CSE 330 LABORATORY -- Week 7, Spring 2018

Instructor: Kerstin Voigt

In this lab we will enhance our BinarySearchTree implementation with additional functionalities that are necessary in order to build the Set and Map data structures on top of binary search trees.

Because these enhancements are fairly involved, this lab will be run in a more instruction-based "follow-the-leader" type manner. Follow the instructions and make sure you keep up. If you find that the pace is too fast, please speak up and we slow down …

Step 1: Obtain a fresh copy of the BinarySearchTree implementation with `BinarySearchTreeLab7.h` from the instructor's directory.

Step 2: Make additions to the code that will have each BinaryNode (node that contains the data values) have one addition pointer to its "parent" node. With this additional link, we will be able to traverse the tree in both directions: (1) down to the leaf nodes, and **(2) upward to the root node.**

Follow the leader in making all additions to the code.

Step 3: Test your new version of class BinarySearchTree by making sure that building a binary search tree, and removing elements are still working.

Step 4: Insert **into** class BinarySearchTree under 'public:' the following iterator class:

```
struct BinaryNode;

class iterator
 {
   public:

      iterator() : current(0) {}

      iterator(BinaryNode* t) : current(t) {}

      T & operator *() const
      {
        return current->element;
       }

      iterator & operator++()
      {
       // MUCH TO BE FILLED IN …
       return *this;
      }
```

```
        iterator & operator++(int)
        {
             iterator old *this;
             ++( *this);
             return old;
        }

        bool operator ==(iterator other) const
        {
              return current == other.current;
        }

        bool operator != (iterator other) const
        {
              return current != other.current;
        }

protected:

        BinaryNode * current;


        bool is_root(BinaryNode *t)
        {
              // FILL IN
             return true; // replace
        }

        bool is_left_child(BinaryNode * t)
        {
              // FILL IN
             return true; // replace
        }

        bool is_right_child(BinaryNode * t)
        {
              // FILL IN
              return true;  // replace
        }

        BinaryNode * leftmost(BinaryNode * t)
        {
              // FILL IN
              return t;  // replace
        }

        BinaryNode * follow_parents_until_leftchild(BinaryNode * t)
        {
             // FILL IN
             return t;  // replace
        }

        friend class BinarySearchTree<Comparable>;

};
```

Step 5: Add member functions begin() and end() to class BinarySearchTree:

```
iterator begin() const
 {
       BinaryNode* t = root;
       while (t->left != 0)
             t = t->left;
       iterator beg(t);
       return beg;
 }

 iterator end() const
 {
       iterator end(0);
       return end;
```

Step 6: Test your code again. It should still compile and run without using any of the partially implemented iterators.

Step 7: FOLLOW THE LEADER in implementing iterator::operator ++() and all necessary "helper functions"

Step 8: Test the user of iterators for BinarySearchTrees …. int main() will be provided.

Credit for this lab: After working diligently on the above, sign up on the signup sheet.