**Programming Assignment 2 Report**

**By: Tai Dao**

**BST Design and Concepts**

**Node Class**

I made a Node class to represent each node in the binary search tree. This node class typically contains pointers to the left child, right child, and parent node with getters and setters for each. These pointers typically point to nothing until the tree is built up. It also contains a key which is an object called Process.

I made this class implement compareTo for whenever I needed to do comparisons between one node and another. This compareTo compares based on the key’s (The key is the Process) priority.

**Tree Class**

I’ve implemented everything based on CLRS. The only thing different is that I added a method called

**public** **void** **processInsert**(**Process** p) {

**Node** **z** = **new** Node(p);

treeInsert(**this**, z);

}

This is needed because I can’t just insert a Process into a tree with treeInsert. I needed to first create a Node with its key set as the Process that was to be inserted. Then I could call treeInsert().

Another method I modified was treeSearch. It wasn’t mentioned in the pseudocode in CLRS, but I needed to add some return nulls for when there is no leftChild / rightChild. This is for error handling, for the case that the google is searching for a priority # that doesn’t exist.

**Process Class**

In my design I have one Object called Process. When this object is created with the constructor Process(int currentPID) a random priority from 0-9999 is assigned and a processID is assigned based on the currentPID. The currentPID is tracked in main and starts from 0. It implements comparable in order to make it possible to compare each process to each other based on priority. It has getters for the processID & priority and a setter for priority.

It also has another constructor Process() which is utilized to create a Process with the lowest possible priority of 0 and a null process PID.

**BstTester**

20 processes are created with random priority and inserted into the binary search tree on start. Then a menu with options appear to test the Tree. Everything else is pretty self-explanatory.

The menu allows the user to display a sorted list of processes in the BST. It also allows the user to enter in the priority of a process that they want deleted from the BST. This calls a treeSearch to find the Node to delete then calls treeDelete to delete that Node.

Search wasn’t required for this assignment. I just thought it was implied in order for me to test treeDelete I needed search.

Lastly this menu allows the user to insert a random process into the BST.

**Problems encountered with implementation of BST**

Well at first I was unsure of how you would insert a Process into a BST that only contains nodes. I ended up solving this problem by created a new Node with the key set to the Process every time I wanted to insert a new Process into the BST.

Initially for my treeSearch I was using Process as the parameter for key. Then, I realized I wanted Process’s priority as the key.

**Lessons learned**

**Procedures Included in – ‘Procedures Readme.txt’**

**Screenshots Included in – ‘Screenshots’ folder.**