Project 1: Large Map

Instructions: Storing and searching based on key-value pairs is a common problem in Computer Science. Complex data structures exist for general purpose key-value search and sort. If given some constraints very efficient ones can be created from simple array(s).

Our Map shall map a student's first and last name to an id. In other words we are mapping a String to an integer. Names shall consist of:

- A single array of characters
- A combination of first and last name separate by a space.
- The first letter of each name shall be capitalized, the rest is in lower case.

In between each first and last name shall be a space. Every student's name shall map to a single 32-bit integer. All IDs will be strictly above 0.

It is imperative you think about how we can simplify this problem with the given constraints. A general solution will be too inefficient to solve in the given time. Here is the interface to consider:

```
class Map{
public:
    /* Adds (inserts) val with the associated key.
     * Returns if successful or not. (It is not successful if we are out of
     * memory, or if the key already exists.)
     */
    bool add(const char *key, int val);

    /* Searches for the key. If found it sets ret to the correct val and
     * returns true. Otherwise this function returns false.
     */
    bool get(const char *key, int &ret);

    /* Returns the size (memory consumed) by this data structure. */
    int size();

    /* Removes the current value from the Map AND frees up any memory that
     * it can.
     * Returns true if there was something to remove, false otherwise.
     */
    bool remove(const char *key);

    /* Returns the number of names with a given prefix.
     * EX: If we have {John, Jonathan, Paul, Mark, Luke, Joanna} then
     * howMany("Jo") == 3
     */
    int howMany(const char *prefix);

    /* Frees all memory */
```

```
30      ~Map();
31  private:
32      /* Store you data here. I highly recommend talking to me about your
33       * data structure before implementing. */
34  };
```

Important Questions:
The crux of this project is determining what your data structure will consist of. I have provided a common list of first names and last name (firstnames.txt and lastnames.txt.) If you were to create an array of all possible combinations, it would take up over 4.5 GB. A list of all possible ids would be just under a GB for a total of about 5.5GB of memory–which may not fit in your RAM! Hence take some care in testing to ensure you stay within RAM.

The main thing to keep in mind is that we are storing the ids (integers), while the student's name is used to index to an ID. How can we efficiently store the names to map to a integer? It is vital you think about the data structure **before** you start coding. I suggest drawing out some pictures if you are having problems.

Hints:
So far we have have learned arrays, pointers, and (soon) linked lists. That should be enough to design your data structure. No hashmaps for this problem–they will not solve the prefix problem efficiently. You will be judged on memory footprint as well as performance!

Code Names:
I plan to present speed and memory results for this project. By CSU policy I am not allowed to display students real name without explicit permission. Instead I can use an alias. If you desire to know how your code stacked up against others, please provide a code name. (Ex: CodeWizard, Wombat, BeastFromTheEast, CodeWarrior, 1337HAXOR, etc...)

Warning:
Data Structure projects are *significantly* more demanding than labs. It is imperative you start thinking about the data structure now! If you hit a road block (and you should/will), please come talk to me. When you talk to me be prepared to show me what you have thought of so far. If you come in with an idea I will guide your idea to a good solution. If you come in without and idea, I will tell you to keep thinking about it. Keep in mind there are many good, fair, and terrible solutions to Data Structure projects.

Grading:
You program will first be graded based on accuracy. If your program passed my test cases, then additional points will be awards for performance. Namely:
Accuracy: 30 %
Performance of Search ('get'): 30 %
Performance of howMany(): 30 %
Memory Footprint: 10 %
I have generated some graphs in objectives to give you an idea of performance. Please keep in mind that these timing were taken from my machine, which at the time of Writing was:
Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz
MemTotal: 8083184 kB

To be clear, a HashMap/HashTable implementation will issued a 0.

STL Tools: From the STL you may use vector, string, list, and iostream. You may use any standard C library tools.

Write some test cases:
Part of this project grade will be how well you can write test cases. You will be in charge of all test cases.

How to turn in:
Turn in via GitHub. Ensure the file(s) are in your directory and then:

- $ git add <files>
- $ git commit
- $ git push

Due Date: February 13, 2019 2359

Teamwork: No teamwork, your work must be your own.