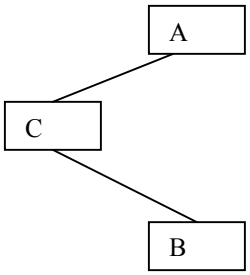
Project Description: Squeak Smalltalk Exercise

Define a binary tree class and construct a tree internally via Smalltalk statements, e.g.
node1 _ BinTree new: 'A' "build a new tree with node labeled A".
node2 _ BinTree new: 'C'.
node3 _ BinTree new: 'B'.
node1 addLeftKid: node2.
node2 addRightKid node3.

At this point, the tree looks like:



Perform traversals on the tree; print the node labels as they are encountered
Inorder traversal (left root right) C B A
Preorder traversal (root, left, right) A C B
Postorder traversal (left, right, root) B C A

Note: view a tree as a ***collection*** (ordered) of nodes; use *first*, *next* to yield nodes in indicated order; you should have a *BinTree* class and traversal classes with constructors - e.g. you would like to do something like:

suppose *theTree* is the variable referencing the tree just constructed;
traversal _ InorderTraversal new: theTree.
"the statement above performs initialization; it DOES NOT create a new collection"
nextLabel _ traversal first.
(nextLabel notNil) whileTrue: [nextLabel print. nextLabel _ traversal next]
...

DO NOT create a collection, besides the original BinTree; *simply keep track where you are traversing in the tree* (for each traversal type) so you can determine which node to yield for the next request of first/next. DO NOT include a link from each node to its parent. **For this part of the exercise you are only required to provide the code for the *inorder* case, not the *pre/post-order* cases.**

Notes:
Since you will be simulating recursion you will need to maintain an explicit stack to be used for continuing each time the user requests a *next* call. The stack will maintain a sequence of nodes leading back to the root from the current node being scrutinized. DO NOT traverse the tree more than once when implementing *first/next*.

In summary, you will need to

- build trees as described above – use my class names and other id's where provided; I might run your code so my test harness will expect to call classes/methods that I have named above (*InorderTraversal, first, next, BinTree, addLeftKid:, addRightKid:)* .
- provide in-/pre-/post-order traversal code without stopping at each step
- provide inorder traversal using first/next

Turn in:

1. Test cases
2. Class diagrams
3. Documentation on your traversal algorithms
4. Smalltalk code – use Squeak Smalltalk

A note on documentation: since you will develop your own inorder traversal algorithm to be used with *first/next*, it is essential to provide good algorithm documentation so I can easily understand the methodology you used. To this end, it's best to use lots of pictures, especially pictures depicting the various stages of the traversal process in a step-by-step fashion. It's ok to provide more detail than necessary but it's not ok to leave me guessing on the details of the algorithm if you do not use visual aids to assist in the description. For instance, it would be best to provide a step-by-step description of your algorithm using a sample tree to help me understand precisely how the traversal is performed and what information/data structures are used, as well as how the data structures are used.