*“Computer Engineers don’t fall in love. They just ‘link’ with someone”*

* *Sheer Angel*

Dear reader, I welcome you to a problem named ‘[**Add Last in Linked List**](https://www.pepcoding.com/resources/online-java-foundation/linked-lists/add-last-linked-list-official/ojquestion)**’**. Trust me, knowing this basic concept will help you solve hard category problems based on linked lists.

***Problem Statement***

* You are given a partially written LinkedList class.
* 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.

*Note*: You just need to complete the code of the ***addLast*** function, all other input/output methods and data members are managed for you.

I recommend you to watch the [question video](https://www.youtube.com/watch?v=r9FxRiIZhK4), where the problem statement is explained in detail.

***Solution***

Firstly, we should create a node with data = val and it’s next pointer should point to null for the time being.

There are two cases which need to be handled:

1. If the linked list is empty, i.e. the size of the linked list is 0.
2. If there is at least one node already present in the linked list, i.e. the size of the linked list is greater than or equal to 1.

If the linked list is empty, then head and tail are currently pointing to null. Hence, the node we are about to add should become both the head node as well as the tail node of the linked list.

Otherwise, if the linked list is not empty, then we will have to append the node at the end of the linked list, i.e. the next pointer of the last node should point to the new node inserted. Also, this new node should now be the tail node of the linked list.

In either of the two cases, we are increasing the size of the linked list by 1.

Please refer to the [solution video](https://www.youtube.com/watch?v=fjjORH3nWy4) if you find difficulty in understanding the algorithm completely.

***Pseudo Code/ Algorithm***

1. Create a new Node, say temp, using ***Node temp = new Node()***.
   1. Make temp’s data equal to val, i.e. ***temp.data = val***.
   2. Make temp’s next point to null, i.e. ***temp.next = null***.
2. If size of linked list is 0, (linked list is empty)
   1. Make head as well as tail point to temp, i.e. ***head = tail = temp***.
3. Else, (linked list has at least one node)
   1. Make the next pointer of tail node point to temp, i.e. ***tail.next = temp***.
   2. Update tail pointer to this new temp node, i.e. ***tail = temp***.
4. Increment size of linked list by 1, i.e. ***size++***.

***Implementation (Java)***

How about first trying by yourself without reading the code we provide?

import java.io.\*;

import java.util.\*;

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) {

Node temp = new Node();

temp.data = val;

temp.next = null;

if (size == 0) {

head = tail = temp;

} else {

tail.next = temp;

tail = temp;

}

size++;

}

}

public static void testList(LinkedList list) {

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

System.out.println(temp.data);

}

System.out.println(list.size);

if (list.size > 0) {

System.out.println(list.tail.data);

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

LinkedList list = new LinkedList();

String str = br.readLine();

while(str.equals("quit") == false){

if(str.startsWith("addLast")){

int val = Integer.parseInt(str.split(" ")[1]);

list.addLast(val);

}

str = br.readLine();

}

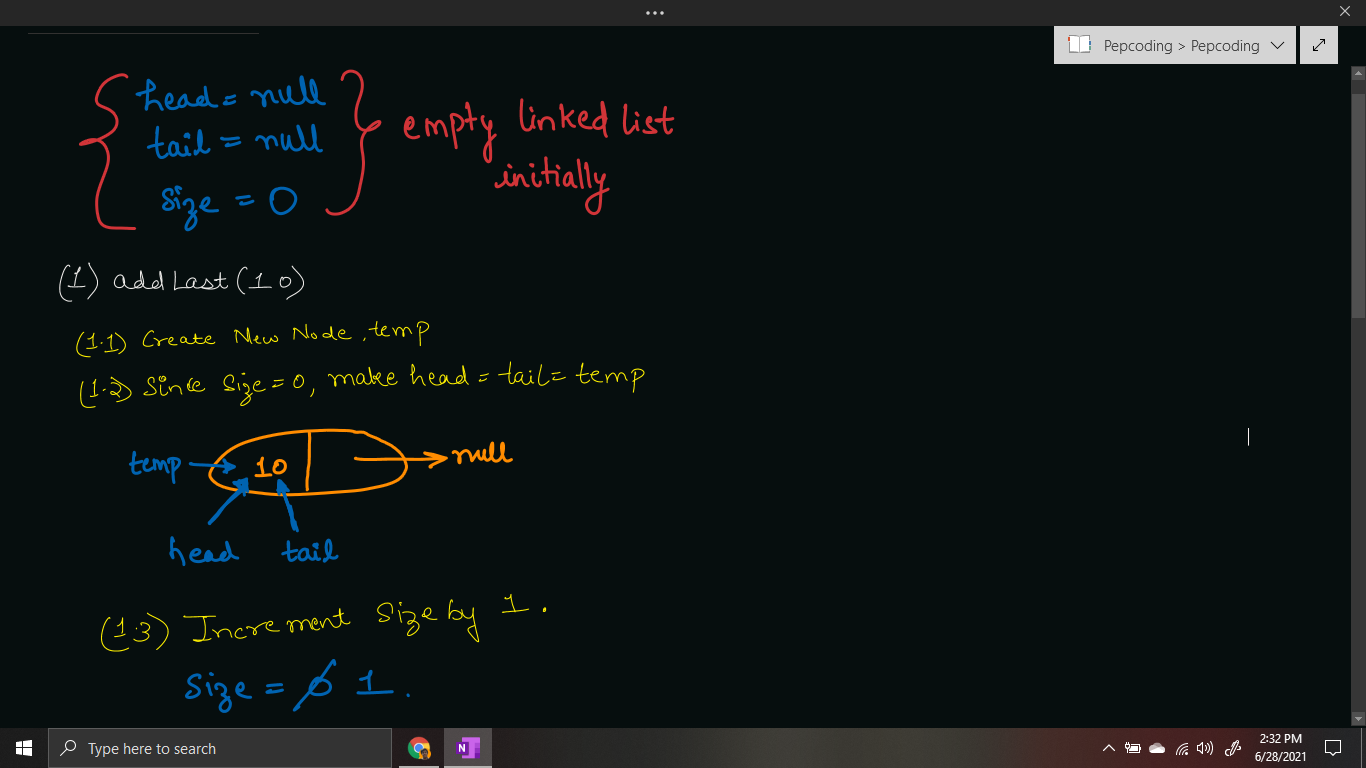
testList(list);

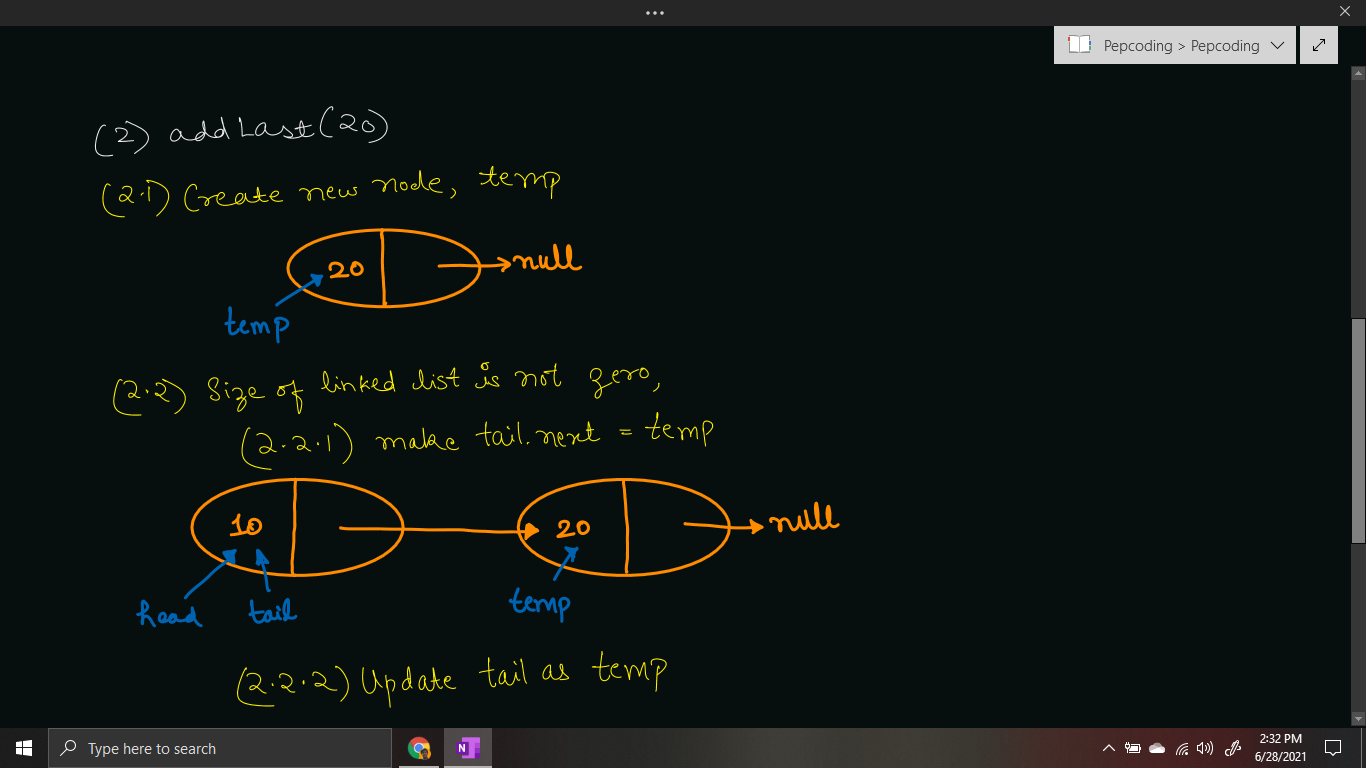
}

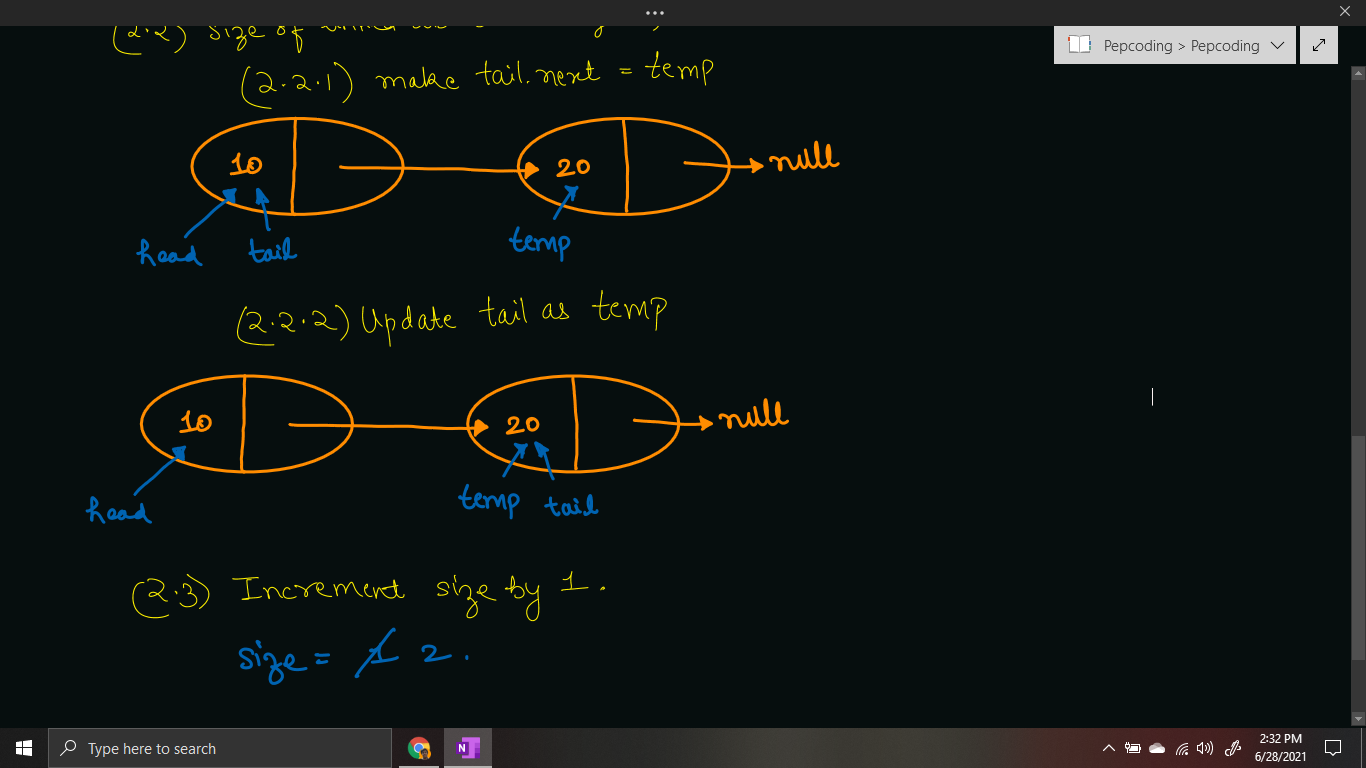
}

This code is written and explained by our team in [this video](https://www.youtube.com/watch?v=fjjORH3nWy4). Please refer to it if you are stuck somewhere.

***Example:*** Let us dry run the algorithm using a few addLast operations.







***Time & Space Complexity Analysis***

**Time Complexity** -

First, we are creating a new node, which is a constant O(1) operation. Then, if the size = 0, then make head and tail point to temp, which is again O(1). Else, we are updating tails’ next as temp and updating tail as temp, which is O(1) again. Also, updating size by 1 is constant O(1) operation.

Hence, the total time complexity is ***O(1)*** only.

**Space Complexity** -

We are creating only ***one*** node which has data and a pointer to the next node, hence auxiliary space required is constant O(1).

Note that the space complexity is equal to the size of the linked list, i.e. O(n), but we are required to find the extra space taken by the addLast function only.

***Extra Gyaan (Knowledge):***

* In this linked list problem, we were maintaining the tail pointer, and just adding the new node as tails’ next node.
* But, what if we were ***only given the head node*** of the linked list? We would have to traverse the entire list to find the tail node and then append the new node after tail.
* It would have made addLast as *O(n)* operation, as we first would have required traversing through the linked list, which is of size n.

Don’t worry, we will see how to traverse a linked list and display it in the next problem. I hope you enjoyed solving the problem with me. The next problem waiting for you is to [display](https://www.pepcoding.com/resources/online-java-foundation/linked-lists/display-linked-list-official/ojquestion) an entire linked list.

***Suggestion***:

Please try to be consistent in your daily coding practice routine. Giving a considerable amount of effort consistently for a long duration (5-6 months) will be more productive than being rigorous and trying to grab all the knowledge in just 2-3 weeks.

Contributor : [Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)