

## Hash Table

**Date assigned:** Wednesday, November 16, 2011  
**Date due:** Monday, November 28, 2011, 11:59pm  
**Points:** 50

We are implementing the Hash Table ADT as defined in the header file `hashtable.h` on Zeus in `/home/CS300Public/2011/hashtable`. The Hash Table will rely on your **Dynamic List** to implement chaining for collision handling. You must keep your chains in sorted order (ascending). A new `list.h` exists in the `hashtable` directory for you to use. The **key** to the hash table is a `char [100]` and the **data** stored is an `int` and a `double`.

The user of the hash table must supply the hash function to the `hashtable` as a function pointer and a compare function as a function pointer to compare keys in the `hashtable`.

The `hashtable` has the expected set of functions to insert, update, delete, and find data in the hash table. A function `htGetStats` must fill the `HT_Stats` structure when called. This finds the longest chain, the number of elements in the hash table, the average access time, and the number of empty buckets.

Additionally, an `htVisit` function and `htVisitAndUpdate` function are required, to allow the user to have a function called on every element in the `hashtable`. The `htVisitAndUpdate` function allows the user's function to update the data in the hash table during the visit.

In addition to implementing the data structure, you must provide a Makefile and test driver (`hashtableDriver.c` that produces an executable named `hashtableDriver`) that thoroughly tests your data structure. More about the driver is at the end of this assignment.

You may add any helper functions you need to `hashtable.c`. You may not alter `hashtable.h` or `list.h` in anyway.

1. Your code is to be written in C using Eclipse 3.7. Programs written in other environments will not be graded. Create an Eclipse project named `cs300_hashtable_PUNetID`. This project should contain the directories: `src`, `include`, and `bin`. The driver, `hashtableDriver`, should be created at the top level of the project, not in the `bin` directory.
2. The Makefile must contain the necessary targets to build the `hashtableDriver` as well as a `clean` and `dist` target similar to the identically named targets in your Stack assignment. There is no `testMe` target required for this assignment. Typing **make** on the command line should build `hashtableDriver`.
3. Submit a file called `cs300_hashtable_PUNetID.tar.gz` into the CS300 Drop Box by 11:59pm on the day in which the assignment is due. This file must contain your **Hash Table** project and **Dynamic List** project. Submit a color, double sided, stapled packet of code by that same deadline. The packet should be in the following order:  
Makefile

Hash Table Driver(.h then .c if you have both, otherwise just .c)  
hashtable.c (do not print list.h)  
Any extra .h/.c pairs you have.

4. Test one function at a time. This will lessen your level of frustration greatly.
5. You are to use the coding guidelines from V6.0 of the coding standards.

**Goals for this assignment:**

1. Code and test your program one function at a time.
2. Write efficient/clean code
3. Use the debugger to effectively develop a correct solution
4. Thoroughly test your code.

The hashtable.h header file contains a list of ERRORCODEs that each function can produce. Further, the ERRORCODEs are listed in order of precedence. If a function could produce multiple ERRORCODEs, the function must return the ERRORCODE highest on the list.

**The Hash Table Driver**

The hashtable driver **MUST**: read data from the file: “data/words.txt” and insert that data into the hashtable. This file is supplied on the class website. The format of this file is:  
**key int**

The driver **MUST** implement the following hash function: sum the ASCII values in the key and mod by the size of the table. This hash function must be used when you submit your code. You **MUST** provide a second hash function of your own devising (OR just implement the mid-square algorithm discussed in class). Be sure to thoroughly test this second hash function!

The driver **MUST** implement a compare function to compare two keys. Longer strings are greater.

The driver **MUST** implement and test a function suitable for use with **htVisit** and a function suitable for use with **htVisitAndUpdate**.

**This project will be reused in the next assignment.**