Homework 1 - a *hash* table

COP3503 assigned June 6, 2017
Michael McAlpin, Instructor due June 20, 2017

1 Objective

Build a hash table that supports searching, insertion, deletion, printing, and integer hash key creation based on text or string input data. In the event of collisions, this separate chaining hash table will use a singly linked list to store *duplicate* keys.

2 Requirements

1. Read the input file formatted as follows. The input file will contain at least one command per line, either insert, delete, search, print, or quit. These are defined in detail below. And if appropriate, a second parameter may be required. This string would contain a name, typically, less than seven characters. This name will be the data used to generate the hash. For example, one of the input files, named **5inserts.txt** contains the following:

```
i homer
i marge
i nelson
i gloria
i duffman
p
```

2. The specific commands are **i** for insert, **d** for delete, **s** for search, **p** for print, and **q** for quit.

   (a) Insert

   The insert command uses the single character **i** as the command token. The command token will be followed by a single space, then the name which will be the characters used to calculate the hash key as defined below. The program will then insert the key and the data in the hash table. In the event that there is a duplicate key, the new key (and data) would be added to the appropriate slot's linked list.

   (This command's success can be verified by using the *print* command.)

   (b) Delete

   The delete command uses the single character **d** as the command token. The command token will be followed by a single space, then the name which will

contain the characters used to calculate the hash key as defined below. The program will then delete that *key* and corresponding data from the hash table.
In the event that the *key* cannot be found, the program will issue an error message and recover gracefully to continue to accept commands.
(This command's success can be verified by using the *print* command.)

(c) Search

The insert command uses the single character **s** as the command token. The command token will be followed by a single space, then the name which will contain the characters used to calculate the hash key as defined in the formula below.
Upon successful location of the key, the corresponding key and data shall be output.
In the event that the key cannot be located, the program will advise the user with a message. See the Output section for an example.

(d) Print

The print command uses the single character **p** as the command token. This command will invoke the *print* function which will output all the slots in the hash table and subsequently, all the data in those slots. Given the size of the test data, the data ouput for each slot should contain the slot number, the hash key, and the data for that key. In the event that there is more than one data element in the slot, each element should be output, in the linked list order, until there is no more data in the slot. (See the Output section below for detailed formatting information.)
This command is critical for verification of all the commands specified above.

(e) Quit

The quit command uses the single character **q** as the command token. In the event the quit command is invoked, the program exits. There is no requirement for data persistence.

3. The hashing algorithm, as discussed in lecture and in the text, shall be based on *Horner's Rule* which produces the following pseudocode [1].

$$h \leftarrow 0;\ for\ i \leftarrow 0\ to\ s-1\ do\ h \leftarrow (h \cdot C + ord(c_i)) mod\ n$$

where C is a constant larger than every $ord(c_i)$.
In this assignment use **27** as the value for C and the **first** command line parameter as the *array size* for the value of n. (See below.)

2.1 Functions

While there are no specific design requirements (with one exception), it might be a meaningful suggestion to consider breaking this problem into several small classes. For example, a *hash* class and a *linked list* class would seem the minimal set of classes.

[1] See Section 6.5 on page 235 for the math and implementation pseudocode.

2.1.1 Required Function(s)

void **complexityIndicator**

Prints to **STDERR** the following:

- NID
- A difficulty rating of difficult you found this assignment on a scale of 1.0 (easy-peasy) through 5.0 (knuckle busting degree of difficulty).
- Duration, in hours, of the time you spent on this assignment.
- Sample output:

```
ff210377@eustis:~/COP3503$ ff210377;3.5;18.5
```

3 Testing

Make sure to test your code on Eustis **even if it works perfectly on your machine** . If your code does not compile on Eustis you will receive a 0 for the assignment. There will be five (5) input files and five (5) output files provided for testing your code, they are respectively shown in Table 1 and in Table 2.

Filename	Description
5inserts.txt	Five names with no duplicate names.
5in2out.txt	Five names added followed by two deletes. One will be a delete of a non-existent name.
5in1del2srch.txt	Five names added, one valid delete, followed by a valid search, then an invalid search.
5in1dup.txt	Five names will be inserted with one name being a duplicate, to verify linked list performance.
10in.txt	10 names inserted with no duplicates.

Table 1: Input files

The expected output for these test cases will also be provided as defined in Table 2. To compare your output to the expected output you will first need to redirect *STDOUT* to a text file. Run your code with the following command (substitute the actual names of the input and output file appropriately):

```
java Hw01 5 inputFile > output.txt
```

The run the following command (substitute the actual name of the expected output file):

```
diff output.txt expectedOutputFile
```

If there are any differences the relevant lines will be displayed (note that even a single extra space will cause a difference to be detected). If nothing is displayed, then congratulations - the outputs match! For each of the five (5) test cases, your code will need to output to *STDOUT* text that is identical to the corresponding *expectedOutputFile*. If your code crashes for a particular test case, you will not get credit for that case.

4 Submission - via WebCourses

The Java source file(s). Make sure that the *main* program is in Hw01.java.

Use reasonable and customary naming conventions for any classes you may create for this assignment.

5 Sample output

```
ff210377@eustis:~/COP3503$ java Hw01 5 5inserts.txt
ff210377;3.5;18.5
5inserts.txt contains:
i homer
i marge
i nelson
i gloria
i duffman
p
The Hash Table contains:
0. List (first->last): 0/duffman;
1. List (first->last):
2. List (first->last): 2/marge;2/gloria;
3. List (first->last): 3/homer;
4. List (first->last): 4/nelson;
ff210377@eustis:~/COP3503$ java Hw01 >5in-myOutput.txt
ff210377;3.5;18.5
ff210377@eustis:~/COP3503$ diff 5in-myOutput.txt 5in-expectedOutput.txt
mi113345@eustis:~/COP3503$
```

Note The **ff210377;3.5;18.5** output shown above is the output from the *complexityIndicator* function to **STDERR**.

Command	Output filenames
java Hw01 5 5inserts.txt	5in-ExpectedOut.txt
java Hw01 5 5in2out.txt	5in2-ExpectedOut.txt
java Hw01 5 5in1del2srch.txt	5in1d-ExpectedOut.txt
java Hw01 5 5in1dup.txt	5in1dup-ExpectedOut.txt
java Hw01 20 10in.txt	10in-ExpectedOut.txt

Table 2: Commands with input files and corresponding output files.

6 Grading

Grading will be based on the following rubric:

Percentage	Description
-100	Cannot compile on *Eustis*.
- 70	Cannot read input files.
- 25	Cannot insert a name into the hash table correctly.
- 25	Cannot search for a name in the hash table correctly. This includes a search for a non-existent name.
- 25	Cannot print the contents of the hash table correctly.
- 25	Cannot delete an matching entry in the hash table correctly. This includes the error case of correctly handling an attempted delete of a non-exstent name.
- 10	Output does not match *expectedOutput.txt* exactly.

Table 3: Grading Rubric