CS 302 – Assignment #07

Purpose: Learn concepts regarding hash tables.
Due: Tuesday (3/14) → Must be submitted on-line before class.
Points: 125 Part A → 75 pts, Part B → 50 pts

Assignment:

Part A:

In order to qualify for the lower bulk mail rates, the addresses must be certified as being correct (≥ 97% valid). This time we will verify that the ZIP code, city, and state are valid and matching. Specifically, we will create a program to validate each address (city, state, and ZIP) in a file of addresses. To store the master set of valid city, state, and zip codes, you will need to implement a ***hash table***[1] data structure.

A main will be provided that performs a series of tests. Refer to the UML descriptions for implementation details.

Part B:

Create and submit a brief write-up, not to exceed ~750 words, including the following:

- Name, Assignment, Section.
- Summary of the ***hash table*** data structure.
- For the hashing, explain what occurs when the load factor is reached (which may occur multiple times).
- Replace the primary hash function with;
  - The 1st alternate hash function (sum of ASCII values) and time the execution (unix time command).
  - Then, the 2nd alternate hash function (ZIP code converted to integer) and time that execution.

  This testing should use the small addresses file (addrsSM.txt). Report the results for each test (run time and collision rate) and compare to the original hash function. Note which hash function is better and explain why.
- Explain the difference between the hash function used in the assignment and a cryptographic hash[2].
- Provide the Big-Oh for the various hash operations (insert, remove, hash, and rehash).
- Suggest some things that can be done to improve the overall performance for the hash table.

Submission:

- Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 23:55.
- Submit a PDF copy of the write-up.

All necessary files must be included in the ZIP file. The grader will download, uncompress, and type **make** (so you must have a valid, working *makefile*).

---

1 For more information, refer to: http://en.wikipedia.org/wiki/Hash_table
2 For more information, refer to: https://en.wikipedia.org/wiki/Cryptographic_hash_function

Class Descriptions

- Hash Table Class
  The hash table class will implement functions specified below.

hashTable
-hashSize: unsigned int
-reSizeCount: unsigned int
-collisionCount: unsigned int
-entries: unsigned int
-**addrHash: string
-loadFactor: static constexpr double = 0.65
-initialHashSize: static constexpr unsigned int = 10000
+hashTable()
+~hashTable()
+insert(const string, const string, const string): bool
+lookup(const string string &, string &) const: bool
+remove(const string): bool
+printHash() const: void
+showHashStats() const: void
-hash(string) const: unsigned int
-rehash(): void

Function Descriptions

- The *hashTable()* constructor should initialize the hash table to an empty state. The hashSize should be initialized and the other variables initialized to 0, and the initial hash table should be dynamically created.
- The *~hashTable()* destructor should deallocate the dynamically allocated memory.
- The *insert()* function should insert the passed key into the hash table and, if successful, return true. If the key is already in the has table, the function should return false. If the hash table entries exceeds the load factor (count/tableSize), the table must be rehashed via the private *rehash()* function before the insertion is performed. The *hash()* function must be used determine the table location. If a collision occurs, the collision count should be incremented and linear probing should be used. The collision count and entries count should be updated as appropriate.
- The *lookup()* function should search the hash table for the passed ZIP code string and, if found, return the city and state (via reference) and return true. The *hash()* function must be used determine the table location. If the item is not found at that location, linear probing should be used. This function should ***not*** increment the collision count. If the passed ZIP code string is not found, the function should return false.
- The *showHashStats()* function is a utility function to print the current hash size, current hash table resize count, and the collision count. Refer to the example output for formatting examples.
- The *printHash()* function should print all non-empty entries in the hash table. *Note*, this function is used for testing.
- The *remove()* function should search the hash table for the passed string and, if found, remove it (by marking with a '*'). The *hash()* function must be used determine the table location. If a collision occurs linear probing should be used. In order to ensure that later searches are not hindered, the delete should use a tombstone (a ‘*’) when deleting an

entry (instead of just blanking the entry). Refer to the class text for additional information regarding tombstones.

- The *rehash()* function should create a new hash table twice the size of the current hash table, extract all entries from the current hash table, insert them into the new table (via the *insert()* function), and delete the old hash table.
- The *hash()* function should return a hash from the passed string. The function should implement a hash based on Horner's Rule as follows:

```
// Hash based on Horner's rule
int   hash = 0;
for (int i=wrd.length()-1; i >= 0; i--)
     hash = (wrd[i] + 128*hash) % hashSize;
return     hash;
```

Note, in addition, two alternative hash function functions should be implemented as follows;

- Create a hash by summing the ASCII values of each character in the string. The final returned hash must be mod'ed with the current hash table size.
- Create a hash that converts the ZIP code string to a integer. The final returned hash must be mod'ed with the current hash table size.

These alternative hash functions will only be used for testing as part of the final write-up.

- Zip Codes Class

The zip codes class should inherit from the ***hashTable*** class and will implement functions specified below.

ZipCodes public hashTable
-goodCount: int
-badCount: int
-masterZipCodes: hashTable
+zipCodes()
+~zipCodes()
+readMasterZipCodes(const string): bool
+checkZips(const string): bool
+showStats() const: void

Function Descriptions

- The *zipCodes()* constructor should initialize the class variables.
- The *~zipCodes()* destructor should reset the class variables.
- The *readMasterZipCodes()* function should read the passed master zip codes file and *insert()* the words in the hash table. If the master zip codes file read operations are successful, the function should return true and false otherwise.
- The *checkZips()* function should read the passed zip codes data file, parse out the zip code and see if the zip code is in the hash table. The city and state for both the data and the results from the lookup() should be converted to upper case ro ensure a valid comparison. If all three match (ZIP, city, and state), the good counter should be incremented and if not, the bad counter should be incremented. If the master zip codes file read operations are successful, the function should return true and false otherwise.

- The *showStats()* function should display the statistics in the format shown in the examples, including the calculation of the percentage good and percentage bad. If the percentage good is ≥ 97%, the message "Validation – PASSED" should be displayed otherwise the message "Validation – FAILED" should be displayed. Refer to the example output for formatting.

Refer to the example executions for output formatting. Make sure your program includes the appropriate documentation. See Program Evaluation Criteria for CS 302 for additional information. ***Note, points will be deducted for especially poor style or inefficient coding.***

Test Script

A test script which will be used for scoring the final submission is provided for reference.

Example Execution:

Below is an example output for the hash test program and program execution for the checkZips main.

```
ed-vm% ./hashTest
----------------------------------------
CS 302 - Assignment #7
Hash Table Test Program.


Hash Dump (for testing)
----------------------------------------
a : a : a :
balloon : balloon : balloon :
bye : bye : bye :
their : their : their :
ball : ball : ball :
answer : answer : answer :
there : there : there :
by : by : by :
any : any : any :
the : the : the :


Test Hash Zero

Hash Stats
   Current Entries Count: 10
   Current Hash Size: 10000
   Hash Resize Operations: 0
   Hash Collisions: 0

----------------------------------------

Hash Dump (for testing)
----------------------------------------


Test Hash Zero

Hash Stats
   Current Entries Count: 0
   Current Hash Size: 10000
   Hash Resize Operations: 0
   Hash Collisions: 0

************************************************************
Test Hash One
```

```
Hash Stats
   Current Entries Count: 0
   Current Hash Size: 10000
   Hash Resize Operations: 0
   Hash Collisions: 1

****************************************************************
Test Hash Two A

Hash Stats
   Current Entries Count: 456976
   Current Hash Size: 1280000
   Hash Resize Operations: 7
   Hash Collisions: 35823633

Test Hash Two

Hash Stats
   Current Entries Count: 0
   Current Hash Size: 1280000
   Hash Resize Operations: 7
   Hash Collisions: 35823633

----------------------------------------
Game Over, thank you for playing.
ed-vm%
ed-vm%
ed-vm%
ed-vm% ./checkZips -z free-zipcode-database-Primary.csv -d addrsSM.csv
****************************************************************
CS 302 - Assignment #7
ZIP Code Checking Program

Hash Stats
   Current Entries Count: 42522
   Current Hash Size: 80000
   Hash Resize Operations: 3
   Hash Collisions: 1882117
-----------------------------------
Zip Code Validation Statistics:

Master Zip Codes Tree Statistics:

Zip Codes Data File Statistics:
   Total Zip Codes: 500
     Good: 484
     Bad: 16
   Percentage Good: 96.8
   Percentage Bad: 3.2
   Validation - FAILED

****************************************************************
Game Over, thank you for playing.
ed-vm%
ed-vm%
```