Dear reader, welcome to the article on the problem named **‘**[**Transform to Left Cloned Tree’**](https://www.pepcoding.com/resources/online-java-foundation/binary-tree/transform-to-left-cloned-tree-official/ojquestion).This problem and the next problem ‘[**Transform to Normal from Left-Cloned Tree**](https://www.pepcoding.com/resources/online-java-foundation/binary-tree/transform-to-normal-from-left-cloned-tree-official/ojquestion)’ can help you understand how recursion works in binary trees.

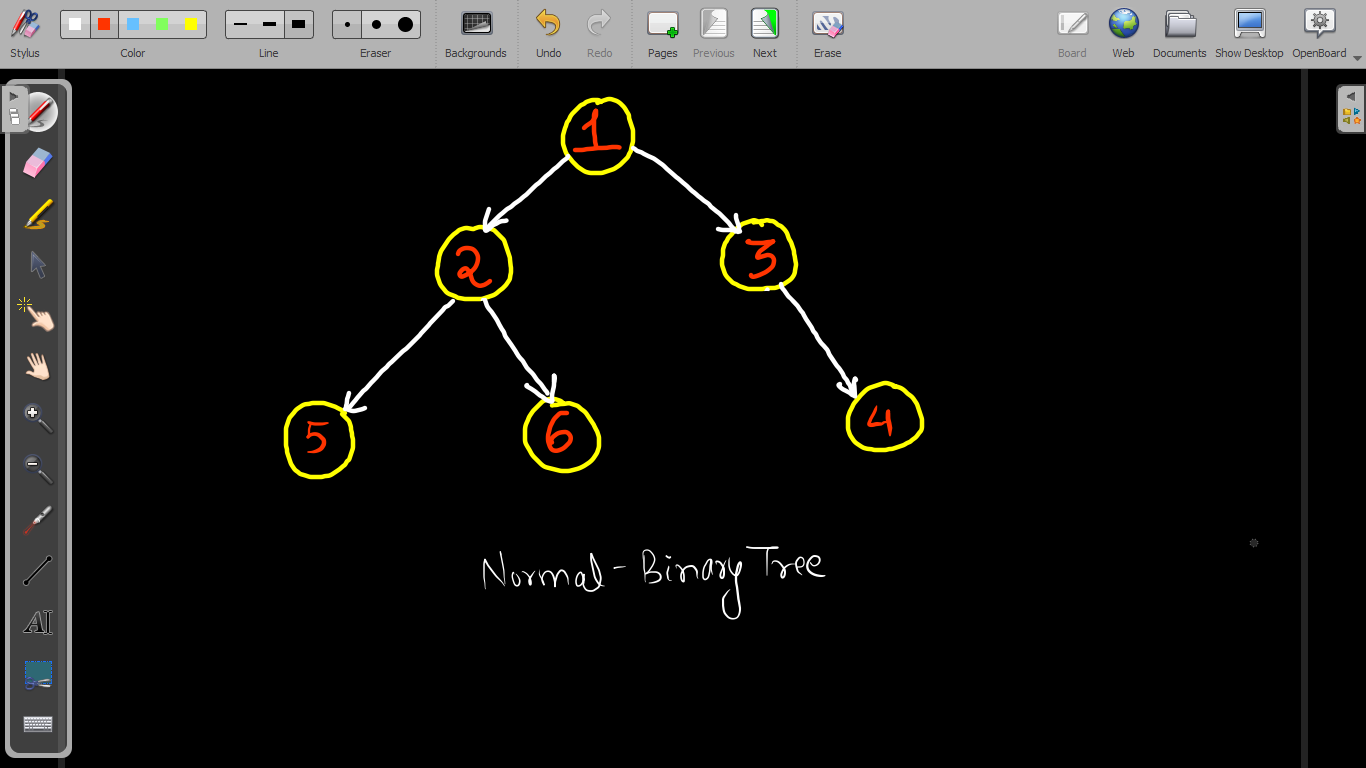
***Problem Statement:***

* You are given a partially written BinaryTree class. (Input and Output is managed for you.)
* You are required to complete the body of ***createLeftCloneTree*** function. The function is expected to create a new node for every node equal in value to it and inserted between itself and it's left child.

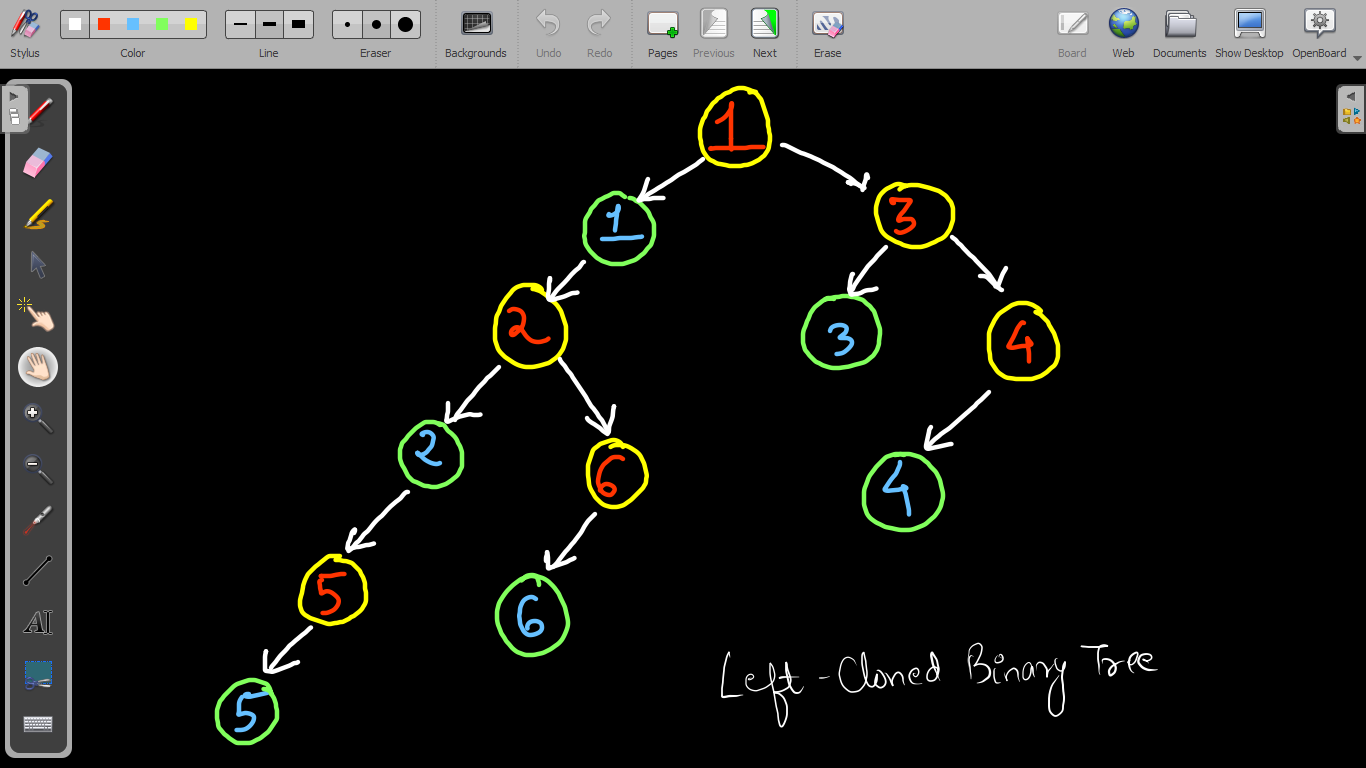
For more details, check out the [question video](https://www.youtube.com/watch?v=jEYKDu4iyik&list=TLGGi2aSoBU8E60wNTA3MjAyMQ).

***Example:***

Input:



Output:



***Solution:***

I told you that we can achieve this task easily using the help of recursion. So let us develop the ***low-level thinking*** by defining our expectation, faith and how we would meet our expectation with faith.

**Expectation**: I need my entire tree rooted at *node* to transform into a left cloned tree.

**createLeftCloneTree(root)**

**Faith**: I will have faith on the left and right subtrees that they know how to transform into a left-cloned tree.

**leftcloned = createLeftCloneTree(root.left)**

**rightcloned = createLeftCloneTree(root.right)**

**Meeting Expectation using Faith:**

Now, we have left-cloned tree of left subtree in *leftcloned* and left-cloned tree of right subtree in *rightcloned* nodes.

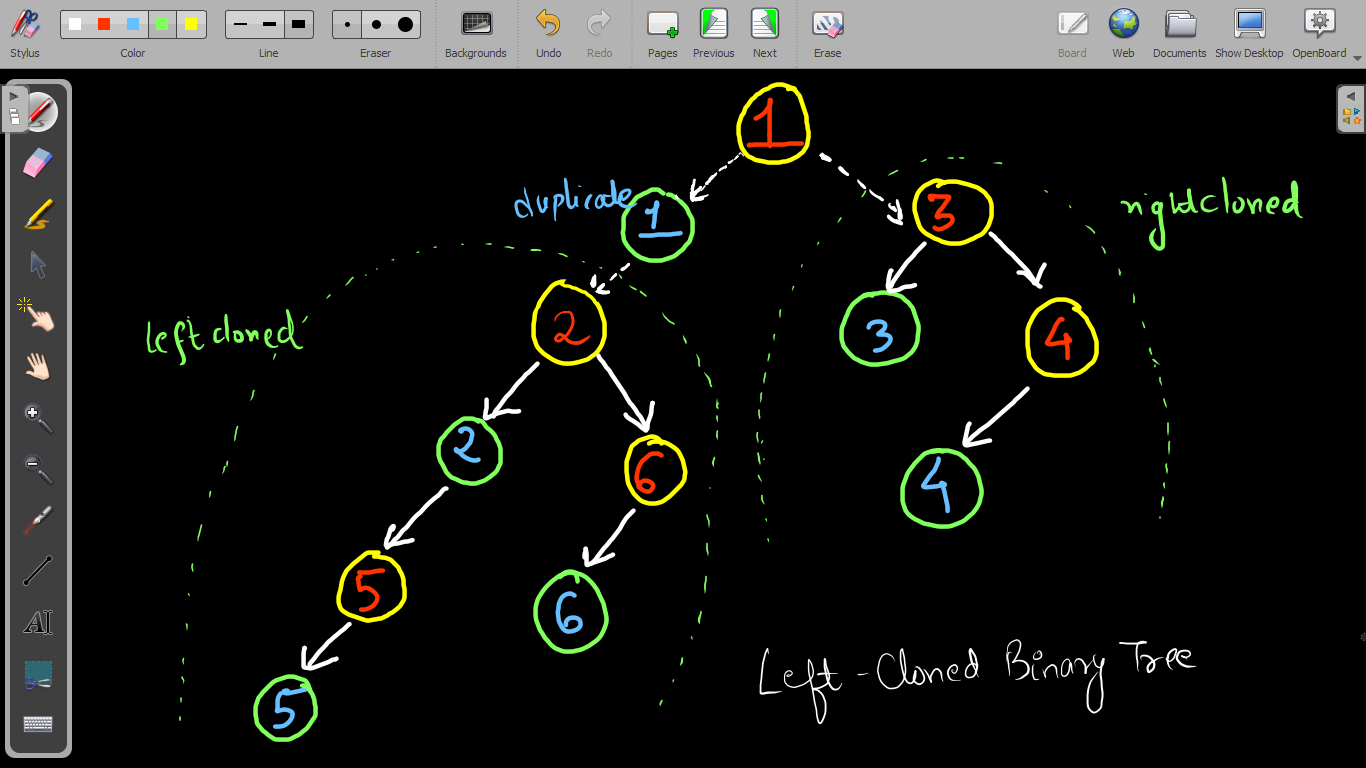
So, we need to do some work in ***POSTORDER*** in order to left-clone the entire tree rooted at *node*.

First, we will create a duplicate node of the root node with the same value. Since the inserted node should be sandwiched between the root node and the left subtree, we will make the left child of this node as *leftcloned*, and update the root node’s left child as this new node.

**node.left = new Node(node.data, leftcloned, null)**

Since, the right subtree will not contain the duplicate node of root, hence we can directly update the root node’s right as *rightcloned*.

**node.right = rightcloned**

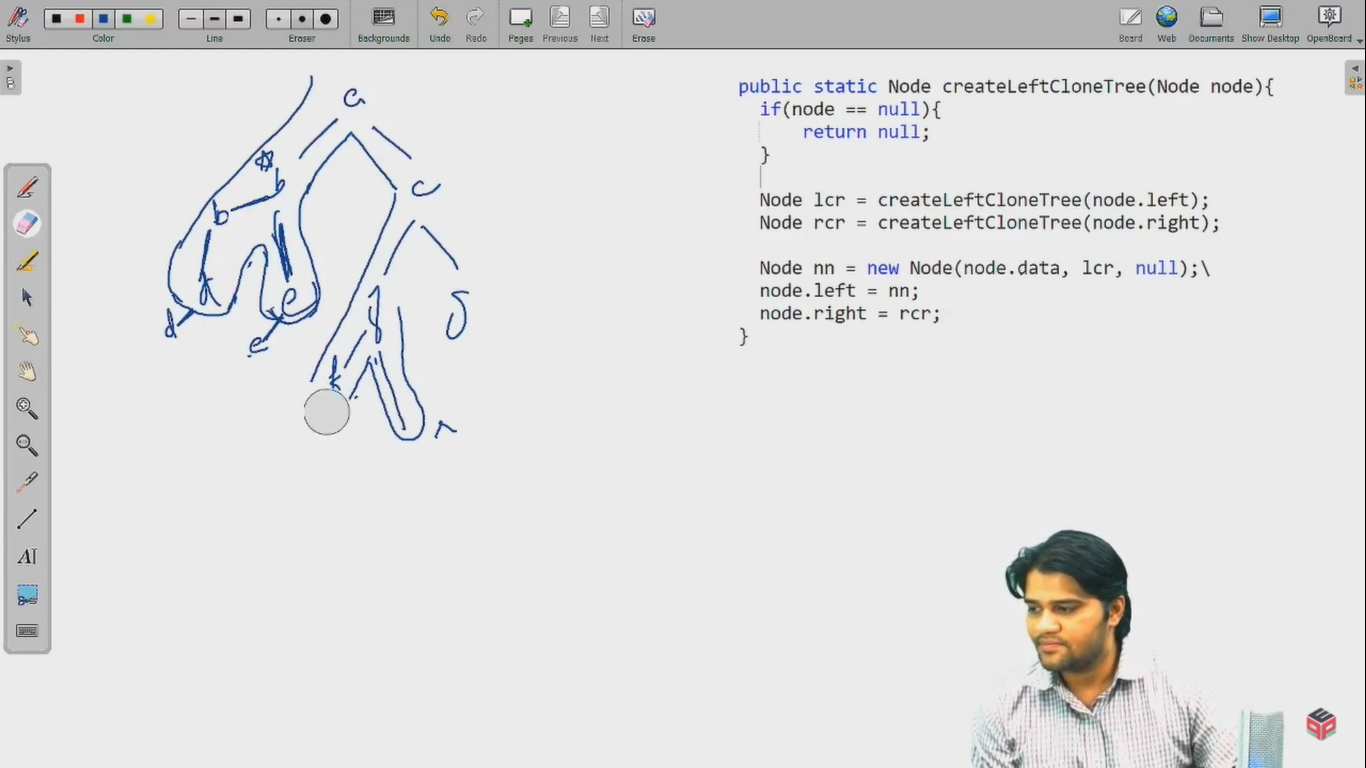


Now, simply return the root *node*: **return node**

What should be the ***base case***?

If the root node is a ***null*** node, then simply return null, as we cannot duplicate a null node.

***Pseudo Code/ Algorithm***



***Implementation***

*Note*: Before reading the Code, we recommend that you must try to come up with the solution on your own. Now, hoping that you have tried by yourself, here is the Java code.

import java.io.\*;

import java.util.\*;

public class Main {

public static class Node {

int data;

Node left;

Node right;

Node(int data, Node left, Node right) {

this.data = data;

this.left = left;

this.right = right;

}

}

public static class Pair {

Node node;

int state;

Pair(Node node, int state) {

this.node = node;

this.state = state;

}

}

public static Node construct(Integer[] arr) {

Node root = new Node(arr[0], null, null);

Pair rtp = new Pair(root, 1);

Stack<Pair> st = new Stack<>();

st.push(rtp);

int idx = 0;

while (st.size() > 0) {

Pair top = st.peek();

if (top.state == 1) {

idx++;

if (arr[idx] != null) {

top.node.left = new Node(arr[idx], null, null);

Pair lp = new Pair(top.node.left, 1);

st.push(lp);

} else {

top.node.left = null;

}

top.state++;

} else if (top.state == 2) {

idx++;

if (arr[idx] != null) {

top.node.right = new Node(arr[idx], null, null);

Pair rp = new Pair(top.node.right, 1);

st.push(rp);

} else {

top.node.right = null;

}

top.state++;

} else {

st.pop();

}

}

return root;

}

public static void display(Node node) {

if (node == null) {

return;

}

String str = "";

str += node.left == null ? "." : node.left.data + "";

str += " <- " + node.data + " -> ";

str += node.right == null ? "." : node.right.data + "";

System.out.println(str);

display(node.left);

display(node.right);

}

public static Node createLeftCloneTree(Node node){

if(node == null){

return null;

}

Node leftcloned = createLeftCloneTree(node.left);

Node rightcloned = createLeftCloneTree(node.right);

node.left = new Node(node.data, leftcloned, null);

node.right = rightcloned;

return node;

}

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

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

int n = Integer.parseInt(br.readLine());

Integer[] arr = new Integer[n];

String[] values = br.readLine().split(" ");

for (int i = 0; i < n; i++) {

if (values[i].equals("n") == false) {

arr[i] = Integer.parseInt(values[i]);

} else {

arr[i] = null;

}

}

Node root = construct(arr);

root = createLeftCloneTree(root);

display(root);

}

}

This code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=TO7trQloRXc&list=TLGGDF6uAn_YS-4wNTA3MjAyMQ). Do check it out to understand the concept completely.

* What is the ***time complexity*** of the above code?

We are traversing the entire tree and duplicating each node, hence the total time complexity is ***O(n)*** where n = number of nodes in the tree.

* What is the ***space complexity*** of the above code?

If we take the space taken by the duplicate nodes created, then the space complexity will be ***O(n)***.

Also, the recursion stack space will take ***O(d)*** space where d = maximum depth of the binary tree.

Hope that you liked the article on *Transform to Left-Cloned Tree*.

Subscribe to Pepcoding’s youtube channel for more such amazing video content on Data Structures & Algorithms.

You can suggest any improvements to the article on our telegram channel, or on the youtube channel’s comment section.

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