1. Define a class Stack with the following methods:

   (a) initialize initializes a stack by defining an instance variable @items to nil.

   (b) push pushes an item on the stack. It takes one argument, the item to push on the stack. The return value should be the stack itself.

   (c) pop pops an item off the stack. It takes no arguments, and returns the item popped from the stack. Calling pop on an empty stack raises an exception.

   (d) empty? tests whether the stack is empty.

   Internally the stack should use a linked list to store the items. Define a class Link for this purpose.

   In your tests, provide tests for each of the methods of stack, including a test that demonstrates raising an exception when popping an empty stack.

2. Implement binary trees by defining two classes Leaf and BinaryNode with the methods described below. A Leaf has one value and a BinaryNode has a value at that node, and also left and right child nodes. Leaf and BinaryNode should both include the Enumerable mixin.

   (a) Leaf's initialize takes one argument, the value at that node.

   (b) BinaryNode's initialize takes three arguments: the value at the node, and the left and right children (both assumed to be either leaves or binary nodes – no need to check).

   (c) each for both Leaf and BinaryNode takes a block and invokes it with each value in the tree. The tree should be traversed using an inorder traversal (i.e. for a binary node, call the block on the left child, then on the value, then on the right child.

   (d) == tests whether two nodes are structurally equal (so for binary node you should recursively test whether the left and right children are equal, as well as the value at the node). Following the Duck Typing philosophy, == should not build in the class name Leaf or BinaryNode. However, you can test whether the argument is an instance of the same class as the receiver (self.class).

   Write unit tests that demonstrate that each of your methods is working correctly. In particular, you should have a test that only passes if the tree is traversed using an inorder traversal (and should fail if it does say a postorder traversal, or doesn’t visit every node).

   By using the Enumerable mixin, you get a large number of useful methods automatically, such as min, max, sort, collect, find, reject, and others. Also include a unit test that shows that min gives the correct answers for both leaves and binary nodes.

   Hint: for BinaryNode, you may want to convert from blocks to Procs and back. (There is another way to write the code that avoids this; which is better is a matter of taste.) If you do want to do this, there is an example linked from the 341 Ruby web page. In any case, the each method for Leaf is straightforward – for this method don't convert the block to a Proc.
**Turnin:** Turn in your two files, one with the definitions definitions and the other with your unit tests. You don’t need to turn in a script showing your program running — the TAs can just run the unit tests for that.