DOMAIN WINTER WINNING CAMP

Student Name: Abhay Bansal
UID: 22BCS15306
Branch: BE-CSE
Section/Group: 620-A
Semester: 5th
Date of Performance: 27/12/24

1. Create a binary tree with elements [1, 2, 3, 4, 5, null, 6]

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

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

TreeNode* createBinaryTree() {
    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);
    return root;
}
int height(TreeNode* root) {
    if (!root) return 0;
    return 1 + max(height(root->left), height(root->right));
}
void printLevel(TreeNode* root, int level) {
    if (!root) {
        cout << "null ";
        return;
    }
    if (level == 1) {
        cout << root->val << " ";
    } else if (level > 1) {
        printLevel(root->left, level - 1);
        printLevel(root->right, level - 1);
    }
}
void levelOrderTraversal(TreeNode* root) {
    int h = height(root);
```

```
    for (int i = 1; i <= h; i++) {
        printLevel(root, i);
    }
    cout << endl;
}

int main() {
    TreeNode* root = createBinaryTree();
    cout << "Level-order traversal of the binary tree: ";
    levelOrderTraversal(root);
    return 0;
}
```

Output Clear

```
Level-order traversal of the binary tree: 1 2 3 4 5 null 6


=== Code Execution Successful ===
```

2. Check the binary tree is symmetric or not

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

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

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);
}

bool isSymmetric(TreeNode* root) {
    if (!root) return true;
    return isMirror(root->left, root->right);
}

TreeNode* createBinaryTree() {
    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() {
    TreeNode* root = createBinaryTree();
    if (isSymmetric(root)) {
        cout << "The binary tree is symmetric." << endl;
    } else {
        cout << "The binary tree is not symmetric." << endl;
    }
    return 0;
}
```

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

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

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

void inorderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    inorderTraversal(root->left);
    cout << root->val << " ";
    inorderTraversal(root->right);
}

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

```
    root->right->left = new TreeNode(3);

    inorderTraversal(root);
    cout << endl;

    return 0;
}
```

```
Output                                                    Clear
1 3 2


=== Code Execution Successful ===
```

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

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

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

int calculateDepth(TreeNode* node) {
    int depth = 0;
    while (node) {
        depth++;
        node = node->left;
    }
    return depth;
}

bool exists(int index, TreeNode* root, int depth) {
    int left = 0, right = (1 << depth) - 1;
    TreeNode* current = root;

    while (left < right) {
        int mid = (left + right) / 2;
        if (index & (1 << mid)) {
            current = current->right;
            left = mid + 1;
        } else {
```

```
            current = current->left;
            right = mid;
        }
    }
    return current != nullptr;
}

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

    int depth = calculateDepth(root);
    int left = 0, right = (1 << depth) - 1;

    while (left <= right) {
        int mid = (left + right) / 2;
        if (exists(mid, root, depth)) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return (1 << depth) - 1 + left;
}

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);

    int nodeCount = countNodes(root);
    cout << "Number of nodes in the complete binary tree: " << nodeCount << endl;

    return 0;
}
```

```
Output                                                    Clear
Number of nodes in the complete binary tree: 7


=== Code Execution Successful ===
```

5. Given the root of a binary tree, you need to find the sum of all the node values in the binary tree.

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

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
TreeNode* buildTree(const string& input, int& index) {
    if (index >= input.size() || input[index] == ',') {
        index++;
        return nullptr;
    }
    int num = 0;
    while (index < input.size() && input[index] != ',' && input[index] != ' ') {
        num = num * 10 + (input[index] - '0');
        index++;
    }
    TreeNode* node = new TreeNode(num);
    node->left = buildTree(input, index);
    node->right = buildTree(input, index);
    return node;
}
int sumOfNodes(TreeNode* root) {
    if (!root) return 0;
    return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
}
int main() {
    string input;
    cout << "Enter the tree nodes (comma separated): ";
    getline(cin, input);
    int index = 0;
    TreeNode* root = buildTree(input, index);
    int sum = sumOfNodes(root);
    cout << "Sum of all nodes: " << sum << endl;
    return 0;
}
```

Output Clear

```
Enter the tree nodes (comma separated): 1,2,3,4,5
Sum of all nodes: 15


=== Code Execution Successful ===
```

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

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

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

TreeNode* buildTree(const string& nodes, int& index) {
    if (index >= nodes.length() || nodes[index] == 'n') {
        index += 4;
        return nullptr;
    }

    int val = stoi(nodes.substr(index, nodes.find(' ', index) - index));
    index = nodes.find(' ', index) + 1;

    TreeNode* node = new TreeNode(val);

    if (index >= nodes.length()) {
        return node;
    }

    node->left = buildTree(nodes, index);
    if (index >= nodes.length()) {
        return node;
    }

    node->right = buildTree(nodes, index);

    return node;
}

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) {
```

```
        cout << "null ";
        return;
    }
    cout << root->val << " ";
    printTree(root->left);
    printTree(root->right);
}

int main() {
    cout << "Enter tree nodes separated by spaces: ";
    string input;
    getline(cin, input);

    int index = 0;
    TreeNode* root = buildTree(input, index);

    cout << "Original tree: ";
    printTree(root);
    cout << endl;

    TreeNode* invertedRoot = invertTree(root);

    cout << "Inverted tree: ";
    printTree(invertedRoot);
    cout << endl;

    return 0;
}
```

Output Clear

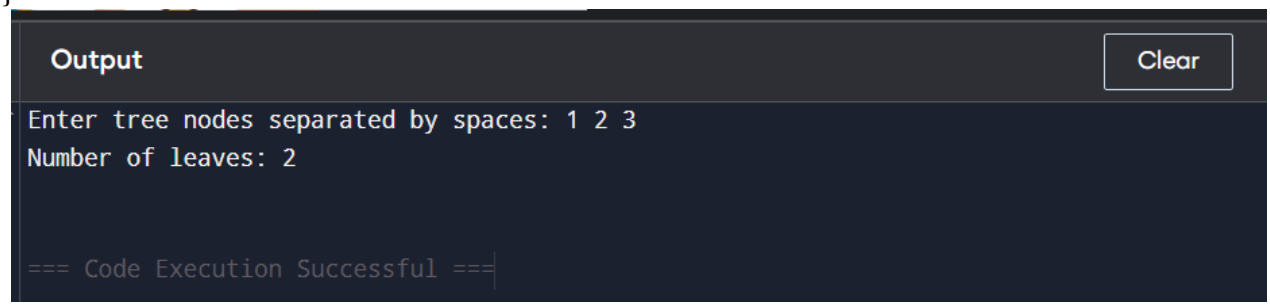
```
Enter tree nodes separated by spaces: 1 2 3
Original tree: 1 2 3 null null null null
Inverted tree: 1 null 2 null 3 null null


=== Code Execution Successful ===
```

7. 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.

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

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
```

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

TreeNode* buildTree(const string& nodes) {
    if (nodes.empty()) {
        return nullptr;
    }
    istringstream iss(nodes);
    string token;
    iss >> token;
    int val = atoi(token.c_str());
    TreeNode* root = new TreeNode(val);

    TreeNode* current = root;
    while (iss >> token) {
        int val = atoi(token.c_str());
        TreeNode* newNode = new TreeNode(val);

        if (!current->left) {
            current->left = newNode;
        } else {
            current->right = newNode;
            current = root;
            while (current->left && current->right) {
                current = current->left;
            }
        }
    }
    return root;
}

int countLeaves(TreeNode* root) {
    if (!root) {
        return 0;
    }
    if (!root->left && !root->right) {
        return 1;
    }
    return countLeaves(root->left) + countLeaves(root->right);
}

void deleteTree(TreeNode* root) {
    if (!root) {
        return;
    }
    deleteTree(root->left);
    deleteTree(root->right);
    delete root;
}

int main() {
```

```
    cout << "Enter tree nodes separated by spaces: ";
    string input;
    getline(cin, input);

    TreeNode* root = buildTree(input);
    if (!root) {
        cout << "Invalid input." << endl;
        return 1;
    }
    int numLeaves = countLeaves(root);
    cout << "Number of leaves: " << numLeaves << endl;
    deleteTree(root);

    return 0;
}
```

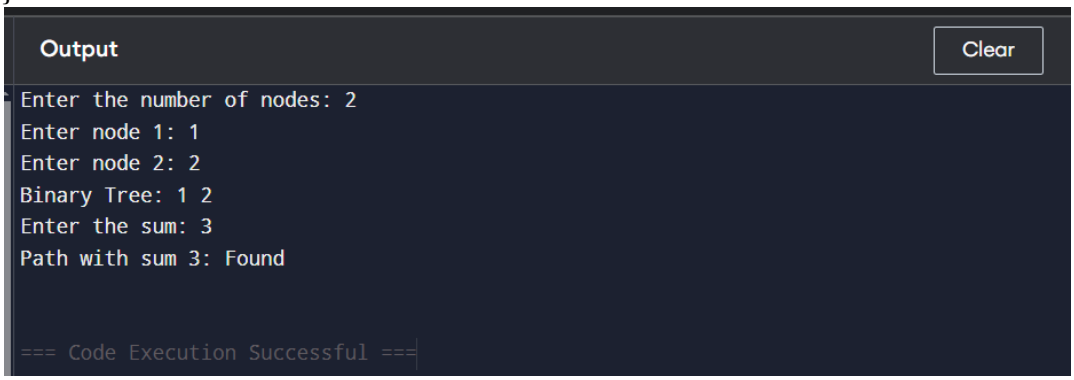


8. Given a binary tree and a sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. Return false if no such path can be found.

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

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

class Solution {
public:
    bool hasPathSum(TreeNode* root, int sum) {
        if (!root) return false;
        return dfs(root, sum);
    }

    bool dfs(TreeNode* node, int sum) {
        if (!node) return false;
        if (!node->left && !node->right && node->val == sum) return true;
        return dfs(node->left, sum - node->val) || dfs(node->right, sum - node->val);
```

```
    }
};

TreeNode* createNode(int val) {
    TreeNode* newNode = new TreeNode(val);
    return newNode;
}

void insertNode(TreeNode** root, int val) {
    if (*root == NULL) *root = createNode(val);
    else if (val < (*root)->val) {
        if ((*root)->left == NULL) (*root)->left = createNode(val);
        else insertNode(&((*root)->left), val);
    } else {
        if ((*root)->right == NULL) (*root)->right = createNode(val);
        else insertNode(&((*root)->right), val);
    }
}

void printTree(TreeNode* root) {
    if (root == NULL) return;
    cout << root->val << " ";
    printTree(root->left);
    printTree(root->right);
}

int main() {
    Solution solution;
    TreeNode* root = NULL;

    int n;
    cout << "Enter the number of nodes: ";
    cin >> n;

    for (int i = 0; i < n; i++) {
        int val;
        cout << "Enter node " << i + 1 << ": ";
        cin >> val;
        insertNode(&root, val);
    }

    cout << "Binary Tree: ";
    printTree(root);
    cout << endl;

    int sum;
    cout << "Enter the sum: ";
    cin >> sum;

    bool result = solution.hasPathSum(root, sum);
```

```
    cout << "Path with sum " << sum << ": " << (result ? "Found" : "Not Found") << endl;

    return 0;
}
```



9. Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

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

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

class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if (!root) return NULL;
        if (root == p || root == q) return root;

        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);

        if (left && right) return root;
        return left ? left : right;
    }
};

void printTree(TreeNode* root) {
    if (root == NULL) return;
    cout << root->val << " ";
    printTree(root->left);
    printTree(root->right);
}

int main() {
    Solution solution;
```

```
int n;
cout << "Enter the number of nodes: ";
cin >> n;

TreeNode* root = NULL;

for (int i = 0; i < n; i++) {
    int val;
    cout << "Enter node " << i + 1 << ": ";
    cin >> val;

    if (i == 0) {
        root = new TreeNode(val);
    } else {
        TreeNode* current = root;
        while (true) {
            if (val < current->val) {
                if (current->left == NULL) {
                    current->left = new TreeNode(val);
                    break;
                }
                current = current->left;
            } else {
                if (current->right == NULL) {
                    current->right = new TreeNode(val);
                    break;
                }
                current = current->right;
            }
        }
    }
}

cout << "Binary Tree: ";
printTree(root);
cout << endl;
int p, q;
cout << "Enter the values of two nodes: ";
cin >> p >> q;
TreeNode* nodeP = NULL;
TreeNode* nodeQ = NULL;
TreeNode* current = root;

while (current != NULL) {
    if (current->val == p) nodeP = current;
    if (current->val == q) nodeQ = current;
    if (nodeP != NULL && nodeQ != NULL) break;

    if (p < current->val && nodeP == NULL) current = current->left;
    else if (q < current->val && nodeQ == NULL) current = current->left;
```

```
        else current = current->right;
    }

    TreeNode* lca = solution.lowestCommonAncestor(root, nodeP, nodeQ);
    cout << "Lowest common ancestor of " << p << " and " << q << ": " << lca->val << endl;

    return 0;
}
```

```
Output                                                    Clear
Enter the number of nodes: 4
Enter node 1: 1
Enter node 2: 2
Enter node 3: 3
Enter node 4: 4
Binary Tree: 1 2 3 4
Enter the values of two nodes: 2 4
Lowest common ancestor of 2 and 4: 2


=== Code Execution Successful ===
```

10. Given a binary search tree (BST), write a function to find the kth smallest element in the tree.

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

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

class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        int count = 0;
        int result = -1;
        inorder(root, k, count, result);
        return result;
    }

    void inorder(TreeNode* node, int k, int& count, int& result) {
        if (node == NULL || count >= k) return;
        inorder(node->left, k, count, result);
        count++;
        if (count == k) {
```

```
            result = node->val;
            return;
        }
        inorder(node->right, k, count, result);
    }
};

int main() {
    Solution solution;
    int n;
    cout << "Enter the number of nodes: ";
    cin >> n;
    TreeNode* root = NULL;

    for (int i = 0; i < n; i++) {
        int val;
        cout << "Enter node " << i + 1 << ": ";
        cin >> val;

        if (i == 0) {
            root = new TreeNode(val);
        } else {
            TreeNode* current = root;
            while (true) {
                if (val < current->val) {
                    if (current->left == NULL) {
                        current->left = new TreeNode(val);
                        break;
                    }
                    current = current->left;
                } else {
                    if (current->right == NULL) {
                        current->right = new TreeNode(val);
                        break;
                    }
                    current = current->right;
                }
            }
        }
    }
    int k;
    cout << "Enter the value of k: ";
    cin >> k;
    int result = solution.kthSmallest(root, k);
    cout << "Kth smallest element: " << result << endl;
    return 0;
}
```

Output Clear

```
Enter the number of nodes: 4
Enter node 1: 1
Enter node 2: 2
Enter node 3: 3
Enter node 4: 4
Enter the value of k: 1
Kth smallest element: 1


=== Code Execution Successful ===
```