Asgn 2 – Fall 2015
CS3310 – Kaminski

World Data Storage/Lookup
Direct Address (Binary) file

********** **OVERVIEW** **********

A2 is a modification of A1. The major changes involve the DataStorage implementation:

- DataStorage is now EXTERNAL rather than INTERNAL storage.
- As such, it's a PERMANENT DataStorage FILE rather than a temporary Backup file (actually created as a file rather than created in memory then dumped/loaded to/from the Backup file) .
- It's a BINARY file (rather than BST Backup file being a csv text file).
- It's structured as DIRECT ADDRESS (DA) rather than as a BST.
- As a DA file, it needs fixed-length records, so uses fixed-length fields.
- The PrimaryKey (PK) is ID rather than CODE.
- TransCodes will still be A/S/I/D, but . . .
    - arguments for S and D will be id rather than code
    - A means ShowAll in ID ORDER rather than code-order
    - I transactions can be valid OR INVALID (i.e., duplicate ID)
- DYNAMIC DELETE (actual removal of data) rather than STATIC DELETE (tombstones)
- Slight changes in what's written to Log file, including:
    - No #records visited since DA is O(1) & has fixed #I/O's is always fixed
    - Different error messages for invalid transaction requests.
    - 2 additional status messages (DataStorage file) & 2 dropped (Backup file)
- Different actual input data files than A1: RawData2.csv, TransData2.txt
- Because of duplicate-id problem, potential for new error messages in Log file (from dataStorage.insert method)

Unless otherwise specified (here in these specs OR in class), assume things stay the same for A2 as was specified for A1.

Ideally not much outside of DataStorage needs to be changed.

- We'll need a new PrettyPrint program though. (Any volunteers?)
- If you have errors in A1, fix them so you don't lose points AGAIN for A2.
- If your A1 doesn't work (or doesn't work right), I will see if someone in class can send me a good, working A1 which I can share with you to use as an A2-starter program. (I provide no warranty!).

********** **TransData2.txt FILE** **********

Sample records

```
I 51,AZE,Azerbaijan,Asia,86600,7734000,62.9
A
S 22
D 4
```

********** **Backup.csv FILE** **********

NOT USED IN A2 !

********** **RawData2.csv FILE** **********

NOTE: .csv file → variable-length FIELDS → variable-length RECORDS

NOTE: a TEXT file

record description (corrected from A1 specs) – SAME format as A1's RawData file

- id - 1- 3 digits [uniquely identifies a country]
- code - 3 capital letters [uniquely identifies a country]
- name - all chars (may contain spaces or special characters) [uniquely identifies a country]
- continent - one of: Africa, Antarctica, Asia, Europe, North America, Oceania, South America
- ~~region – NOT USED IN THIS PROJECT~~
- area (physical size of the country) - a positive integer
- ~~yearOfIndep – NOT USED IN THIS PROJECT~~
- population - a positive integer or 0 [which could be a very large integer]
- lifeExpectancy - a positive float with 1 decimal place
- ~~Rest of fields – NOT USED IN THIS PROJECT~~
- <CR><LF>

********** **Log.txt FILE** **********

^^^^^^^^^^^^^^^^^^^^^ Status messages ^^^^^^^^^^^^^^^^^^^^^

Same as A1 EXCEPT No Backup file opened/closed messages (since there's no Backup file) & ADD:

```
-->> OPENED DataStorage file
-->> CLOSED DataStorage file
```

^^^^^^^^^^^^^^^^^^^^^^ Error messages ^^^^^^^^^^^^^^^^^^^^^^

From data.storage.insert when called by Setup – IF it encounters a duplicate id (record already in the file for that location), it prints this error message:

```
ERROR:  DUPLICATE ID 17 when inserting Latvia, Poland has id 17
```

^^^^^^^^^^^^^^^^^ Transaction requests/results ^^^^^^^^^^^^^^^^^

NOTE: Data below not necessarily correct for A2's RawData2 file. I'm just demonstrating FORMAT.

```
S 4 >>
CDE ID- NAME------------- CONTINENT---- ------AREA ---POPULATION LIFE
ZWE 004 Zimbabwe         Africa          390,757    11,669,000 37.8

I AFG,2,Afganistan,Asia,652090,22720000,45.9
    >> OK, Afganistan inserted

S 2 >>
CDE ID- NAME------------- CONTINENT---- ------AREA ---POPULATION LIFE
AFG 002 Afganistan       Asia            652,090    22,720,000 45.9

D 27 >> OK, Canada deleted
```

```
S 27 >> invalid country id

S 999999999 >> invalid country id

S 0 >> invalid country id

D 888888888 >> invalid country id

D 0 >> invalid country id

I 4,ARG,Argentina,South America,2780400,37032000,75.1
    >> invalid (duplicate) country id

I 27,UMI,United States Minor Outlying Islands,Oceania,16,0,0.0
    >> OK, United States Mino inserted
```

NOTE: Yes, locations where records were deleted are RE-USED for inserts
NOTE: Long strings (like name) are truncated on right to correct size in DataStorage, BEFORE they're written to the file.

```
A
CDE ID- NAME-------------- CONTINENT---- ------AREA ---POPULATION LIFE
AFG 002 Afganistan         Asia             652,090    22,720,000 45.9
ZWE 004 Zimbabwe           Africa           390,757    11,669,000 37.8
. . .
UMI 027 United States Mino Oceania              16             0 00.0
. . .
MAR 052 Morocco            Africa           446,550    28,351,000 69.1
============================
```

NOTE: ShowAll (A) transactions do NOT show empty locations, while PrettyPrint does.

```
D 28 >> OK, Iceland deleted
```

^^^^^^^^^^^^^^^^^^^^ PrettyPrint results ^^^^^^^^^^^^^^^^^^^^

NOTE: Data below not necessarily correct for A2's RawData2 file. I'm just demonstrating FORMAT.
NOTE: LOC numbers are NOT actual data in DataStorage file. They are generated by PrettyPrint .
NOTE: PrettyPrint displays empty locations, while ShowAll (A) transactions do NOT.

```
N is 31, MaxID is 52

LOC> CDE ID- NAME-------------- CONTINENT---- ------AREA ---POPULATION LIFE
001> EMPTY
002> AFG 002 Afganistan         Asia             652,090    22,720,000 45.9
003> EMPTY
004> ZWE 004 Zimbabwe           Africa           390,757    11,669,000 37.8
. . .
027> UMI 027 United States Mino Oceania              16             0 00.0
028> EMPTY
. . .
052> MAR 052 Morocco            Africa           446,550    28,351,000 69.1
============================
```

********** **DataStorage.bin FILE** **********

Direct Address (DA) file structure
- with mapping: Relative Record Number (RRN) = id
- (and then byteOffset = . . . as a function of rrn, just before the seek)
- DA file → fixed-length records → fixed-length fields

Binary file (not Text file)
- so numeric data stored as short/int/long/float/double
- no field-separators (as A1 had since it was a csv file)
- no record-separators (as A1 had with it's <CR><LF> after each record)

Types of records:
- 1 header record containing n and maxId, both short's
- Data records, each with the following record format:
  - code - 3 chars
  - id – short
  - name – RIGHT-PAD WITH SPACES OR RIGHT-TRUNCATE to 18 chars
  - continent - RIGHT-PAD WITH SPACES OR RIGHT-TRUNCATE to 13 chars
  - area – int
  - population – long
  - lifeExpectancy – float

NOTES:
- insert method in DataStorage class is responsible for padding/truncating alphanumeric fields – it's NOT RawData's job. Ideally insert is also responsible for converting string/character data into the required numeric data types – it's NOT RawData's job.
- Use Dynamic Delete (actually delete records), NOT Static Delete (tombstones). To delete a record, re-write it to look like what that location contained before any record was written there in the first place
- You should NOT have to initialize the file space (in constructor, when called by Setup) before inserting any records – you should be able to assume that the system did that
- Use DirectAddressSearch to locate a record for Select/Delete transactions – do NOT use linear searching.
- insert method has to do duplicate-id-checking BEFORE inserting any record into the file – whether that's during Setup or UserApp – and if there's already a record in that location, do NOT insert the new record. Use DirectAddressSearch to check for an empty location (i.e., seek, read, IF it's empty… ELSE seek, write).
- Two symptoms of an empty location: 1) All 0 bits case 2) read failed
- insert method has hard-coded checks for invalid id's including 0 and any id > REASONABLE_ID (which is a constant currently set to 60).
- select and delete methods have hard-coded check for 0 id requests. However, they DO use DirectAddressSearch to check for ALL OTHER id's, even id 99999999. Do NOT use maxId or REASONABLE_ID to do a comparison – YOU MUST ACTUALLY VISIT THE FILE.
- ShowAll (A) transactions do NOT use random access. Sequential access is appropriate (except for a single seek to the start of the file to byte 0 at the beginning of showAll).

More on this in class and in various notes to read on the website.