CSCI-1200 Data Structures — Spring 2015
Lab 12 — Hash Tables

In this lab, you will experiment with our hash table implementation of a set. The key differences between the `ds_set` class (based on a binary search tree) and the `ds_hashset` class (based on a hash table, of course), are the performance of insert/find/erase: $O(log\ n)$ vs. $O(1)$, and the order that the elements are traversed using iterators: the `set` was *in order*, while the `hashset` is in no apparent order.

http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/12_hash_tables/ds_hashset.h
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/12_hash_tables/test_ds_hashset.cpp

Checkpoint 1

For the first part of this checkpoint, implement and test the `insert` function for the hashset. The insert function must first determine in which bin the new element belongs (using the hash function), and then insert the element into that bin *but only if it isn't there already.* The insert function returns a pair containing an iterator pointing at the element, and a bool indicating whether it was successfully inserted (true) or already there (false).

For the second part of this checkpoint, experiment with the hash function. In the provided code we include the implementation of a good hash function for strings. Are there any collisions for the small example? Now write some alternative hash functions. First, create a trivial hash function that is guaranteed to have many, many collisions. Then, create a hash function that is not terrible, but will unfortunately always place anagrams (words with the same letters, but rearranged) in the same bin. Test your alternate functions and be prepared to show the results to your TA.

To complete this checkpoint: Show a TA your debugged implementation of `insert` and your experimentation with alternative hash functions.

Checkpoint 2

Next, implement and test the `begin` function, which initializes the iteration through a `hashset`. Confirm that the elements in the set are visited in the same order they appear with the `print` function (which we have implemented for debugging purposes only).

Finally, implement and test the `resize` function. This function is automatically called from the `insert` function when the set gets "too full". This function should make a new top level vector structure of the requested size and copy all the data from the old structure to the new structure. Note that the elements will likely be shuffled around from the old structure to the new structure.

To complete this checkpoint: Show a TA these additions and the test output.

Checkpoint 3

Use the remainder of the lab time to learn how to use the STL `unordered_set` and/or `unordered_map` on your system. You will need to add:

```
#include <unordered_set>
#include <unordered_map>
```

And you may also need to add the compile flag `-std=c++11` to the compilation line when building your executable. Use the internet to search for instructions as necessary.

Write a program to load and store the words from the text files for Homework 7 Word Frequency Maps into a hash table and count and output the total number of *unique* words in the file.

Spend the rest of the lab time to work on Homework 9 and ask your TAs plenty of questions!