OVERVIEW

A3 is a variation of A1 (sort of) which has a name index added, and several other changes. The CountryData is a text file which is direct address on id - this file has already been created manually (in a text editor). [*The format is DIFFERENT from A1 though].* The name index is implemented as an **internal binary search tree** (BST). In order to port it between program runs (i.e., Setup & UserApp and UserApp run today & UserApp run tomorrow), the internal data structure is saved to a backup file and restored from the backup file (handled by NameIndex's constructor & finishUp methods).

[If you use A1 as your starting point, remove those parts of A1 which are not relevant to A3 (including cleaning up comments)].

Changes to A1

(***not including the NameIndex details****, which are covered elsewhere*)

- PrettyPrint utility program just prints out IndexBackup.bin file to Log file, and NOT CountryData file
- Setup program
  - Creates the NameIndex (inside of NameIndex class)
  - Does NOT create CountryData file
- UserApp only allows for QN and LN transCodes – others aren't operational
- RawData class only has to accommodate id and name since Setup is only creating the INDEX and not the actual CountryData file (already created manually)
- CountryData file records have changed:
  - fields have changed (different fields, different field-sizes, fixed-length fields)
  - there's a <CR><LF> after each record
  - empty locations are all spaces rather than "all-0-bits"
- CountryData class really only needs a single transaction-handling PUBLIC service method for this project, `GetThisData(rrn, ui)`, besides the constructor & finishUp methods – though it may have (as needed) private methods, data storage, getters/setters
- Log file has a different format (see below) including node-counts for QN's

Required Code Files

Programs: Setup, UserApp, PrettyPrint
Classes: RawData, CountryData, UI, NameIndex, BSTNode

The 5 Data Files

1. RawDataA3.csv **(provided)**
   - id, code, name, continent, population, lifeExpectancy, <CR><LF>
2. CountryData.txt **(provided)**
   - code, name, continent, population, lifeExpectancy, <CR><LF>
3. TransDataA3.csv **(provided)** (with <CR><LF> after each record)
   - LN (for ListAllByName)
   - QN,France (for QueryByName)
     - NOTE: name will MAY or MAY NOT be space-padded on right to be of the correct length for the index's name, and it MAY or MAY NOT use the capitalization which index has – queryByName method must adapt to the user data on this when comparing
     - NOTE: a .csv file
   - This is a TEST PLAN file which contains embedded comments. Any line beginning with a % is a comment, not actual transaction data.
4. Log.txt - See description below
5. IndexBackup.bin – created/used inside NameIndex class
   - a binary file containing:
     - 1 headerRecord containing:
       rootPtr (a short)
       n (a short)
     - N indexNodeRecords: each record contains:
       leftChPtr (a short)
       keyValue (that is, name) (a "string")
       dataRecordPtr (DRP) (that is, id) (a short)
       rightChPtr (a short)
   - NOTE: there are no field-separators (e.g., commas) nor record-terminators (e.g., <CR><LF>'s) in the file
   - It's just a serial file - written out & read in "sequentially"
     [physically sequentially - NOT in key-sequence by any field],
     So NO random access is needed (so fixed-length records not needed)

Log File

Status messages at the appropriate times

```
>> Setup done – 84 countries stored in CountryData file
>> UserApp done – 19 transactions processed
>> PrettyPrint done
>> open RawData file
>> close RawData file
>> open CountryData file
>> close CountryData file
>> open TransData file
>> close TransData file
>> open Log file
>> open IndexBackup file                    ← ← NEW
>> close IndexBackup file                   ← ← NEW
```

Transaction processing (echo transaction request before showing data)
[NOTE: data below is not accurate based on A3 data – it's just to show format & other issues]
[NOTE: the . . . part for LN is filled in, of course]

```
LN
CDE NAME-------------- CONTINENT---- ---POPULATION L.EX
AND Andorra            Europe               78,000 83.5
BWA Botswana           Africa            1,622,000 39.3
CAN Canada             North America    31,147,000 79.4
. . .
IND India              Asia          1,013,662,000 62.5
. . .
ZMB Zambia             Africa            9,169,000 37.2
+ + + + + + + + + END OF DATA - 26 countries + + + + + + + +
QN,Russian Federation
RUS Russian Federation Europe          146,934,000 67.2
  [4 BST nodes visited]
QN,ruSSian FEDERAtion
RUS Russian Federation Europe          146,934,000 67.2
  [4 BST nodes visited]
QN,Russian
** ERROR: no country with name Russian
  [6 BST nodes visited]
QN,United
** ERROR: no country with name United
  [7 BST nodes visited]
QN,United States of America
** ERROR: no country with name United States of America
  [7 BST nodes visited]
```

PrettyPrint

[NOTE: data below is not accurate based on A3 data – it's just to show format & other issues]
[NOTE: the . . . part is filled in, of course]
[NOTE: The SUB is NOT actually stored in the file – it's printed out just to help developer]

```
RootPtr is 0,  N is 26
[SUB]  LCH  NAME--------------  DRP  RCH
[000]  001  China               001  003
[001]  002  India               007  004
[002]  006  United States       024  011
. . .
[026]  -01  Germany             027  -01
```

################################ Name Index ################################

- All storage associated with NameIndex and all handling of NameIndex (including IndexBackup.bin file) are done inside of NameIndex class (which uses BSTNode class objects)
- NameIndex is implemented as a Binary Search Tree (BST). But Setup and UserApp themselves do NOT know ANYTHING about how NameIndex is actually implemented (neither the internal/external decision, nor the data structure)
- BST's and their algorithms will be discussed in class & see readings on course website
- The BST is INTERNAL data structure. That means it MUST BE
  1) built entirely IN MEMORY during Setup's run, and
  2) stored entirely IN MEMORY all during UserApp's run.
- NameIndex is only stored on a file as a way to PORT it from Setup to UserApp program, and from UserApp (run today) to UserApp (run tomorrow) so the data's not LOST when either program stops executing. It is NOT considered as an EXTERNAL index because it is NOT being PROCESSED from the FILE itself.
- This uses **array storage** for node storage. *[More on this in class. This is not the conventional way to store BST's. Nor is this the conventional Array Storage for binary trees as used for heaps. This uses EXPLICIT pointers, not implicit ones. Don't just use what's described in a/the book or on the internet) for "Binary Tree storage using an array" !!!]*
- The internal storage structure uses array storage with explicit "pointers" (i.e., array subscripts).
- Use -1 for "points nowhere". 0 won't work since that's a valid storage location in arrays.
- A BST data structure includes additional fields besides what's needed for node storage (just as a stack needs a topPtr, a linked list needs a headPtr, . . .).
  - A BST needs a **`rootPtr`**.
  - And because YOU'RE doing the space management for this (because we're using array storage for nodes), we also need **N**.
    - Since arrays start at 0, N specifies nextEmpty location in array]
    - N is a counter which needs initializing and incrementing at the appropriate times
  - These 2 fields are stored in memory during program execution, and saved in the headerRecord in the IndexBackup file

- The index is an array of **bstNode objects**, where each node looks like:
  - leftChPtr is the SUBSCRIPT where the left child node is stored
  - keyValue (i.e., the Name as a string)
  - drp (dataRecordPtr)
    (which is just ID since CountryData file is DirectAddress on id)
  - rightChPtr is the SUBSCRIPT where the right child node is stored

############################ IndexBackup File ############################

NameIndex class's

- finishUp method calls a private (local) method to SAVE the internal index to the external backup file (including the headerRec and all the valid bstNode objects)
- constructor calls a private (local) method to RESTORE the external backup file data back into the internal index structure in memory (including headerRec fields)
  - NOTE: If the file doesn't exist (which will be the case when Setup calls the constructor), then:
    - don't RESTORE the index from the file
    - but instead, do whatever initialization of internal storage is needed to prepare to start building the BST

n and rootPtr must also be saved as the HeaderRecord in the file.

MORE ON SAVE AND RESTORE private methods in class

########################## Notes on Comparisons ##########################

insertCountry and queryByName (in NameIndex class) both need to compare targetName (inputName or transName) with name in the BST.

- Ignore case when comparing – so "mExICO" successfully finds "Mexico"
- Ignore trailing spaces when comparing – so "France " from the TransData file matches "France" in the NameIndex and vice versa
- Only treat full-matches as successful, so "United" should NOT match "United States" or "United Kingdom"
- Use C#'s CompareOrdinal method rather than Compare or CompareTo so that special characters follow strict ASCII-order in terms of "less than". Java's compareTo uses ASCII-order by default.
- See further notes in the TransData file (TestPlan comments)

######################## What DATA does user want? ########################

User wants (in the Log file) the ACTUAL DATA from CountryData in response to QN and LN requests. The user doesn't care at all whether there's an index or that it's implemented as a BST.

HOWEVER, you MUST print out # nodes visited for each QN to prove that you really are using a BST search and not a linear search.

For a QN France, for example

- UserApp calls nameIndex.queryByName, sending in France as the targetName
- queryByName does a bstSearch of nameIndex (**using a proper BST search algorithm**)
    - NOTE: it MAY or MAY NOT find a match
- If it finds a match, at name[13] in the index (for example), it needs to use drp[13] to call countryData.getThisCountry(drp[13], ui)
- getThisCountry then uses direct address to read record 13 and display it to the user (in Log file)

For an LN request,

- UserApp calls nameIndex.listAllByName
- which uses inOrderTraversal algorithmto visit each node in the proper order
- each VISIT to a node will require a call to countryData.getThisCountry(drp[i],ui)
    - which will then read and display the data (to the Log file)

So NameIndex class needs to be able to access CountryData class.

Cautions – USE PROPER BST Algorithms

queryByName processing: *The BST is stored in an array, so linear search is possible for QN transactions – but* **DON'T USE LINEAR SEARCH. YOU'LL LOSE LOTS OF POINTS IF YOU DON'T DO A PROPER BST SEARCH.**

listAllByName Processing: **MUST use inOrderTraversal algorithm. YOU'LL LOSE LOTS OF POINTS IF YOU DO A PHYSICAL SORT.**