# CSC 220 Data Structures

## Program #6

## Trees

Your objective is to implement the main operations of a binary search tree from the lecture. To be clear, this entails creating a class for your tree nodes, then creating another class for your binary search tree.

Create a class called **`Node`** (this is different from your Linked List Node class) that has three fields: a field called `left` for the left child link, a field called `right` for the right child link and a field called `data` for the int data being held. These fields should be private with public accessors and mutators (i.e. `getLeft, getRight, getData, setLeft, setRight` and `setData`). This `Node` class can either be a public class in its own file (i.e. Node.java) or a class without the public keyword in the same file as your BST class (described below).

Create a public class called **`BST`** (and put it in a file called BST.java) that has the following:

- Fields
  - public field called `root` that points to the root of the tree. This **must** be public for the visualization to work.
  - Any other fields you need
- Public methods
  - A default constructor (i.e. a constructor that doesn't take any parameters). This will initialize `root` to null.
  - `insert(value)`
    - Takes an int as input and inserts it into the logical place in the tree
    - You don't have to worry about duplicate values (see 'Edge Cases and Notes' for more information)
    - You may use either recursion or iteration here.
  - `search(value)`
    - This method takes an int as input and returns true if the given value is in the tree, false otherwise. You don't have to return the location in the tree, just a boolean indicating whether the value is found.
    - If root is null, you should return false (since the value could not be found)
    - You may use either recursion or iteration here.
  - `delete(value)`
    - This method takes an int as input, searches for it in the tree and deletes it according to the method described in the nodes.
    - If the value given is not found or if the root is null, this operation does nothing
    - Do **NOT** even attempt this method until everything else works!
    - You may use either recursion or iteration here.

- `min()`
  - Returns the smallest value in the tree.
  - If the root is null, return -1
  - You may use either recursion or iteration here.
- `max()`
  - Returns the largest value in the tree.
  - If the root is null, return -1
  - You may use either recursion or iteration here.
- `inorder()`
  - Traverses the tree according to the inorder traversal described in the notes.
  - Do not print values. Instead, construct a String, concatenating the node values as they are processed with either a comma or a space in between values. Return this String.
  - If the root is null, return an empty string (i.e. "" with nothing in between the quotes)
  - You **must** use recursion here.
- `preorder()`
  - Traverses the tree according to the preorder traversal described in the notes.
  - Do not print values. Instead, construct a String, concatenating the node values as they are processed with either a comma or a space in between values. Return this String.
  - If the root is null, return an empty string (i.e. "" with nothing in between the quotes)
  - You **must** use recursion here.
- `postorder()`
  - Traverses the tree according to the postorder traversal described in the notes.
  - Do not print values. Instead, construct a String, concatenating the node values as they are processed with either a comma or a space in between values. Return this String.
  - If the root is null, return an empty string (i.e. "" with nothing in between the quotes)
  - You **must** use recursion here.

**Edge Cases and Notes:**

You do **not** have to use generics for this. For this assignment, just use int as the data type for the data stored in your binary search tree nodes.

Do **not** worry about handling duplicate values. You can safely assume that I will not attempt to store the same value twice in your tree, so your tree does not have to handle this situation.

Every function should handle the case where the root is null. See the function description above for instructions on how to handle it for that function.

You may use helper functions to help you with the recursion. For example, you could have a public `inorder` method (i.e. `public String inorder()`) that simply calls a private `inorderRecursive` method (i.e. `private String inorderRecursive(Node cur)`) and sends it the root node to start the inorder traversal (i.e. `inorderRecursive(root)`). This is helpful since this means a developer using your tree class would only need to call `tree.inorder()`, assuming they have a BST object named "tree", without needing access to the root.

**Main part:**
Make a separate public class called **Main** and put it into its own file. Put your entry point inside this class. Make sure to put **Vis.class** and **Color.class** in the **same folder** as your .java files if you plan on running your code on the command line (more on this below).

Create an object of your BST class (call it whatever you like). You can now call whichever operations you want on your object. This will initialize your tree by giving it a set of nodes it will contain. When you are ready to see the visualize of your tree and test it in real-time, call

```
Vis.test(nameOfYourTreeObject);
```

Here is an example (this code would appear inside your main function in your Main class):

```
// instantiate our BST object
BST tree = new BST();

// load it up with some initial values
tree.insert(5);
tree.insert(15);
tree.insert(2);

// are you on Windows?
Vis.runOnWindows = false; // set to true if running on Windows

// test it out
Vis.test(tree);
```

The test function will take your BST object and show it on the screen (in beautiful ASCII art fashion). It will also display a menu wherein you can test out any operation you want. All the operations you must implement are able to be tested here.

**Running in an IDE:**

The included “Color.class” and “Vis.class” files will work just fine if compiling and running on the command line. For using an IDE, I have included “vis.jar”. If you are using Eclipse, go to Project > Properites > Java Build Path > Libraries, make sure “Classpath” is selected below and click on “Add External JARs…”. Navigate to where “vis.jar” is on your computer (wherever you downloaded it), select vis.jar and then click open. Now click “Apply and Close”. Everything should run as expected. You might have to set `Vis.runOnWindows` to `true` to prevent seemingly random characters from appearing in the output. If you are using another IDE, you will need to look up online how to include a JAR file.

**VERY IMPORTANT NOTE:**
I designed this assignment to encourage an iterative development cycle. What I mean by that is that you should be getting **one** thing to work at a time. Before the 1990's, we used to think that the best way to program was to try to do everything at once. In the mid 1990's, a different school of thought appeared that stated that we needed to do one thing, make sure it works, and then move on to the next thing. This was called "agile development". From that point on, agile development became the most common

means of development, with studies showing three times more likelihood of a successful project over using the old approach. So, please, start out by just getting your `Node` class to work. Test it by creating a `Node` object in your main, and calling `getData()`, `getLeft()` and `getRight()` on it. Now create your `BST` class and just give it a constructor and a field called `root`. Test that in your main. Now work on your `insert` function. You can test this using `Vis.test()`. Only when that completely works in every case you can think it (insert many different values to test it), do you move on to doing `search` or another operation. Don't Worry! If you accidently call a function that you haven't implemented, the `Vis.test()` function won't crash your program. It might even be helpful to call ALL functions from the menu just to see what they say ☺

**For submission:**
Your main method (i.e. entry point) **must** call the `Vis.test()` function sending it your BST object. I will test your code in this interactive environment. Feel free to load up your tree with whatever values you like before calling `Vis.test()` but do note that there is a limit to the number of nodes `Vis` will display so do not go pass this limit.

Submit all .java files you have written by putting them into one zip file and submitting the zip. Do not submit any .class nor .jar files.

**Rubric:**

| # | ITEM | POINTS |
|---|---|---|
| 1 | insert method | 6 |
| 2 | search method | 6 |
| 3 | delete method | 6 |
| 4 | min method | 4 |
| 5 | max method | 4 |
| 6 | inorder method | 4 |
| 7 | preorder method | 4 |
| 8 | postorder method | 4 |
| 9 | uses Vis correctly for visualization | 2 |
| | **TOTAL** | **40** |

| # | PENALTIES | POINTS |
|---|---|---|
| 1 | Doesn't compile | -50% |
| 2 | Doesn't execute once compiled (i.e. it crashes) | -25% |
| 3 | Late up to 1 day | -25% |
| 4 | Late up to 2 days | -50% |
| 5 | Late after 2 days | -100% |