# Domain Winter Camp DAY-6

## Name :- Abhiraj Patel

## Section/Group:-901-Kpit-B

## UID :- 22BCS11329

### Problem 1

```cpp
#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) return;
        inorderHelper(node->left, result);  // Traverse left subtree
        result.push_back(node->val);       // Visit node
        inorderHelper(node->right, result); // Traverse right subtree
```

input

```
Inorder Traversal: 1 3 2
```

### Problem 2

```cpp
#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) {}
};

// Function to calculate the depth of the tree.
int getDepth(TreeNode* node) {
    int depth = 0;
    while (node) {
        depth++;
        node = node->left;
    }
    return depth;
}

int countNodes(TreeNode* root) {
    if (!root) return 0;

    int leftDepth = getDepth(root->left);
    int rightDepth = getDepth(root->right);
```

input

```
Number of nodes: 6
```

**Problem 3**

```cpp

// 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) {}
};

// Function to find the maximum depth of the binary tree.
int maxDepth(TreeNode* root) {
    if (!root) return 0; // Base case: if the tree is empty, depth is 0.

    // Recursively find the depth of the left and right subtrees.
    int leftDepth = maxDepth(root->left);
    int rightDepth = maxDepth(root->right);

    // Return the maximum depth between the two subtrees plus 1 (for the current node).
    return max(leftDepth, rightDepth) + 1;
}

int main() {
    // Example usage
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
```

input

```
Maximum Depth: 3
```

**Problem 4**

```cpp
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) {}
};

// Recursive function to perform preorder traversal.
void preorderHelper(TreeNode* root, vector<int>& result) {
    if (!root) return;

    result.push_back(root->val);       // Visit the current node.
    preorderHelper(root->left, result); // Traverse the left subtree.
    preorderHelper(root->right, result); // Traverse the right subtree.
}

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

int main() {
    // Example usage
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
```



input

```
Preorder Traversal: 1 2 3
```

**Problem 5**

```cpp
// 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) {}
};

// Recursive function to find the sum of all nodes.
int sumOfNodes(TreeNode* root) {
    if (!root) return 0; // Base case: if the node is null, its sum is 0.

    // Sum the value of the current node and the sum of its left and right subtrees.
    return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
}

int main() {
    // Example usage
    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->right = new TreeNode(6);

    cout << "Sum of all nodes: " << sumOfNodes(root) << endl; // Output: 21
```

input

```
Sum of all nodes: 21
```