CS3310 – Kaminski - Fall 2014
Asgn 5 Project Specs

TEST MODE World Data App 2.5
external CodeIndex (as B Tree)
mainData (as Direct Address)

######################## OVERVIEW OF ASGN 5 ########################

A5 is part of a larger project like A3. However, **YOUR TASK** for A5 is mainly to develop the UserApp aspects of the project – with the major focus being on the selectByCode functionality – i.e.,

- search the code index, implemented as an external BTree, (a binary file) in order to
- directly access the main data file (a text file which is direct address on id).

Because this is just TEST MODE, the data in the files is not real, but was made up to facilitate testing of UserApp since Setup program isn't yet completed.

To speed things up A5 uses a simpler overall program structure than A1-A3. There is just ONE PROGRAM and three classes for this testing phase. You MUST have these 4 (each in its own physical file):

1. The program's `main` method is the overall TestDriver which Deletes Log.txt and deals with different test data sets, specifying different fileNameSuffixes for the ? parts
2. UserApp (procedural) CLASS – has userAppMain method (similar to A1-A3's UserAppPROGRAM's `main` in terms of format/functionality) is the controller. It also does the actual handling of both `A5TransData?.txt` and `Log.txt` files (rather than putting those in a separate OOP class as you did for A1-A3)
3. CodeIndex OOP CLASS
   – handles EVERYTHING to do with `CodeIndex?.bin`
4. BTreeNode OOP CLASS – OPTIONAL (or handle that inside CodeIndex class)
5. ActualData OOP CLASS
   – handles EVERYTHING to do with `FakeActualData?.txt`

############################ THE DATA FILES ############################

CodeIndex?.bin

- 3-letter words were used instead of valid countryCodes, which suffices for testing.
- `CodeIndex?.txt` files were was created with NotePad (ASCII text files with spaces as field-separators and <CR><LF>'s). But the file is NOT in the proper format which Setup would actually (someday) create. So these files are **NOT USED IN YOUR PROGRAM**.
- A conversion utility creates `CodeIndex?.bin` files with the equivalent data with:
  - NO field separators and NO record separators
  - The original TEXT file records have an array of triples (tp/kv/drp) plus the extra tp. But the BINARY file records' data is rearranged into **3 parallel (ish) arrays** instead, ,i.e., **all M tp's, then all M-1 kv's, then all M-1 drp's**
  - numeric fields (m, rootPtr, n, tp's, drp's) are all short's using _____ Endian
  - kv fields (the code's) are fixed-length character arrays (3 byte ASCII chars) and NOT strings (with preceding byteCounts or lengths).
  - RRN's for start with 1 not 0
  - HeaderRec in (which doesn't count as being at RRN 1) contains:
    M RootPtr N (all short's with no field-separators nor <CR><LF>)

FakeActualData?.txt

- Files were created with NotePad - so they're ASCII text files (with spaces as field-separators and <CR><LF>'s). Setup will (someday) create a binary file – but we'll just use the TEXT files for A5.
- Setup will (someday) make this a hash file on name. But since the focus here is on the BTree index, the file is just direct address on id – so the index's DRPs point to the correct records.
- Real country data wasn't used – it was easier to make up fake data, which suffices for testing.
- To simplify, only 3 fields were used:
  id – 2 char's (followed by a space)
  code – 3 char's (followed by a space)
  restOfData – 16 char' s (which may include spaces) followed by a <CR><LF>
- Records are fixed-length (as they need to be for Direct Address).
- RRN's start with 1 not 0
- NO HeaderRec in these files

A5TransData?.txt

- All records in the form: `SC USA` (followed by a <CR><LF>)

############################ 3 TEST DATA SETS ############################

There are 3 data sets with the following fileNameSuffixes: 1, 2, 3

As the TestDriver, `main` contains a FOR loop from 1 to 3,
calling userAppMain method, sending it i for the fileNameSuffix.
During each run of userAppMain, the same fileNameSuffix is used for:
CodeIndex?.bin and FakeActualData?.txt and A5TransData?.txt
userAppMain provides this fileNameSuffix to the 2 constructors in:
CodeIndex class and ActualData class

############################## USING M ##############################

The 3 different CodeIndex files facilitate testing the program with various M values for the different M-way Btree indexes.

The 3 CodeIndex files contain M values of 5, 8 and 9 respectively,
but the program MUST NEVER USE OF THESE ACTUAL HARDCODED
Things must be based on some FUNCTION OF M, e.g., for loop from 0 to M-1 or 0 to M-2.

This forces the program to be robust enough to handle various "any" M values
(up to MAX_M = 73),

QUESTIONS: What should M be using a 512-byte block for CodeIndex:

1. With the implementation specified in this asgn for KV, TP, DRP?
2. If countryCode was stored as a string (with 1-byte preceding length field) instead of a charArray (using 8-bit ASCII codes)?

3. If countryCode used 16-bit Unicode rather than ASCII codes for the charArray?
4. If int's were used for TP & DRP, and countryCode was a charArray using ASCII codes?

################################ Log file ################################

```
====================================
PROCESSING A5TransData1

SC IMP -->> 10 IMP ish        49132    [nodes read:  1]
SC CMU -->> ERROR - code not in index  [nodes read:  3]
. . .
====================================
```

Similarly for A5TransData2 and A5TransData3.

NOTES:

- No status message needed.
- `main` (the TestDriver) DELETEs the file at the start, before ever calling userAppMain
- userAppMain opens the Log.txt file in APPEND (and closes it at the end of userAppMain) – so all 3 test set results appear in the SINGLE Log file

########################## CodeIndex CLASS ##########################

This class contains ALL handling of the external code index BINARY file.

main (the TestDriver) does NOT deal with CodeIndex at all. (It does NOT open the file).
userAppMain does NOT open the file – it's done in the CodeIndex's constructor.
CodeIndex's finishUp method closes the file.

The project MUST use the BINARY versions of the files, NOT the TEXT versions.

userAppMain supplies fileNameSuffix to the constructor,
so it knows which of the 3 CodeIndex files to open

Header record data is read into memory ONCE (for a particular CodeIndex file)
in the constructor , just after opening the file

Do NOT read in the entire FILE into MEMORY. This is an EXTERNAL storage structure,
not an internal data structure. **BIG LOSS OF POINTS IF YOU DO.**

This class (OR BTreeNode class) contains storage for a SINGLE node (or perhaps for a "big node" – see below). There is NEVER more than ONE node in memory at once – so re-use the same storage space (or same object) each time you read in a node into memory. (If you're using a separate node class, don't keep declaring new objects for every node you read – just keep re-using the same storage space).

To simplify searching by reducing the number of loop-stopping conditions to 2 (rather than 3), define the KV array to be of size M rather than M-1 (even though that's NOT what's in the FILE record/node itself), and initialize that extra KV as]]] (ONCE at the start, before ever reading in any nodes – and since nothing ever over-writes that, you'll never have to re-initialize that extra KV).

selectByCode is a public method which calls readOneNode and searchOneNode as needed.
YOU MUST USE THESE 3 METHODS (WITH THESE NAMES)
– OR BIG POINT LOSS

The actual B Tree handling methods are private, including:
readOneNode
searchOneNode
readOneNode's body could contain a single physical read of the whole block
OR it might contain 3 for loops, each containing a physical read of a field
OR. . .
BUT THIS METHOD MUST READ IN AN <u>ENTIRE NODE</u> INTO MEMORY
AND NOT JUST PART OF A NODE
searchOneNode must contain a loop rather than if/else's
since it has to be able handle any value for M

NOTE: TRUTH IN ADVERTISING !!!
These methods MUST do what their names say and NOTHING MORE

NOTES:

- searching must allow for a successful search or an unsuccessful search
- only search as far into the node as needed
- use 1-pass node-searching - do NOT do 2-pass searching
i.e., don't search a node for a match,
then if there's no match, go back and search the node for <

WARNING: If you linear search the CodeIndex file rather than a root-to-hit-target path or root-to-leaf path, you will get 0 points for the asgn ! ! !

WARNING: If you linear search the ActualData file rather than using CodeIndex to locate the right KV, to get its corresponding DRP, to be able to do DirectAddress using the DRP, you will get 0 points for the asgn ! ! !