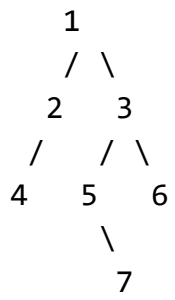
Problem Statement:

You are given a binary tree where each node contains an integer value. Write a C# program that identifies and returns the rightmost node at each level of the binary tree.

Example:

Consider the binary tree below:



For the given binary tree, the program should output: `[1, 3, 6, 7]`.

Explanation:

- The rightmost node at the first level is `1`.
- The rightmost node at the second level is `3`.
- The rightmost node at the third level is `6`.
- The rightmost node at the fourth level is `7`.

Requirements:

1. Implement a class `TreeNode` to represent nodes in the binary tree.
2. Write a method `List<int> GetRightmostNodes(TreeNode root)` that takes the root of the binary tree as input and returns a list of integers representing the rightmost nodes at each level.

Constraints:

- The binary tree can have up to `10^4` nodes.
- Node values are unique and can be any integer.

Notes:

- Consider edge cases like an empty tree or a tree with only one node.

Sample Input:

```
TreeNode root = new TreeNode(1)
{
    left = new TreeNode(2)
    {
        left = new TreeNode(4)
    },
    right = new TreeNode(3)
    {
        left = new TreeNode(5)
        {
            right = new TreeNode(7)
        },
        right = new TreeNode(6)
    }
};
```

Sample Output:

```
List<int> result = GetRightmostNodes(root);
// Output: [1, 3, 6, 7]
```

Test Cases:

Here are some test cases to validate your C# code for detecting the rightmost nodes in a binary tree, covering various edge cases:

Test Case 1: Empty Tree

Input:

```
TreeNode root = null;
```

Expected Output:

```
List<int> result = GetRightmostNodes(root);
// Output: []
```

Explanation:
An empty tree should return an empty list as there are no nodes.

Test Case 2: Single Node Tree

Input:

```
TreeNode root = new TreeNode(1);
```

Expected Output:

```
List<int> result = GetRightmostNodes(root);
// Output: [1]
```

Explanation:
A tree with only one node should return that node as the only rightmost node.

Test Case 3: Full Binary Tree

Input:

```
TreeNode root = new TreeNode(1)
{
    left = new TreeNode(2)
    {
        left = new TreeNode(4),
        right = new TreeNode(5)
    },
    right = new TreeNode(3)
    {
```

```
        left = new TreeNode(6),
        right = new TreeNode(7)
    }
};
```

Expected Output:

```
List<int> result = GetRightmostNodes(root);
// Output: [1, 3, 7]
```

Explanation:
In this full binary tree, the rightmost nodes at each level are 1, 3, and 7.

Test Case 4: Right-Skewed Tree

Input:

```
TreeNode root = new TreeNode(1)
{
    right = new TreeNode(2)
    {
        right = new TreeNode(3)
        {
            right = new TreeNode(4)
        }
    }
};
```

Expected Output:

```
List<int> result = GetRightmostNodes(root);
// Output: [1, 2, 3, 4]
```

Explanation:
In a right-skewed tree, each level has only one node, which is also the rightmost node.

Test Case 5: Left-Skewed Tree

Input:

```
TreeNode root = new TreeNode(1)
{
    left = new TreeNode(2)
    {
        left = new TreeNode(3)
        {
            left = new TreeNode(4)
        }
    }
};
```

Expected Output:

```
List<int> result = GetRightmostNodes(root);
// Output: [1, 2, 3, 4]
```

Explanation:
In a left-skewed tree, even though the tree leans left, the rightmost node at each level is the only node present.

Test Case 6: Complete Binary Tree with Missing Nodes

Input:

```
TreeNode root = new TreeNode(1)
{
    left = new TreeNode(2)
    {
        left = new TreeNode(4),
        right = new TreeNode(5)
    },
```

```
    right = new TreeNode(3)
    {
        right = new TreeNode(7)
    }
};
```

Expected Output:

```
List<int> result = GetRightmostNodes(root);
// Output: [1, 3, 7]
```

Explanation:
This tree is missing some nodes on the right side at different levels, but the rightmost nodes are still correctly identified.

Test Case 7: Tree with Single Child Nodes

Input:

```
TreeNode root = new TreeNode(1)
{
    left = new TreeNode(2)
    {
        right = new TreeNode(4)
    },
    right = new TreeNode(3)
};
```

Expected Output:

```
List<int> result = GetRightmostNodes(root);
// Output: [1, 3, 4]
```

Explanation:
Here, the tree has some nodes with only one child, and the rightmost nodes should still be detected correctly.