COP3530
Data Structures
Summer 2019
Project 3

In this project, you are going to implement the Binary Tree ADT with a linked structure. Then, you will write a simple program to test your Binary Tree implementation.

The first step in this project is to write and compile the following *Node.java* file.

Node.java

```
class Node {

    int data;
    Node left;
    Node right;

    Node() {
        data = 0;
        left = null;
        right = null;
    }

    public int getData() {
        return data;
    }

    public Node getLeft() {
        return left;
    }

    public Node getRight() {
        return right;
    }

    public void setData(int newData) {
        data = newData;
    }

    public void setLeft(Node newLink) {
        left = newLink;
    }

    public void setRight(Node newLink) {
        right = newLink;
    }

}
```

The second step in this project is to complete the following *Tree.java* file which implements the Binary Tree ADT using a linked structure.

COP3530
Data Structures
Summer 2019
Project 3

Tree.java

```
class Tree {

    Node root;

    Tree ( ) {
    //Sets root to null

    }

    public Node getRoot( ) {
    //Returns root

    }

    public void setRoot(Node newRoot ) {
    //Sets root to newRoot

    }

    public boolean empty( ) {
    //Returns true if the tree is empty and false otherwise

    }


    public int count( ) {
    //Returns the number of nodes in the tree.
    //Uses function countNodes().

      return countNodes(root);
    }

    public int countNodes(Node N) {
    //Recursive function which returns the number of nodes
    //in the subtree rooted at N.

    }

    public boolean equal(Tree T) {
    //Returns true if our tree equals tree T. Returns false otherwise.
    //Uses function equalNodes().

    }

    public boolean equalNodes(Node N, Node M) {
    //Recursive function which returns true if the subtrees rooted
    //at N and M are equal. Returns false otherwise.

    }

    public boolean search(int element) {
    //Returns true if element belongs in the tree. Returns false otherwise.
```

```
//Uses function searchNodes().

  return searchNodes(root, element);
}

public boolean searchNodes(Node N, int element) {
//Recursive function which returns true if element belongs to the
//subtree rooted at N. Returns false otherwise.

}

public void printLDR() {
//Prints the elements of the tree in in-order (LDR).
//Uses function printLDRNodes().

}

public void printLDRNodes(Node N) {
//Recursive function which prints the elements of the subtree
//rooted at N in in-order (LDR).

}

public void printLRD() {
//Prints the elements of the tree in post-order (LRD).
//Uses function printLRDNodes().

}

public void printLRDNodes(Node N) {
//Recursive function which prints the elements of the subtree
//rooted at N in post-order (LRD).

}

public void printDLR() {
//Prints the elements of the tree in pre-order (DLR).
//Uses function printDLRNodes().

}

public void printDLRNodes(Node N) {
//Recursive function which prints the elements of the subtree
//rooted at N in pre-order (DLR).

}

public Tree copy() {
//Returns a copy of our tree.
//Uses function copyNodes().

  Tree cpTree = new Tree();
  Node cpNode = copyNodes(root);
  cpTree.setRoot(cpNode);
  return cpTree;
```

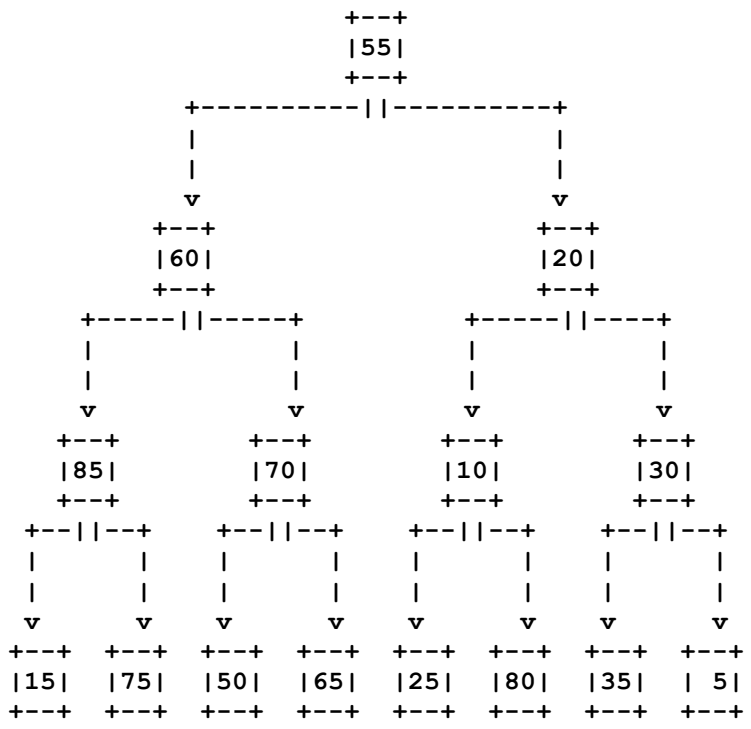
```
}

public Node copyNodes(Node N) {
//Recursive function which returns a copy of the subtree rooted at N.

}

public void leaves( ) {
//Prints the leaves of our tree. The root is not a leave.
//Uses function leavesNodes().

}

public void leavesNodes(Node N) {
//Recursive function which prints the leaves of the subtree
//rooted at N.

}

public void internal() {
//Prints the internal nodes of our tree.
//The root is not an internal node.
//Uses function internalNodes().

}

public void internalNodes(Node N) {
//Recursive function which prints the internal nodes of the
//subtree rooted at N.

}

public void path(int element) {
//Prints the path from root to element if element exists in the tree.
//Uses function pathNodes().

}

public void pathNodes(int element, Node N) {
//Recursive function which prints the path from N to element.
//Assumes element belongs in the subtree rooted at N.

}

public int height() {
//Returns the height of the tree.
//Uses function heightNodes().

}

public int heightNodes(Node N) {
//Recursive function which returns the height of the subtree
//rooted at N.

}
```

```
    public void descendents(int element) {
    //Prints all the descendents of element if element exists in the tree.
    //Uses function descendentsNodes().

    }

    public void descendentsNodes(Node N, int element) {
    //Recursive function which prints all the descendents of element in
    //the subtree rooted at N.
    //Assumes element belongs in the subtree rooted at N.

    }

}
```

The third step in this project is to complete the following *Project3.java* file.

The program basically tests our *Tree.java* implementation.

To test your program, you will construct (by hardcoding) the binary tree given below with the Node methods Node(), setData(), setLeft(), and setRight(), and Tree methods Tree() and setRoot().



Your program must behave like the execution examples given below.

COP3530
Data Structures
Summer 2019
Project 3

Important Note: I will construct (by hardcoding) other binary trees to test your Binary Tree implementation. So please test your program with lots of other binary trees before submission.

Project3.java

```
import java.util.Scanner;

class Project3 {

  public static void main(String args[]) {

    Scanner scan = new Scanner(System.in);

    Node N55 = new Node();
    N55.setData(55);

    Node N60 = new Node();
    N60.setData(60);

    Node N20 = new Node();
    N20.setData(20);

    N55.setLeft(N60);
    N55.setRight(N20);

  }
}
```

Execution of Project3.java

Example 1

```
Checking function size()
------------------------
The size of the binary tree T1 is: 15

Checking function height()
--------------------------
The height of the binary tree T1 is: 3

Checking function leaves()
--------------------------
The leaves of the binary tree T1 are:
15 75 50 65 25 80 35 5

Checking function internal()
----------------------------
The internal nodes of the binary tree T1 are:
60 85 70 20 10 30

Checking functions printLDR(), printDLR(), and printLRD()
---------------------------------------------------------
```

```
Enter 1 (in-order traversal of T1),
      2 (pre-order traversal of T1), or
      3 (post-order traversal of T1): 3
The nodes of the binary tree T1 in post-order traversal are:
15 75 85 50 65 70 60 25 80 10 35 5 30 20 55

Checking function search()
--------------------------
Enter element to search: 50
50 belongs in the binary tree T1

Checking function path()
------------------------
Enter element to find path for: 80
The path from the root of T1 to node 80 is:
55 20 10 80

Checking function descendents()
-------------------------------
Enter element to find descendents for: 30
The descendents of 30 in T1 are:
35 5

Checking function copy()
------------------------
Let T2 be a copy of T1
The nodes of binary tree T2 in in-order traversal are:
15 85 75 60 50 70 65 55 25 10 80 20 35 30 5

Checking function equal()
-------------------------
Let data part of node 80 be 90 in binary tree T1
Binary trees T1 and T2 are not equal
Let data part of node 80 be 80 in binary tree T1
Binary trees T1 and T2 are equal
```

Example 2

```
Checking function size()
------------------------
The size of the binary tree T1 is: 15

Checking function height()
--------------------------
The height of the binary tree T1 is: 3

Checking function leaves()
--------------------------
The leaves of the binary tree T1 are:
15 75 50 65 25 80 35 5

Checking function internal()
----------------------------
The internal nodes of the binary tree T1 are:
60 85 70 20 10 30
```

COP3530
Data Structures
Summer 2019
Project 3

```
Checking functions printLDR(), printDLR(), and printLRD()
----------------------------------------------------------
Enter 1 (in-order traversal of T1),
      2 (pre-order traversal of T1), or
      3 (post-order traversal of T1): 2
The nodes of the binary tree T1 in pre-order traversal are:
55 60 85 15 75 70 50 65 20 10 25 80 30 35 5

Checking function search()
--------------------------
Enter element to search: 99
99 does not belong in the binary tree T1

Checking function path()
------------------------
Enter element to find path for: 5
The path from the root of T1 to node 5 is:
55 20 30 5

Checking function descendents()
-------------------------------
Enter element to find descendents for: 60
The descendents of 60 in T1 are:
15 85 75 50 70 65

Checking function copy()
------------------------
Let T2 be a copy of T1
The nodes of binary tree T2 in in-order traversal are:
15 85 75 60 50 70 65 55 25 10 80 20 35 30 5

Checking function equal()
-------------------------
Let data part of node 80 be 90 in binary tree T1
Binary trees T1 and T2 are not equal
Let data part of node 80 be 80 in binary tree T1
Binary trees T1 and T2 are equal
```