CS 302 – Assignment #07

Purpose: Learn concepts regarding hash tables.
Due: Monday (10/22) → Must be submitted on-line before class.
Points: 125 Part A → 75 pts, Part B → 50 pts

Assignment:

Part A:

In computer science, an *associative array*[1] is an abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection. Many languages provide direct support for associative arrays (Python, Perl, Ruby, etc.) as primitives. For example;

```
myArr["Jorgensen"] = 3.14159;
```

is a legal Python statement. There are many useful applications for associative arrays. The data structure typically used to implement an associative array is a Hash Table[2].

Design and implement a C++ templated ***assocArray*** class (which uses a hash table data structure) to provide associative array functionality. The class should include a simple iterator[3].

Create a main, using the ***assocArray*** class, to read a series of movie ratings into an associative array. The main should call a function to read, check, and process command line arguments for the input and output files (**`-i <inputFile> -o <outputFile>`**) in either order. The main will read the movies, store them in an array, and output the full movie list with ratings to a formatted output file. For duplicate movies, the movies should be averaged. The ***assocArray*** class iterator should be used to display all the movies in the associative array. The specific format is "**`Movie:`** ", the movie title, left justified with a maximum width of 50, "**`Score:`** " with the movie score with one decimal place. The main should also show the overall ratings average, top three rated movies, bottom three rated movies and hash statistics to the console. See the sample execution for formatting examples.

Part B:

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

- Name, Assignment, Section.
- Summary of what an associative array is and why it might be useful.
- 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.
- Temporarily replace the primary hash function with an alternate hash function (which sums the ASCII values). Time the execution (Linux time command) using the original hash and then using this alternate hash. Report the results for each test (run time and collision totals). Explain which hash function is better and explain why.

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

- Explain the difference between the hash function used in the assignment and a *cryptographic hash*[4].
- Provide the Big-Oh for the various hash operations (insert, erase, hash, and rehash).
- Suggest some things that might improve the overall performance for the hash table.

Make File:

You will need to develop a make file. You should be able to type:

```
make
```

Which should create the executables. The makefile should be able to build both executables; one for the testing (i.e., assocTest) and the other for the movie ratings program (i.e., movieRatings).

Submission:

- Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 4:00 PM.
- 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*).

Class Descriptions

- Associative Array Class
  The hash table class will implement functions specified below.

assocArray<myType>
-hashSize: unsigned int
-reSizeCount: unsigned int
-collisionCount: unsigned int
-entries: unsigned int
-currIteratorIndex: unsigned int
-*hashKeys: string
-*hashValues: myType
-loadFactor: static constexpr double = 0.65
-initialHashSize: static constexpr unsigned int = 1013
-hashSizesCount: static constexpr unsigned int = 13
-hashSizes[13]: static constexpr unsigned int = {9973, 30011, 60013, 120017, 240089, 480043, 960017, 1920013, 3840037, 7680103, 15360161, 30720299, 61440629};
+assocArray()
+~assocArray()
+operator[] (const string): myType&
+begin(): string
+end(): string
+next(): string

4 For more information, refer to: https://en.wikipedia.org/wiki/Cryptographic_hash_function

+exists(const string) const: bool
+erase(const string): bool
+printHash() const: void
+showStats() const: void
-hash(const string) const: unsigned int
-rehash(): void

Immediately after the class declaration, include the following:

```
template <class myType>
constexpr unsigned int assocArray<myType>::hashSizes[];
```

This will ensure that the constant array will be visible to the class members.

Function Descriptions

- The *assocArray()* constructor should initialize the hash table to an empty state. The *hashSize* class variable should be set to *initialHashSize* and the other variables initialized to 0, and the initial hash arrays, *hashKeys[]* and *hashValues[]*, should be dynamically created and initialized appropriately.
- The *~hashTable()* destructor should deallocate the dynamically allocated memory.
- The overloaded **operator[]** function should check the load factor, generate the initial index using the hash function, search for the passed string using quadratic probing. If the string is found, return the hash value. If the string is not found, insert the passed string into the keys and return the passed value (which, via the overloading will return a reference to that location which will set the value into the *hashValues* array). If the hash table entries exceeds the load factor (entries/tableSize), the table must be rehashed via the private *rehash()* function before the search or insertion is performed. The *hash()* function must be used determine the table location. If a collision occurs, the collision count should be incremented. The entries count should be updated as appropriate.
- The *begin()* function should return the key valid string key value (from the start of the *hashKeys[]* array). The function should set the *currIteratorIndex* class variable. If the current hash is empty, the function should return the empty string.
- The *next()* function should return the next valid key value. The function should use and updated the *currIteratorIndex* class variable. If there are no more entries, the function should return the empty string.
- The *end()* function should return the empty string and is only included for completness.
- The *showStats()* function is a utility function to print the entries count, current hash size, current hash table resize count, and the collision count. Refer to the example output for formatting examples.
- The *exists()* function should search the hash table for the passed key string and, if found, return true. The *hash()* must be used determine the table location. If the key is not found at that location, quadratic probing should be used. This function should ***not*** increment the collision count. If the passed key string is not found, the function should return false.
- The *printHash()* function should print all non-empty entries in the hash table. *Note*, this function is used for only testing.
- The *erase()* function should search the hash table for the passed word 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 by doubling the size of the current hash table, insert the old values from the old table into the new table (via the *insert()* function), and delete the old hash table. This function should increase the re-size count.
- The *hash()* function should return a hash from the passed string. The function should implement the SDBM[5] hash function. The hash value should be initialized to 0 and for each character in the passed key string, update the hash value by adding the character, the hash value left shifted by 6, the hash value left shifted by 16, and subtract the hash value. The final returned hash must be mod'ed with the current hash table size.
    - *Note*, in addition, an alternate 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 alternate hash function will only be used for testing as part of the final write-up.

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% ./assocTest
---------------------------------------
CS 302 - Assignment #7
Associative Array Test Program.

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

Array Dump (for testing)
---------------------------------------
Key: ball        Value: 8
Key: a           Value: 1
Key: bye         Value: 6
Key: their       Value: 7
Key: answer      Value: 3
Key: by          Value: 5
Key: any         Value: 4
Key: balloon     Value: 9
Key: the         Value: 0
Key: there       Value: 2

Values Sum: 45


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

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

Array Dump (for testing)
---------------------------------------
```

5 For more information, refer to: http://www.partow.net/programming/hashfunctions/#SDBMHashFunction

```
Key: abate       Value: 49
Key: aasvogel    Value: 14
Key: abacus      Value: 21
Key: aardvark    Value: 8
Key: abamps      Value: 30
Key: abasing     Value: 47
Key: abaca       Value: 17
Key: abaci       Value: 19
Key: aback       Value: 20
Key: aba         Value: 16
Key: aahs        Value: 3
Key: abaka       Value: 24
Key: aalii       Value: 5
Key: aals        Value: 7
Key: abasers     Value: 39
Key: aahed       Value: 1
Key: abase       Value: 35
Key: abash       Value: 41
Key: aaliis      Value: 6
Key: abasias     Value: 46
Key: aarrgh      Value: 11
Key: abas        Value: 34
Key: abalones    Value: 27
Key: abasedly    Value: 37
Key: abaft       Value: 23
Key: aah         Value: 0
Key: aal         Value: 4
Key: aas         Value: 13
Key: abacuses    Value: 22
Key: ab          Value: 15
Key: abapical    Value: 33
Key: abatable    Value: 48
Key: aargh       Value: 10
Key: abandons    Value: 32
Key: abased      Value: 36
Key: abaser      Value: 38
Key: abases      Value: 40
Key: abashing    Value: 44
Key: abasia      Value: 45
Key: abampere    Value: 29
Key: abamp       Value: 28
Key: abalone     Value: 26
Key: aarrghh     Value: 12
Key: abandon     Value: 31
Key: abakas      Value: 25
Key: abashed     Value: 42
Key: abashes     Value: 43
Key: aahing      Value: 2
Key: aardwolf    Value: 9
Key: abacas      Value: 18

Values Sum: 1225


Test Array One Stats
Hash Stats
   Current Entries Count: 0
   Current Hash Size: 1013
   Hash Resize Operations: 0
   Hash Collisions: 3

****************************************************************
Test Array Two Stats
Hash Stats
   Current Entries Count: 0
   Current Hash Size: 960017
   Hash Resize Operations: 7
   Hash Collisions: 2137311
```

```
--------------------------------------
Game Over, thank you for playing.
ed-vm%
ed-vm%
ed-vm%
ed-vm% ./movieRatings
Usage: ./movieRatings -i <moviesFile> -o <outputFile>
ed-vm%
ed-vm%
ed-vm% ./movieRatings -f none.txt -o tmp.txt
Error, invalid input file specifier.
ed-vm%
ed-vm%
ed-vm% ./movieRatings -i none.txt -o tmp.txt
Error, unable to input file.
ed-vm%
ed-vm%
ed-vm% ./movieRatings -i movies.txt -out tmp.txt
Error, invalid output file specifier.
ed-vm%
ed-vm%
ed-vm% echo "hello" > tmp.txt
ed-vm% chmod -w tmp.txt
ed-vm% ./movieRatings -i movies.txt -o tmp.txt
Error, unable to output file.
ed-vm% rm tmp.txt
rm: remove write-protected regular file 'tmp.txt'? y
ed-vm%
ed-vm%
ed-vm%
ed-vm%
ed-vm% ./movieRatings -i movies.txt -o tmp.txt
**************************************************************
CS 302 - Assignment #7

Overall Average: 6.91269

Lowest Rated Movies:
     Movie: The Brownstone Incident-1958                    Score: 0.7
     Movie: Land of Shadows-1987                            Score: 0.8
     Movie: The Tri-Tones-1957                              Score: 0.9

Highest Rated Movies:
     Movie: The Chinese Word for Horse-1977                 Score: 10.3
     Movie: Closed for Business-1997                        Score: 10.2
     Movie: You Shook Me-2014                               Score: 10.0

Hash Stats
   Current Entries Count: 809832
   Current Hash Size: 1920013
   Hash Resize Operations: 8
   Hash Collisions: 3377925

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