CS3310 – Kaminski - Fall 2012
Asgn 3 Project Specs

World Data App – part of version 1.3
1) *Code Index changed to a Hash Table*
2) Simplified overall project

########## **OVERVIEW** ##########

Conceptually, imagine this asgn is a variation to asgn 2, where the CodeIndex is changed from a BST implementation to a Hash Table implementation

HOWEVER, **in actuality**, this project is NOT a modification of asgn 1 or 2. Start this project from scratch. For asgn 3 there is:

- NO MainData file (so no DataStorage class)
- NO Status messages sent to LogSession file
- NO AutoTesterUtility program (you'll be running the3 individual programs MANUALLY on a SINGLE test data set with no fileNamePrefix's)
- NO duplicate key checking

Code modules:

- 3 programs: **Setup** **UserApp** **ShowIndexFileUtility**
  - Setup & UserApp use Object-Oriented Programming
  - ShowIndexFileUtility is just a short procedural program
- 3 instantiable classes: **RawData** **CodeIndex** **UI**
  - UI handles both TransData.txt & LogSession.txt files
  - CodeIndex handles IndexBackup.bin file
    - or you MIGHT have a separate class for IndexBackup access by CodeIndex (not Setup or UserApp)
  - CodeIndex must be a SINGLE, SHARED class (shared by Setup & UserApp)
  - UI is only used by UserApp, not Setup
  - you MIGHT also have other classes like: HashNode

Data files: **RawData.csv** **IndexBackup.bin**
TransData.txt **LogSession.txt**

########## **TranCodes** ##########

IN (then a space, then a whole csv raw data record)
DC (then a space, then a 3-char code)
QC (then a space, then a 3-char code)

UserApp contains a switch statement using the tranCode (which a UI method provided) to determine which public service method in CodeIndex to call.

########## **CodeIndex Public Service Methods** ##########

InsertACountry – used by both Setup and UserApp

- Inserts appropriate data into CodeIndex (there's NO MainData file)
- Only uses locations from "Available Node Pool A" (FRESH locations) NOT from "Available Node Pool B" (TOMBSTONED locations)
- Do not do DUPLICATE-CODE checking

DeleteByCode – used by UserApp

- Deletes appropriate data from CodeIndex (there's NO MainData file)
- Uses STATIC delete algorithm (TOMBSTONES), NOT DYNAMIC delete (i.e., make locations available for re-use)
- This is NOT a dummy stub
- LINEAR SEARCH → 0 points for this part of the asgn!!!

QueryByCode – used by UserApp

- Finds appropriate data in CodeIndex and just displays the DRP (since there is NO MainData file)
- LINEAR SEARCH → 0 points for this part of the asgn!!!

Any mention of the actual hash table implementation is private to the class – either as part of the code body of a public method, or in the header of a private method in the class

########## **Hashing** ##########

Code Index is implemented as an **INTERNAL** Hash Table using **STATIC** hashing (not dynamic hashing).

- The table is implemented as an array of nodes (or 3 parallel arrays).
- The static array(s) would be of size MAX_ARRAY_SIZE (a named local constant set to 300 for now) – or you can use a dynamic array(s).
- MAX_N_HOME_LOC is a named local constant set to 20 for now
- The home area of the array(s) includes locations: [0] through [MAX_N_HOME_LOC – 1].
- The collision area of the array(s) includes locations: [MAX_N_HOME_LOC] through [MAX_ARRAY_SIZE – 1] (if static)

The DRP is just id for the RRN (since there is no actual MainData.bin file).

2 Counters: nHomeRec (Number of actual good data entries in Home area)
nCollRec (Number of actual good data entries in Collision area)

HashFunction:

- This MUST BE a physically separate callable (private) method with the name HashFunction in CodeIndex class:
  - input: code & MAX_N_HOME_LOC
  - output: homeAddress
- the ALGORITHM:
  - convert 3 char's in code to their 3 ASCII codes
    (giving 3 numbers, all between 65 to 90, inclusive)
  - multiply those 3 numbers together
    *(giving numbers between 65*65*65=274,625 to 90*90*90=729,000)*
  - use division-remainder algorithm (i.e., divide step #2 result by MAX_N_HOME_LOC and use the remainder)
    (giving a number between 0 & MAX_N_HOME_LOC – 1, inclusive)
  - return homeAddress

Collision resolution:

- the ALGORITHM: **Chaining** with **separate** overflow.

Chains

- A chain is a linked list for a single synonym family.
- Chains are unordered.
- The data in the home area is NOT part of the chain. The chain is made up of just the collisions for that synonym family.
- Since there is only a single headPtr of a chain – it is stored in the link field of the home data node.
- There will be a total of MAX_N_HOME_LOC chains (i.e., one per synonym family), some of which could be empty.
- Use -1 to indication "points nowhere" (i.e., null ptr).
- Inserting in a linked list requires:
  1. Physically store the node in the nextEmpty location in the collision area - i.e., at MAX_N_HOME_LOC + nCollRec
  2. Insert the node on the FRONT of the linked list (since it's the cheapest spot)
  3. Increment nCollRec

Initialization of physical storage?????

- Is this necessary? (Yes, for the HOME area, but not for the COLLISION area)
- How are locations being "given out" in the home area?
  (Based on the calculated homeAddress from the HashFunction)
- How are locations given out in the collision area?
  (Based on the nextEmpty location)
- Will it be necessary to test whether a location is empty or not
  - In the home area? (Yes)
  - In the collision area? (No)
- At the end of Setup, will there be any empty locations in the array
  - in the home area? (Yes, probably)
  - in the collision area? (At the end, since it's theoretically infinite)

########################## IndexBackup.bin file ############################

A BINARY file (no field-separators, no <CR><LF>) containing:

1) the header record with these 3 int fields in this order:
   nHomeRec, nCollRec, MAX_N_HOME_LOC
2) regular records (which may be empty locations) contain these fields in this order:
   code (3 char), drp (int), link (int)

IMPORTANT: Do NOT dump out any empty locations at the end of the collision area. So do NOT use MAX_ARRAY_SIZE to control saving the file. Use the other fields in the header record, as needed.

#################### LogSession File ####################################

Output from UserApp (via UI class)

```
QC FRA >>> 003 [3 nodes visited]
QC WMU >>> INVALID COUNTRY CODE [2 nodes visited]
DC USA >>> OK, USA deleted [2 nodes visited]
DC WMU >>> INVALID COUNTRY CODE [2 nodes visited]
IN 48,CSG, . . .
       >>> OK, CSG inserted [4 nodes visited]
```

Output from ShowIndexFileUtility

To be specified later in class