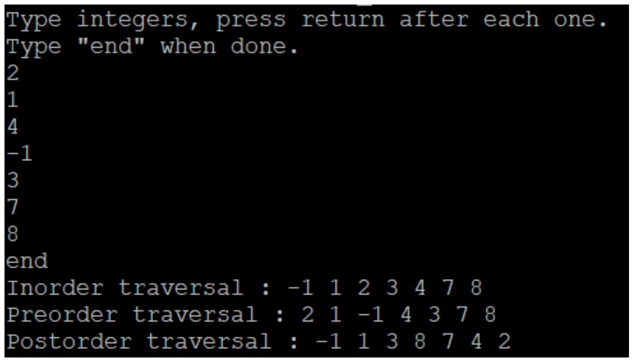
CSCI241, Winter 2020, Lab 4
Check Canvas for due date and time

Unlike the homework assignments for CSCI241, you are **encouraged** to work with your peers in completing the labs. However, each student must have their own submission; you cannot exchange files. If any of this is unclear, please ask for further clarification.

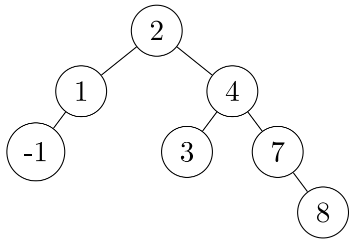
1 Overview and Goals

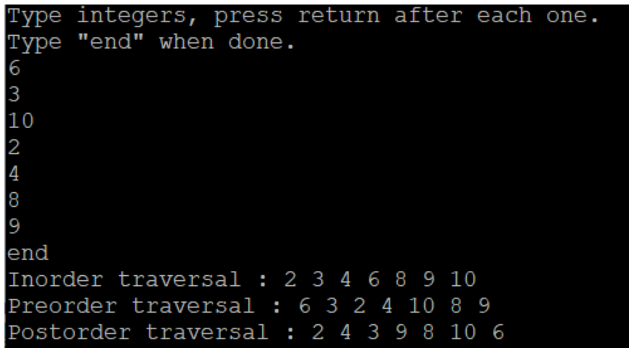
For this lab, you will implement the classes `BST` and `Node` for the purpose of creating a binary search tree. You will implement the in-order, post-order, and pre-order print routines. The `main` method is in the BST class. The tree traversal methods for the BST will be recursive. Use this lab to become familiar with the recursive implementations of trees, which is what you'll need to implement for homework 3.

Screenshots of the behavior of the program are shown in Figure 1, along with schematics of the two BSTs that are created.



```
Type integers, press return after each one.
Type "end" when done.
2
1
4
-1
3
7
8
end
Inorder traversal : -1 1 2 3 4 7 8
Preorder traversal : 2 1 -1 4 3 7 8
Postorder traversal : -1 1 3 8 7 4 2
```





```
Type integers, press return after each one.
Type "end" when done.
6
3
10
2
4
8
9
end
Inorder traversal : 2 3 4 6 8 9 10
Preorder traversal : 6 3 2 4 10 8 9
Postorder traversal : 2 4 3 9 8 10 6
```

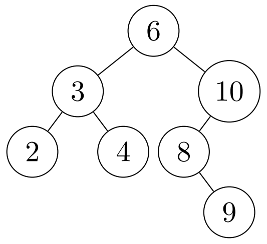


Figure 1: Example invocations (left) of `BST`, in which the user is prompted to input integer values, which are used to build a BST. The schematics of the BSTs build are shown right.

Unlike lab 3, the instructions for this lab are much more high-level. You are provided with the function signatures that you need to implement, but the details are left to you. Hint : Refer to the lecture slides for pseudocode – in many cases, almost "real" code – the recursive implementations of insert, as well as in-order, pre-order, and post-order traversal.

2 Create a new branch

In your local git (log onto a lab linux computer, and using the terminal, navigate to the directory where you have your code for CSCI241 and where you cloned your git repo), create a new branch named *lab4* – remember to make a new branch from the *master* branch only; refer to lab 1 for a refresher – and then create a new directory, *lab4*. Do all of your work in your lab4 directory. Commit and push often. See previous labs for a refresher on how to check out a branch, how to commit, and how to push to a remote branch.

3 Code Description, `Node` and `BST` classes.

For this lab, you will write two .java files, *Node.java* and *BST.java*. The BST java file contains the `main` routine. Use the IDE of your choice to create these two .java classes.

The high-level description of the `Node` class is the following :

- class variable `value`, private and of type int, is the value of each node
- class variables `left` and `right`, both of type `Node`, refer to the left and right children of "this" node. One or both of these values can be null.
- The constructor
- The function `getValue()` which returns an int and is public. Because the value of the node is private, if you need to retrieve the value of the node from code that is outside of this class, a public getter function is needed.
- An *insert* function, which is the recursive implementation of adding a value (node) into the BST.

The skeleton of *Node.java* (indicating the exact function signatures you need), is the following:

```
public class Node{

    private int value;
    Node left, right;

    public Node(int value){
    }

    public int getValue(){
    }

    public void insert(int value){
    }
}
```

The `Node` class is a simple class with just three class variables, and three functions, one of which is the constructor.

The BST class is a bit more complex than the Node class, because it contains the traversal functions, as well as main, which prompts the user for inputs and processes what is typed via the keyboard.

The high-level description of the BST class is the following :

- Private and static variable overallRoot, which is of type Node. This follows exactly the example from the lectures slides
- The constructor
- The function insert which calls the insert function on the root of the tree, and the function getRoot, which is the public getter function invoked when the printInOrder, printPostOrder, and printPreOrder functions are invoked inside of main.
- The functions printInOrder, printPostOrder, printPreOrder which traverse a tree in the order specified, and print the nodes' values as a node is processed. Refer to the lecture slides for the pseudocode of these three functions.
- The main function. Much like you did for the previous lab, this function is reading input from the keyboard, so its declaration specifies that it throws an IOException. The function sets up a buffered reader, creates a new BST by calling the constructor, prints a few instructions to the screen (see sample output at the onset of this lab), and prompts a user for integer (negatives are fair) values, which are inserted into the BST. After a user specified *end*, the in-order, pre-order, and post-order print functions are invoked.

The skeleton of BST (indicating the exact function signatures you need), is the following:

```
public class BST{

    private static Node overallRoot;

    public BST(){    }

    public static Node getRoot(){    }

    public static void insert(int value){    }

    public static void printInOrder(Node node){    }

    public static void printPreOrder(Node node){    }

    public static void printPostOrder(Node node){    }

    public static void main(String[] args) throws IOException {    }
}
```

The BST class has a single class variable, and 7 functions.

4 File header, Import Statements

The top of your *BST.java* file should contain your name, date, and a short description of your program. Also include the necessary IOException Buffered and InputStream Reader imports.

5 Compiling and Running

To compile and run your program (which you should do incrementally), issue the following:

```
javac BST.java
java BST
```

6 Pushing to your git

Add, commit, and push your .java files only, to your lab4 branch. **By convention, compiled code, nor object files, are NEVER (except in special, rare cases, and this is not one of them!) pushed to a git repo**.

```
git add BST.java
git add Node.java
git commit -m "Added this-and-that code"
git push origin lab4
```

Rubric

Node.java and *BST.java* have been pushed to the lab4 branch, which contains the lab4 directory. Class files nor anything other than .java files, are excluded	2
The asked-for functions (with EXACT signatures, return types, etc.) have been declared in *Node.java* and *BST.java.*	5
Each of the two classes `Node` and `BST`, and asked-for functions, correctly implement the asked-for functionality – recursive in the case of the traversals, and insert functions.	30
The `main` function prompts the user for the value(s) of a BST, and that BST is built, followed by the execution of the three traversals.*end.*	10
The program compiles via `javac BST.java`, and upon invocation, exhibits the correct behavior.	5
The code is commented – including your name, date, brief description, etc. at the top of the .java file, variable names are appropriately named, and good coding practices are followed.	3
Total	55 points

To confirm that you've correctly pushed, open your browser and navigate to the gitlab for CS, and double check that all of the files, as well as branch and directory, are there.

Note : It is a good idea to NOT modify the git repo using the web interface. If you do add files, remove files, etc., via the web interface (insteada of via the command line), you'll need to `pull` the changes when logged in to your linux account.