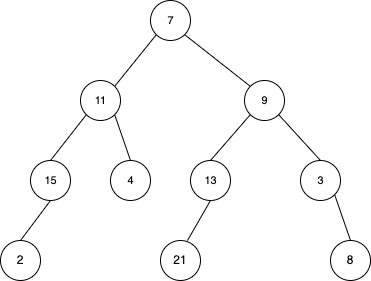
**COMP 203 Lab 7**

**Trees**



**SOLUTION**

class Node<E> { // variables 2pt

E data;

Node<E> left, right;

public Node(E data) {//constructors 3pt

this.data = data;

this.left = this.right = null;

}

}

public class BinaryTree<E>

private Node<E> root; //object from node class 2pt

public BinaryTree() {//constructor 3pt

this.root = null;

}

// PreOrderTraversal

public void preOrderTraversal(Node<E> root) { //10pt

if (root != null) {

System.out.print(root.data + " ");

preOrderTraversal(root.left);

preOrderTraversal(root.right);

}

}

// PostOrderTraversal

public void postOrderTraversal(Node<E> root) {//10pt

if (root != null) {

postOrderTraversal(root.left);

postOrderTraversal(root.right);

System.out.print(root.data + " ");

}

}

// InOrderTraversal

public void inOrderTraversal(Node<E> root) {//10pt

if (root != null) {

inOrderTraversal(root.left);

System.out.print(root.data + " ");

inOrderTraversal(root.right);

}

}

// Size

public int size(Node<E> root) { //5pt it can be iterative version too.

if (root == null) {

return 0;

} else {

return 1 + size(root.left) + size(root.right);

}

}

// isChild

public boolean isChild(Node<E> parent, Node<E> child) { //5pt

if (parent == null || child == null) {

return false;

}

return isChildHelper(parent, child);

}

private boolean isChildHelper(Node<E> parent, Node<E> child) {

if (parent == null) {

return false;

}

if (parent.left == child || parent.right == child) { //important here

return true;

}

return isChildHelper(parent.left, child) || isChildHelper(parent.right, child); }

// isEmpty

public boolean isEmpty() {//5pt

return root == null;

}

public static void main(String[] args) {

BinaryTree<Integer> binaryTree = new BinaryTree<>();

// crating tree 15pt

binaryTree.root = new Node<>(7);

binaryTree.root.left = new Node<>(11);

binaryTree.root.right = new Node<>(9);

binaryTree.root.left.left = new Node<>(15);

binaryTree.root.left.right = new Node<>(4);

binaryTree.root.left.left.left = new Node<>(2);

binaryTree.root.right.left = new Node<>(13);

binaryTree.root.right.right = new Node<>(3);

binaryTree.root.right.left.left = new Node<>(21);

binaryTree.root.right.right.right = new Node<>(8);

// testing all methods 6x5=30 points as follows:

System.out.println("PreOrder Traversal:");

binaryTree.preOrderTraversal(binaryTree.root); //5pt

System.out.println("\nPostOrder Traversal:");

binaryTree.postOrderTraversal(binaryTree.root); ); //5pt

System.out.println("\nInOrder Traversal:");

binaryTree.inOrderTraversal(binaryTree.root); ); //5pt

System.out.println("\nSize of the tree: " + binaryTree.size(binaryTree.root)); ); //5pt

Node<Integer> parentNode = binaryTree.root.left;

Node<Integer> childNode = binaryTree.root.left.right;

System.out.println("Is " + childNode.data + " a child of " + parentNode.data + "? " +

binaryTree.isChild(parentNode, childNode)); ); //5pt

System.out.println("Is the tree empty? " + binaryTree.isEmpty());); //5pt

}

}