CS201 – Data Structures and Discrete Math **Fall 2014**

Programming Project 3

Due: Monday, 11/17/14 at 11:59 pm

Hash Tables

For this project, you are to write your own Hash Table class in C. The hash table is to use chaining (linked lists) to resolve collisions. The hash table is to use a string as input/key to the hash function. The hash function must use the simple function of adding together all the ASCII characters in the string and MODing by the size of the hash table. In addition to storing a string in the hash table, each string will also have an associated integer value that is to be stored in the hash table. The default size of the hash table must always be a prime number. This value can initially be any value that you wish (53 is offered as a good suggestion).

The linked list used by the chaining portion of the hash table must be written by you. You are not allowed to use library code in your program for the linked list.

The exact structure names and function names in your program is left entirely up to you. It is suggested that they are chosen to provide easy execution of the commands needed for this program.

The hash table is to be used in a program that will be interactive using a text based interface. You are to prompt the user for a command, read in the user's command and process that command. Each command will be given on a single input line (a single command cannot be split over multiple lines of input). The elements in the command can be separated by any amount of "white space" characters. The "white space" characters are the space and tab characters. You are welcome to use the string tokenizer to help parse the input or you may parse it yourself. If you user enters in input that is not 100% correct, you are to produce some meaningful error message, ignore that line of input and prompt the user for another line of input. If a line only contains white space or is completely empty, you are to ignore the line of input, but you are not to display a message.

The commands that your program is to implement are as follows. Note the "angle brackets" of < and > are to help in the description of the command. They are not actually part of the syntax of the command.

- **a <name> <integer>**

  Add or update the hash table with the String <name> and value <integer>. The hash table will only have one instance of each <name> in the table. When adding a new <name> name to the hash table, display a message starting the position in the hash table the information was added and whether any collisions occurred while adding the information.

  The <name> can be any sequence of characters that does not contain a white space character (spaces or tabs). The names do not need to follow the identifier naming syntax as specified by the C language. This allows us to do strange things like storing the value of 35 under the string "53".

- **r <name>**

Retrieve the integer value associated with the given <name> from the hash table and display the value in some meaningful form. If the <name> does not exist in the hash table, display a message stating this.

- **d <name>**

  Delete the information associated with the given <name> from the hash table. Display a meaningful message stating whether the <name> was deleted or whether it did not exist.

- **l**

  List all of the names stored in the hash table. These names are to be grouped together according to the position they are stored in the hash table and that position number must be included in the list. If a position does not have any information stored there, do not list the position number.

- **c**

  Clear all of the information from the hash table.

- **s <int>**

  Resize the hash table to size of the smallest prime number greater than or equal to the given integer value <int>. I.E. if the <int> value is 8, the table will be of size 11. If the <int> value is 17, the table will be of size 17. The current information in the hash table must be **"rehashed"** into the hash table of this new size. When finished with the rehashing operation, you are to print a message stating the current size of the hash table and how many collisions occurred during the rehashing operation. There is lots of code on the internet about determining whether a number is prime and finding prime numbers. Most use the MOD operation to check in number X is evenly divisible by number Y.

- **f <filename>**

  You are to read commands from the file with the given <filename>. This file can have any commands listed in it including another **f** command or the **q** command. When the end of the file has been reached, your program should first print an appropriate message and return the either the command line or the previous file (if multiple **f** commands are nested in the files).

  The **f** command does need to check to see if <filename> specified by this **f** is currently in use by a previous **f** command. If this is the case, display an appropriate message and ignore this **f** command. To do this, every time you open a file to read commands from, you should store that name in some storage structure and every time you get to the end the file you should remove that name from the storage structure. Then when you encounter and **f** command, you just have to determine is the name of the file is already stored in the storage structure. This sound like an ideal usage for a HASH TABLE!!! In fact, you are required to use a second instance of your hash table to keep track of the filenames in use. Since, we would not care what numeric value is being stored, you can just associate the value of zero with each name (or perhaps you may come up with a better numeric value to associate with each filename). Another item to note is that this is actually a good location to use recursion in keeping track of the open files, but this is a suggestion and not a requirement of the program.

- **w <filename>**

  You are to write the contents of the hash table out to the file <filename> such that the current hash table can be recreated by reading in the contains of the file via the **f** command. It is assumed that the output written to the file will:
    - first contain a **c** command to ensure the hash table is initially empty,
    - then contain an **s** command to get the hash table to the right size, and
    - finally be a sequence of **a** commands to add the values into the hash table.

- #

  This command is a comment. You are to ignore the rest of the information on this line.

- **q**

  Quit the program immediately. You are to quit the program even if you are in the middle of reading command from a file. Print an appropriate message upon completion of the program.

All data structure code in your program must be written by you. You are not allowed to use any pre-written/library stacks, queues, lists, sets, arrays, hash tables, etc. You may assume that all names in this program are 100 characters or less. Thus a C character array of static size 101 can be used.

The work you turn in must be 100% your own. You are not allowed to share code with any other person (inside this class or not). You may discuss the project with other persons; however, you may not show any code you write to another person nor may you look at any other person's written code.

Programming Style

Your program must be written using good programming style which includes:

- Meaning variable names
- Header comments for the file
- Header comments for each function
- Inline comments
- Proper indentation of code
- Blank lines between code sections
- Use of functions

Program Submission

You are to submit the programs for this lab via the Assignments Page in Blackboard.

To help the TA, name your file with your net-id and the assignment name, like:

- ptroy1ProjX.c