Homework 2

September 2020

1 Introduction

In this assignment, you will be improving on your Caesar Cipher assignment by implementing a data structure to improve the performance. What that means is that you will be picking from an appropriate data structure to see which will speed up your program. Along with using data structures as a means to improve speed,we will enforce professional C software engineering practices. These include a Google Test Suite and separate header and code files to break up your code into testable modules. In addition, the main module will import the other modules and its could should be relatively short by comparison to that in the other modules.

2 Assignment

In other words, you will be taking your old code and making it both better and faster. You have a wide array of data structures to pick from to accomplish this task. You must draw on your previous knowledge of data structures to make an appropriate choice.

We recommend using any of these to help increase speed,

1. **Hash Table**
2. **Tree** - Binary or Height-Balanced (e.g. AVL or R-B tree)
3. **Queue**
4. **Heap**
5. **Sorted Array**

A stipulation to implementing any data structure is that you must implement your structure(s) from **scratch**. You are not permitted to use any 3rd party libraries in your code.

You may not copycode from any source when making your data structure(s). To ensure that some actual thought goes into this homework assignment *before the deadline*, you will pick a data structure and register it in Sakai. This mini-assignment (counted as participation) will ask you which data structures you will be using and why. You may actually select more than one data structure, but you must think very carefully about how you will use it to improve your program. Once you choose a data structure, you are committed to using it to improve the performance of your program.

This grade will fall under participation.

The next requirement is to write unit test for your code. Unit tests were introduced in Prof. Thiruvathukal's Zoom lecture on 8 September 2020. You would be wise to start from the *googletest-mva* repository (see Sakai Lessons if you cannot find it) and adapt it to serve your purposes.

Google offers an amazing testing suite that works great with C. You must:

1. Install the Google Test Suite on your own
2. Install `cmake` and make sure your code builds with it and the generated `Makefile`.

3. Implement at least 15 tests. Grading in this category will be ranked by those who do the most rignorous testing to those who do the least.

The tests must be thorough.

We will not be giving any resources to test your code. It is your job to think about how you'd test every function in your program. (If you don't have many functions, this is a *red flag.*) Make sure you are making extensive use of functions and abstract types (the C equivalent of *classes.*)

The old file with all the original shifts will not be used. Although, we will provide a dictionary file of all possible words.

Finally, the last component to the assignment is speed. We will perform a stress test and timings on all solutions, which will be ranked from fastest to slowest on an Intel-based processor.

If you successfully completed the first homework, you must beat your old speed which will be given to you. If not, you will basically need to complete all of the requirements of homework 1 and homework 2, which will each be graded separately.

The speed calculation will given will be from our machine, which is very fast. So this number in its own right will not be helpful to you.

You should time your code before making changes. You might want to consider keeping a folder of timings as your program continues to evolve (and, hopefully, become faster).

Lastly, you must also explain why the new method is faster in your documentation. It is important that you have detailed commments and documentation in this assignment. In addition, you must *cite* any sources you use that help you to implement your improvements. Even if you have the best solution, you are expected to have clear detailed explanations for how you achieved it.

3 Rubric

You will be graded using the following criteria.

1. 1 point for proper GitHub usage and software engineering practices.
2. 1 point for a `cmake` file to compile your code.
3. 1 point for your speed analysis and improvement of your original code.
4. 3 points for your tests and their thoroughness.
5. 4 points for your data structure implementation.