Asgn 3 – Fall 2015
CS3310 – Kaminski

World Data Storage – the index?
Comparison of HashTables for CodeIndex

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

This is a utility program used to compare different internal hash tables structures for potential use as the CodeIndex for a direct address (on id) DataStorage.bin file (like A2's). In order to do this, pairings of various hash functions (HF) and various collision resolution algorithms (CRA) are used to build a hash table and compute its average search path (for successful searches, not unsuccessful). This is done for a small RawData test file (26 countries) and for the full RawData file (all 239 countries).

This is a SINGLE program. It is not a modification of A1 or A2 (although you may use parts of your A1/A2 code). This program only CREATES the data structure and POPULATES it with data, but it does NOT USE the data (e.g., UserApp). There is NO DataStorage.bin file created, just the CodeIndex (multiple times). Nor is there any Backup file to save the internal structure to a file. There are only 3 files involved: RawDataTest.csv, RawDataAll.csv & Log.txt.

********** **2 RawData?.csv FILES** **********

- RawDataTest.csv file & RawDataAll.csv file
- files have a slightly different formats than A1 & A2 RawData files
- These were originally SQL data files used to populate a database, so each record starts with `INSERT INTO `Country` VALUES (` which is not relevant for A3
- Char fields have single quotes around them and should be removed before use.
- Only 2 fields needed from a record for A3:
  code & id, the first 2 actual data fields in the record

********** **The 4 Hash Functions** **********

- Each hashFunction is a single overloaded method within a separate class containing ALL the hashFunctions
- Private methods may be included in this class for common code
- Each is named hashFunction
- Input: whichHashFunction (i.e., the HF number), code, maxNHomeLoc
- Output: homeAddress

The hashFunctions:

1) Convert 3 char's to their ASCII codes, multiply 3 codes,
   use division-remainder by maxNHomeLoc
2) Convert 3 char's to their ASCII codes, add 3 codes,
   use division-remainder by maxNHomeLoc
3) Convert 3 char's to their ASCII codes, concatenate 3 codes ($1^{st} + 2^{nd} + 3^{rd}$),
   use division-remainder by maxNHomeLoc
4) Convert 3 char's to their ASCII codes, concatenate 3 codes ($3^{rd} + 2^{nd} + 1^{st}$),
   use division-remainder by maxNHomeLoc

********** **The 2 Collision Resolution Algorithms** **********

1) Linear, Embedded
   *Use Linear, WITH **WRAP-AROUND** – i.e., the next location to try after [maxNHomeLoc – 1] is [0] NOT [maxNHomeLoc]*
2) Chaining, Separate
   *Add to **FRONT** of chain, NOT the back – and do NOT keep it as an ordered list*

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

- A single file containing all the results from the various hash tables and their stats.
- When RawData**Test** file is used, the hash table IS printed here, along with its stats since the printout isn't that long. However, when RawData**All** file is used, just the stats are printed (with appropriate labelling), and the actual hash table is NOT printed.
- Here's the format for a couple sample hash tables printout - there will be lots more than what's shown – see "Controller" section below
- The data shown here are not necessarily accurate, based on data in the RawData? Files – I'm just demonstrating which results need to be shown, and their printout format
  - *NOTE: The data for the Test/1/2/20 case "should" be correct – but not for subsequent test cases*
- The . . . would be filled in, of course, for the HASH TABLE

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CASE:  1
RAW DATA FILE:  Test
HASH FUNCTION:  1  (with maxNHomeLoc:  20)
COL RESOL ALG:  2  (Chaining, Separate)
N_HOME: 11,  N_COLL:  15  --> 26
AVE SEARCH PATH (for successful):  (11+52)/(11+15) --> 2.4
HASH TABLE:
LOC  CODE DRP LINK
000> JPN  013 034
001>          -01
002>          -01
003>          -01
004> MEX  012 029
. . .
034> ATA  059 033
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
. . .
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CASE:  6
RAW DATA FILE:  Test
HASH FUNCTION:  1  (with maxNHomeLoc:  30)
COL RESOL ALG:  1  (Linear, Embedded)
N_HOME: 8,  N_COLL:  15  --> 26
AVE SEARCH PATH (for successful):  (11+52)/(11+15) --> 2.4
HASH TABLE:
LOC  CODE DRP
000> JPN  013
001>
002>
003>
004> MEX  012
. . .
029> ATA  059
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
. . .
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CASE: 8
RAW DATA FILE:  All
HASH FUNCTION:  1  (with maxNHomeLoc:  260)
COL RESOL ALG:  2  (Chaining, Separate)
N_HOME: 179,  N_COLL:  60  --> 239
AVE SEARCH PATH (for successful):  (179+168)/(179+60) --> 1.5
HASH TABLE:  big table --> saved paper by not printing it
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
. . .
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

*************************** The CONTROLLER (`main`) **************************

- The program's main is the big controller – although if the method gets too long, modularize it using private local methods.
- This code repeatedly calls hashTable.createHashTable
- There are 11 test cases to be done (in a single run of the program) IN THIS ORDER !!!!

case	RawData file	Hash Function	Collision Resolution Algor	maxNHomeLoc
1	Test	1	2 (Chaining,Separate)	20
2	Test	2	2	20
3	Test	3	2	20
4	Test	4	2	20
5	Test	1	2	30
6	Test	1	1 (Linear,Embedded)	30
7	All	1	2 (Chaining,Separate)	240
8	All	1	2	260
9	All	1	2	350
10	All	1	1 (Linear,Embedded)	240
11	All	1	1	260

**************************** HashTable OOP class ****************************

Memory for the Hash Table

- Use a static array of size (a named constant) MAX_SIZE = 500 (or a dynamic array)
  - So that's 0 through MAX_SIZE – 1 (i.e., locations 0-299)
- Class only needs a SINGLE array of nodes (or 3 parallel arrays) – you do NOT need more than ONE HASH TABLE at once since the program
  - builds a hash table (calculating the necessary stats) from RawData file
  - prints the descriptive header info
  - prints the stats
  - perhaps prints the table (just for small hash tables which use "Test" data file)

  and then this particular HashTable is NO LONGER NEEDED !!!

  So the SPACE IS REUSED for the next HashTable

- A hash table node contains: code, drp, link
  - NOTE: link is used for "chaining, separate" collision resolution algorithm but NOT for "linear, embedded" collision resolution algorithm

createHashTable method

- This is the MINI CONTROLLER which creates a SINGLE hash table using data from the designated RawData file.
- This method will be called MULTIPLE times (by the program main), with different input parameters each time.

public method createHashTable(whichData, // Test or All for RawData?.csv
whichHF, // 1-4 for which Hash Function
whichCRA, // 1-2 for Lin,Emb or Ch,Sep
maxNHomeLoc) // see note below

Printing the HashTable

1. Print the HashTable or not?
   If whichData says "Test", automatically print it – otherwise do NOT print it

2. Print the LINK column or not?
   If whichCRA says 2 (Chaining,Separate), then print it – if it's a 1 then do NOT print it

3. How many nodes to print? (i.e., what controls the FOR loop?)
   Don't use MAX_SIZE constant (since it's mainly empty)
   Use maxNHomeLoc (for CRA 1) or maxNHomeLoc+nColl (for CRA 2)
   [But note that you're starting at [0], so stop at . . .]

maxNHomeLoc?

- Since arrays start at 0, not 1, maxNHomeLoc means: [0] → [maxNHomeLoc – 1]
- For Chaining, SEPARATE CollResAlgor, maxNHomeLoc means:
  - The # locations allocated for the HOME area, [0] → [maxNHomeLoc – 1]
  - The COLLISION area is SEPARATE, using locations:
    [maxNHomeLo] → [maxNHomeLoc + nColl – 1]
  - So if maxNHomeLoc was 10, & nColl was 5, then the HOME area is 0 to 9,
    the COLLISION area is 10 to 14
- For Linear, EMBEDDED CollResAlgor, maxNHomeLoc means
  - The # locations allocated for the HOME area AND the COLLISION area together (since collisions are EMBEDDED within the home area), so
    [0] → [maxNHomeLoc – 1] for BOTH "intermingled"
  - So if maxNHomeLoc was 30, then the HOME area is 0 to 29,
    the COLLISION area is 0 to 29

Efficiency note:

- The first 6 tests use RawDataTest, the next 5 use RawDataAll
- So you could read all of RawDataTest records into a temporary area in memory, then build your hashTable from there rather than the file
- Similarly for RawDataAll file