# Overview

In this lab, you will write methods to traverse (walk through) a BST using an [in-order traversal](https://en.wikipedia.org/wiki/Tree_traversal#In-order_(LNR)). You will do this two different ways. First, you will write a recursive method which traverses the entire tree and returns a string result. Second, you will use an iterator to be able to step through the tree one node at a time. Both will visit nodes in the same order, but the iterator will take several different methods to be able to get the current node and move correctly to the next.

# Objectives

* Practice creating a class
* Practice with BSTs (binary search trees)
* Practice with recursion
* Practice creating Iterators
* Apply test cases to your program

# Set-up

1. Create a folder on your local machine for your Java program, you can name it whatever you like
2. Start Visual Studio Code (VS Code)
3. In VS Code, Open that newly created folder.
4. Download the starter code from the course public folder ([public/16L](https://cs.unh.edu/~cs416/public/16L)) and save it in the src directory of your new project.

# **Implementation**

The starter code includes **BinarySearchTree.java**, **Data.java**, and **TreeApp.java**. You should not change anything in **Data.java**, or **TreeApp.java** but you can run the main method of **TreeApp.java** to test your work. Your task will be to complete several methods in **BinarySearchTree.java**. You will also have to complete some methods within the inner class InOrderIterator inside **BinarySearchTree.java**. You may notice that the hasNext method of that class is already complete.

**public String inOrder()**

A helper method for inOrder( Node n ). Should call inOrder on the root of the tree and return the resulting string. May need to modify the result string to remove trailing comma/space.

**public String inOrder( Node n )**

Performs an in-order traversal of the subtree beginning with n and combines all the subtree's data into a string, with each item separated by a comma and a space. Remember that an in-order traversal means recursively processing the left child, processing the current node, and then recursively processing the right child. Your base case should be when n is null.

**public String leftMost( Node n )**

Will be used by the iterator to find the next node to process as it steps through the tree. Returns the node furthest to the left in the subtree starting at n. This can be done two possible ways:

* Recursively by calling on n.left if n.left is not null and returning n otherwise. If n is null, return null.
* Iteratively by setting n to n.left until you reach a node that has no left child.

**public Node unfinishedAncestor( Node n )**

Will be used by the iterator to find the next node to process as it steps through the tree. Returns the closest unfinished ancestor of n (an unfinished node would be a node which was reached by a left child, because the node itself and the right child still need to be processed.) This can be done two possible ways:

* Recursively by checking whether n is the right child of its parent. If it was, you can call recursively on that parent and return the result. If n is not the right child of its parent (or its parent is null), you can return the parent.
* Iteratively by setting n to n.parent until you reach a node that is a left child of its parent, or until n.parent is null.

**public InOrderIterator()**

Constructor for the inner class. You will need to set next to be the first node that would be visited by an in-order traversal. This will be the left-most node from the root.

**public boolean hasNext()**

Returns true if there are nodes that still need to be traversed, false otherwise. Should be based on whether next is null or not.

**public Data next()**

Returns the data from the current next node. Before returning, updates next to the next node that should be visited by the traversal. If the current next node has a right child, its subtree needs to be traversed, so next should be set to the leftmost node in the subtree of next.right. Otherwise, the whole subtree under next has been traversed and next should be set to the closest unfinished ancestor of next. If there are no nodes left to visit (the current next node is null), throw a NoSuchElementException.