CSE 143
Computer Programming II
Binary Trees
Outline

1. LinkedLists to Binary Trees
2. Why Do We Care About Binary Trees?
3. Printing Recursively
4. Binary Tree Traversals
Consider the following standard LinkedList:

```
0 -> 1 -> 2 -> 3
```

Recall the definition of a ListNode

```java
public class Node {
    public int data;
    public Node next;

    public Node(int data, Node next) {
        this.data = data;
        this.next = next;
    }
}
```

What if we added more fields?
- Multiple data fields?
- Multiple “next” fields?
Nodes with Multiple next Fields

```java
public class Node {
    public int data;
    public Node next1;
    public Node next2;

    public Node(int data, Node next1, Node next2) {
        this.data = data;
        this.next1 = next1;
        this.next2 = next2;
    }
}
```

(front red is next1; yellow is next2)
Introducing Trees

Binary Trees

```java
public class Node {
    public int data;
    public Node left;
    public Node right;

    public Node(int data, Node left, Node right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}
```

(root)

```
0
  ↓
  1
  ↓
  3
  ↓
leaf

  ↓
  2
  ↓
  4
  ↓
  5
  ↓
leaf
leaf
leaf

(red is right; yellow is left)
```
Consider the following LinkedList of a mathematical expression:

front

What's bad about it?

- It doesn't really help us with the structure
- Looking at it doesn’t really show us what’s going on

What about this structure instead?

Now we can see the order of operations much more clearly!
Uses of Trees

- Parsing (Programming Languages, Math, etc.)
  
- Implementing TreeSet
  
- Directory File Structure
More Uses of Trees

- Recursive Trees (including things like games of Tic-Tac-Toe)

- Compression (this will be your last assignment!)
public void print() {
    Node current = this.front;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
}

We’d like to figure out how to print trees. Since LinkedLists are “simpler versions of trees”, they might help.

How do we go in every direction in a tree?

USE RECURSION!
To print a LinkedList...

- Print the **front** of the list
- Print the **next** of the list (recursively)

**Code**

```java
public void print() {
    print(this.front);
}

public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.next);
    }
}
```
To print a BinaryTree...

- Print the **root** of the tree
- Print the **left** of the tree (recursively)
- Print the **right** of the tree (recursively)

**Code**

```java
public void print() {
    print(this.root);
}

public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

Output:

```
>> 1
```
```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```
```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

The output is: `1 2`
Printing a Tree Example

```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

Trace

```
>> 1 2 3
```

OUTPUT
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```
Printing a Tree Example

```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

**Trace**

```
>> 1 2 3
```
public void print(Node c) {
  // c = 1
  if (c != null) {
    // c is null!
    System.out.print(c.data + " ");
    print(c.left);
    print(c.right);
  }
}

public void print(Node c) {
  // c = 2
  if (c != null) {
    // c is null!
    System.out.print(c.data + " ");
    print(c.left);
    print(c.right);
  }
}

public void print(Node c) {
  // c = 3
  if (c != null) {
    // c is null!
    System.out.print(c.data + " ");
    print(c.left);
    print(c.right);
  }
}

>> 1 2 3
```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

```
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

```
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

Trace

```
1 2 3
```

OUTPUT

```
>> 1 2 3
```
```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

Trace:
```
>> 1 2 3
```
```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

**Trace**

```
>> 1 2 3 4
```
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}

1 public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}

1 public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}

Trace

OUTPUT

>> 1 2 3 4
Printing a Tree Example

public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}

OUTPUT

>> 1 2 3 4
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}

Trace

OUTPUT

>> 1 2 3 4
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

**OUTPUT**

```
>> 1 2 3 4
```
Printing a Tree Example

```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

Trace

```
>> 1 2 3 4 5
```

OUTPUT

```
>> 1 2 3 4 5
```
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}

Trace

>> 1 2 3 4 5
Printing a Tree Example

```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

Trace

```
>> 1 2 3 4 5 6
```

OUTPUT
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}

>> 1 2 3 4 5 6
```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

Trace

OUTPUT

```
>> 1 2 3 4 5 6
```
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}

>> 1 2 3 4 5 6 7
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```
public void print(Node c) {
    // c = 1
    if (c != null) {
        // c = 5
        if (c != null) {
            System.out.print(c.data + " ");
            print(c.left);
            print(c.right);
        }
    }
}
```

Trace

```
>> 1 2 3 4 5 6 7
```

OUTPUT

```
>> 1 2 3 4 5 6 7
```
Printing a Tree Example

```java
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
```

Trace

```
>> 1 2 3 4 5 6 7
```

OUTPUT
public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " ");
        print(c.left);
        print(c.right);
    }
}
1 public void print(Node c) { // c = 1
2     if (c != null) {
3         System.out.print(c.data + " ");
4         print(c.left);
5         print(c.right);
6     }
7 }

Trace

OUTPUT

>> 1 2 3 4 5 6 7
### Pre-Order Traversal
```java
class Node {
    int data;
    Node left, right;
}

public void print(Node c) {
    if (c != null) {
        System.out.print(c.data + " "); // print
        print(c.left); // left
        print(c.right); // right
    }
}
```

### In-Order Traversal
```java
class Node {
    int data;
    Node left, right;
}

public void print(Node c) {
    if (c != null) {
        print(c.left); // left
        System.out.print(c.data + " "); // print
        print(c.right); // right
    }
}
```

### Post-Order Traversal
```java
class Node {
    int data;
    Node left, right;
}

public void print(Node c) {
    if (c != null) {
        print(c.left); // left
        print(c.right); // right
        System.out.print(c.data + " "); // print
    }
}
```
Consider the following binary tree:

```
root
  ▼
 10
  ▼
22  53
  ▼  ▼
 3  4  66  17
       ▼
     12
```

Compute the Pre-Order, In-Order, and Post-Order Traversals:

- **Pre-Order:** 10, 22, 3, 4, 53, 66, 17, 12
- **In-Order:** 3, 22, 4, 10, 66, 53, 17, 12
- **Post-Order:** 3, 4, 22, 66, 12, 17, 53, 10
To Quickly Generate A Traversal

- Trace a path around the tree
- As you pass a node on the proper side, process it:
  - Pre-Order: left
  - In-Order: bottom
  - Post-Order: right

```
3 41 17 4 8 root 66 32
```
Binary Tree methods are just normal recursive functions. The base case/recursive calls will always be similar.

Writing a Binary Tree Method

- The base case is `current == null`.
- First recursive case is `method(current.left)`.
- Second recursive case is `method(current.right)`.

```java
public type method(...) {
    return method(this.root, ...);
}

private type method(TreeNode current, ...) {
    if (current == null) { /* DO BASE CASE */ }

    // Do the left recursive case:
    type leftResult = method(current.left, ...);

    // Do the right recursive case:
    type rightResult = method(current.right, ...);

    /* Use the left and right results... */
    return ...
}
```
Write a method, in the IntTree class, called contains():

```java
public boolean contains(int value);
```

that returns true if the tree contains value and false otherwise.

```java
public boolean contains(int value) {
    return contains(this.root, value);
}

private boolean contains(IntTreeNode current, int value) {
    /* If the tree is null, it definitely doesn’t contain value... */
    if (current == null) {
        return false;
    }

    /* If current *is* value, we found it! */
    else if (current.data == value) {
        return true;
    }

    else {
        boolean leftContainsValue = contains(current.left, value);
        boolean rightContainsValue = contains(current.right, value);
        return leftContainsValue || rightContainsValue;
    }
}
```
Some Tree Tips!

- Trees are just generalized LinkedLists. So, all of the things you learned about references with LinkedLists are going to apply to trees as well.

- Almost all the tree methods you write will be recursive (and will have a private helper that takes in the root).

- Make sure you understand all the traversals; the trick can be very useful.