CS 302 – Assignment #07

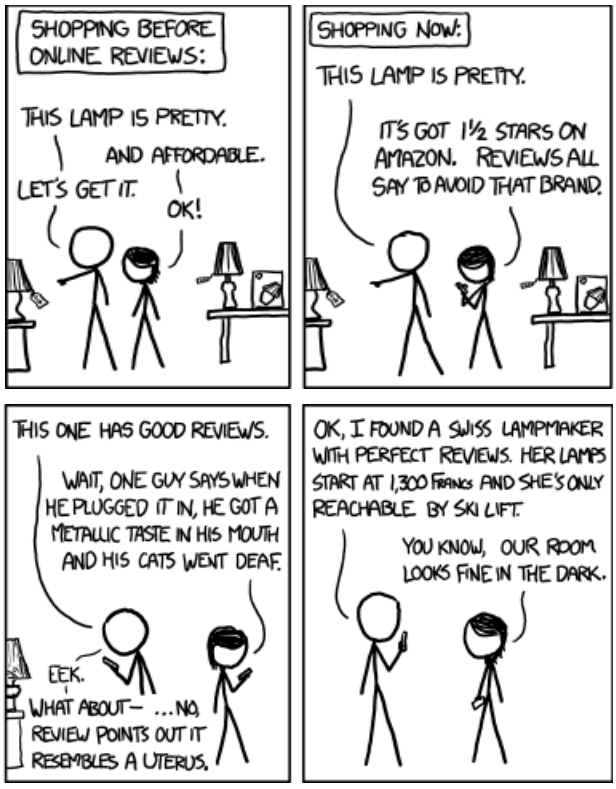
Purpose: Learn concepts regarding hash tables.
Due: Tuesday (10/24) → Must be submitted on-line before class.
Points: 125 Part A → 75 pts, Part B → 50 pts

Assignment:

Part A:

Many web sites collect product reviews. For this project, we will use a ***hash table***[1] data structure to store product review data.

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) and the associated impact.
- Replace the primary hash function with a secondary hash function (which sums the ASCII values). Time the execution (unix time command) using the original hash and then using this secondary hash. This testing should use the medium file (foodsMed.txt). Report the results for each test (run time and collision rate) and compare. 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
-*hashReviews: string
-*hashscores: double
-*hashCounts: unsigned int
-loadFactor: static constexpr double = 0.65
-initialHashSize: static constexpr unsigned int = 1013
-hashSizes[12]: static constexpr unsigned int = {30011, 60013, 120017, 240089, 480043, 960017, 1920013, 3840037, 7680103, 15360161, 30720299, 61440629};
+hashTable()
+~hashTable()
+insert(const string, const double): bool
+search(const string, double &, unsigned int &) const: bool
+remove(const string): bool
+printHash() const: void
+showHashStats() const: void
+findMaxReview(string &, double &, unsigned int &): bool
-insert(const string, const double, const unsigned int): bool
-hash(string) const: unsigned int
-rehash(): void

Note, in order for the hashSizes array to be seen correctly in the implementation file, the following declaration will be required:

```
constexpr unsigned int hashTable::hashSizes[];
```

This must be placed in the top of the implementation file (after the includes, before the functions).

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 arrays should be dynamically created.
- The *~hashTable()* destructor should deallocate the dynamically allocated memory.
- The public *insert()* function should insert the passed ID and score into the hash table. If the hash table entries exceeds the load factor (entries/tableSize), the table must be rehashed via the private *rehash()* function before the insertion is performed. If the key is already in the hash table, the function should increment the count return true. If the key is not already in the hash table, the function should insert it and return true. *Note*, this

function should *not* call the *search()* function. If the product ID string is empty, the function should return false. The *hash()* function must be used determine the table location. If a collision occurs, the collision count should be incremented and quadratic probing must be used. The collision count and entries count should be updated as appropriate.

- The *search()* function should search the hash table for the passed ID string and, if found, return the score and count (via reference) and return true. The *hash()* function must be used determine the table location. If the ID is not found at that location, quadratic probing should be used. This function should ***not*** increment the collision count. If the passed ID 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 public *findMaxReview()* function should find the product with the most reviews. If there is no hash table, the function should return false. Otherwise, the function should find the product with the highest number of reviews and return true along with the product, score, and count true (via reference).
- The *remove()* function should search the hash table for the passed ID string and, if found, remove it (by marking that entry with a '*'). The *hash()* function must be used determine the table location. If a collision occurs quadratic probing must 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 of the next larger size in the array of hash sizes, extract all entries from the current hash table, insert them into the new table, and delete the old hash table.
- The private *insert()* function should insert the passed ID, score, and count into the hash table. This function is only called from the *rehash()* function. The function should insert the entry and return true. If the product ID string is empty, the function should return false. The *hash()* function must be used determine the table location. If a collision occurs, the collision count should be incremented and quadratic probing must be used. The collision count and entries count should be updated as appropriate.
- The *hash()* function should return a hash from the passed string. The function should implement a Fowler-Noll-Vo[3] FNV1a hash function as follows:

```
hash = FNV_offset_basis
for each byte_of_data to be hashed
     hash = hash XOR byte_of_data
     hash = hash × FNV_prime
return hash.
```

The appropriate values for the 32-bit FVN1a implementation are as follows:

```
FNV_prime = 16777619
FNV_offset_basis = 2166136261
```

The final returned hash must be mod'ed with the current hash table size.

  - *Note*, in addition, a secondary hash function should be implemented by summing the ASCII values of each character in the string. Again, the final returned hash must be mod'ed with the current hash table size. This secondary hash function will only be used for testing as part of the final write-up.

3 For more information, refer to: https://en.wikipedia.org/wiki/Fowler-Noll-Vo_hash_function

- Review Data Class
  The review data class should inherit from the ***hashTable*** class and will implement functions specified below.

reviewData public hashTable
-totalReviews: unsigned int
-uniqueProducts: unsigned int
+reviewData()
+~reviewData()
+readMasterReviewData(const string): bool
+getReviews(const string): bool
+showStats() const: void
+showMaxReview() const: void
+printAllReviews() const: void
+printProduct(const string, const double, const unsigned int): void

Function Descriptions

- The *reviewData()* constructor should initialize the class variables.
- The *~reviewData()* destructor should release any dynamically allocated memory and reset the class variables.
- The *readMasterReviewData()* function should read the passed master products file and *insert()* the words in the hash table. If the master products file read operations are successful, the function should return true and false otherwise.
- The *getReviews()* function should read the passed product ID's data file. If the data file read open/read operations are successful, the function should return true and false otherwise. For each product ID in the file, the function should check if a review exists and if so, print it using the *printProduct*() function. If not, it should display a formatted message "Product, <productID> not found." message. Refer to the example output for formatting.
- The *showStats()* function should display the data statistics (in the format shown in the examples) including the current entries count, current hash size, count of hash resize operations, and current count of hash collisions. Refer to the example output for formatting.
- The *showMaxReviews*() function should show the product with the maximum number of reviews via the inherited *findMaxReview*() function.
- The *printAllProducts*() function should print all products in the hash. This is primarily used for debugging. Refer to the example output for formatting.
- The *printProduct*() function should print the passed product information in a formatted manner. The average score should be calculated based on the score total and the review count. This function is used by multiple other functions. 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 reviews main.

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


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

their : 7 : 1 :
bye : 6 : 1 :
by : 5 : 1 :
any : 4 : 1 :
balloon : 9 : 1 :
a : 1 : 1 :
ball : 8 : 1 :
the : 0 : 1 :
answer : 3 : 1 :
there : 2 : 1 :


Test Hash Zero Stats

Hash Stats
   Current Entries Count: 10
   Current Hash Size: 1013
   Hash Resize Operations: 0
   Hash Collisions: 0

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

Hash Dump (for testing -> should be empty)
----------------------------------------



Test Hash Zero Stats (now empty)

Hash Stats
   Current Entries Count: 0
   Current Hash Size: 1013
   Hash Resize Operations: 0
   Hash Collisions: 0

****************************************************************


Test Hash One

Hash Stats
   Current Entries Count: 50
   Current Hash Size: 1013
   Hash Resize Operations: 0
   Hash Collisions: 0

Test Hash One (now empty)

Hash Stats
```

```
   Current Entries Count: 0
   Current Hash Size: 1013
   Hash Resize Operations: 0
   Hash Collisions: 0

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

Hash Stats
   Current Entries Count: 456976
   Current Hash Size: 960017
   Hash Resize Operations: 6
   Hash Collisions: 637944


Test Hash Two (now empty)

Hash Stats
   Current Entries Count: 0
   Current Hash Size: 960017
   Hash Resize Operations: 6
   Hash Collisions: 637944

----------------------------------------
Game Over, thank you for playing.
ed-vm%
ed-vm%
ed-vm% ./reviews -m foodsSm.txt
****************************************************************
CS 302 - Assignment #5
Amazon Review Checking Program

================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> s
-------------------------------------
Review Data Statistics:

Review Hash Stats:
Hash Stats
   Current Entries Count: 55
   Current Hash Size: 1013
   Hash Resize Operations: 0
   Hash Collisions: 2

Review Data Stats:
   Total Reviews: 222
   Unique Products: 0

================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> l
Enter Product: B003EMXLU2

Product: B003EMXLU2
```

```
  Avg Score: 4.50
  Reviews:   4


================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> m

--------------------
Product: B00374ZKQ0
  Avg Score: 2.93
  Reviews:   28

================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> a
B001682QCK : 15.00 : 5 :
B002PXSGA6 : 10.00 : 2 :
B000P56I7Y : 5.00 : 1 :
B001BOAOLY : 27.00 : 6 :
B001EO7N10 : 27.00 : 6 :
B0013Z0PTW : 92.00 : 22 :
B004BY23I8 : 12.00 : 3 :
B000RHXJM2 : 5.00 : 1 :
B000LKVRQA : 42.00 : 10 :
B003XUL27E : 7.00 : 5 :
B003S1WTCU : 2.00 : 1 :
B000NY4SAG : 19.00 : 4 :
B000NY8O9M : 5.00 : 1 :
B00374ZKQ0 : 82.00 : 28 :
B0000D16IP : 6.00 : 2 :
B001LR2CU2 : 5.00 : 1 :
B005OCX5XI : 1.00 : 1 :
B004BRECPW : 6.00 : 2 :
B0039KE8Y2 : 4.00 : 1 :
B001TGY7W6 : 5.00 : 1 :
B002XO3Q52 : 3.00 : 1 :
B004CHDG44 : 5.00 : 1 :
B003JHR4GE : 15.00 : 3 :
B000KOSIP0 : 20.00 : 6 :
B003Q9VWUO : 2.00 : 1 :
B006JSPXZY : 1.00 : 1 :
B004I613EE : 10.00 : 2 :
B005OTVL8C : 5.00 : 1 :
B001IZHZJA : 32.00 : 8 :
B000H28ABW : 64.00 : 14 :
B003VWU7IE : 28.00 : 6 :
B006T7TKZO : 5.00 : 1 :
B0018CLWM4 : 32.00 : 7 :
B0001WYNFA : 5.00 : 1 :
B003EMXLU2 : 18.00 : 4 :
B004CZUOSM : 2.00 : 1 :
B004JLGEII : 21.00 : 5 :
B001FPT1WM : 36.00 : 8 :
B001ELL54Y : 24.00 : 6 :
```

```
B001EQ5O6Y : 71.00 : 18 :
B000H7K114 : 5.00 : 1 :
B004BRECP2 : 3.00 : 1 :
B000KOUOHU : 5.00 : 1 :
B000JT45IA : 5.00 : 1 :
B003IFB148 : 5.00 : 1 :
B002OXLXLG : 5.00 : 1 :
B003O5Q3KE : 12.00 : 4 :
B009GTIHGO : 5.00 : 1 :
B0002ARMS6 : 5.00 : 1 :
B005ZC0RRO : 5.00 : 1 :
B001F2GDJY : 10.00 : 2 :
B004MZ40DW : 6.00 : 2 :
B006Z0Z6WG : 4.00 : 1 :
B003NQMPYM : 5.00 : 1 :
B0015V7GG4 : 24.00 : 5 :
==================================================
Select Option:
   'p' - process input file.
   's' - show statistics
   'a' - show entire tree contents (debug)
   'l' - lookup product
   'm' - find/show product with maximum number of reviews
   'q' - quit
> q

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