Binary Tree to LL in C++

```
#include <iostream>
using namespace std;

// TreeNode class definition
class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int x) {
        val = x;
        left = nullptr;
        right = nullptr;
    }
};

class FlattenBinaryTreeToLinkedList {
public:
    void flatten(TreeNode* root) {
        flattenHelper(root);
    }

private:
    TreeNode* flattenHelper(TreeNode* root) {
        if (root == nullptr) return nullptr;

        TreeNode* leftTail = flattenHelper(root->left);
        TreeNode* rightTail = flattenHelper(root->right);

        if (leftTail != nullptr) {
            leftTail->right = root->right;  // Connect the end of the left subtree to the start of the right subtree
            root->right = root->left;       // Move the left subtree to the right
            root->left = nullptr;           // Nullify the left pointer
        }

        // Return the last node in the flattened tree
        if (rightTail != nullptr) {
            return rightTail;
        } else if (leftTail != nullptr) {
            return leftTail;
        } else {
            return root;
        }
    }

public:
    // Utility function to print the flattened tree
    void printFlattenedTree(TreeNode* root) {
        while (root != nullptr) {
            cout << root->val << " ";
            root = root->right;
        }
        cout << endl;
    }
```

Absolutely! Let's dry run your `flatten` function **with a step-by-step table**, using this binary tree:

The goal is to flatten this tree into a **linked list using right pointers** in **pre-order traversal**:

```
1 -> 2 -> 3 -> 4 -> 5 -> 6
```

🔁 Dry Run Table:

Step	Node Visited	Left Subtree Tail	Right Subtree Tail	Action Taken	Resulting Right Chain (Partial)
1	3	`nullptr`	`nullptr`	Leaf node → return `3`	3
2	4	`nullptr`	`nullptr`	Leaf node → return `4`	4
3	2	3	4	Move left to right: 2->left becomes nullptr, 2->right = 3, 3->right = 4	2 → 3 → 4
4	6	`nullptr`	`nullptr`	Leaf node → return `6`	6
5	5	`nullptr`	6	No left to move → do nothing, return 6	5 → 6
6	1	4 (tail of 2)	6 (tail of 5)	Move left to right: 1->right = 2, 4->right = 5 (attach 5 to end	1 → 2 → 3 → 4 → 5 → 6

```
    // Function to delete a binary tree to free
memory
    void deleteTree(TreeNode* root) {
        if (root == nullptr) return;
        deleteTree(root->left);
        deleteTree(root->right);
        delete root;
    }
};

int main() {
    FlattenBinaryTreeToLinkedList solution;

    // Creating a sample binary tree:
    //     1
    //    / \
    //   2   5
    //  / \   \
    // 3   4   6
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(5);
    root->left->left = new TreeNode(3);
    root->left->right = new TreeNode(4);
    root->right->right = new TreeNode(6);

    cout << "Original Tree:" << endl;
    solution.printFlattenedTree(root); // This will
just print the root node, as the tree is not
flattened yet

    solution.flatten(root);

    cout << "Flattened Tree:" << endl;
    solution.printFlattenedTree(root);

    // Clean up memory
    solution.deleteTree(root);

    return 0;
}
```

				of 2 chain)	

🔚 Final Result:

The flattened tree is:

```
1 → 2 → 3 → 4 → 5 → 6 → nullptr
```

Output:-

```
1 → 2 → 3 → 4 → 5 → 6 → nullptr
```