

## Assignment A5

**Due Date: December 11**

**Report Due Date: December 17**

### Purpose

This is it! Finally!

The purpose of this last (and easy?) project is to implement a hash table using a simple hash function.

### Problem

When looking at old English manuscripts, it is often difficult to determine exactly who the author or authors might be. Researchers in the field have asked computer scientists for help in finding patterns in large texts. These patterns can then help determine authorship. You will write such a program that will count the number of times words are used in a passage of text.

### Input

You should first prompt for a file name. The file will be a plain, ASCII text file containing a passage from a published book. Because different platforms test for EOF slightly differently, assume that the last line of the input contains one item in all capital letters: ENDFILE. Thus, read one word at a time (using a string) until you reach that sentinel value.

Finally, the program should prompt for the size of the hash table (an integer).

### Output

The program should display the five words with the highest counts, the number of times the words appeared in the input, and some performance statistics (see below).

### Specifics

You must implement a hash table that uses linear probing with an accompanying hash function, following these guidelines:

- Use OOP principles in your program. The hash table should be defined in a class, along with its associated methods such as `insert()`.
- The hash function is up to you. You may choose a simple method that will cause more collisions or one that is more complex.
- The hash table must store strings and associated counts. Initialize the table so that all of the counts are 0. This should be done in the constructor.
- Because the user will input the size of the hash table, you must assume that some collisions will result in wrap-around.

As done in previous projects, you must write a short report analyzing your program and hash function. Test the program with various length text and different table sizes. Compute and display timing data and/or number of search steps (that is, how many “jumps” had to be done to get to a

free location). In the report, compare the performance of the linear probing method using various table sizes and different length texts (containing  $N$  words). Use tables and/or graphs to bolster your arguments. Does your program match the theoretical performance of the linear probing? Why or why not?

Note that the performance should NOT include finding and displaying the top five words; rather, we are looking for the performance of **inserting**  $N$  words into the hash table (and counting up the number of matches).

### Notes

- Be sure that your program reads in a text file properly, and that the program ends when the sentinel is reached.
- Implementing this program should be much easier than previous projects. However, be sure to leave some time to run experiments and write your report.
- Email the program to me as usual. Be sure to name your source code and/or your zip/gz file by your last name. Hand in hard copy on December 14 at the latest.  
**Note for Mac users:** Please send either your source code or a zip file with necessary files **only**. I do not want a bunch of extraneous files and folders that seem to be generated in Xcode.
- Turn in your report at the beginning of the final exam on December 17 (or earlier, if you wish). The report will be part of your final exam grade. Expect a question on your project/report on the exam as well.
- Write/print and sign the Wheaton Honor Code Pledge on what you turn in: “I have abided by the Wheaton College Honor Code in this work.”
- Remember to save all of your work until your project is returned.

*I got distracted by learning.*

– Amy Hopkinson '09, in a Theory of Computation class.