Lab Exercise 9  
Cursor Over Binary Search Trees

Just like other Collections we’ve used this semester, binary search trees (BST) may allow iteration over their elements through use of an Iterator. In this lab, instead of using an Iterator, we will implement a cursor over a BST. You will be required to implement the following method for a BST of integers:

- advance()

To help illustrate this, we provided an applet that prints a BST. The applet will highlight the current node when there is one.

Using Eclipse, import “Lab9”:

/afs/cs.uwm.edu/users/classes/cs351/401/pantherid/git/lab9.git

1 advance()

The **advance()** method performs an in-order traversal controlled by the caller so the elements in the tree would be returned in order. We use a `current` reference to keep track of the currently-traversed node. Initially, `current` is null, because there is no current element. The **start()** method sets `current` to the node that contains the smallest integer. An example is given below.

Each time the method **advance()** is called, we can get the new current value using **getCurrent()**. However, updating the reference `current` requires more work. In this lab, we use the parent pointers to find the next element in the tree. Refer to Section 4, “Parent Pointers”, of the handout on the course website for more details: [http://www.cs.uwm.edu/classes/cs351/navigating-trees.pdf](http://www.cs.uwm.edu/classes/cs351/navigating-trees.pdf)

2 Using Parent Pointers

Let us practice the three-step methodology on this task.

2.1 Step 1: Search in the tree for the next element

On the following tree, suppose each edge is bi-directional, i.e., you can move up and down in the tree. Write down the next element next to each element in the tree.

```
100
 /   \
/     \ 
/       \
50  150
 / \   \
/   \   \
10 90 125 200
 / \   \
/   \   \
25 75 130
 /   \
60 135
```
2.2 Step 2: Cases

Now try to figure out a way to solve the problem in general. Consider the following cases, for each of them, find an example in the above tree and generalize your solution.

case 1. The current element has right child

   case 1a. The right child does not have left child, which element is the next?

   case 1b. The right child has left child, which element is the next?

   case 2. The current element does not have right child

   case 2a. The current element is the left child of its parent, which element is the next?

   case 2b. The current element is the right child of its parent, which element is the next?

2.3 Step 3: Code

Now implement the advance() method. Notice that in Step 2, case 1a and case 1b are essentially the same case (so are case 2a and case 2b).

3 Running the program

Run the applet and validate that your advance() method works correctly as your answers in Step 1. You should see a pop-up dialog when you try to advance when there is no current. You should not see any other error messages.

See your TA to receive credit for this lab.