CS 302
Homework, Asst. #08

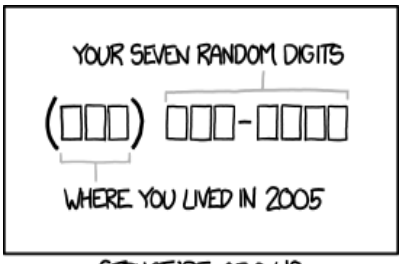
Purpose: Learn concepts regarding hash tables.
Due: Thursday (10/23) → Must be submitted on-line before class.
Points: Part A → 75 pts, Part B → 50 pts

Assignment:

Part A:

Design and implement a C++ class, ***phoneHash***, to implement a ***hash table***[1] data structure. The ***phoneHash*** class will implement a hash table for phone numbers in order to provide a reverse lookup capability. That is, given a phone number, the class will be able to find and return the associated name.

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



STRUCTURE OF A US CELL PHONE NUMBER

Source: www.xkcd.com

Part B:

Create and submit a brief write-up including the following:

- Name, Assignment, Section.
- Summary of the ***hash table*** data structure.
- Compare the hash table data structure to using a balanced binary search tree. Include the associated trade-offs.
- In the test data set, the instructors name was added first. Explain why the name showed near the end of the output.
- Note why the initial table size was set to 503 and explain what occurs when the load factor is reached (multiple times).
- Explain why the hash function used is good or poor for this application.
- Big-O for the various hash operations (insert, search, rehash).

Test Script:

You will need to develop a simple bash test script. The test script should verify that an executable file is passed as an argument, execute the program with the "-p" option using predefined input (provided), capture the output to a file, and compare the program output to a known correct output file (provided). If the files are the same, display Test Successful. If not, display the differences between the files. Refer to the example output to see en example of the output formatting.

Submission:

- Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 23:50.
- Submit a copy of the write-up (open document, word, or PDF format).

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

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

```
make
```

Which should create the executable.

Class Descriptions

- Phone Hash Class
  The phone hash class will implement functions specified below.

phoneHash
-title: string
-count: int
-tableSize: int
-phoneBook: **string
-loadFactor=0.65: static const double
-initialTableSize=503: static const int
+phoneHash()
+~phoneHash()
+entries() const: int
+insert(const string, const string): bool
+remove(const string): bool
+readPhonebook(const string): bool
+search(const string): string
+printPhonebook(): const void
+getTablesize() const: int
+showHash(string) const: int
-verifyPhoneNumber(const string): const bool
-hash(const string): const int
-rehash(): void

Function Descriptions

- The *phonehash()* constructor should initialize the hash table to an empty state.
- The *~phonehash()* destructor should recover all dynamically allocated memory.
- The public *entries()* function should return the total count of elements in the hash table.
- The *insert()* function should insert an entry into the hash table. If an invalid phone number is entered, the function should do nothing. If the hash table entries exceeds the load factor (count/tableSize), the table must be rehashed via the private *rehash()* function. The *hash()* function should be used determine the table location. If a collision occurs, linear probing should be used. The count should be updated accordingly.
- The *remove()* function should remove an entry from the hash table. The count should be updated accordingly.
- The *hash()* function should return a hash from the passed phone number. Specifically, the hash should be based on the last four digits on the phone number modulo the table size. The function should only be called with valid formatted phone numbers (as such, it does

not need to validate the input string).

- The *rehash()* function create a new hash table twice the size of the existing hash table, extract all entries from the current hash table, insert them into the new table, and delete the old hash table. The entries should be placed with the *insert()* function into the new hash table (which may place them at a different location).
- The *readPhonebook()* function should open a file (which is passed), read the phone numbers and names, enter the names into the hash table using the *insert()* function, and close the file. If an error occurs, return false, otherwise return true.
- The *search()* function should search the hash table for the passed phone number and, if found, return the name associated with that number. The function should return the empty string if the passed number is not valid or if it is not found.
- The *verifyPhoneNumber()* function should validate a passed string a valid phone number in the format of xxx-xxx-xxxx (where x is a digit). *Note*, for this assignment, other formats, such as (xxx) xxx-xxxx, are not accepted. If the string meets the required format, return true and return false otherwise.
- The *printPhonebook()* function should print all entries in the hash table. *Note*, this function is only used for testing on small phone books.
- The *getTablesize()* function is a utility function for testing and returns the current table size.
- The *showHash()* function is a utility function and calls the private *hash()* function to return the hash value of a passed phone number. The passed phone number must be validated before calling the private *hash()* function. Return -1 if the passed string is not a valid phone number.

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.***

Example Execution:

Below is an example output for the test script and a program execution for the main.

```
ed-vm% ./revPhone
CS 302 Assignment #8 Test Script

./revPhone -p <in.txt >out.txt
Test Successful

ed-vm%
ed-vm% ./revPhone
*****************************************************************
CS 302 - Assignment #8
Reverse Phone Number Lookup

=================================================================
Initial Testing

-----------------------------------------------------------
CS 302 Students

  555-895-5533   Doofenshmirtz, Heinz
  800-342-6001   Perfect, Joe
  727-724-5001   Sponge, Bob
  727-724-4001   Dallas, Steve
  239-338-3001   Runner, Road
  800-342-2001   Who, Dr.
  702-895-1000   Jorgensen, Ed
  717-724-1002   Bunny, Bugs
```

```
Entries = 8
Table Size = 503
Hash Test = 0
Hash Test = 502

================================================================
Test #0

Entries = 29
Table Size = 503
Hash Test = 0
Hash Test = 502

================================================================
Test #1

Entries = 200
Table Size = 503
Hash Test = 0
Hash Test = 502

================================================================
Test #2

Entries = 500
Table Size = 1006
Hash Test = 503
Hash Test = 502

================================================================
Test #3

Entries = 700
Table Size = 2012
Hash Test = 1509
Hash Test = 1508

================================================================
Test #4

Entries = 1000
Table Size = 2012
Hash Test = 1509
Hash Test = 1508

================================================================
Test #5

Entries = 3892
Table Size = 8048
Hash Test = 5533
Hash Test = 5532

----------------------------------------
Reverse Lookup
Enter Phone Number (return to exit): 702-895-5409
Name: Jorgensen, Edward
Enter Phone Number (return to exit): 702-895-1900
Name: Gewali, Laxmi
Enter Phone Number (return to exit):


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