirst(val);
  }
```



```
int pop() {
  if(size == 0){
      System.out.println("Stack underflow");
      return -1;
  } else {
      return list.removeFirst();
  }
}

int top() {
  if(size == 0){
      System.out.println("Stack underflow");
      return -1;
  } else {
      return list.getFirst();
  }
}
```



Linked List To Queue Adapter

1. You are required to complete the code of our LLToQueueAdapter class.
2. As data members, you've a linkedlist available in the class.
3. Here is the list of functions that you are supposed to complete
   3.1. add -> Should accept new data in FIFO manner
   3.2. remove -> Should remove and return data in FIFO manner. If not available, print "Queue underflow" and return -1.
   3.3. peek -> Should return data in FIFO manner. If not available, print "Queue underflow" and return -1.
   3.4. size -> Should return the number of elements available in the queue
4. Input and Output is managed for you.

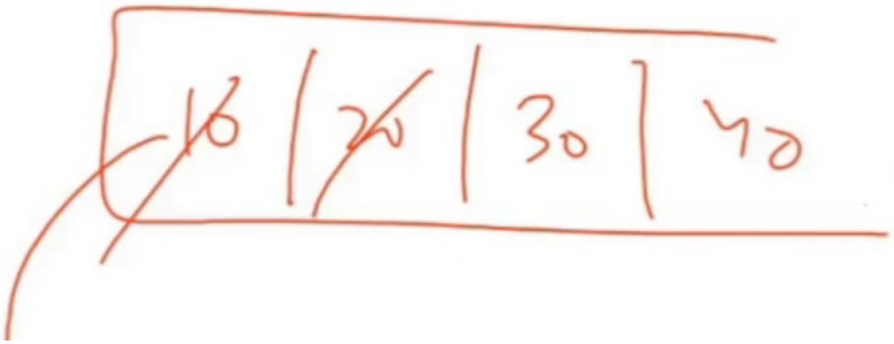
Note -> The intention is to use linked list functions to achieve the purpose of a queue. All the functions should work in constant time.

Sample Input

```
add 10
add 20
add 30
add 40
add 50
add 60
peek
```

Sample Output

```
10
10
20
20
30
30
40
```



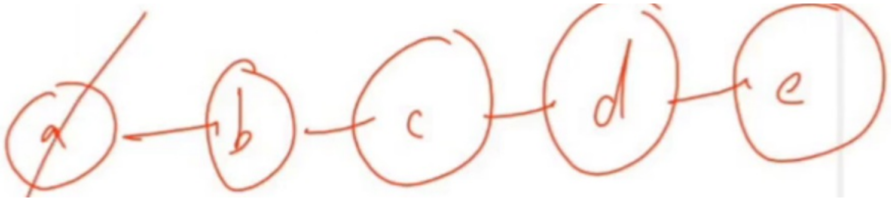
a	aL	aF
r	rF	rL

approach is

al rf

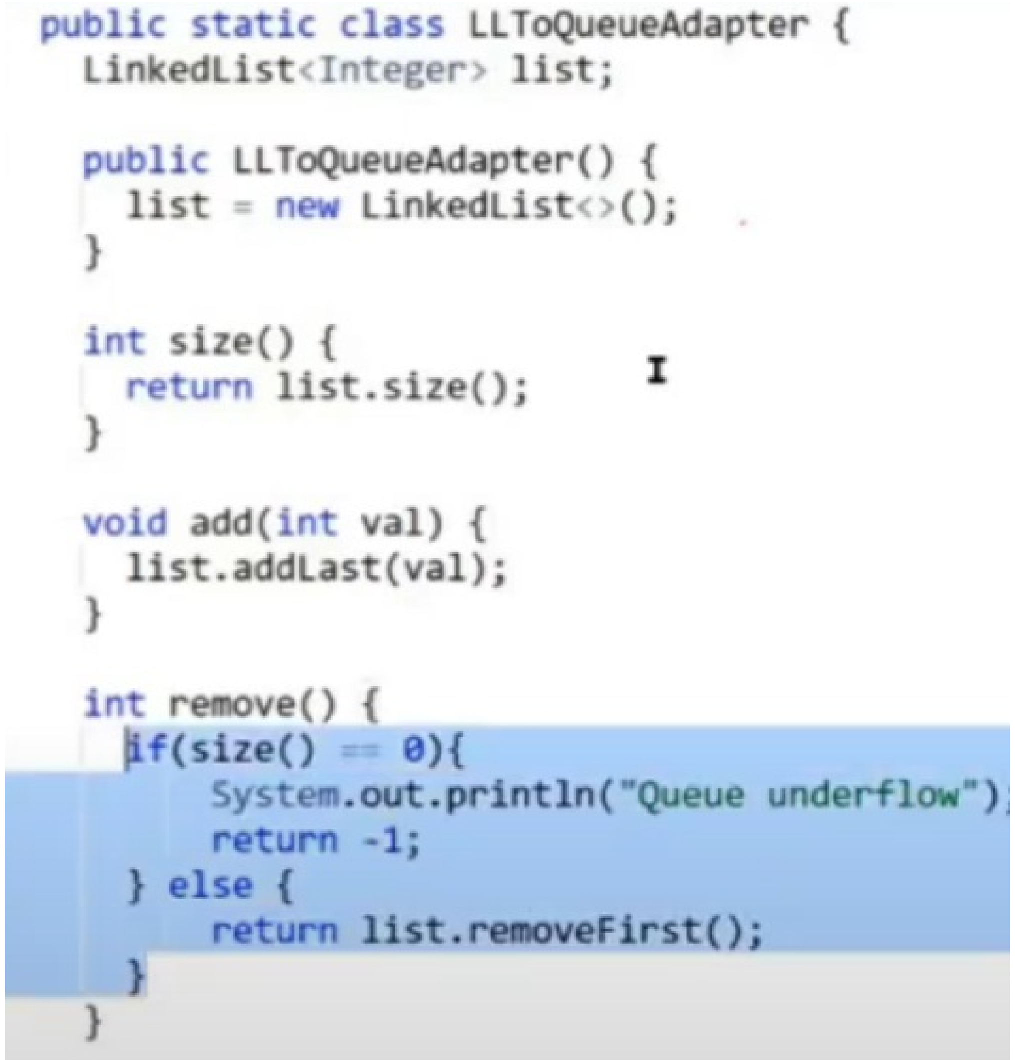
or

af rl

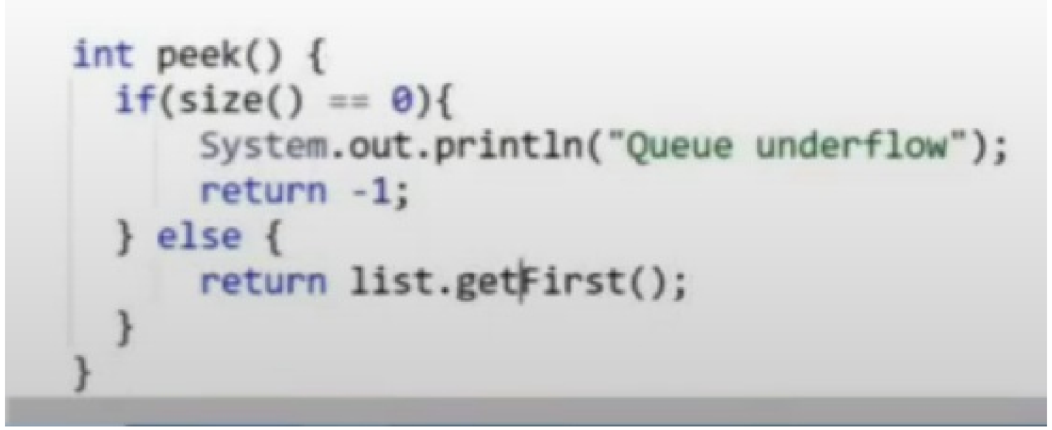


we will us al rf

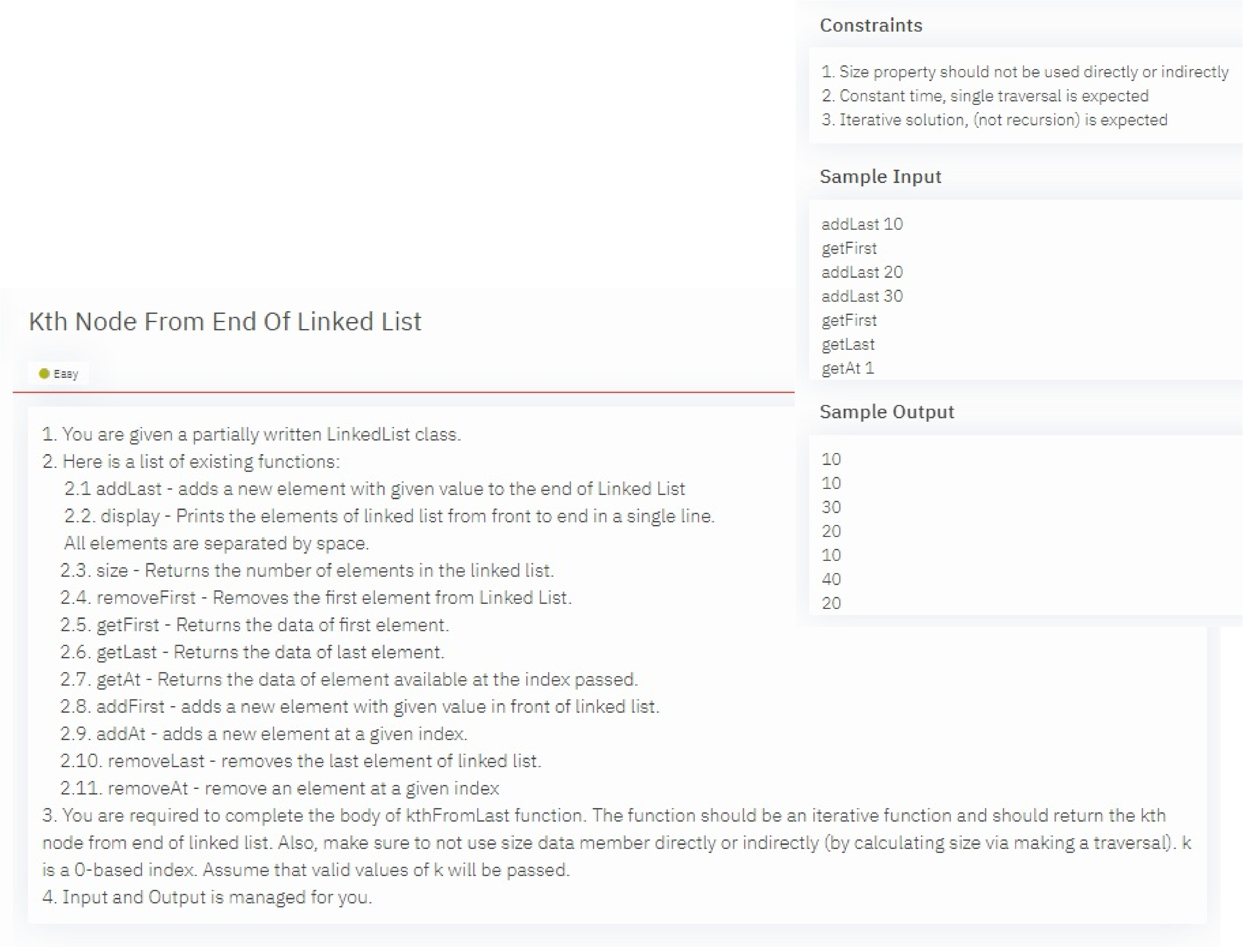
we will us al rf



```
public static class LLToQueueAdapter {
  LinkedList<Integer> list;

  public LLToQueueAdapter() {
    list = new LinkedList<>();
  }

  int size() {
    return list.size();
  }

  void add(int val) {
    list.addLast(val);
  }

  int remove() {
    if(size() == 0){
        System.out.println("Queue underflow");
        return -1;
    } else {
        return list.removeFirst();
    }
  }
```



```
int peek() {
  if(size() == 0){
      System.out.println("Queue underflow");
      return -1;
  } else {
      return list.getFirst();
  }
}
```



Kth Node From End Of Linked List

Easy

1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
   2.1 addLast - adds a new element with given value to the end of Linked List
   2.2. display - Prints the elements of linked list from front to end in a single line.
   All elements are separated by space.
   2.3. size - Returns the number of elements in the linked list.
   2.4. removeFirst - Removes the first element from Linked List.
   2.5. getFirst - Returns the data of first element.
   2.6. getLast - Returns the data of last element.
   2.7. getAt - Returns the data of element available at the index passed.
   2.8. addFirst - adds a new element with given value in front of linked list.
   2.9. addAt - adds a new element at a given index.
   2.10. removeLast - removes the last element of linked list.
   2.11. removeAt - remove an element at a given index
3. You are required to complete the body of kthFromLast function. The function should be an iterative function and should return the kth node from end of linked list. Also, make sure to not use size data member directly or indirectly (by calculating size via making a traversal). k is a 0-based index. Assume that valid values of k will be passed.
4. Input and Output is managed for you.

Constraints

1. Size property should not be used directly or indirectly
2. Constant time, single traversal is expected
3. Iterative solution, (not recursion) is expected

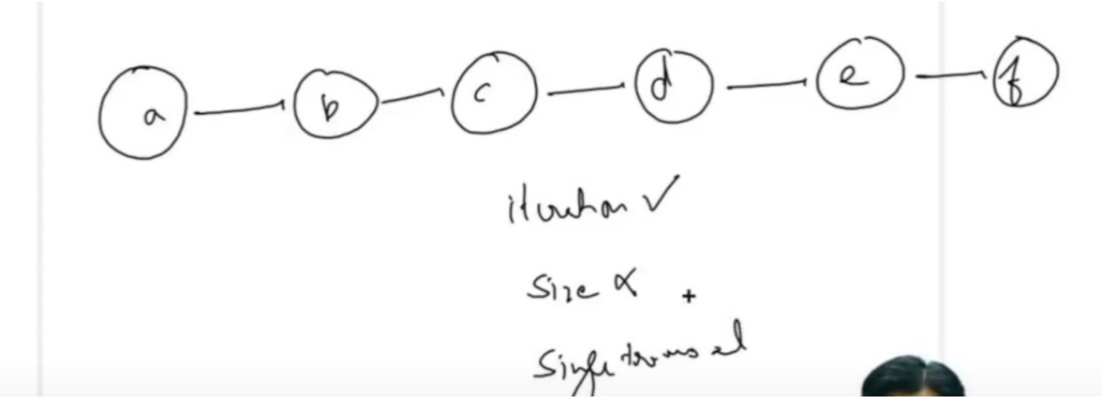
Sample Input

```
addLast 10
getFirst
addLast 20
addLast 30
getFirst
getLast
getAt 1
```

Sample Output

```
10
10
30
20
10
40
20
```

you need to get the kth node from back

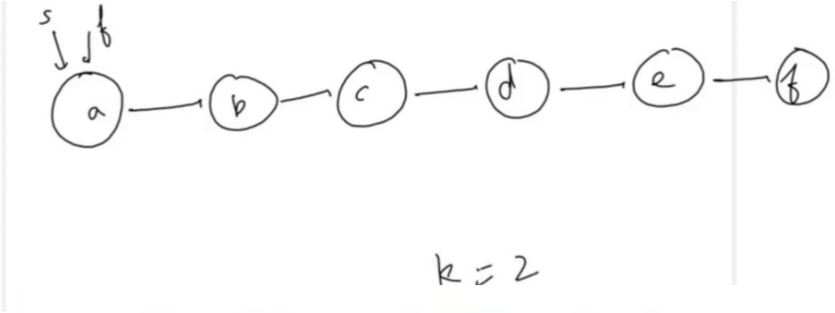


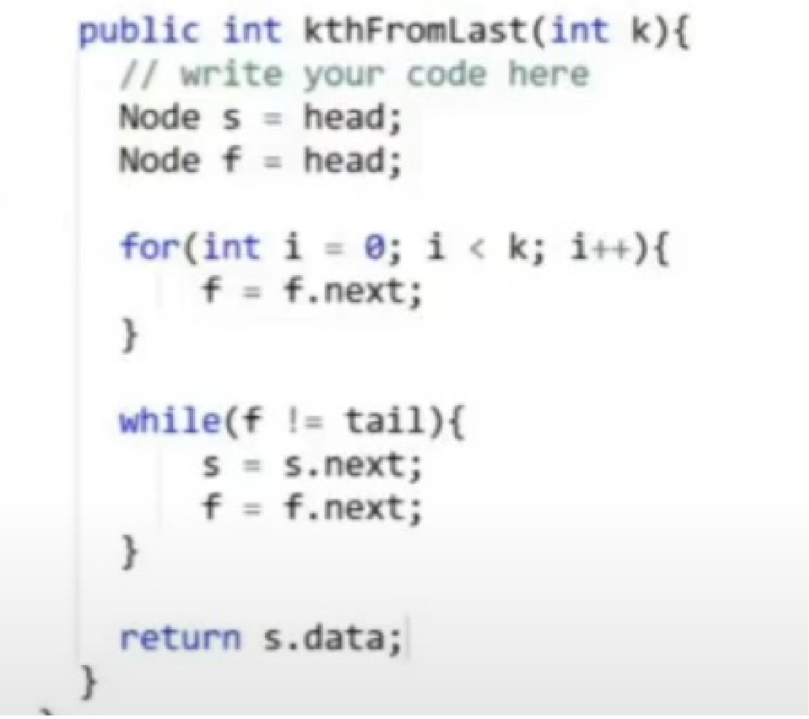
constraint

will not use size property
give iterative soln
single traversal

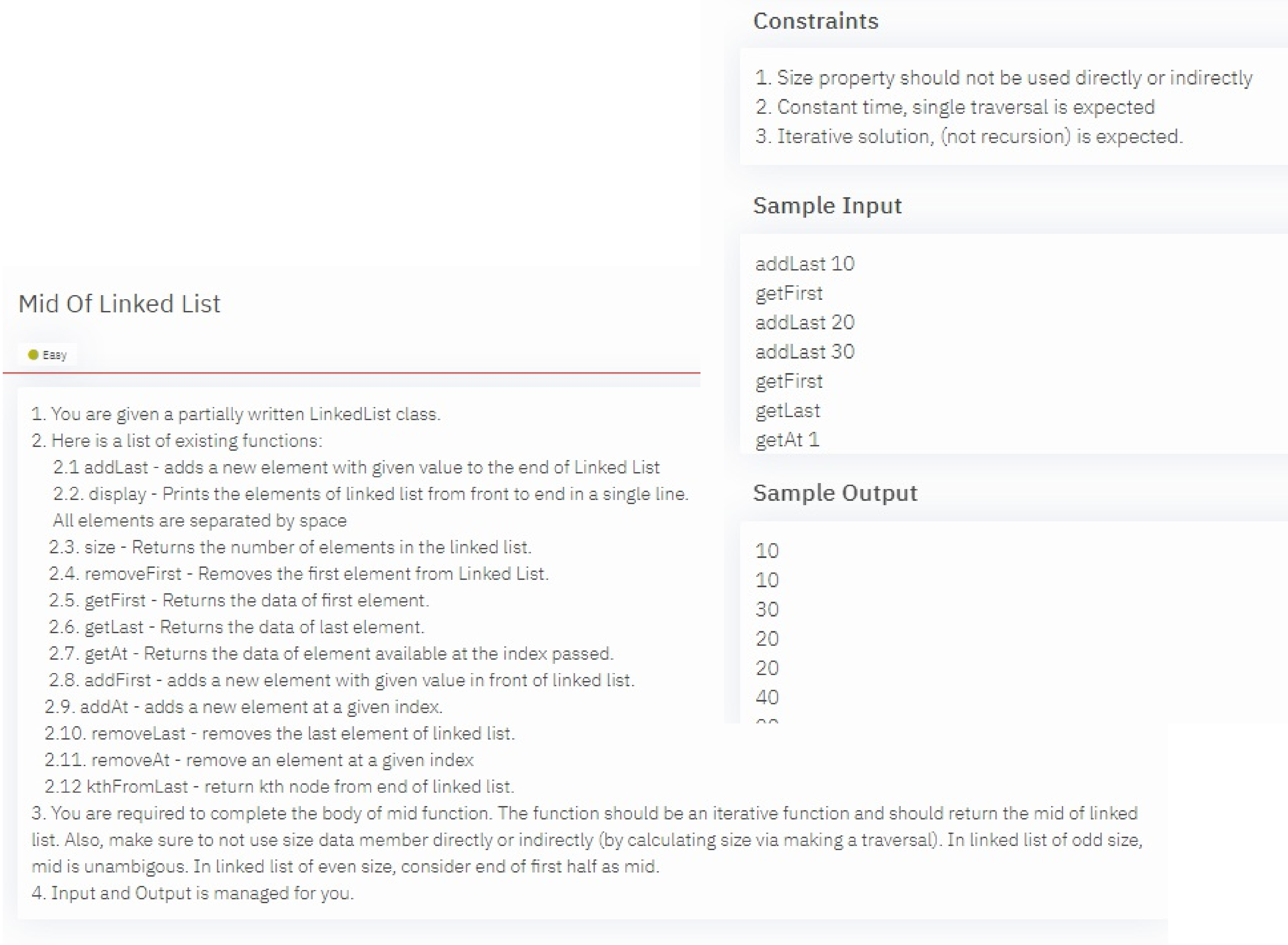
Two pointer approach (slow ,fast)

```
initially f,s=0;
1. f ko k times move karenge
2. now, while k !=last node or tail
        s=s->next
        f=f->next
3. ans=s;
```





```
public int kthFromLast(int k){
  // write your code here
  Node s = head;
  Node f = head;

  for(int i = 0; i < k; i++){
      f = f.next;
  }

  while(f != tail){
      s = s.next;
      f = f.next;
  }

  return s.data;
}
```



Mid Of Linked List

Easy

1. You are given a partially written LinkedList class.
2. Here is a list of existing functions:
   2.1 addLast - adds a new element with given value to the end of Linked List
   2.2. display - Prints the elements of linked list from front to end in a single line. All elements are separated by space
   2.3. size - Returns the number of elements in the linked list.
   2.4. removeFirst - Removes the first element from Linked List.
   2.5. getFirst - Returns the data of first element.
   2.6. getLast - Returns the data of last element.
   2.7. getAt - Returns the data of element available at the index passed.
   2.8. addFirst - adds a new element with given value in front of linked list.
   2.9. addAt - adds a new element at a given index.
   2.10. removeLast - removes the last element of linked list.
   2.11. removeAt - remove an element at a given index
   2.12 kthFromLast - return kth node from end of linked list.
3. You are required to complete the body of mid function. The function should be an iterative function and should return the mid of linked list. Also, make sure to not use size data member directly or indirectly (by calculating size via making a traversal). In linked list of odd size, mid is unambigous. In linked list of even size, consider end of first half as mid.
4. Input and Output is managed for you.

Constraints

1. Size property should not be used directly or indirectly
2. Constant time, single traversal is expected
3. Iterative solution, (not recursion) is expected.

Sample Input

```
addLast 10
getFirst
addLast 20
addLast 30
getFirst
getLast
getAt 1
```

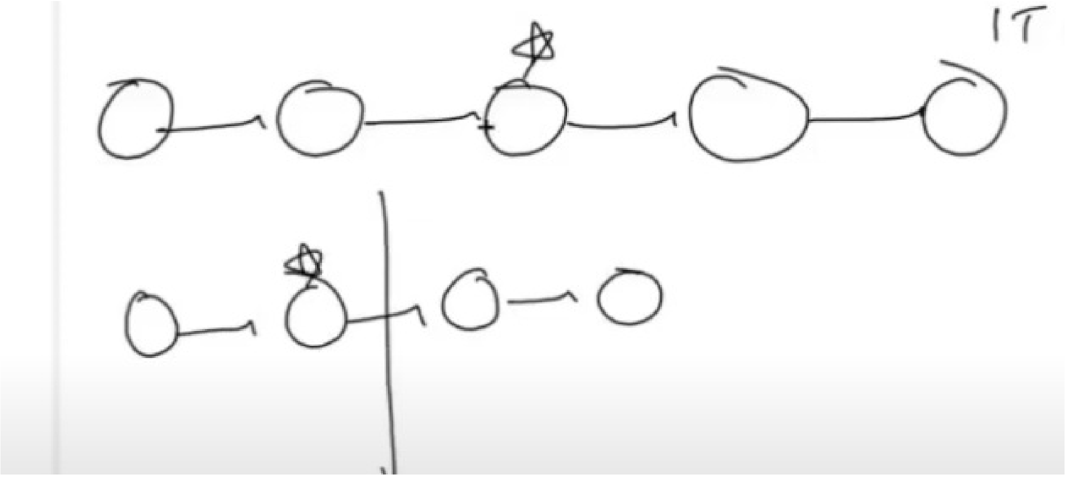
Sample Output

```
10
10
30
20
20
40
```

get mid of linklist

constraint

will not use size property
give iterative soln
single traversal



for odd size , mid element

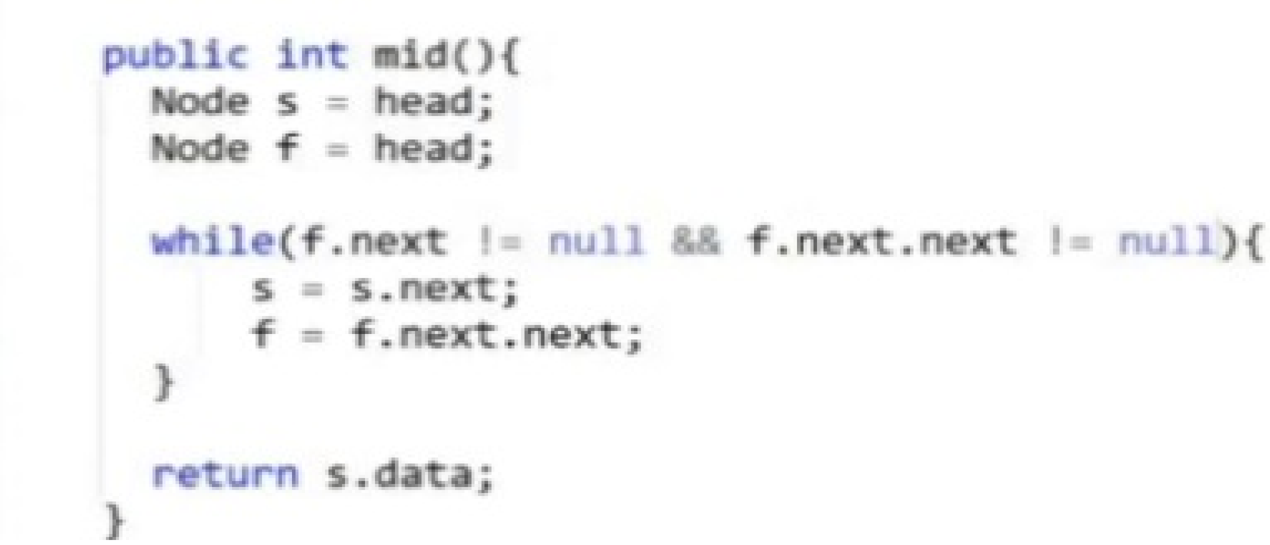
for even size, 1st half ka last element is mid

f.next.next is added for even no of nodes

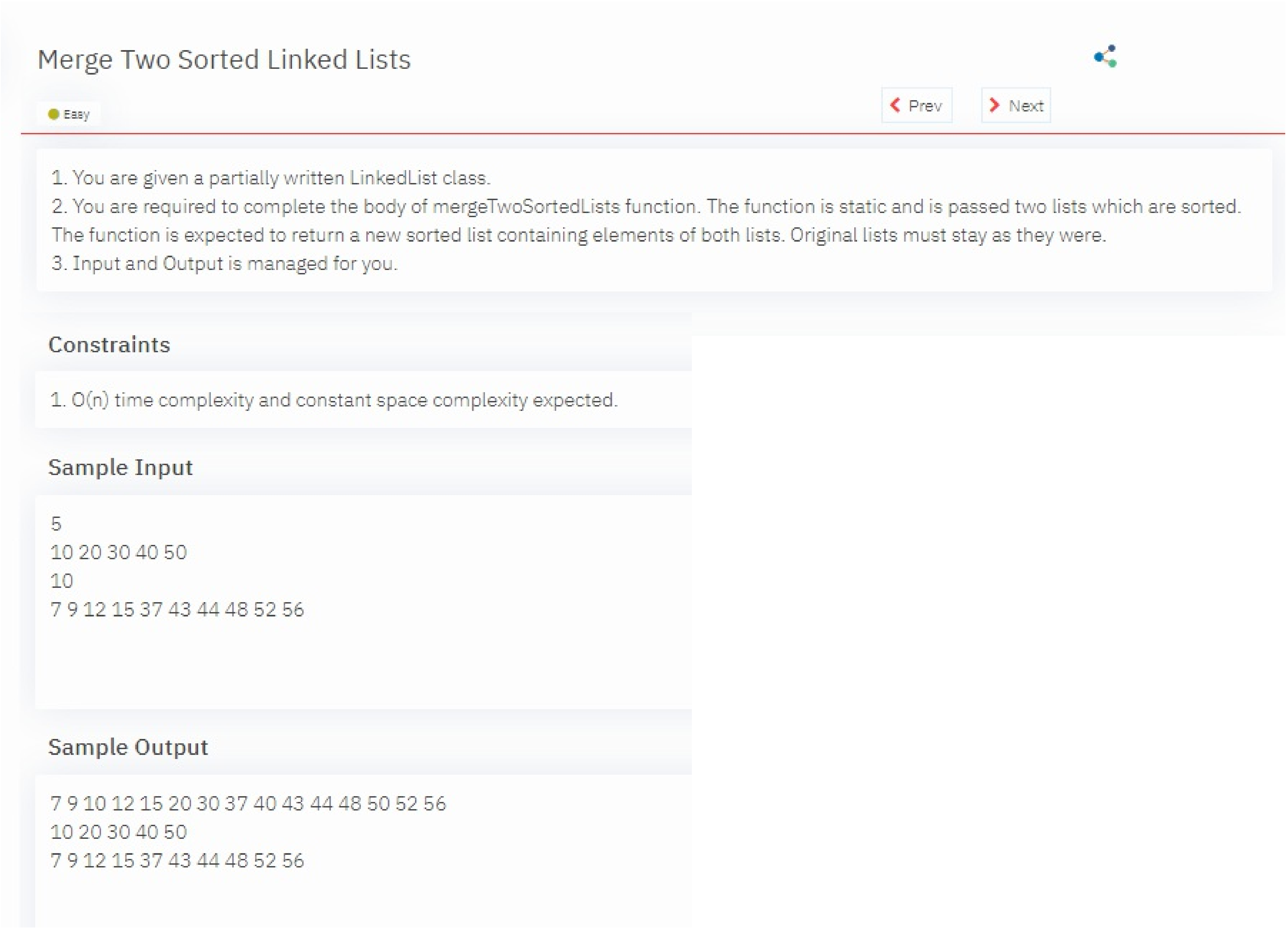
Two pointer approach (slow ,fast)

while fast !=tail
    when slow 1 step chalta he
    fast will move twice

ans = slow.data



```
public int mid(){
  Node s = head;
  Node f = head;

  while(f.next != null && f.next.next != null){
      s = s.next;
      f = f.next.next;
  }

  return s.data;
}
```



Merge Two Sorted Linked Lists

1. You are given a partially written LinkedList class.
2. You are required to complete the body of mergeTwoSortedLists function. The function is static and is passed two lists which are sorted. The function is expected to return a new sorted list containing elements of both lists. Original lists must stay as they were.
3. Input and Output is managed for you.

Constraints

1. O(n) time complexity and constant space complexity expected.

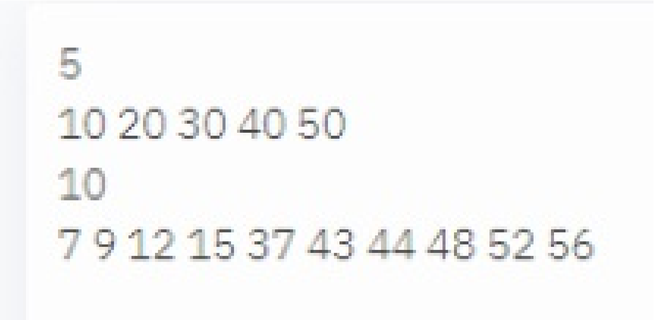
Sample Input

```
5
10 20 30 40 50
10
7 9 12 15 37 43 44 48 52 56
```

Sample Output

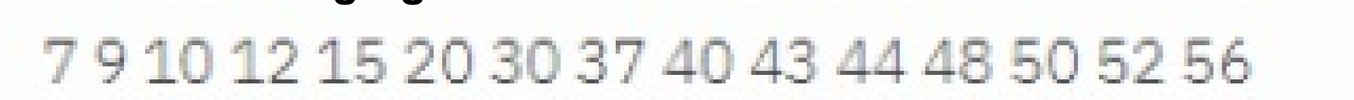
```
7 9 10 12 15 20 30 37 40 43 44 48 50 52 56
10 20 30 40 50
7 9 12 15 37 43 44 48 52 56
```

2 sorted linklist

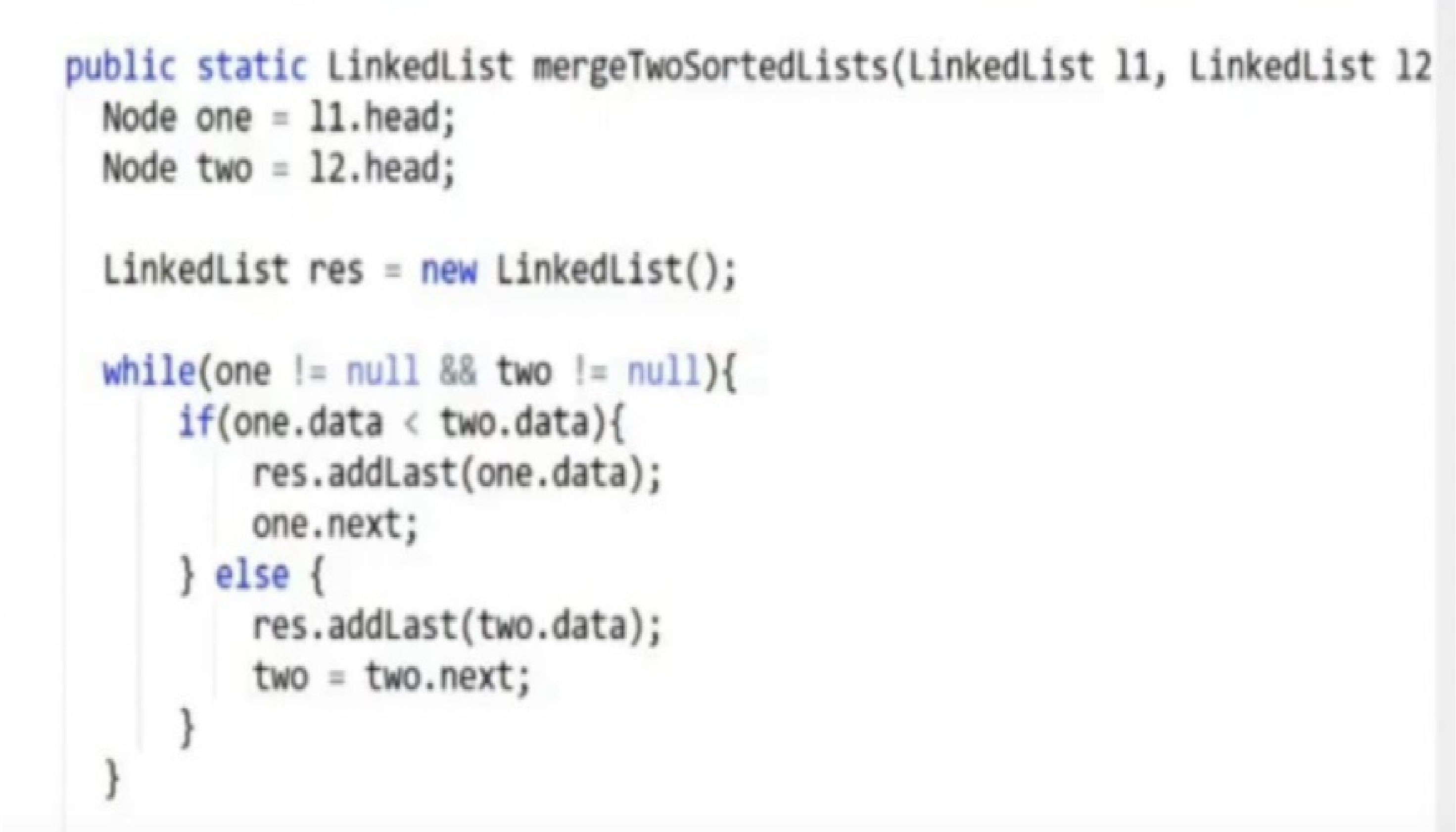


```
5
10 20 30 40 50
10
7 9 12 15 37 43 44 48 52 56
```

after merging

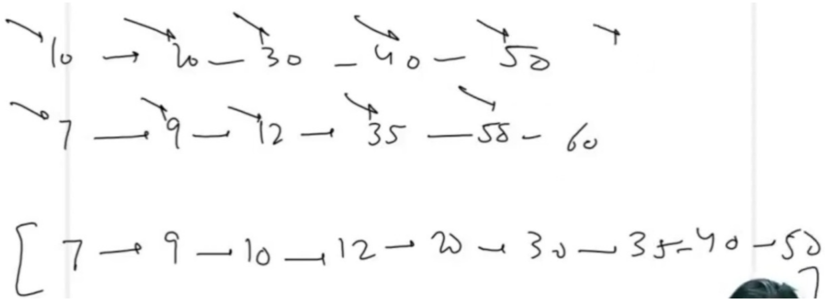


```
7 9 10 12 15 20 30 37 40 43 44 48 50 52 56
```

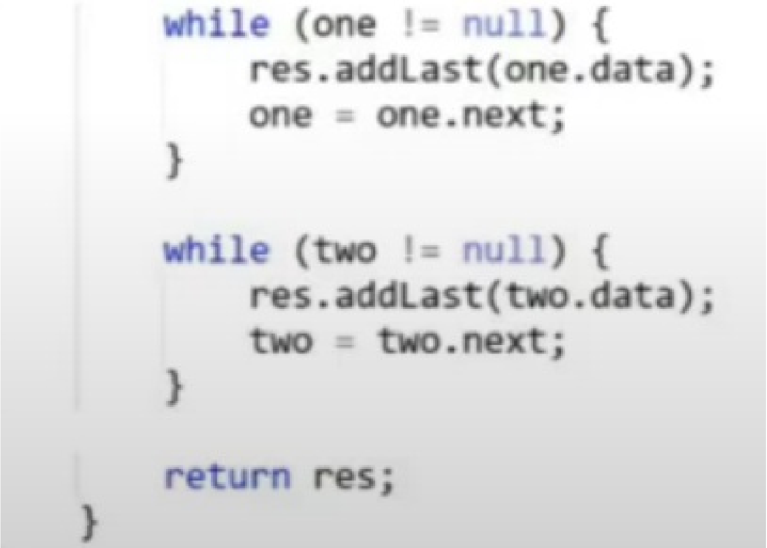


```
public static LinkedList mergeTwoSortedLists(LinkedList l1, LinkedList l2
  Node one = l1.head;
  Node two = l2.head;

  LinkedList res = new LinkedList();

  while(one != null && two != null){
      if(one.data < two.data){
          res.addLast(one.data);
          one.next;
      } else {
          res.addLast(two.data);
          two = two.next;
      }
  }
```

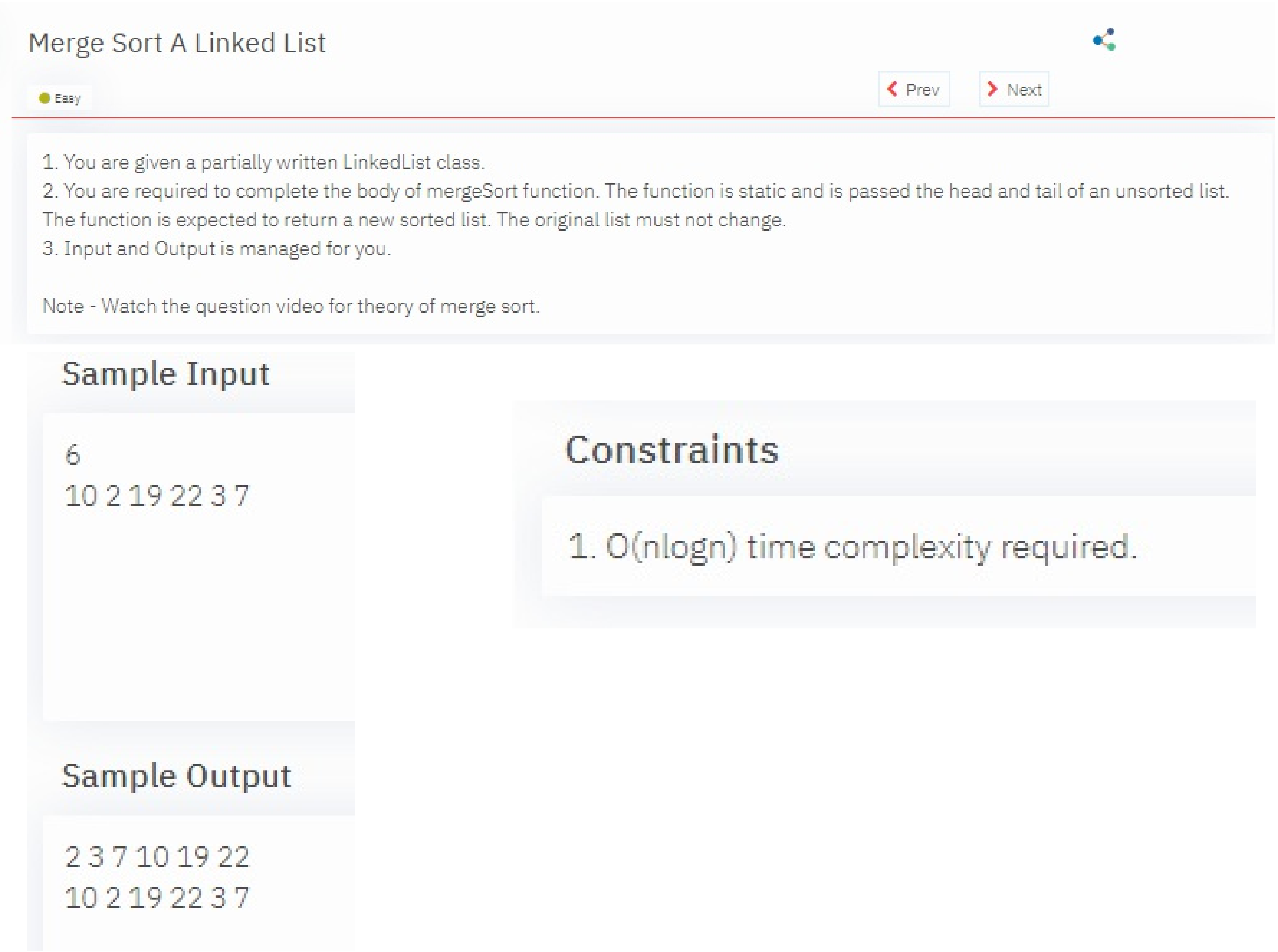
Two pointer approach(i,j)



edge case when one /two khatam ho gaya tab



```
    while (one != null) {
        res.addLast(one.data);
        one = one.next;
    }

    while (two != null) {
        res.addLast(two.data);
        two = two.next;
    }

    return res;
}
```



Merge Sort A Linked List

1. You are given a partially written LinkedList class.
2. You are required to complete the body of mergeSort function. The function is static and is passed the head and tail of an unsorted list. The function is expected to return a new sorted list. The original list must not change.
3. Input and Output is managed for you.

Note - Watch the question video for theory of merge sort.

Sample Input

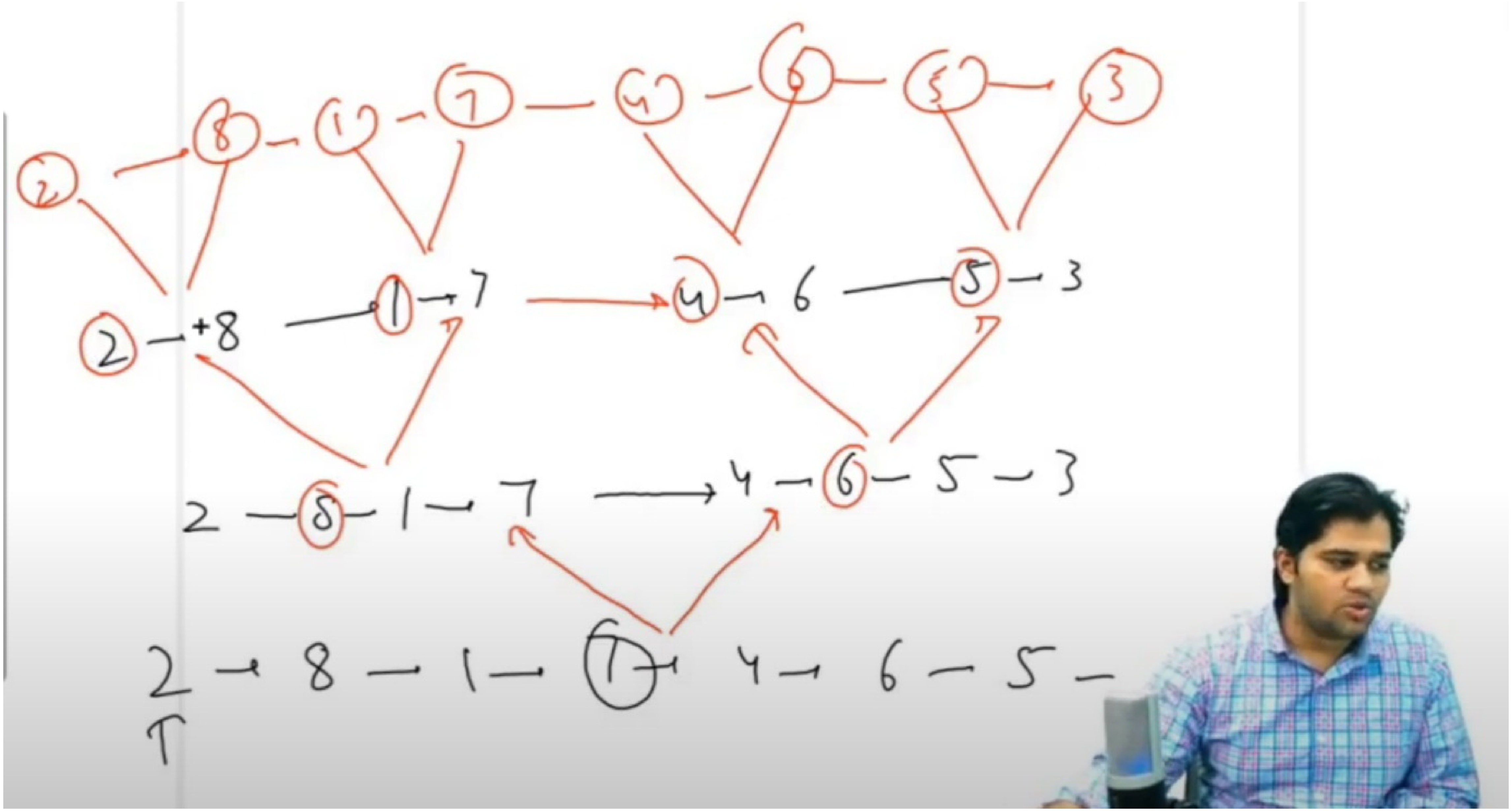
```
6
10 2 19 22 3 7
```

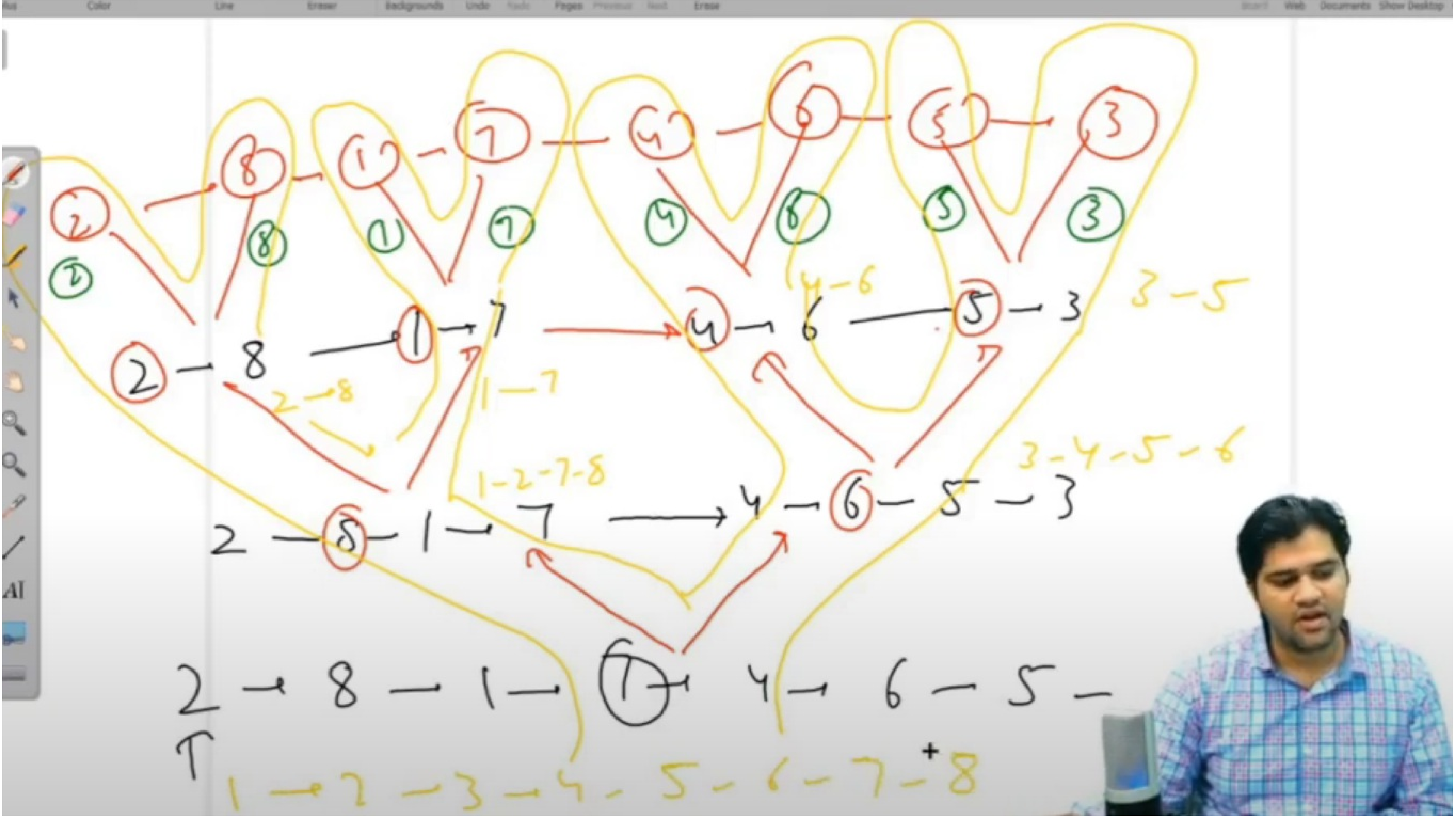
Sample Output

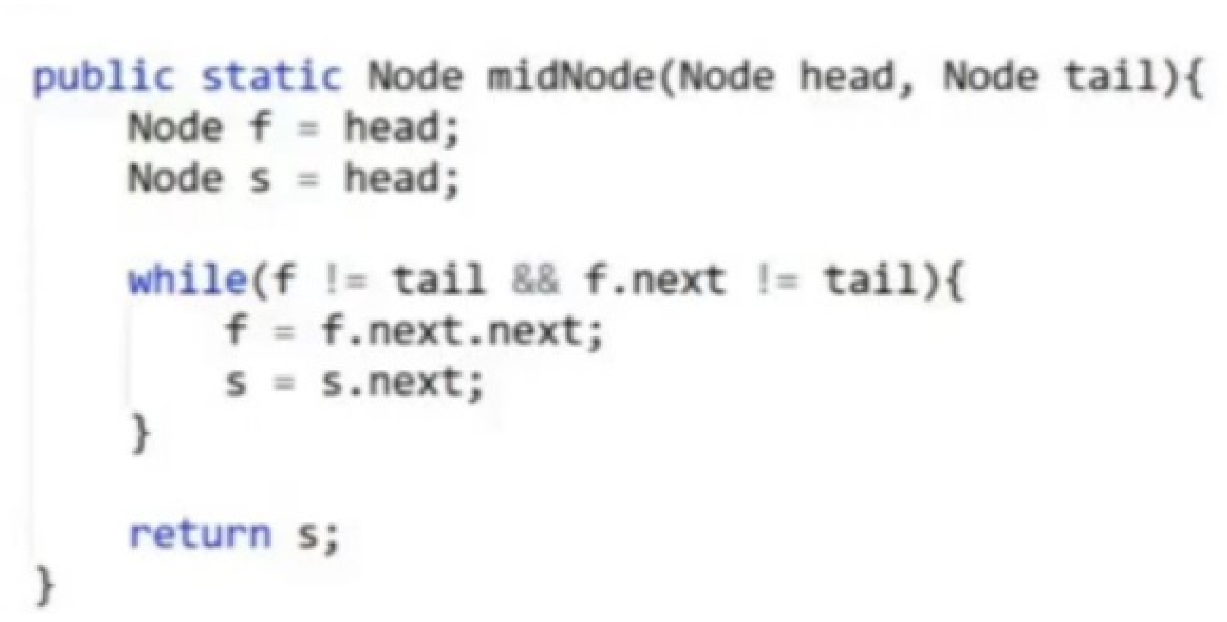
```
2 3 7 10 19 22
10 2 19 22 3 7
```

Constraints

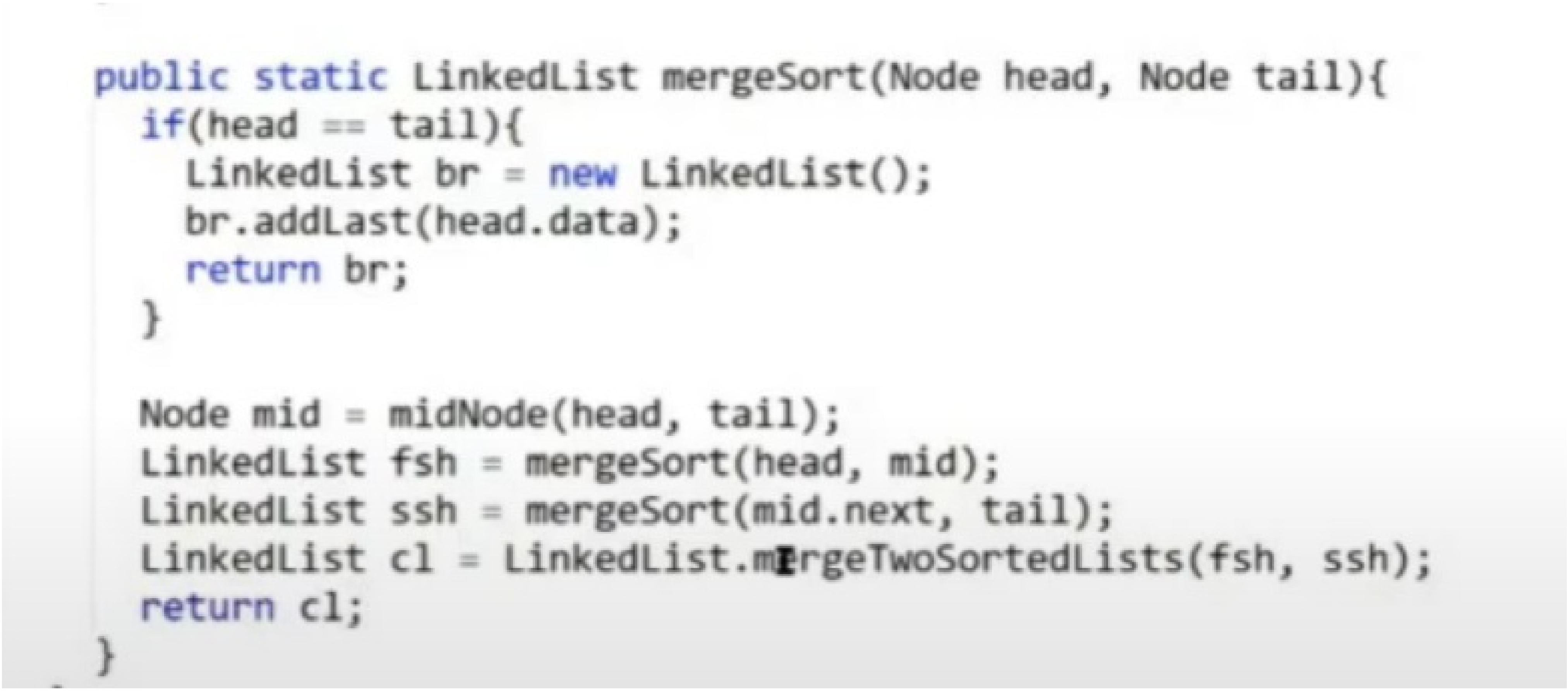
1. O(nlogn) time complexity required.


2 — 8 — 1 — 7 — 4 — 6 — 5 — 3
2 — 8 — 1 — 7 — 4 — 6 — 5 — 3
2 — 8 — 1 — 7 — 4 — 6 — 5 — 3
2 — 8 — 1 — 7 — 4 — 6 — 5 —
T

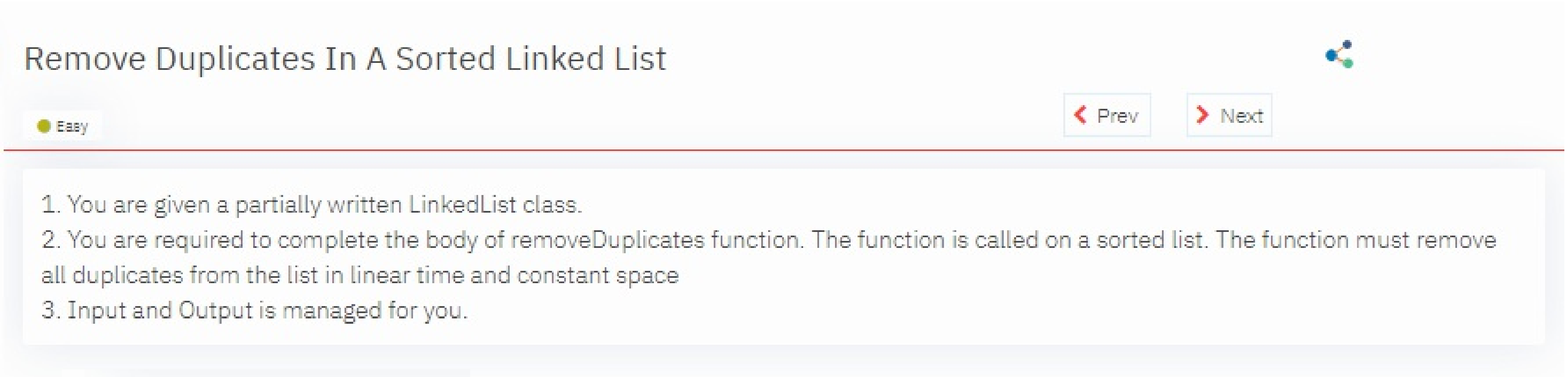

Color
Line
Eraser
Backgrounds
Undo
Redo
Pages
Previous
Next
Erase
Start
Web
Documents
Show Desktop
2 — 8 — 1 — 7 — 4 — 6 — 5 → 3
2
8
1
7
4
6
5
3
2 — 8 — 1 → 7 → 4 → 6 — 5 — 3
4 – 6
3 – 5
2 → 8
1 — 7
1 – 2 – 7 – 8
3 – 4 – 5 – 6
2 — 8 — 1 — 7 → 4 — 6 — 5 — 3
2 → 8 — 1 — 7 — 4 — 6 — 5 —
T
1 → 2 — 3 → 4 – 5 – 6 – 7 – 8



```java
public static Node midNode(Node head, Node tail){
    Node f = head;
    Node s = head;

    while(f != tail && f.next != tail){
        f = f.next.next;
        s = s.next;
    }

    return s;
}
```

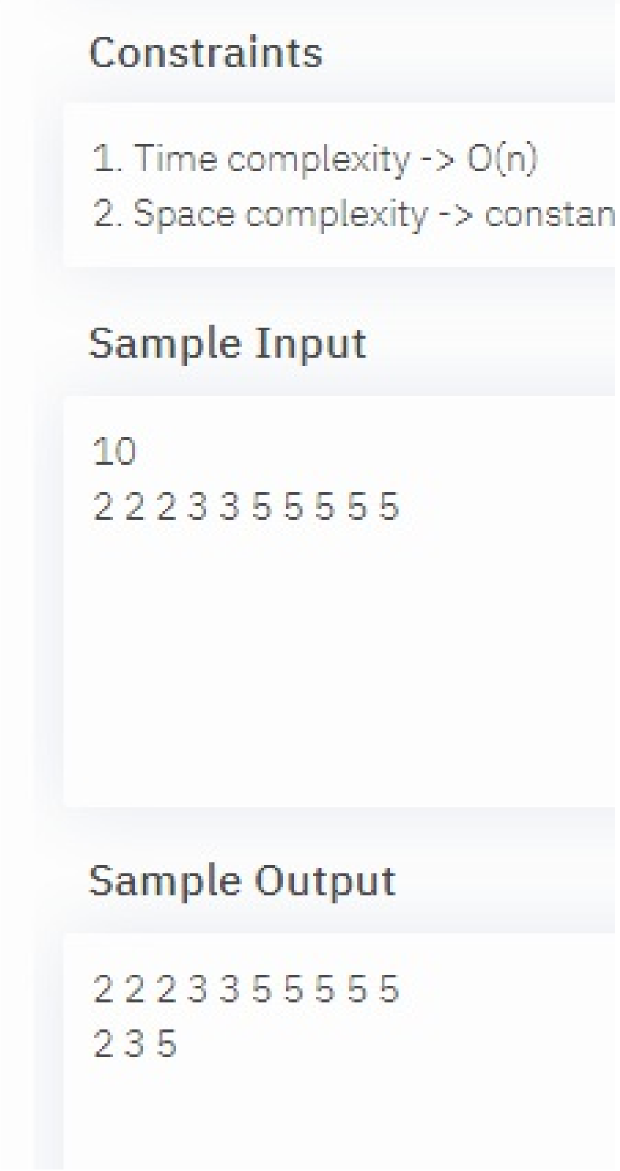


```java
public static LinkedList mergeSort(Node head, Node tail){
  if(head == tail){
    LinkedList br = new LinkedList();
    br.addLast(head.data);
    return br;
  }

  Node mid = midNode(head, tail);
  LinkedList fsh = mergeSort(head, mid);
  LinkedList ssh = mergeSort(mid.next, tail);
  LinkedList cl = LinkedList.mergeTwoSortedLists(fsh, ssh);
  return cl;
}
```



Remove Duplicates In A Sorted Linked List

1. You are given a partially written LinkedList class.
2. You are required to complete the body of removeDuplicates function. The function is called on a sorted list. The function must remove all duplicates from the list in linear time and constant space
3. Input and Output is managed for you.



Constraints

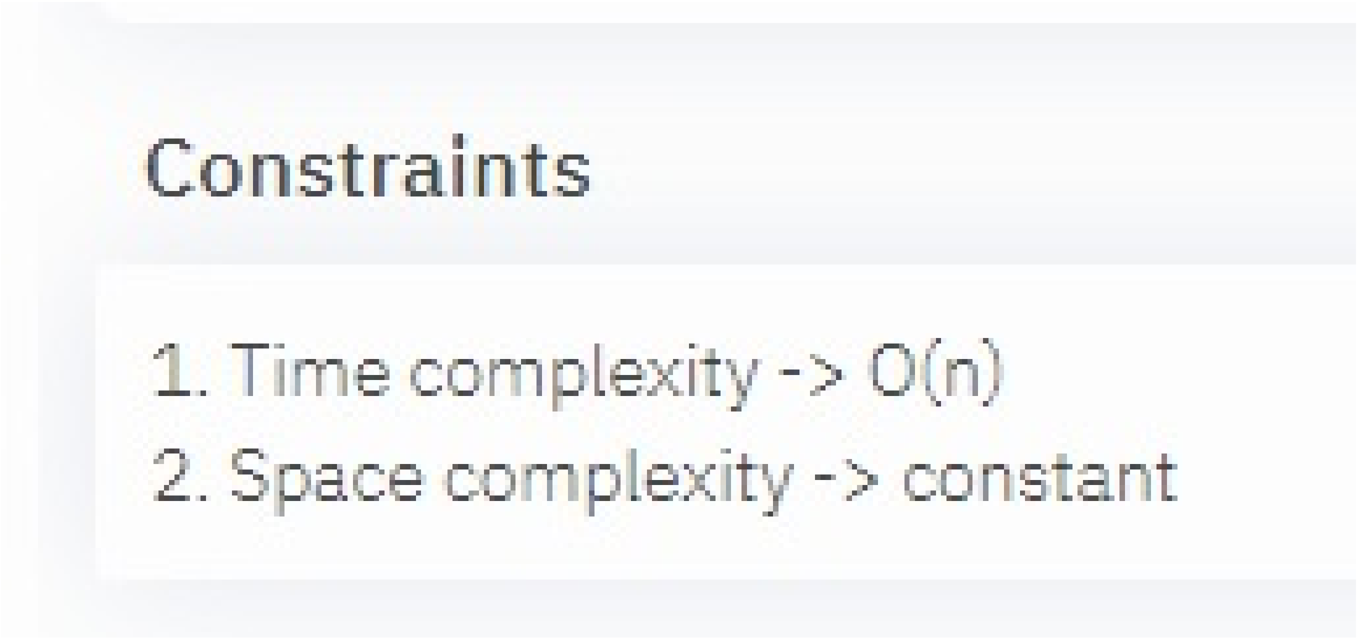
1. Time complexity -> O(n)
2. Space complexity -> constan

Sample Input

```
10
2 2 2 3 3 5 5 5 5 5
```

Sample Output

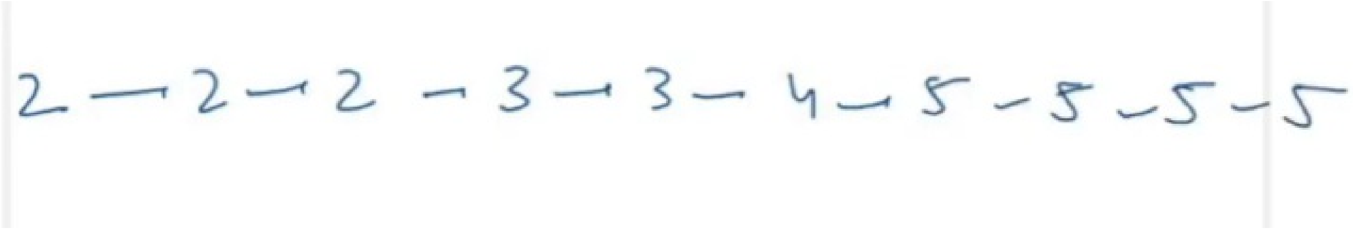
```
2 2 2 3 3 5 5 5 5 5
2 3 5
```



Constraints

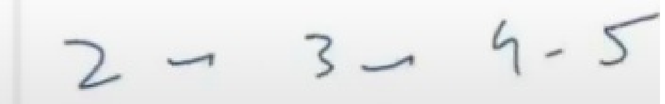
1. Time complexity -> O(n)
2. Space complexity -> constant

i/p



2 → 2 → 2 → 3 → 3 → 4 → 5 → 5 → 5 → 5

o/p



2 → 3 → 4 → 5