CS3310 – Kaminski - Spring 2013
Asgn 3 Project Specs

World Data App 1.1 - UTILITY
test various Hash Table options for use in CodeIndex

############################ OVERVIEW ####################################

A3 is NOT a modification of A1/A2. It's just a **utility** project to evaluate various hashing options for **future** use (in A4?) for the CodeIndex. A3 project creates multiple hash tables using 2 different hash functions (HF), 2 different collision resolutions algorithms (CRA), and 2 different hash table sizes (hence 8 hash tables). It runs this experiment on both the RawDataSample file (26 countries) and the RawDataAll file (all 239 countries).

In order to select the best options for the future A4 project, A3 calculates and prints out the average search path for each of the 16 hash table. (It also prints out some of the hash tables themselves, to aid the developer in testing, when using the smaller sample file of 26 countries).

############################ PROGRAM STRUCTURE ############################

This is a **single program, HashTableTester**, (i.e., the project's controller including a main method and other local static methods, as needed) with **a separate CodeIndex class** (in a physically separate file).

The goal in designing CodeIndex class is to make it somewhat close to being reusable "as is" in A4, with minor modifications once A3 demonstrates the best HF, CRA and hashTableSize. That is, in A4, the developer would pretty much just have to remove methods for the UNused HashFunction and UNused CollisionResolutionAlgorithm, and then hard-code MAX_N_HOME_LOC constant's value with the selected (best) integer value - rather than the controller program supplying the options as parameters to the object's class, as is done in A3.

Main controller:

- Specifies the hard-coded lists of options (parallel arrays or lists) as specified in "Test Parameters" section below - i.e., whichHF, whichCRA, whatSize
- Opens Log file (in truncate mode, for a fresh start for the demo)
- Calls RunExperiment twice, specifying
  - "Sample", then "All" for fileNameSuffix(for RawData?.csv)
  - 2 options lists: whichHF, whichCRA
  - the appropriate 3rd option list: whatSizeSam vs. whatSizeAll
- Closes Log file

RunExperiment (a local method in the controller program):

- Logs which data set is being used: Sample or All
- Fills **local countryCodeArray** with countryCodes from appropriate raw data file
  - NOTE: RawData?.csv, which is in a different format than A1 & A2
  - *[An array is used as the input stream for creating the hash table since the data set is being used 8 times – so it saves having to repeatedly read through the input file 8 times].*
- Loops through the 3 options lists, and for each option-triple:
  - Sets up a **codeIndex object**, providing it with the 3 options to use for this run
    (Let CodeIndex's constructor sets its own object's whichHF switch, whichCRA switch and MAX_N_HOME_LOC value)
  - Calls the project controller's **local Setup procedure** which:
    - loops through the raw data code ARRAT to repeatedly
      - call CodeIndex's Insert1Country procedure
  - calls **CodeIndex's FinishUp** specifying the **printHashTable switch** value:
    true for the "Sample" data set, false for the "All" data set
    (which prints averageSearchPath to the Log file AND IF printHashTable switch is on, prints the actual hash table to Log file too)

############################ TEST PARAMETERS ##############################

```
whichHF:     "times", "times","times", "times","plus",  "plus", "plus",  "plus"
whichCRA:    "linEmb","chSep","linEmb","chSep","linEmb","chSep","linEmb","chSep"
whatSizeSam:  30,      30,     31,      31,     30,      30,     31,      31
whatSizeAll:  250,     250,    251,     251,    250,     250,    251,     251
```

NOTES:

- whatSize**Sam** vs. whatsize**All** is used corresponding to which RawData? data set is used
  - CodeIndex class uses this value for its **MAX_N_HOME_LOC** which is set by the constructor
- whichHF indicates which (of 2) **HashFunction** methods is to be used

1. whichCRA indicates which (of 2) **CollisionResolution** methods is to be used

############################ HASH TABLE STORAGE ##############################

2 parallel arrays (of size ARRAY_SIZE = 300 should suffice for both raw data sets):

2. array of countryCodes - used for both collision resolution algorithms
3. array of links (i.e., HeadPtr's in the HomeArea, NextPtr's in the CollisionArea)
   - ONLY used for CHAINING collision resolution algorithm, NOT for LINEAR

HOME AREA: locations [0] through [MAX_N_HOME_LOC – 1]

COLLISION AREA:

- For LINEAR/EMBEDDED: there's no separate collision area since collisions are embedded in HOME AREA
- For CHAINING/SEPARATE: locations [MAX_N_HOME_LOC] through ??? (theoretically, to infinity)

############################ HASH FUNCTION ##################################

There are 2 HashFunction methods (HashFunction & HashFunctionPlus) in CodeIndex.
(INPUT: countryCode & MAX_N_HOME_LOC, OUPUT: homeAddress)

1. TIMES hash function (the default) is described in the initial worksheet used in class
2. PLUS hash function is the same EXCEPT that the 3 ASCII codes are ADDED together rather than multiplied

Insert1Country (in CodeIndex) decides on which one to use based on the local (to the class) switch that the constructor set (based on which option it received when called)

########################## COLLISION RESOLUTION ALGORITHM ##################

There are 2 CollisionResolution methods in CodeIndex class. These were described in class, and 2 separate worksheets were used to demonstrate each.

Insert1Country (in CodeIndex) decides on which one to use based on the local (to the class) switch that the constructor set (based on which option it received when called)

1. "linEmb" means "**Linear / Embedded**" algorithm
   - **LINEAR** with wrap-around (based on MAX_N_HOME_LOC) – i.e., if a location is full, try next location (with wrap-around) - REPEAT until empty location found
   - **EMBEDDED**: Collisions are stored in the SAME storage area as the HOME area – i.e., locations 0 through MAX_N_HOME_LOC - 1
2. "chSep" means "**Chaining / Separate**" algorithm
   - **CHAINING** A chain is a linked list. All collisions for a particular synonym family form a chain of just those nodes (though the synonym member in the HOME location is NOT part of the chain).
     - Links (including the HeadPtr) are subscript values which either point to a location in the Collision Area OR are -1 (for "end-of-chain")
     - When adding a node to a chain, ALWAYS ADD IT TO THE **END OF THE CHAIN**.
       NOTE: This is NOT THE MOST EFFICIENT during Setup time, but
       1) it allows us to add to the proper counter (see below) which is used in calculating averageSearchPath and
       2) UserApp (in A4) won't care what order the codes in the chain are in any way since there's no "fairness principle" involved (e.g., FIFO, FILO)
     - Inserting into a chain (where the program's doing its own space management, as is the case here) involves 3 steps:
       1. Physically store the new data in nextEmptyLocation in Collision area
       2. Attach it to the data structure (according to "the rules")
       3. Increment space management counter - i.e., nColl
     - NOTE: There'll be MAX_N_HOME_LOC number of chains, some of which might be empty, and hence a -1 in its HeadPtr.
   - **SEPARATE**: COLLISION AREA is physically separate from HOME AREA
     - Collision Area is: locations [MAX_N_HOME_LOC] through [???] (so it's theoretically infinite)
     -

########################## COUNTERS ##

nHome – number of countryCodes in their HOME location
nColl – number of countryCodes that are COLLISIONS

- Needed for space management in Chaining/Separate CRA - nextEmptyLocation in collision area is always: nColl + MAX_N_HOME_LOC

Counter array to store: number of countryCode which are actually stored in:

- Home + 0 → use counter[0] for a code stored in its home location
- Home + 1 → use counter[1]
  - For Linear/Embedded (with wrap-around), this means countryCode is stored **physically next door** to its home spot (or for wrap-around cases where home is MAX_N_HOME_LOC – 1, but it ends up stored in [0])
  - For Chaining/Separate, this would be for codes stored as the 1st node in the synonym family's chain
- Home + 2 → use counter [2] for this
- . . .
- Home + ? → use counter [?] for. . .

For Chaining/Separate, you'll need a "travelDistance" (down the chain) to know which counter to add 1 to.

########################## Log FILE ##

The following is NOT accurate, data-wise. It is just used to demonstrate what needs to be printed out and the required formatting.

```
++++++++++++++++++++ data set: SAMPLE ++++++++++++++++++++

>> TIMES HashFunction, LINEAR / EMBEDDED CollisionRes
>>    MaxNHomeLoc:  30
>> NHome:  17,  NColl:  9,  average search path:  2.3
[00] MEX
[01]
[02] USA
. . .                          [this part filled in, of course]
[29] ATA
------------------------------------------------------------
>> TIMES HashFunction, CHAINING / SEPARATE CollisionRes
>>    MaxNHomeLoc:  30
>> NHome:  21,  NColl:  5,  average search path:  1.4
[00] CHN -1
[01] IND 34
[02]
[03] GBR 32
. . .                          [this part filled in, of course]
[29] ATA 31
[30] DEU -1
[31] USA -1
. . .                          [this part filled in, of course]
[34] FRA 30
------------------------------------------------------------
```

########################## PUBLIC / PRIVATE ################################

All local members of the controller program class, except main, are private
(e.g., RunExperiment, Setup, …, countryCodeArray)
CodeIndex class:
Public: constructor(s), Insert1Country, FinishUp
Private: HashFunction(s), CollisionResolution(s), Hash table array(s), counters (array & nHome & nColl), Switches for whichHF, whichCRA, MAX_N_HOME_LOC