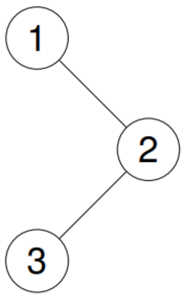
**Name – Shreyansh Vishnoi UID – 22BCS15373**
**Section – 620-B**

## DAY – 5

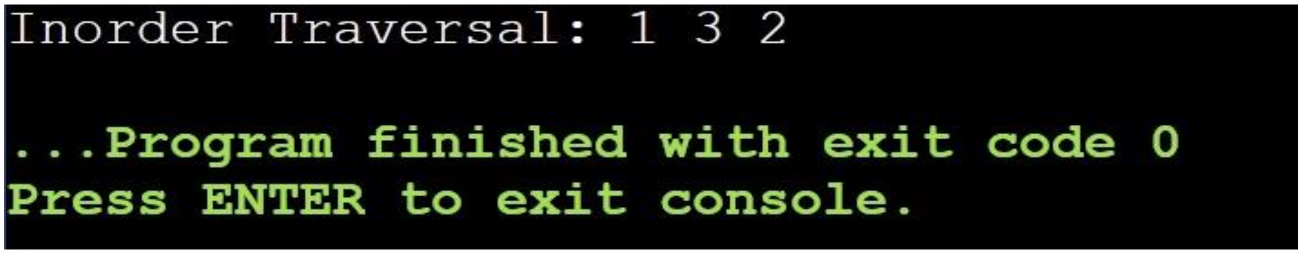
**Binary Tree Inorder Traversal**

**1. Given the root of a binary tree, return the inorder traversal of its nodes' values.**

**Example 1:**
**Input: root = [1,null,2,3]**
**Output: [1,3,2]**
**Explanation:**



```
#include <iostream>
#include <vector>

using namespace std;

// Definition for a binary tree node.
struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode* right;

   TreeNode() : val(0), left(nullptr), right(nullptr) {}
   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
   TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};

class Solution {
public:
   vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorderHelper(root, result);
        return result;
    }

private:
    void inorderHelper(TreeNode* node, vector<int>& result) {
        if (node != nullptr) {
            inorderHelper(node->left, result); // Traverse left subtree
            result.push_back(node->val);      // Visit the root
            inorderHelper(node->right, result); // Traverse right subtree
        }
    }
};

int main() {
    // Example: Constructing the tree [1, null, 2, 3]
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);

    Solution solution;
    vector<int> result = solution.inorderTraversal(root);

    // Printing the result
    cout << "Inorder Traversal: ";
    for (int val : result) {
        cout << val << " ";
    }

    // Output: Inorder Traversal: 1 3 2
    return 0;
}
```

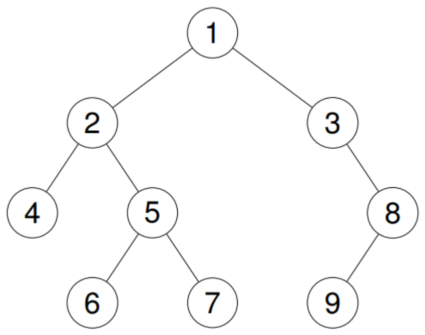


```
Inorder Traversal: 1 3 2

...Program finished with exit code 0
Press ENTER to exit console.
```

**2. Input: root = [1,2,3,4,5, null, 8 , null,null,6,7,9]**

**Output: [4,2,6,5,7,1,3,9,8]**

**Explanation:**

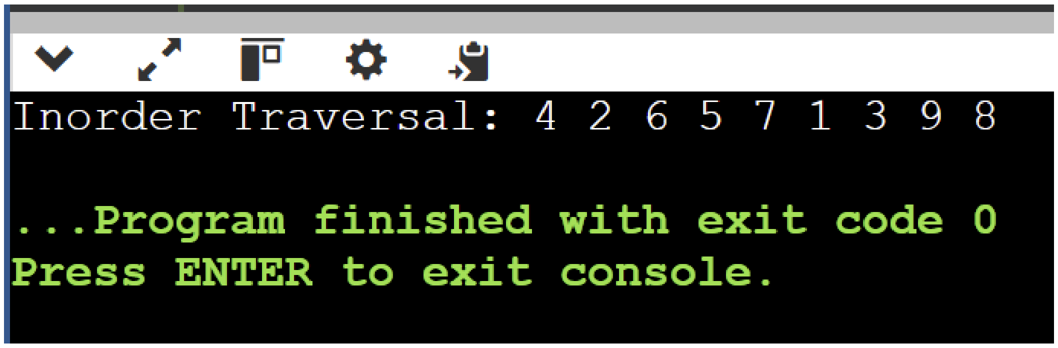


**Constraints:**

**The number of nodes in the tree is in the range [0, 100].**

**-100 <= Node.val <= 100**

```
#include <iostream>
#include <vector>
#include <queue>
#include <memory>
using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;

    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};

// Helper function to build a binary tree from a vector
TreeNode* buildTree(const vector<int>& nodes) {
    if (nodes.empty() || nodes[0] == -1) return nullptr;

    TreeNode* root = new TreeNode(nodes[0]);
    queue<TreeNode*> q;
    q.push(root);

    size_t i = 1;
    while (i < nodes.size()) {
        TreeNode* current = q.front();
        q.pop();

        if (i < nodes.size() && nodes[i] != -1) {
            current->left = new TreeNode(nodes[i]);
            q.push(current->left);
        }
        i++;

        if (i < nodes.size() && nodes[i] != -1) {
            current->right = new TreeNode(nodes[i]);
            q.push(current->right);
        }
        i++;
    }

    return root;
}

class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorderHelper(root, result);
        return result;
    }

private:
    void inorderHelper(TreeNode* node, vector<int>& result) {
        if (node != nullptr) {
            inorderHelper(node->left, result); // Traverse left subtree
            result.push_back(node->val);      // Visit the root
            inorderHelper(node->right, result); // Traverse right subtree
        }
    }
};

int main() {
    // Example Input: [1,2,3,4,5,null,8,null,null,6,7,9]
    vector<int> nodes = {1, 2, 3, 4, 5, -1, 8, -1, -1, 6, 7, 9};

    // Build the binary tree
    TreeNode* root = buildTree(nodes);

    // Solution and traversal
    Solution solution;
    vector<int> result = solution.inorderTraversal(root);

    // Print the output
    cout << "Inorder Traversal: ";
    for (int val : result) {
        cout << val << " ";
    }

    // Output: [4, 2, 6, 5, 7, 1, 3, 9, 8]
    return 0;
```



```
Inorder Traversal: 4 2 6 5 7 1 3 9 8

...Program finished with exit code 0
Press ENTER to exit console.
```

**3. Count Complete Tree Nodes Given the root of a complete binary tree, return the number of the nodes in the tree.**

**According to Wikipedia, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.**

**Design an algorithm that runs in less than O(n) time complexity**

**Example 1:**
**Input: root = [1,2,3,4,5,6]**

**Output: 6**

**Example 2:**

**Input: root = []**

**Output: 0**

**Example 3:**

**Input: root = [1]**

**Output: 1**

**Constraints:**

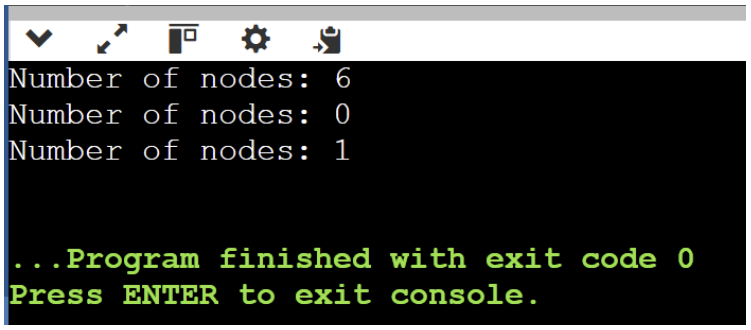
**The number of nodes in the tree is in the range [0, 5 * 104].**

**0 <= Node.val <= 5 * 104**

**The tree is guaranteed to be complete.**

**Reference: https://leetcode.com/problems/count-complete-tree-nodes/description/**

```
#include <iostream>
#include <cmath> // For pow

using namespace std;

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;

    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};

class Solution {
public:
    int countNodes(TreeNode* root) {
        if (!root) return 0;

        int leftHeight = getLeftHeight(root);
        int rightHeight = getRightHeight(root);

        if (leftHeight == rightHeight) {
            // The tree is a perfect binary tree
            return (1 << leftHeight) - 1; // 2^height - 1
        } else {
            // Recur for left and right subtrees
            return 1 + countNodes(root->left) + countNodes(root->right);
        }
    }

private:
    int getLeftHeight(TreeNode* node) {
        int height = 0;
        while (node) {
            height++;
            node = node->left;
        }
        return height;
    }

    int getRightHeight(TreeNode* node) {
        int height = 0;
        while (node) {
            height++;
            node = node->right;
        }
        return height;
    }
};

int main() {
    // Example 1: [1,2,3,4,5,6]
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);

    Solution solution;
    cout << "Number of nodes: " << solution.countNodes(root) << endl;
    // Output: 6

    // Example 2: []
    cout << "Number of nodes: " << solution.countNodes(nullptr) << endl;
    // Output: 0

    // Example 3: [1]
    TreeNode* singleNodeTree = new TreeNode(1);
    cout << "Number of nodes: " << solution.countNodes(singleNodeTree) << endl;
    // Output: 1

    return 0;
```



```
Number of nodes: 6
Number of nodes: 0
Number of nodes: 1


...Program finished with exit code 0
Press ENTER to exit console.
```

## 4. Binary Tree - Find Maximum Depth

**A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.**

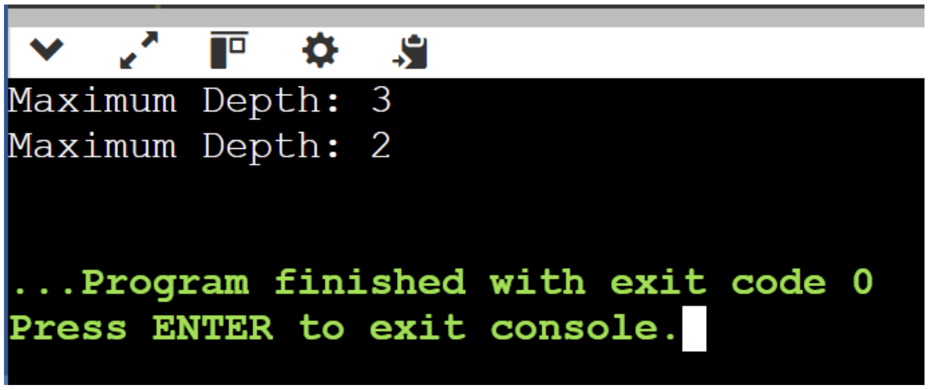
**Example 1:**
**Input: [3,9,20,null,null,15,7]**
**Output: 3**

**Example 2:**
**Input: [1,null,2]**
**Output: 2**

**Constraints:**
**The number of nodes in the tree is in the range [0, 104].**
**-100 <= Node.val <= 100**

**Reference:** **https://leetcode.com/problems/maximum-depth-of-binary-tree/description/**#include
<iostream>

```
#include <algorithm> // For max

using namespace std;

// Definition for a binary tree node.

struct TreeNode {

    int val;

    TreeNode* left;

    TreeNode* right;

    TreeNode() : val(0), left(nullptr), right(nullptr) {}

    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

    TreeNode(int x, TreeNode* left, TreeNode* right) :

val(x), left(left), right(right) {}

};

class Solution {

public:

    int maxDepth(TreeNode* root) {
        if (!root) return 0; // Base case: if the tree is empty,
depth is 0
        return 1 + max(maxDepth(root->left), maxDepth(root-
>right));
    }
};
int main() {
    // Example 1: [3,9,20,null,null,15,7]
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);
    Solution solution;
    cout << "Maximum Depth: " <<
solution.maxDepth(root) << endl;
    // Output: 3
    // Example 2: [1,null,2]
    TreeNode* root2 = new TreeNode(1);
    root2->right = new TreeNode(2);
    cout << "Maximum Depth: " <<
solution.maxDepth(root2) << endl;
    // Output: 2
    return 0;
```


```
Maximum Depth: 3
Maximum Depth: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

5. **Binary Tree Preorder Traversal**

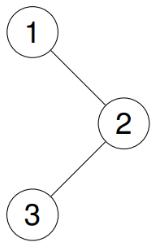
**Given the root of a binary tree, return the preorder traversal of its nodes' values.**

**Example 1:**

**Input: root = [1,null,2,3]**

**Output: [1,2,3]**

**Explanation:**

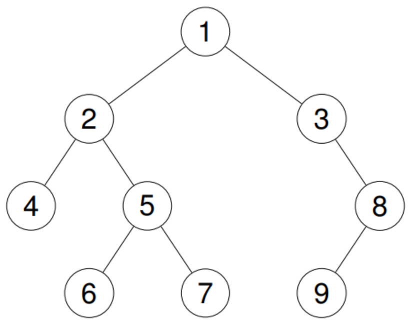


**Example 2:**

**Input: root = [1,2,3,4,5,null,8,null,null,6,7,9]**

**Output: [1,2,4,5,6,7,3,8,9]**

**Explanation:**



**Constraints:**

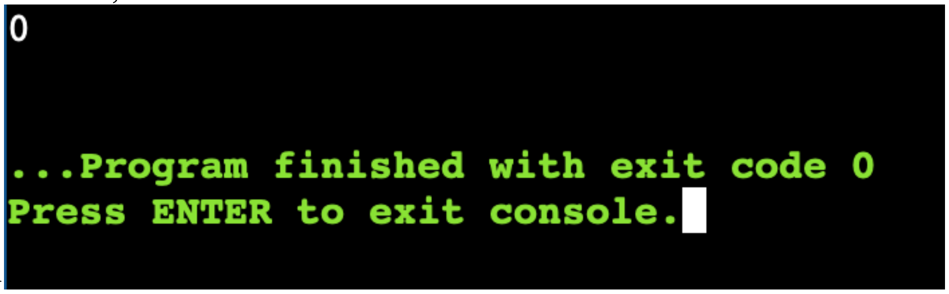
**The number of nodes in the tree is in the range [1, 100].**

**1 <= Node.val <= 1000**

**Reference:** [https://leetcode.com/problems/binary-tree-preorder-traversal/description/](https://leetcode.com/problems/binary-tree-preorder-traversal/description/)

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

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;

    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};

class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> result;
        preorderHelper(root, result);
        return result;
    }

private:
    void preorderHelper(TreeNode* node, vector<int>& result) {
        if (node) {
            result.push_back(node->val); // Visit the root
            preorderHelper(node->left, result); // Traverse the left subtree
            preorderHelper(node->right, result); // Traverse the right subtree
        }
    }
};
```

```
int main() {
    // Example 1: [1, null, 2, 3]
    TreeNode* root1 = new TreeNode(1);
    root1->right = new TreeNode(2);
    root1->right->left = new TreeNode(3);

    Solution solution;
    vector<int> result1 = solution.preorderTraversal(root1);

    cout << "Preorder Traversal Example 1: ";
    for (int val : result1) {
        cout << val << " ";
    }
    cout << endl;

    // Example 2: [1, 2, 3, 4, 5, null, 8, null, null, 6, 7, 9]
    TreeNode* root2 = new TreeNode(1);
    root2->left = new TreeNode(2);
    root2->right = new TreeNode(3);
    root2->left->left = new TreeNode(4);
    root2->left->right = new TreeNode(5);
    root2->right->right = new TreeNode(8);
    root2->left->right->left = new TreeNode(6);
    root2->left->right->right = new TreeNode(7);
    root2->right->right->left = new TreeNode(9);

    vector<int> result2 = solution.preorderTraversal(root2);

    cout << "Preorder Traversal Example 2: ";
    for (int val : result2) {
        cout << val << " ";
    }
    cout << endl;

    return 0;
```



```
0


...Program finished with exit code 0
Press ENTER to exit console.
```

```
}
```

**6. We are given an array asteroids of integers representing asteroids in a row.**

**For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.**

**Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.**

**Example 1:**

**Input:** asteroids = [5,10,-5]

**Output:** [5,10]

**Explanation:** The 10 and -5 collide resulting in 10. The 5 and 10 never collide.

**Example 2:**

**Input:** asteroids = [8,-8]

**Output: []**

**Explanation:** The 8 and -8 collide exploding each other.

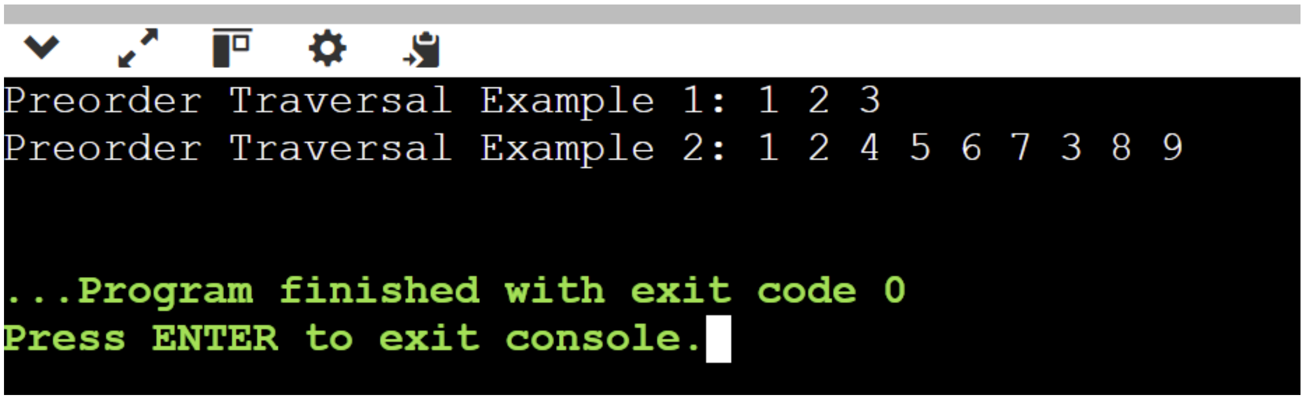
**Example 3:**

**Input:** asteroids = [10,2,-5]

**Output:** [10]

```
#include <iostream>

#include <vector>

#include <stack>
```

```
}

using namespace std;

vector<int> asteroidCollision(vector<int>& asteroids) {
    stack<int> st; // Stack to simulate asteroid collisions

    for (int asteroid : asteroids) {
        bool destroyed = false;

        while (!st.empty() && asteroid < 0 && st.top() > 0) {
            if (abs(asteroid) == st.top()) {
                st.pop(); // Both asteroids explode
                destroyed = true;
                break;
            } else if (abs(asteroid) > st.top()) {
                st.pop(); // Top asteroid is smaller and explodes
            } else {
                destroyed = true; // Current asteroid is smaller and explodes
                break;
            }
        }

        if (!destroyed) st.push(asteroid); // Push the current asteroid if not destroyed
    }
    vector<int> result(st.size());
    for (int i = st.size() - 1; i >= 0; i--) {
        result[i] = st.top(); // Transfer stack to result in reverse order
        st.pop();
    }

    return result;
}

int main() {
    vector<int> asteroids1 = {5, 10, -5};
    vector<int> asteroids2 = {8, -8};
    vector<int> asteroids3 = {10, 2, -5};

    vector<int> result1 = asteroidCollision(asteroids1);
    vector<int> result2 = asteroidCollision(asteroids2);
    vector<int> result3 = asteroidCollision(asteroids3);

    for (int x : result1) cout << x << " "; // Output: 5 10
    cout << endl;
    for (int x : result2) cout << x << " "; // Output: (empty)
    cout << endl;
    for (int x : result3) cout << x << " "; // Output: 10

    cout << endl;



    return 0;

}
```



```
Preorder Traversal Example 1: 1 2 3
Preorder Traversal Example 2: 1 2 4 5 6 7 3 8 9


...Program finished with exit code 0
Press ENTER to exit console.
```

**8 Binary Tree - Sum of All Nodes**
**Given the root of a binary tree, you need to find the sum of all the node values in the binary tree.**

**Example 1:**
**Input: root = [1, 2, 3, 4, 5, null, 6]**

**Output: 21**

**Explanation: The sum of all nodes is 1 + 2 + 3 + 4 + 5 + 6 = 21.**

**Example 2:**
**Input: root = [5, 2, 6, 1, 3, 4, 7]**

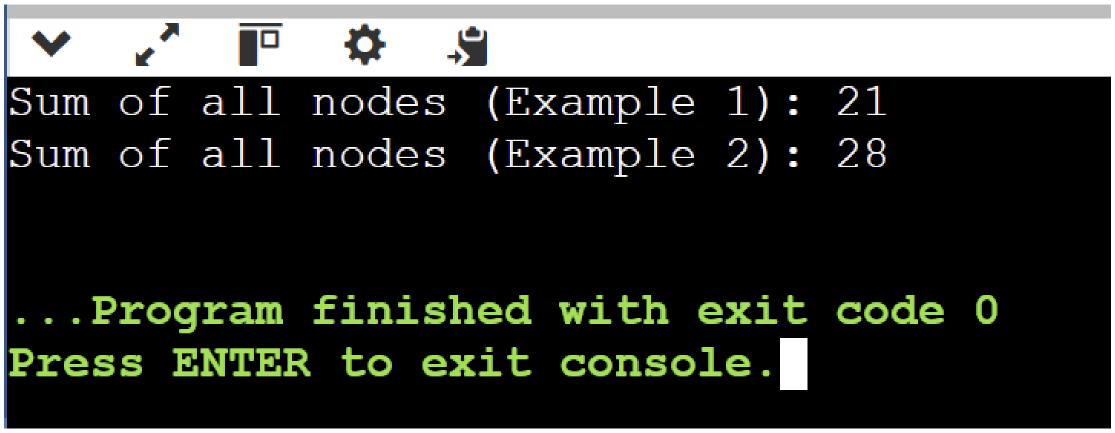
**Output: 28**

**Explanation: The sum of all nodes is 5 + 2 + 6 + 1 + 3 + 4 + 7 = 28.**

**Reference: [http://leetcode.com/problems/sum-of-left-leaves/](http://leetcode.com/problems/sum-of-left-leaves/)**

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

// Definition for a binary tree node.
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;

    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};

class Solution {
public:
    int sumOfNodes(TreeNode* root) {
        if (!root) return 0; // Base case: if the tree is empty, sum is 0
        return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
    }
};

int main() {
    // Example 1: [1, 2, 3, 4, 5, null, 6]
    TreeNode* root1 = new TreeNode(1);
    root1->left = new TreeNode(2);
```

```
root1->right = new TreeNode(3);
root1->left->left = new TreeNode(4);
root1->left->right = new TreeNode(5);
root1->right->right = new TreeNode(6);

Solution solution;
cout << "Sum of all nodes (Example 1): " << solution.sumOfNodes(root1) << endl;
// Output: 21

// Example 2: [5, 2, 6, 1, 3, 4, 7]
TreeNode* root2 = new TreeNode(5);
root2->left = new TreeNode(2);
root2->right = new TreeNode(6);
root2->left->left = new TreeNode(1);
root2->left->right = new TreeNode(3);
root2->right->left = new TreeNode(4);
root2->right->right = new TreeNode(7);

cout << "Sum of all nodes (Example 2): " << solution.sumOfNodes(root2) << endl;
// Output: 28

return 0;
```



```
Sum of all nodes (Example 1): 21
Sum of all nodes (Example 2): 28


...Program finished with exit code 0
Press ENTER to exit console.
```

**10. Same Tree**

**Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.**

**Example 1:**
**Input: p = [1,2,3], q = [1,2,3]**
**Output: true**

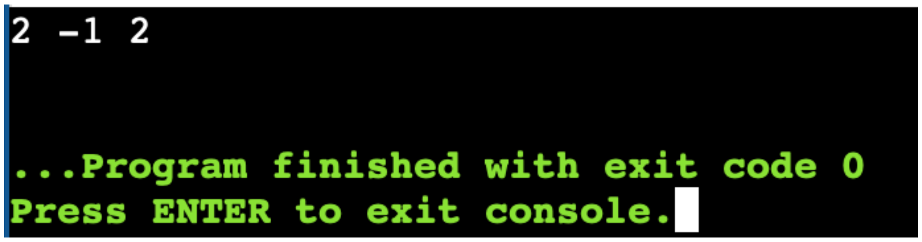
**Example 2:**
**Input: p = [1,2], q = [1,null,2]**
**Output: false**

**Constraints:**
**The number of nodes in both trees is in the range [0, 100].**
**-104 <= Node.val <= 104**

**Reference: [https://leetcode.com/problems/same-tree/description/?envType=study-plan-v2&envId=top-interview-150](https://leetcode.com/problems/same-tree/description/?envType=study-plan-v2&envId=top-interview-150)**

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

// Definition for a binary tree node.
struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode* right;

   TreeNode() : val(0), left(nullptr), right(nullptr) {}
   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
   TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};
```

```
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        // If both trees are empty, they are the same
        if (!p && !q) return true;
        // If one is empty and the other is not, they are not the same
        if (!p || !q) return false;
        // If the values of the current nodes are different, they are not the same
        if (p->val != q->val) return false;

        // Recursively check the left and right subtrees
        return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
    }
};

int main() {
    // Example 1: p = [1, 2, 3], q = [1, 2, 3]
    TreeNode* p1 = new TreeNode(1);
    p1->left = new TreeNode(2);
    p1->right = new TreeNode(3);

    TreeNode* q1 = new TreeNode(1);
    q1->left = new TreeNode(2);
    q1->right = new TreeNode(3);

    Solution solution;
    cout << "Are trees p1 and q1 the same? " << (solution.isSameTree(p1, q1) ? "true" : "false") <<
endl;
    // Output: true

    // Example 2: p = [1, 2], q = [1, null, 2]
    TreeNode* p2 = new TreeNode(1);
    p2->left = new TreeNode(2);

    TreeNode* q2 = new TreeNode(1);
    q2->right = new TreeNode(2);

    cout << "Are trees p2 and q2 the same? " << (solution.isSameTree(p2, q2) ? "true" : "false") <<
endl;
    // Output: false

    return 0;
}
```


```
2 -1 2


...Program finished with exit code 0
Press ENTER to exit console.
```