

CSE 143X

Accelerated Computer
Programming I/II

Binary Trees

```

                                     1
                                    0
                                   00 00
                                  100 00
                                 10001 11
                                001 10
                               00 000 101
                              00 000 1001
                             000 00 0000
                            0001 000 001
                           11001 0001 010
                          11001 101100
                         11011 1001
                        1101 10101 01100
                       11001 1100 01100
                      0101 1100 01100
                     0111000110010
                    01000000100
                   0110000110
                  011100000 101
                 011100010111
                111 011100110
               01011100110
              011000110
             1011110110
            00111101101
           0011110010001
          11100111110010000111
```

Outline

- 1 `LinkedLists` to `BinaryTrees`
- 2 Why Do We Care About Binary Trees?
- 3 Printing Recursively
- 4 Binary Tree Traversals

Consider the following standard LinkedList:



Recall the definition of a ListNode

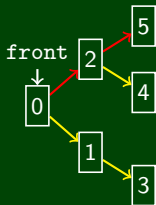
```
1 public class Node {
2     public int data;
3     public Node next;
4
5     public Node(int data, Node next) {
6         this.data = data;
7         this.next = next;
8     }
9 }
```

What if we added more fields?

- Multiple data fields?
- Multiple “next” fields?

Nodes with Multiple next Fields

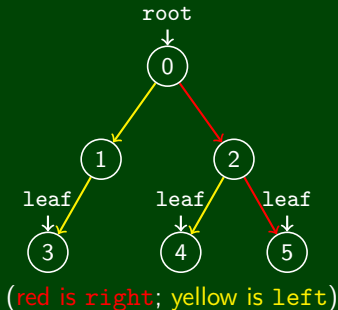
```
1 public class Node {  
2     public int data;  
3     public Node next1;  
4     public Node next2;  
5  
6     public Node(int data, Node next1, Node next2) {  
7         this.data = data;  
8         this.next1 = next1;  
9         this.next2 = next2;  
10    }  
11 }
```



(yellow is next2; red is next1)

Binary Trees

```
1 public class Node {  
2     public int data;  
3     public Node left;  
4     public Node right;  
5  
6     public Node(int data, Node left, Node right) {  
7         this.data = data;  
8         this.left = left;  
9         this.right = right;  
10    }  
11 }
```



Consider the following LinkedList of a mathematical expression:

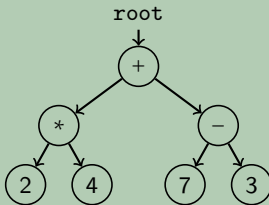
front



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?

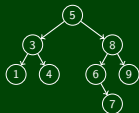


Now we can see the order of operations much more clearly!

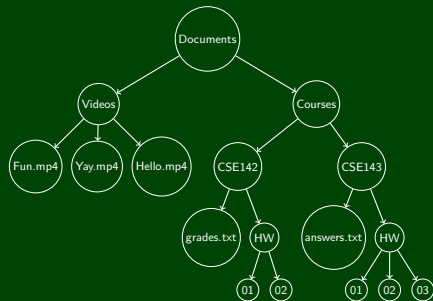
- Parsing (Programming Languages, Math, etc.)



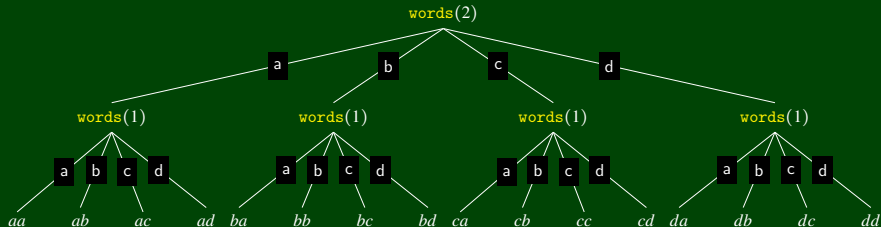
- Implementing TreeSet



- Directory File Structure



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



- Compression (this will be your last assignment!)

```
1 public void print() {  
2     Node current = this.front;  
3     while (current != null) {  
4         System.out.print(current.data + " ");  
5         current = current.next;  
6     }  
7 }
```

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

```
1 public void print() {  
2     print(this.front);  
3 }  
4  
5 public void print(Node c) {  
6     if (c != null) {  
7         System.out.print(c.data + " ");  
8         print(c.next);  
9     }  
10 }
```

To print a BinaryTree...

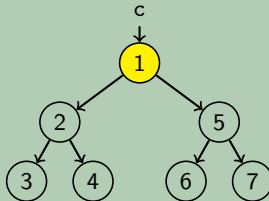
- Print the **root** of the tree
- Print the **left** of the tree (recursively)
- Print the **right** of the tree (recursively)

Code

```
1 public void print() {
2     print(this.root);
3 }
4
5 public void print(Node c) {
6     if (c != null) {
7         System.out.print(c.data + " ");
8         print(c.left);
9         print(c.right);
10    }
11 }
```

```
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

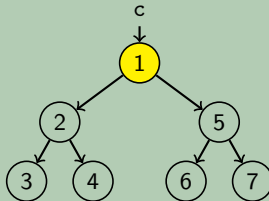


OUTPUT

>> 1

```
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

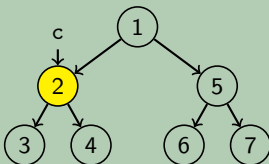


OUTPUT

>> 1

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 3     System.out.print(c.data + " ");
6                 4     print(c.left);
7                 5     print(c.right);
8             }
9         }
10    }
```

Trace

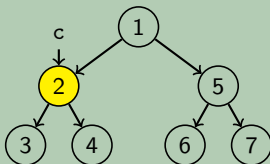


OUTPUT

>> 1 2

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

Trace



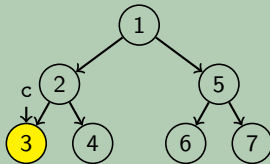
OUTPUT

```
>> 1 2
```



```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 3         1 public void print(Node c) { // c = ③
6                     4             2     if (c != null) {
7                         5                 System.out.print(c.data + " ");
8                             6                 print(c.left);
9                             7                 print(c.right);
10                            6             }
11                           7         }
12                          }
13                         }
```

Trace

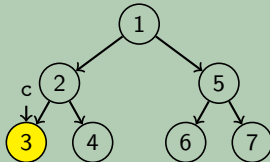


OUTPUT

>> 1 2 3

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 3     public void print(Node c) { // c = ③
6                     4         if (c != null) {
7                         5         System.out.print(c.data + " ");
8                             6         print(c.left);
9                             7         print(c.right);
10                        }
11                    }
12                }
13            }
14        }
15    }
```

Trace



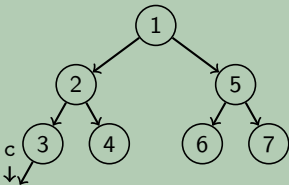
OUTPUT

>> 1 2 3

```

1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 1 public void print(Node c) { // c = ③
6                     2     if (c != null) {
7                         1 public void print(Node c) { // c = null
8                             2     if (c != null) { // c is null!
9                                 3         System.out.print(c.data + " ");
10                                4         print(c.left);
11                                5         print(c.right);
12                                6     }
13                            7 }
14                        6 }
15                    7 }
16                5 }
17            4 }
18        3 }
19    2 }
20 }
    
```

Trace

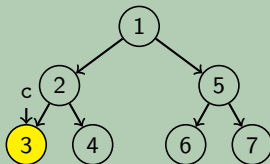


OUTPUT

>> 1 2 3

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 3     public void print(Node c) { // c = ③
6                     4         if (c != null) {
7                         5             System.out.print(c.data + " ");
8                             6             print(c.left);
9                                 7             print(c.right);
10                            }
11                        }
12                    }
13                }
14            }
15        }
16    }
17 }
```

Trace



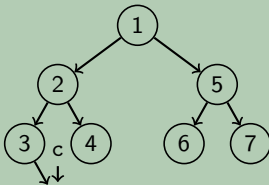
OUTPUT

```
>> 1 2 3
```

```

1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 1 public void print(Node c) { // c = ③
6                     2     if (c != null) {
7                         1 public void print(Node c) { // c = null
8                             2     if (c != null) { // c is null!
9                                 3         System.out.print(c.data + " ");
10                                4         print(c.left);
11                                5         print(c.right);
12                                6     }
13                            7 }
14                        6 }
15                    7 }
16                5 }
17            4 }
18        3 }
19    2 }
20 }
    
```

Trace

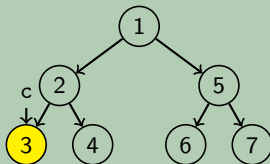


OUTPUT

>> 1 2 3

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 3         1 public void print(Node c) { // c = ③
6                     4             2     if (c != null) {
7                         5                 3         System.out.print(c.data + " ");
8                             6                     4             print(c.left);
9                                 5                 print(c.right);
10                                6             }
11                                7         }
12                            7     }
13                        }
```

Trace

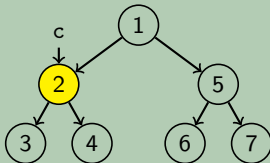


OUTPUT

>> 1 2 3

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

Trace

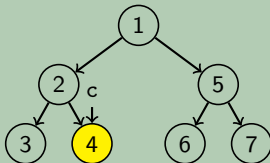


OUTPUT

>> 1 2 3

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 3     1 public void print(Node c) { // c = ④
6                     4     2     if (c != null) {
7                         5     3     System.out.print(c.data + " ");
8                             6     4     print(c.left);
9                                 7     5     print(c.right);
10                                6     }
11                                7     }
12                            }
13                            }
14                        }
```

Trace

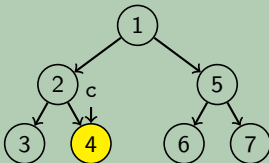


OUTPUT

>> 1 2 3 4


```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 3     public void print(Node c) { // c = ④
6                     4         2     if (c != null) {
7                         5         3     System.out.print(c.data + " ");
8                             6         4         print(c.left);
9                                 7         5         print(c.right);
10                                6     }
11                                7     }
12                            }
13                            }
```

Trace

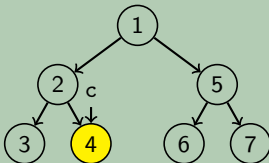


OUTPUT

>> 1 2 3 4

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 3     public void print(Node c) { // c = ④
6                     4         2     if (c != null) {
7                         5         3         System.out.print(c.data + " ");
8                             6         4         print(c.left);
9                                 7         5         print(c.right);
10                                6     }
11                                7     }
12                                }
13                                }
```

Trace

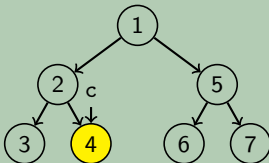


OUTPUT

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

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ②
4             2     if (c != null) {
5                 3     public void print(Node c) { // c = ④
6                     4     2     if (c != null) {
7                         5     3     System.out.print(c.data + " ");
8                             6     4     print(c.left);
9                                 7     5     print(c.right);
10                                6     }
11                                7     }
12                                }
13                            }
```

Trace

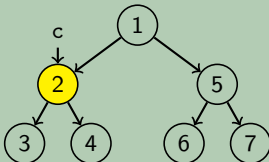


OUTPUT

>> 1 2 3 4

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

Trace

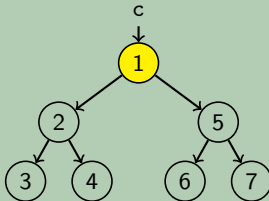


OUTPUT

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

```
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

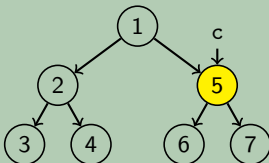


OUTPUT

>> 1 2 3 4

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ⑤
4             2     if (c != null) {
5                 3     System.out.print(c.data + " ");
6                 4     print(c.left);
7                 5     print(c.right);
8             }
9         }
10    }
```

Trace

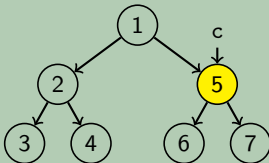


OUTPUT

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

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

Trace

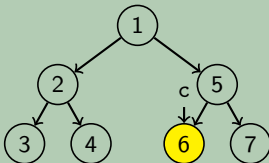


OUTPUT

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

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ⑤
4             2     if (c != null) {
5                 3     1 public void print(Node c) { // c = ⑥
6                     4     2     if (c != null) {
7                         5     3     System.out.print(c.data + " ");
8                             6     4     print(c.left);
9                                 7     5     print(c.right);
10                                6     }
11                                    7     }
12                                }
13                                }
```

Trace



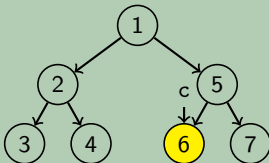
OUTPUT

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



```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ⑤
4             2     if (c != null) {
5                 3     1 public void print(Node c) { // c = ⑥
6                     4     2     if (c != null) {
7                         5     3     System.out.print(c.data + " ");
8                             6     4     print(c.left);
9                                 7     5     print(c.right);
10                                6     }
11                                7     }
12                            }
13                            }
14                        }
```

Trace

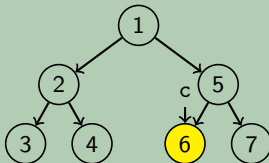


OUTPUT

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

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ⑤
4             2     if (c != null) {
5                 3     1 public void print(Node c) { // c = ⑥
6                     4     2     if (c != null) {
7                         5     3     System.out.print(c.data + " ");
8                             6     4     print(c.left);
9                                 7     5     print(c.right);
10                                6     }
11                                7     }
12                            }
13                        }
14                    }
```

Trace

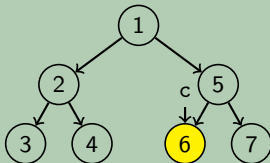


OUTPUT

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

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ⑤
4             2     if (c != null) {
5                 3     1 public void print(Node c) { // c = ⑥
6                     4     2     if (c != null) {
7                         5     3     System.out.print(c.data + " ");
8                             6     4     print(c.left);
9                                 7     5     print(c.right);
10                                6     }
11                                7     }
12                                }
13                            }
```

Trace

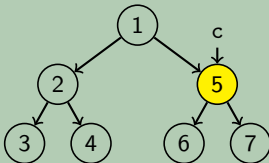


OUTPUT

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

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ⑤
4             2     if (c != null) {
5                 3         System.out.print(c.data + " ");
6                 4         print(c.left);
7                 5         print(c.right);
8             }
9         }
10    }
```

Trace

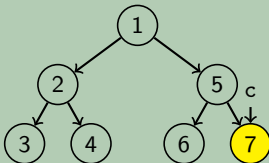


OUTPUT

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

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ⑥
4             2     if (c != null) {
5                 3     1 public void print(Node c) { // c = ⑦
6                     4     2     if (c != null) {
7                         5     3     System.out.print(c.data + " ");
8                             6     4     print(c.left);
9                                 7     5     print(c.right);
10                                6     }
11                                    7     }
12                                }
13                            }
```

Trace

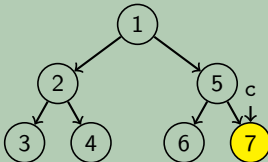


OUTPUT

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

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ⑥
4             2     if (c != null) {
5                 3     1 public void print(Node c) { // c = ⑦
6                     4     2     if (c != null) {
7                         5     3     System.out.print(c.data + " ");
8                             6     4     print(c.left);
9                                 7     5     print(c.right);
10                                6     }
11                                7     }
12                            }
13                        }
14                    }
15                }
16            }
17        }
```

Trace

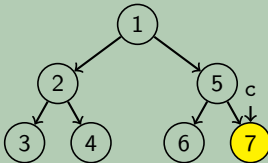


OUTPUT

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

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ⑥
4             2     if (c != null) {
5                 3     public void print(Node c) { // c = ⑦
6                     4         if (c != null) {
7                         5             System.out.print(c.data + " ");
8                             6             print(c.left);
9                             7             print(c.right);
10                    }
11                }
12            }
13        }
14    }
15 }
```

Trace

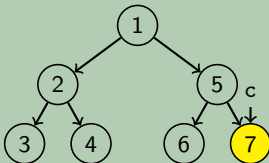


OUTPUT

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

```
1 public void print(Node c) { // c = ①
2     if (c != null) {
3         1 public void print(Node c) { // c = ⑥
4             2     if (c != null) {
5                 3     1 public void print(Node c) { // c = ⑦
6                     4     2     if (c != null) {
7                         5     3     System.out.print(c.data + " ");
8                             6     4     print(c.left);
9                                 7     5     print(c.right);
10                                6     }
11                                7     }
12                                }
13                            }
```

Trace



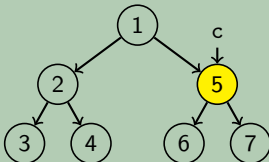
OUTPUT

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



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

Trace

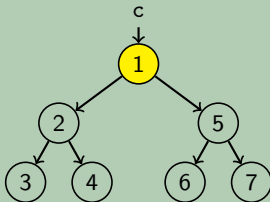


OUTPUT

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

```
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

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

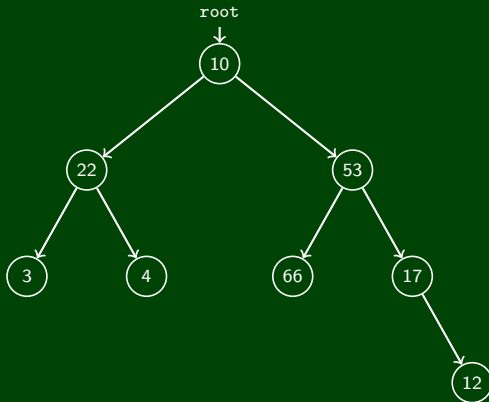
In-Order Traversal

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

Post-Order Traversal

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

Consider the following binary tree:

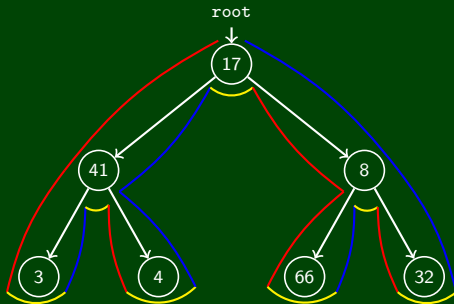


Compute the Pre-Order, In-Order, and Post-Order Traversals:

- 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**



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)`

```
1 public type method(...) {
2     return method(this.root, ...);
3 }
4 private type method(TreeNode current, ...) {
5     if (current == null) { /* DO BASE CASE */ }
6
7     // Do the left recursive case:
8     type leftResult = method(current.left, ...);
9
10    // Do the right recursive case:
11    type rightResult = method(current.right, ...);
12
13    /* Use the left and right results... */
14    return ...;
15 }
```

contains()

Write a method, in the `IntTree` class, called `contains()`:

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

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

```
1 public boolean contains(int value) {
2     return contains(this.root, value);
3 }
4 private boolean contains(IntTreeNode current, int value) {
5     /* If the tree is null, it definitely doesn't contain value... */
6     if (current == null) { return false; }
7
8     /* If current *is* value, we found it! */
9     else if (current.data == value) { return true; }
10
11     else {
12         boolean leftContainsValue = contains(current.left, value);
13         boolean rightContainsValue = contains(current.right, value);
14         return leftContainsValue || rightContainsValue;
15     }
16 }
```

- 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.