**Assignment 4**

**Implement Binary Tree Traversals**:

* Perform **preorder**, **inorder**, and **postorder** traversals of a binary tree (both recursive and iterative).

Iterative:

#*include* <iostream>

#*include* <vector>

using namespace std;

class Node {

public:

    int data;

    Node\* left;

    Node\* right;

*Node*(int x) {

        data = x;

        left = nullptr;

        right = nullptr;

    }

};

class BinaryTree {

public:

    Node\* root;

*BinaryTree*() {

        root = nullptr;

    }

*// Method to create a predefined binary tree*

    void *createTree1*() {

        root = new *Node*(1);

        root->left = new *Node*(2);

        root->right = new *Node*(3);

        root->*left*->left = new *Node*(4);

        root->*left*->right = new *Node*(5);

    }

    void *createTree2*() {

        root = new *Node*(10);

        root->left = new *Node*(20);

        root->right = new *Node*(30);

        root->*left*->left = new *Node*(40);

        root->*left*->right = new *Node*(50);

        root->*right*->left = new *Node*(60);

        root->*right*->right = new *Node*(70);

    }

*// Postorder Traversal*

    void *postOrderTraversal*(Node*\** node, vector<int>*&* result) {

*if* (node == nullptr) *return*;

        vector<Node\*> stack1, stack2;

        stack1.*push\_back*(node);

*while* (!stack1.*empty*()) {

            Node\* curr = stack1.*back*();

            stack1.*pop\_back*();

            stack2.*push\_back*(curr);

*if* (curr->left != nullptr) stack1.*push\_back*(curr->left);

*if* (curr->right != nullptr) stack1.*push\_back*(curr->right);

        }

*while* (!stack2.*empty*()) {

            result.*push\_back*(stack2.*back*()->data);

            stack2.*pop\_back*();

        }

    }

*// Iterative function for inorder traversal*

    void *inOrderTraversal*(Node*\** node, vector<int>*&* result) {

        vector<Node\*> stack;

        Node\* curr = node;

*while* (curr != nullptr || !stack.*empty*()) {

*while* (curr != nullptr) {

                stack.*push\_back*(curr);

                curr = curr->left;

            }

            curr = stack.*back*();

            stack.*pop\_back*();

            result.*push\_back*(curr->data);

            curr = curr->right;

        }

    }

*// Preorder Traversal*

    void *preOrderTraversal*(Node*\** node, vector<int>*&* result) {

*if* (node == nullptr) *return*;

        vector<Node\*> stack;

        stack.*push\_back*(node);

*while* (!stack.*empty*()) {

            Node\* curr = stack.*back*();

            stack.*pop\_back*();

            result.*push\_back*(curr->data);

*if* (curr->right != nullptr) stack.*push\_back*(curr->right);

*if* (curr->left != nullptr) stack.*push\_back*(curr->left);

        }

    }

    void *displayTraversals*() {

*if* (root == nullptr) {

            cout *<<* "Tree is empty." *<<* *endl*;

*return*;

        }

        vector<int> result;

*// Postorder Traversal*

*postOrderTraversal*(root, result);

        cout *<<* "Postorder Traversal: ";

*for* (int val : result) {

            cout *<<* val *<<* " ";

        }

        cout *<<* *endl*;

        result.*clear*();

*// Inorder Traversal*

*inOrderTraversal*(root, result);

        cout *<<* "Inorder Traversal: ";

*for* (int val : result) {

            cout *<<* val *<<* " ";

        }

        cout *<<* *endl*;

        result.*clear*();

*// Preorder Traversal*

*preOrderTraversal*(root, result);

        cout *<<* "Preorder Traversal: ";

*for* (int val : result) {

            cout *<<* val *<<* " ";

        }

        cout *<<* *endl*;

    }

};

int *main*() {

    BinaryTree tree1, tree2;

    tree1.*createTree1*();

    cout *<<* "Traversals for Tree 1:" *<<* *endl*;

    tree1.*displayTraversals*();

    tree2.*createTree2*();

    cout *<<* "\nTraversals for Tree 2:" *<<* *endl*;

    tree2.*displayTraversals*();

*return* 0;

}



**Recursive:**

#*include* <iostream>

#*include* <vector>

using namespace std;

class Node {

public:

    int data;

    Node\* left;

    Node\* right;

*Node*(int x) {

        data = x;

        left = nullptr;

        right = nullptr;

    }

};

class BinaryTree {

public:

    Node\* root;

*BinaryTree*() {

        root = nullptr;

    }

*// Method to create a predefined binary tree*

    void *createTree1*() {

        root = new *Node*(1);

        root->left = new *Node*(2);

        root->right = new *Node*(3);

        root->*left*->left = new *Node*(4);

        root->*left*->right = new *Node*(5);

    }

    void *createTree2*() {

        root = new *Node*(10);

        root->left = new *Node*(20);

        root->right = new *Node*(30);

        root->*left*->left = new *Node*(40);

        root->*left*->right = new *Node*(50);

        root->*right*->left = new *Node*(60);

        root->*right*->right = new *Node*(70);

    }

*// Recursive Postorder Traversal*

    void *postOrderTraversal*(Node*\** node, vector<int>*&* result) {

*if* (node == nullptr) *return*;

*postOrderTraversal*(node->left, result);

*postOrderTraversal*(node->right, result);

        result.*push\_back*(node->data);

    }

*// Recursive Inorder Traversal*

    void *inOrderTraversal*(Node*\** node, vector<int>*&* result) {

*if* (node == nullptr) *return*;

*inOrderTraversal*(node->left, result);

        result.*push\_back*(node->data);

*inOrderTraversal*(node->right, result);

    }

*// Recursive Preorder Traversal*

    void *preOrderTraversal*(Node*\** node, vector<int>*&* result) {

*if* (node == nullptr) *return*;

        result.*push\_back*(node->data);

*preOrderTraversal*(node->left, result);

*preOrderTraversal*(node->right, result);

    }

    void *displayTraversals*() {

*if* (root == nullptr) {

            cout *<<* "Tree is empty." *<<* *endl*;

*return*;

        }

        vector<int> result;

*// Postorder Traversal*

*postOrderTraversal*(root, result);

        cout *<<* "Postorder Traversal: ";

*for* (int val : result) {

            cout *<<* val *<<* " ";

        }

        cout *<<* *endl*;

        result.*clear*();

*// Inorder Traversal*

*inOrderTraversal*(root, result);

        cout *<<* "Inorder Traversal: ";

*for* (int val : result) {

            cout *<<* val *<<* " ";

        }

        cout *<<* *endl*;

        result.*clear*();

*// Preorder Traversal*

*preOrderTraversal*(root, result);

        cout *<<* "Preorder Traversal: ";

*for* (int val : result) {

            cout *<<* val *<<* " ";

        }

        cout *<<* *endl*;

    }

};

int *main*() {

    BinaryTree tree1, tree2;

    tree1.*createTree1*();

    cout *<<* "Traversals for Tree 1:" *<<* *endl*;

    tree1.*displayTraversals*();

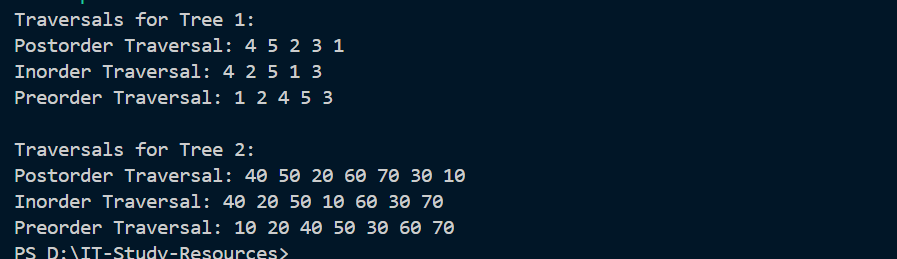
    tree2.*createTree2*();

    cout *<<* "\nTraversals for Tree 2:" *<<* *endl*;

    tree2.*displayTraversals*();

*return* 0;

}

****