

# Domain Winter Camp DAY-6

Name :- Priti

Section/Group:-901-Kpit-B

UID :- 22BCS12729

## Problem 1

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 // Definition for a binary tree node.
6 struct TreeNode {
7     int val;
8     TreeNode* left;
9     TreeNode* right;
10    TreeNode() : val(0), left(nullptr), right(nullptr) {}
11    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
12    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
13 };
14
15 class Solution {
16 public:
17     vector<int> inorderTraversal(TreeNode* root) {
18         vector<int> result;
19         inorderHelper(root, result);
20         return result;
21     }
22
23 private:
24     void inorderHelper(TreeNode* node, vector<int>& result) {
25         if (!node) return;
26         inorderHelper(node->left, result); // Traverse Left subtree
27         result.push_back(node->val);       // Visit node
28         inorderHelper(node->right, result); // Traverse right subtree
29     }
30 }
```

input

Inorder Traversal: 1 3 2

## Problem 2

```

1  #include <iostream>
2  using namespace std;
3
4  // Definition for a binary tree node.
5  struct TreeNode {
6      int val;
7      TreeNode *left;
8      TreeNode *right;
9      TreeNode() : val(0), left(nullptr), right(nullptr) {}
10     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
11     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
12 };
13
14 // Function to calculate the depth of the tree.
15 int getDepth(TreeNode* node) {
16     int depth = 0;
17     while (node) {
18         depth++;
19         node = node->left;
20     }
21     return depth;
22 }
23
24 int countNodes(TreeNode* root) {
25     if (!root) return 0;
26
27     int leftDepth = getDepth(root->left);
28     int rightDepth = getDepth(root->right);

```

▼ 🔍 📄 ⚙️ 🗑️

input

Number of nodes: 6

### Problem 3

```

3
4 // Definition for a binary tree node.
5 struct TreeNode {
6     int val;
7     TreeNode *left;
8     TreeNode *right;
9     TreeNode() : val(0), left(nullptr), right(nullptr) {}
10    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
11    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
12 };
13
14 // Function to find the maximum depth of the binary tree.
15 int maxDepth(TreeNode* root) {
16     if (!root) return 0; // Base case: if the tree is empty, depth is 0.
17
18     // Recursively find the depth of the left and right subtrees.
19     int leftDepth = maxDepth(root->left);
20     int rightDepth = maxDepth(root->right);
21
22     // Return the maximum depth between the two subtrees plus 1 (for the current node).
23     return max(leftDepth, rightDepth) + 1;
24 }
25
26 int main() {
27     // Example usage
28     TreeNode* root = new TreeNode(3);
29     root->left = new TreeNode(9);
30     root->right = new TreeNode(20);
31     root->right->left = new TreeNode(15);

```



input

Maximum Depth: 3

## Problem 4

```
6 struct TreeNode {
7     int val;
8     TreeNode *left;
9     TreeNode *right;
10    TreeNode() : val(0), left(nullptr), right(nullptr) {}
11    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
12    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
13 };
14
15 // Recursive function to perform preorder traversal.
16 void preorderHelper(TreeNode* root, vector<int>& result) {
17     if (!root) return;
18
19     result.push_back(root->val); // Visit the current node.
20     preorderHelper(root->left, result); // Traverse the left subtree.
21     preorderHelper(root->right, result); // Traverse the right subtree.
22 }
23
24 vector<int> preorderTraversal(TreeNode* root) {
25     vector<int> result;
26     preorderHelper(root, result);
27     return result;
28 }
29
30 int main() {
31     // Example usage
32     TreeNode* root = new TreeNode(1);
33     root->right = new TreeNode(2);
```



input

Preorder Traversal: 1 2 3



## Problem 5

```
4 // Definition for a binary tree node.
5 struct TreeNode {
6     int val;
7     TreeNode *left;
8     TreeNode *right;
9     TreeNode() : val(0), left(nullptr), right(nullptr) {}
10    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
11    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
12 };
13
14 // Recursive function to find the sum of all nodes.
15 int sumOfNodes(TreeNode* root) {
16     if (!root) return 0; // Base case: if the node is null, its sum is 0.
17
18     // Sum the value of the current node and the sum of its left and right subtrees.
19     return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
20 }
21
22 int main() {
23     // Example usage
24     TreeNode* root = new TreeNode(1);
25     root->left = new TreeNode(2);
26     root->right = new TreeNode(3);
27     root->left->left = new TreeNode(4);
28     root->left->right = new TreeNode(5);
29     root->right->right = new TreeNode(6);
30
31     cout << "Sum of all nodes: " << sumOfNodes(root) << endl; // Output: 21
```



input

Sum of all nodes: 21