DOMAIN WINTER WINNING CAMP

Student Name: Unnati Srivastava **UID: 22BCS13475**

Branch: BE-CSE **Section/Group: TPP_FL_603-A**

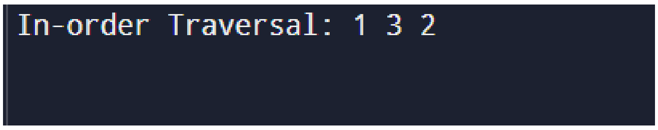
DAY 6:

QUES 1: Binary Tree Inorder Traversal

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

Solution:

```
#include <iostream>
#include <vector>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
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);
        result.push_back(node->val);
        inorderHelper(node->right, result);
    }
};
TreeNode* createSampleTree() {
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);
    return root;
}
int main() {
    Solution solution;
    TreeNode* root = createSampleTree();
    vector<int> result = solution.inorderTraversal(root);
    cout << "In-order Traversal: ";
    for (int val : result) {
        cout << val << " ";
    }
    cout << endl;

    return 0;
}
```



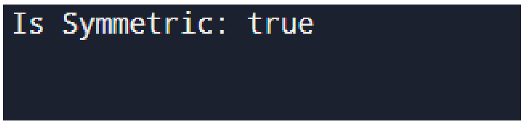
```
In-order Traversal: 1 3 2
```

QUES 2: **Symmetric Tree**

Solution:

```
#include <iostream>
```

```
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if (!root) return true;
        return isMirror(root->left, root->right);
    }
private:
    bool isMirror(TreeNode* t1, TreeNode* t2) {
        if (!t1 && !t2) return true;
        if (!t1 || !t2) return false;
        return (t1->val == t2->val) &&
               isMirror(t1->left, t2->right) &&
               isMirror(t1->right, t2->left);
    }
};
TreeNode* createSampleTree() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(2);
    root->left->left = new TreeNode(3);
    root->left->right = new TreeNode(4);
    root->right->left = new TreeNode(4);
    root->right->right = new TreeNode(3);
    return root;
}
int main() {
    Solution solution;
    TreeNode* root = createSampleTree();
    bool result = solution.isSymmetric(root);
    cout << "Is Symmetric: " << (result ? "true" : "false") << endl;
    return 0;
}
```



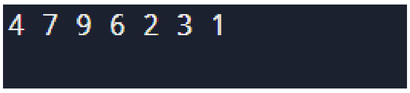
```
Is Symmetric: true
```

QUES 3: **Invert Binary Tree**

Given the root of a binary tree, invert the tree, and return its root.

Solution:

```
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    TreeNode* invertTree(TreeNode* root) {
        if (!root) return nullptr;
        swap(root->left, root->right);
        invertTree(root->left);
        invertTree(root->right);
        return root;
    }
};
void printTree(TreeNode* root) {
    if (!root) return;
    cout << root->val << " ";
    printTree(root->left);
    printTree(root->right);
}
int main() {
    TreeNode* root = new TreeNode(4);
    root->left = new TreeNode(2);
    root->right = new TreeNode(7);
    root->left->left = new TreeNode(1);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(9);
    Solution solution;
    root = solution.invertTree(root);
    printTree(root);
    return 0;
}
```



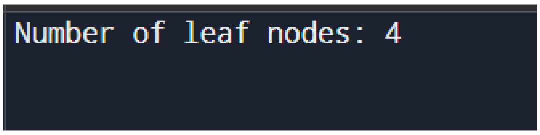
```
4 7 9 6 2 3 1
```

QUES 4: Leaf Nodes of a Binary Tree

Given a Binary Tree, the task is to count leaves in it. A node is a leaf node if both left and right child nodes of it are NULL.

Solution:

```
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    int countLeaves(TreeNode* root) {
        if (!root) return 0;
        if (!root->left && !root->right) return 1; // Leaf node
        return countLeaves(root->left) + countLeaves(root->right);
    }
};
int main() {
    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);
    root->right->right = new TreeNode(7);
    Solution solution;
    cout << "Number of leaf nodes: " << solution.countLeaves(root) << endl;
    return 0;
}
```



```
Number of leaf nodes: 4
```

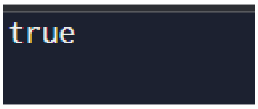
QUES 5: Path Sum

Given a binary tree and a sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. Return false if no such path can be found.

Solution:

```
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    bool hasPathSum(TreeNode* root, int targetSum) {
        if (!root) return false; // Base case: Empty tree
        if (!root->left && !root->right) // Check if it's a leaf
            return root->val == targetSum;
        return hasPathSum(root->left, targetSum - root->val) ||
               hasPathSum(root->right, targetSum - root->val);
    }
};
```

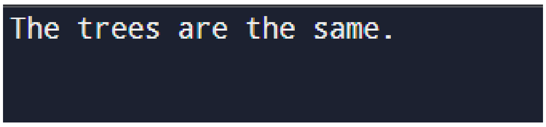
```
int main() {
    TreeNode* root = new TreeNode(5);
    root->left = new TreeNode(4);
    root->right = new TreeNode(8);
    root->left->left = new TreeNode(11);
    root->left->left->left = new TreeNode(7);
    root->left->left->right = new TreeNode(2);
    root->right->left = new TreeNode(13);
    root->right->right = new TreeNode(4);
    root->right->right->right = new TreeNode(1);
    Solution solution;
    int targetSum = 22;
    cout << (solution.hasPathSum(root, targetSum) ? "true" : "false") << endl;
    return 0;
}
```



```
true
```

Ques :6 Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

```
#include <iostream>


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


bool isSameTree(TreeNode* p, TreeNode* q) {
    // Base case: both nodes are null
    if (p == nullptr && q == nullptr) {
        return true;
    }
    // If one is null and the other is not, they are not the same
    if (p == nullptr || q == nullptr) {
        return false;
    }
    // Check if current node values are the same and recurse for left and right subtrees
    return (p->val == q->val) && isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}

// Example usage
int main() {
    TreeNode* root1 = new TreeNode(1);
    root1->left = new TreeNode(2);
    root1->right = new TreeNode(3);

    TreeNode* root2 = new TreeNode(1);
    root2->left = new TreeNode(2);
    root2->right = new TreeNode(3);

    if (isSameTree(root1, root2)) {
        std::cout << "The trees are the same." << std::endl;
    } else {
        std::cout << "The trees are not the same." << std::endl;
    }

    // Clean up memory
    delete root1->left;
    delete root1->right;
    delete root1;
    delete root2->left;
    delete root2->right;
    delete root2;

    return 0;
}
```



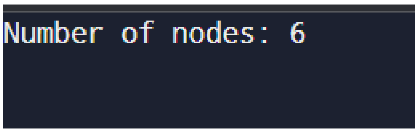
```
The trees are the same.
```

Ques 7: **Count Complete Tree Nodes**

Given the root of a complete binary tree, return the number of the nodes in the tree.

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

```
#include <iostream>
#include <cmath>

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

int computeHeight(TreeNode* node) {
    int height = 0;
    while (node != nullptr) {
        height++;
        node = node->left; // Traverse down the left subtree
    }
    return height;
}

int countNodes(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }

    int leftHeight = computeHeight(root->left);
    int rightHeight = computeHeight(root->right);

    if (leftHeight == rightHeight) {
        // Left subtree is a perfect binary tree
        return (1 << leftHeight) + countNodes(root->right);
    } else {
        // Right subtree is a perfect binary tree of height (h - 1)
        return (1 << rightHeight) + countNodes(root->left);
    }
}

// Example usage
int main() {
    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);

    std::cout << "Number of nodes: " << countNodes(root) << std::endl;

    // Clean up memory
    delete root->left->left;
    delete root->left->right;
    delete root->right->left;
    delete root->left;
    delete root->right;
    delete root;

    return 0;
}
```

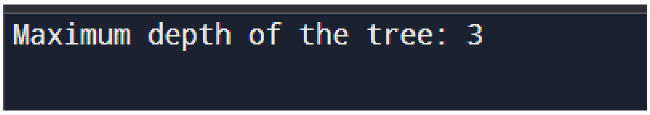


```
Number of nodes: 6
```

Ques 8: **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.

```
#include <iostream>
#include <algorithm> // For std::max

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

int maxDepth(TreeNode* root) {
    if (root == nullptr) {
        return 0; // Base case: empty tree has depth 0
    }
    // Recursive case: 1 + max depth of left and right subtrees
    return 1 + std::max(maxDepth(root->left), maxDepth(root->right));
}

// Example usage
int main() {
    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);

    std::cout << "Maximum depth of the tree: " << maxDepth(root) << std::endl;

    // Clean up memory
    delete root->left->left;
    delete root->left->right;
    delete root->left;
    delete root->right;
    delete root;

    return 0;
}
```



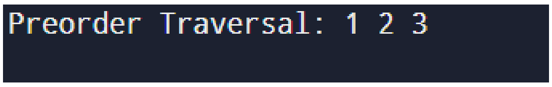
```
Maximum depth of the tree: 3
```

Ques 9; Binary Tree Preorder Traversal

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

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

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

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

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

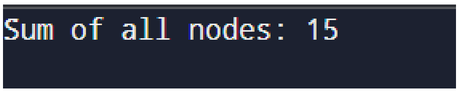
```
// Example usage
int main() {
    TreeNode* root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);

    std::vector<int> result = preorderTraversal(root);
    std::cout << "Preorder Traversal: ";
    for (int val : result) {
        std::cout << val << " ";
    }
    std::cout << std::endl;

    // Clean up memory
    delete root->right->left;
    delete root->right;
    delete root;

    return 0;
}
```



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

Ques 10: **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.

```
#include <iostream>

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

int sumOfNodes(TreeNode* root) {
    if (root == nullptr) {
        return 0; // Base case: Empty tree contributes 0 to the sum
    }
    // Recursive case: Sum of current node value + left subtree + right subtree
    return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
}

// Example usage
int main() {
    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);

    std::cout << "Sum of all nodes: " << sumOfNodes(root) << std::endl;

    // Clean up memory
    delete root->left->left;
    delete root->left->right;
    delete root->left;
    delete root->right;
    delete root;


    return 0;
}
```



```
Sum of all nodes: 15
```