**CS41 Programming Assignment #4**

**Total Point:** 100 ***+40 extra credit*** **Due Date:** **05/23/2017**, Tuesday

**Assignment Description:** Implement a dictionary application using hash table.

**Tasks:**

**A.** **Implement a Dictionary class using a HashTable of strings.** The table is stored as an **array of pointers** to a linked list of strings. That is, each element in the array is a pointer to a linked list that stores all the data that hashed to that index.

Your Dictionary (Hash Table) class should include the following functions:

* A **constructor** that accepts the size of the table as a parameter and initializes the table with NULL pointer.
* A function ***insert(str)*** that adds the string to the dictionary if it is not already there. The dictionary does not allow duplicate entry.

**Hint:** you may have to use the ***find(str)*** function to check if the string is already present in the table before doing the insertion.

* *A function****remove(str)****that removes the string from the table if it is in the dictionary. (****For extra credit****)*
* A function ***find(str)*** that returns ***true*** if the string **str** is in the table, else returns ***false***.
* A function **size()** that returns the number of words in the dictionary.
* A helper **hash function** to compute the hash code of a string. The function takes a string as its parameter, and adds up the ASCII values of all of the characters in that string to get an integer as the hash value. For simplicity, you can convert the string into lowercases before feeding it to the hash function.

Here is one possible hashing implementation using the string class in STL:

int hash ( const string s ) // (refer to p. 627 in textbook)

{

int value = 0;

for ( int i = 0; i < s.length(); i++ )

value += static\_cast<int>(s.[i]);

return (value % table.size);

}

(Hints: Refer to Fig.12.9 on page 623 of textbook as a sample class definition)

**B.** **Write a test driver program** that uses (and test) your hash table class functions.

**1. Build the dictionary**. The program reads strings from a text file that contain the words to be inserted into the dictionary.

* The file **wordlistab.txt** (which will be provided) contains about 550 words starting with letter a or b. You have to create an array (or vector) that is large enough to store these words. Choose a prime number close to 600 (e.g. 599, 601, 643, … ) as the size of array to decrease the chance of collisions.
* Use the **separate chaining** method to resolve the collision.
* Keep a **counter** to save the number of collisions that occur during this building process. Print out the **number of collisions** and total **number of words** that are added to the dictionary at end of building.
* After the dictionary is successfully built, **print out the contents of the first 20 elements of the table** (i.e. the list of words that are pointed to by these elements) **and the total number of words in the dictionary.**

**2. Exercise the Operations**

* After the dictionary is successfully built, invoke all the operations defined in the Dictionary ADT: Prompts the user to enter words for searching ***(and removing, for extra credit).*** Print messages about the result of operations accordingly.
* Invoke the ***find()*** operation with a list of words such as:

“an apple as appaitizer and read boook before breakfast are best”

and print out the words that are not in the dictionary. You can input more sentences to test the find() function.

**3.** *To test the delete() function, prompt the user to enter the words to be deleted (one-by-one), call the delete() function. If the word is in the dictionary, remove that word and print the results of operation (e.g. word find and deleted”, “word not found”, “other undeleted words in the same bucket are…”, etc.).* ***(For extra credit)***

4. Turn in source code, program input/output dialog, and a copy of the executable (.exe or jar file in USB or CD) for grading.