CS3310 – Kaminski - Fall 2014 CountriesOfTheWorld App 1.0

Asgn 2 Project Specs – PART B

[Part A discusses A2 overview & DA dataTable aspects]

NameIndex OVERVIEW

In A2, NameIndex is still an **internal** index (with name as the key) as it was in A1.
However, for A2:

1. It MUST be implemented as a **binary search tree** data structure (vs. a sequential list)
   - *[FAQ: No, you may NOT use the built-in tree structure]*

2. It MUST use an array of BST nodes (objects) rather than parallel arrays (if that's what you did in A1)
   - where bstNodes are objects of the BSTNode class type
   - *[FAQ: Yes, there are MANY bstNode objects (in memory at the same time) since this is an INTERNAL data structure, not an EXTERNAL one]*
   - *[FAQ: This is NOT the conventional way to implement BST's (found in Data Structure books or online). NOR is this the conventional Array Storage for binary trees used for heaps which uses implicit pointers. We're using this ARRAY of objects approach because we need to dump & load it to port it between program-runs. More on this in class.]*
   - You MUST use explicit "pointers" (i.e., array subscripts) rather than C-style points or Java/C# references.
   - You MUST use -1 for "points nowhere".
     [FAQ: NULL won't work since these "pointers" are int subscripts].
     [FAQ: 0 won't work since that's a valid storage location in arrays].

3. It uses BST algorithms for the code-bodies for NameIndex methods including:
   a. insertIntoNameIndex(name, drp)
   b. selectByName(name)
   c. selectAllByName()
   d. deleteByName(name) - which is still a dummy stub

   [FAQ: No, you may NOT use the built-in "magic" tree handler methods – they must be explicitly written, "manual labor" method bodies].

Other BST Issues

Node Storage

A bstNode object is of BSTNode class type
(where the class is a sub-class or separate class from NameIndex class).

BSTs contain these fields:	LeftChildPtr, KeyData, RightChildPtr
Indexes contain these fields:	KeyData, DataRecordPtr
So a single bstNode contains these 4 fields:	leftChPtr, name, drp, rightChPtr

*[FAQ: Use a static array of bstNode objects of size MAX_N_LOCATIONS = 40 for now.
However, given our RawData test file, we'll only really need 25 locations,
plus a few more potentially during UserApp doing some Inserts.
But we're NOT implementing delete, so. . .]*

*[FAQ: Do NOT use 40 or MAX_N_LOCATIONS to control such things as:
dumping to the Backup file (use n instead)
loading from the Backup file (use "watch for EOF" instead)
pretty-printing (use "watch for EOF" instead)].*

Header data

A BST data structure needs an additional field (besides node storage): rootPtr

Manual space management needs an additional field: n
(which can be used for nextEmptyPtr since we're starting with storage location 0, not 1, and we're not implementing DELETE)
[FAQ: insertIntoNameIndex increments n]

"Pointers": leftChPtr, rightChPtr, rootPtr, n (when used as nextEmptyPtr)
These are actually integer subscript values, which "point to" some location in the array.
[FAQ: Yes, rootPtr starts out at 0 and stays there because we're not doing any DELETEs].

Space management (of array storage)

- Use manual "static space management" using n (which is really nextEmptyPtr)
  - n is initialized (in the constructor?)
  - n is incremented in insertIntoNameIndex
- We're not implementing deleteByName, so we don't have to worry about returning a deleted node to the available node pool. So n is always the next empty location

Use proper BST algorithms

[FAQ: You don't have to use "MY" algorithms, specifically. You may use iterative or recursive algorithms. But you MUST tailor ANY algorithms to deal with MY SPECIFIC REQUIREMENTS FOR HOW THE BST IS IMPLEMENTED].

- selectByName uses the BST search algorithm
  Doing a LINEAR search will result in TAKING LOTS OR POINTS OFF
- deleteByName is still a dummy stub (i.e., there's a proper callable method interface)
- selectAllByName uses binary tree's inorder traversal
  Doing any kind of SORT will result in TAKING LOTS OR POINTS OFF
  [FAQ: Note that a BST is in logical sequence (i.e., logically sorted), but it's NOT in physical sequence (i.e., physically sorted). So if you looked at the array (or the Backup file), you would NOT see the names in alphabetical order. HOWEVER, if you use the BT InOrder Traversal algorithm, then you can visit the nodes in alphabetical order, and so use the DRPs in the order needed to print out the DATA in country-name order]
- insertIntoNameIndex uses the BST insert algorithm