Homework 8. Deadline: 11.28. 11:59pm

Part 1. (50 points)

1. Given input {4371, 1323, 6173, 4199, 4344, 9679, 1989} and a hash function h(x)=x (mod 10), show the resulting Hash Table using a secondary hash function h2 (x) =7– x (mod 7). Table size is 10.

2. Draw the 11-entry hash table that results from using the hash function, h(i) =(2i + 5) mod 11, to hash the keys {12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5} assuming collisions are handled by linear probing. Table size is 11.

3. Demonstrate the insertion of the keys 5, 28, 19, 15, 20, 33, 12, 17, and 10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be h(k) = k mod 9.

4. Convert word "coffee" into an integer (hash table index) using Horner's method. (Use base = 32). Show your work.

5. Someone wants to store an **ordered** dictionary in a hash table. Is it a good idea? Explain.

Part 2. Hash Table and Spell Checker (150)

You will create a simple spell checker. First you will write a hash table that will be used to store the words. Then your program will read a set of words from a file (dictionary) that is given as the first argument, insert the words into a hash table.
Then the spelling checker will read a list of words from another file that is given as a second argument. Each word will be looked up in the dictionary. If it is incorrect, it will be written to the standard output together with a list of suggested corrections.

You will be given the following files:

- These files must be turned in
  - HashTable.java (Contains the main function)
  - SpellChecker.java (Contains the main function)
- Do not change the following files in any way (or turn them in)
  - IHashTable.java
  - HashTableTest.java
  - SpellCheckerTest.java
  - ISpellChecker.java

1. Hash table

- *insert(i):* Inserts element *i* in the hash table. Your program should return either "item successfully inserted," or "item already present."
- *lookup(i):* Uses the hash table to determine if *i* is in the data structure. Your program should return either "item found" or "item not found".
- *delete(i):* Use the hash table to determine where *i* is, delete it from the hash table. You program should return either "item successfully deleted" or "item not found".
- *print(PrintStream):* Print out the hash table. (see below)

Some implementation notes:

1. Assume all elements to be inserted are of type String.
2. For this assignment, you need to choose a good hash function to convert strings into indices. (Tuesday class topic).
3. Do not forget about double- and-rehash when the load factor is too large.
4. For collision resolution use separate chaining**.**
5. Use a combination of deletion from the list and the "lazy deletion" method by marking slots in the hash table as deleted.

For consistency, you can assume that the sequence of operations is provided in the following format, with each command on a separate line (in a text file, given as a command line argument)

Example:

```
insert "marina"
insert "table"
lookup "fifty"
insert "fifty"
lookup  "fifty"
delete "marina"
insert "fifty"
lookup "marina"
```

In addition, your program should print out (one line per command) information after each operation as follows:

```
item marina successfully inserted

item table successfully inserted
item fifty not found
item fifty successfully inserted
item fifty found
item marina successfully deleted
item fifty already present
item marina not found
```

Print method output (please use this output format to make our grading easier).

```
0:
1: marina, fifty
2:
3: table
4:
5:
6:
7:
```

Important: to motivate you start early, I will create a milestone submission. After 1 week I expect you to be done with part 1 (more or less). You will submit your code and I will reward you with **10 extra points**.

If later you find an error in your code you can fix it, it is ok for the code to be somewhat buggy.

2. Spell Check Part

After your hash table was created and tested, you are going to use it to create a dictionary.

The dictionary will consist of a list of words, separated by whitespace. You can assume that the words will be given in lowercase. You will need to insert them into a hash table that grows dynamically as necessary to hold the dictionary while keeping the load factor low enough.

You should keep track and report the following information for the hash table:

- The number of times you had to expand the table.
- The load factor in the table.
- The number of insertions that encountered a collision.
- The length of the longest known collision chain

Note that all but the first of these statistics will need to be reinitialized whenever you expand the table.

Write it out to the text file, HW8.txt in the following format r, c, n are integers, alpha is floating point number)

r resizes, load factor *alpha*, *c* collisions, *n* longest chain

Dictionary.

- You may wish to create a very small sample dictionary of your own for initial testing.
- A slightly larger dictionary of 341 words should help you to get most of your bugs out. (provided)
- When you're ready, try a dictionary with over 34,000 words (provided).

Ready to spell check? Let's get started!

Your program will read a list of words from another input file, the words can be any case, upper and lower (so what do you need to do first before you check?)

- If a word is found in the dictionary, your program should produce no output.
- If the word is not found, you should generate suggested corrections and write them, together with the original word (converted to lowercase), as a single output line. (see below)

Generating Corrections

The easiest way to generate corrections in a spell checker is a trial-and-error method. If we assume that the misspelled word contains only a single error, we can try all possible corrections and look each up in the dictionary.

For example, assume that we check "Pol"

Usually, spelling checkers have looked for four possible errors:

1) a wrong letter ("pol"),
2) an inserted letter ("pogl"),
3) a deleted letter ("po"),
4) or a pair of adjacent transposed letters ("plo").

To simplify this assignment, you will only need to deal with the first possibility, a wrong letter. When a word isn't found in the dictionary, you will need to look up all variants that can be generated by changing one letter. For example, given "pol," you should look up "aol", "bol", "col", etc. through "zol", then "pal", "pbl", "pcl" through "pzl", and so forth. Whenever you find a match in the dictionary, you should add it to your output line.

Possible outputs:

```
pol: pal,  pod, pop, pot, pox
xzzyz:
```

If every word in the input is found in the dictionary, the spell checker should produce no output (empty text file).

Submission: read carefully.

We will test your hash table from 1 separately from a spell checker method. Therefore, after you are done with a hash table, you need to submit it to HW8.HashTable assignment. Files that go there are:

```
HashTable.java
```

The spell checker has its own submission: HW8.SpellChecker. Files that go there are:

```
HashTable.java
SpellChecker.java
```

Extra Credit.

You can implement additional errors that spell checker looks into.

Case 2. Additional 5 points.
Case 3: Additional 5 points.
Case 4: Additional 10 points.