Lab 01 Hash Tables

**Fufillment**

* All of the requirements are met

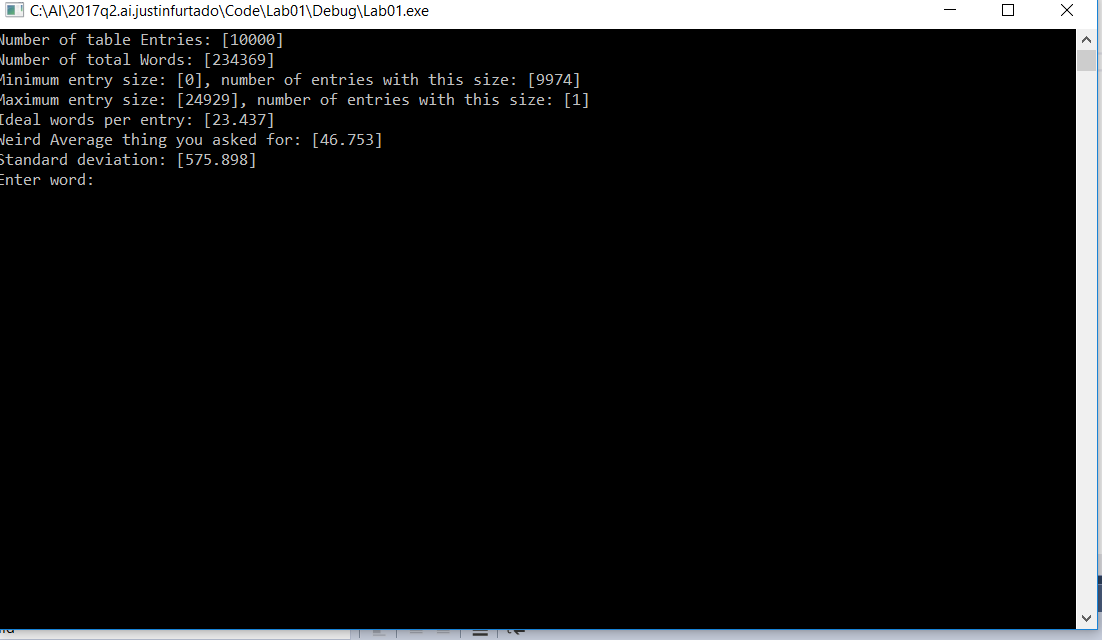
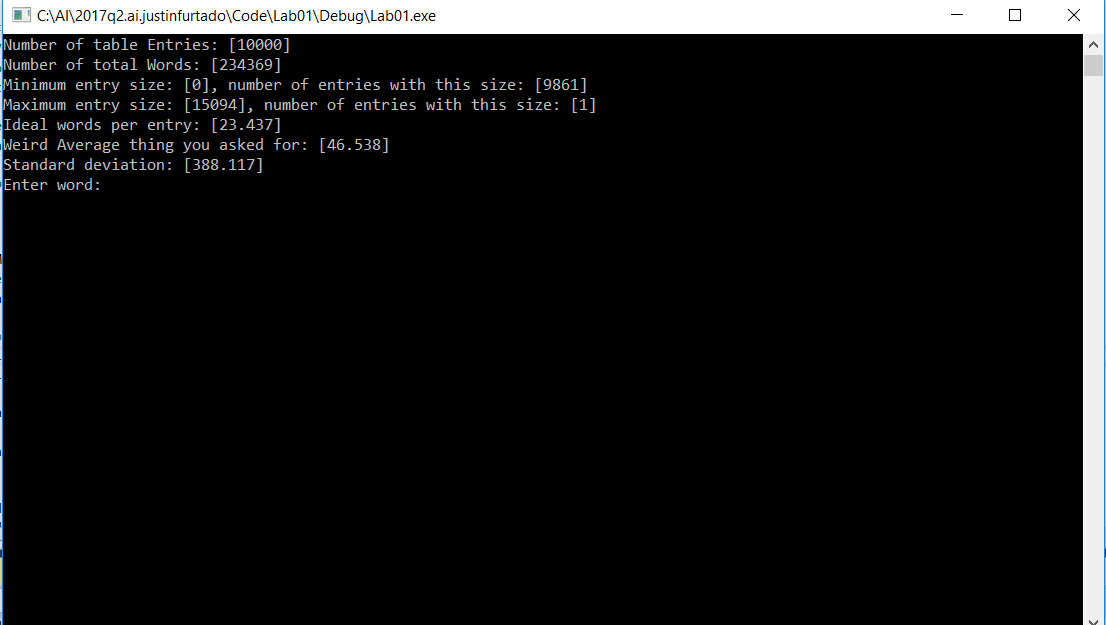
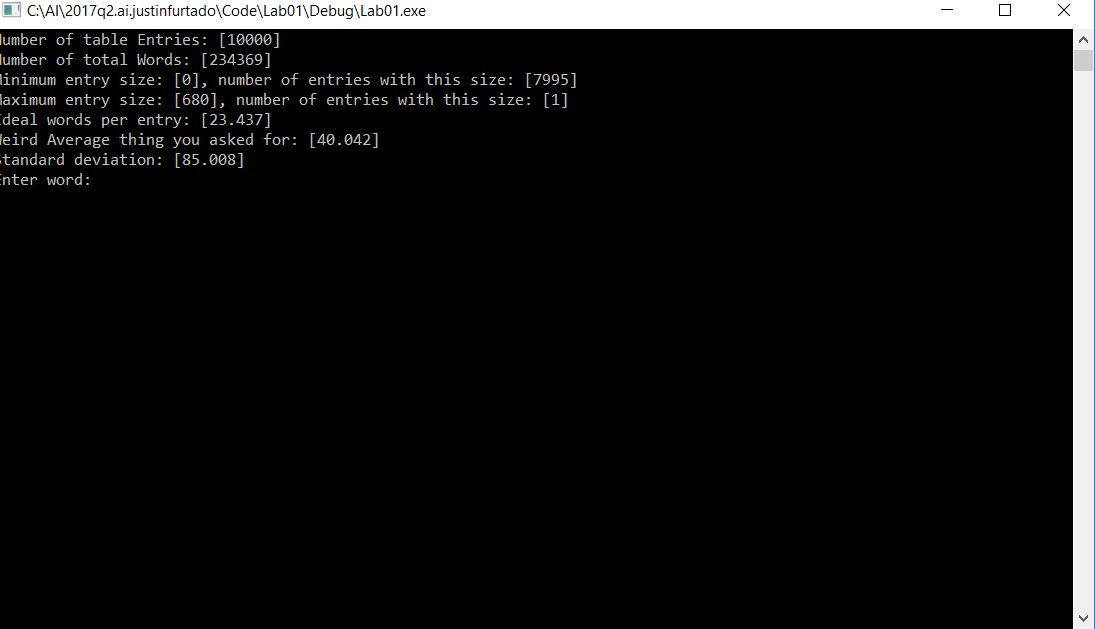
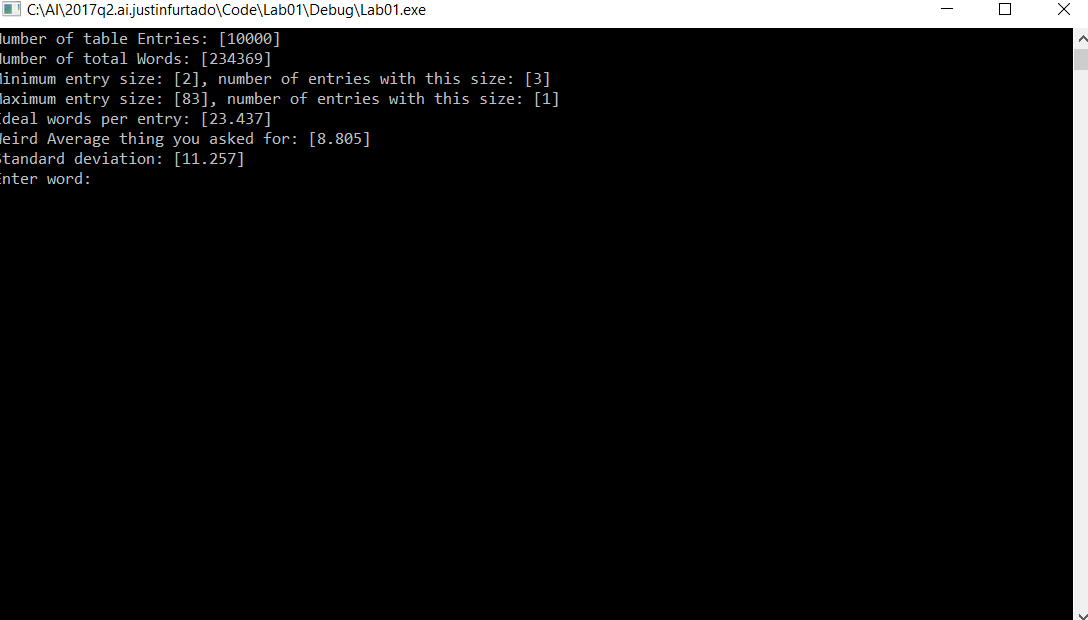
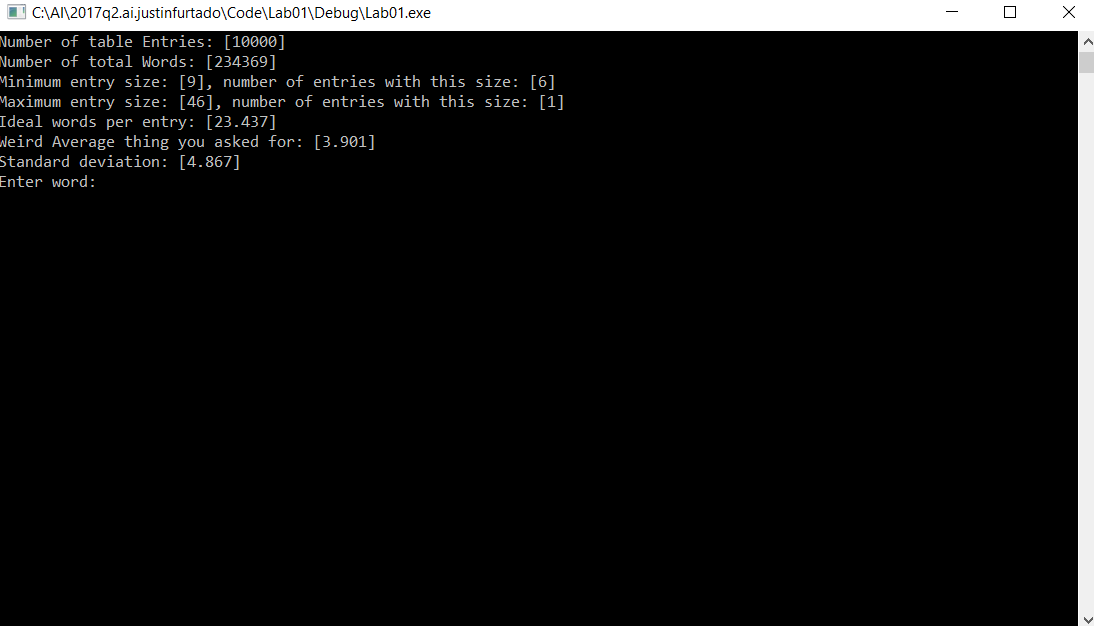
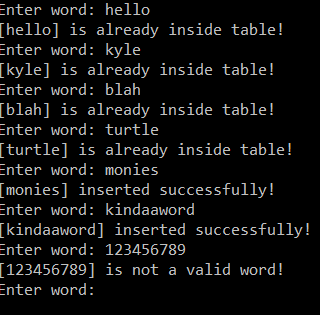
**Execution**

* This program expects two command line arguments
  + First, the relative path to the text file to parse for the word database
  + Second, the number of entries for the hash table to contain

**Controls**

* Entering a string of only “-“ will close the application
* Entering a word when prompted with “Enter word” will attempt to add the word to the dictionary if it is valid and does not already exist within the dictionary and report accordingly

**Screenshots**

* The hash that simply returns zero is absolutely horrible for large data sets
  + I got bored waiting for it to finish so no screenshot for the big data set, though it does work for smaller data sets
* The hash that returns the ascii value for the first character is better than returning zero, but still takes multiple minutes for 234369 words because the words are not even close to uniformly distributed as you can see by the stats
  + The range is massive, and the standard deviation is large
* 
* The hash that returns the sum of the first three ascii values is much better (though still awful) as can be seen in the simple statistics shown below
  + The range is smaller but still large, there are less empty buckets, and the standard deviation is lower, meaning that, on average, the number of entries per table spot is closer to the ideal number
* 
* For the function that sums the values of all of the characters, the hash is much more efficient.
  + Not only does it create the table much faster, but also the range is lower, there are less empty slots in the hash table and, again, the standard deviation is lower
* 
* For the hash function with bit shifts and xors that you suggested, the hash becomes even better than the above
  + Again, faster creation, smaller range, this time ZERO empty slots in the table, massively reduced standard deviation
* 
* Finally my custom hash function works even better than that
  + The smallest range, still no empty buckets, lowest completion time, lowest standard deviation
    - There are about 23.4 words per entry ideally, and on average, with my hash, the amount of words per entry is only 4.8 words away from that ideal number! I’d say that’s pretty good, though I assume there are even better algorithms in existence elsewhere
* 
* Screenshot showing word entry post-file-processing.
* 
* I did try different data sets, but I’m not showing stats for them because they were all smaller and less interesting to look at, plus writing a long lab doc takes time and I want to be done. With the other data sets, the hashes tended to fall in the same order when it comes to how uniformly distributed the data was, with only slight variations
* The data set I decided on finally can be found in the text file provided, its pretty much just a giant list of words
* From the experimentation with this lab, it seems that the more uniformly the data is distributed, the faster the lookups are
  + This makes sense because it means all the linked-lists have approximately the same number of elements, minimizing the iteration needed to find any element
* Hash tables are really useful, I can see plenty of applications for this
* Standard deviation was going above and beyond ☺

**Post-Mortem**

* This lab was relatively easy and took me less than five hours
* It was pretty interesting and should be helpful later on
* The allotted time for the lab seemed to be pretty appropriate, it was easy to casually work on and did not cause much stress or interference with other classes