**Project 3**

**Arteen Abrishami**

**UID 205-577-156**

Notable Obstacles

**Initial Development:**

*main idea for theJumbler:*

*reorder word and reset it to its own character but in ASCII order*

*reorder each entry in the dict[] that matches the size of word*

*temporarily store into a local string (if meets other specifications) (as a return)*

*compare against word and if matches, add the unordered entry in the dict[] to results[]*

*Issue:*

*Overloaded stack space in g32*

*Worked perfectly otherwise (have left the code for the functions related to it in the file but //'d out for my own future review/yours)*

*Processed all words in milliseconds*

- figuring out how to work infile/ifstream and linking my file on Xcode (1 hour)

- Solution: re linked working directory to the one of the executable through Product -> Change Scheme

- initial writing of lexiconBuilder, confused by the formatting (20-30 minutes)

Solution: figuring out how if statements with regard to getline/reading in work

- Writing recursive loops instead of for/while (30-45 minutes)

Solution: Pretty simple, wrote the for/while loops and extrapolated from there - have left the NonRecursive function //'d out for now

- count not incrementing properly, had to write cerr statements, figure out returns, etc. definitely took me the longest after writing theJumbler (1-2 hours)

Solution: fixed return statements, had to do with the helper function being returned instead of just count (as a reference)

- not working properly with the allocated stack size in g32, works in milliseconds on Xcode with the same stack size

- new idea is to check differently (took so long that I have titled it below as secondary development)

**5-6 hours total**

**Secondary Development:**

*main idea for theJumbler: test if statement with a boolean to determine if word should be added or not, will check to see letter by letter to see whether or not the letter is present in both word and the dict[] string passed in.*

*No issues, works perfectly in all cases, processes in milliseconds for all manner of words within the 8 mb stack space on both g32 and Xcode*

- main issue was that it required me to think recursively again but in a different way, and I tried too hard and made it too complicated, when really it was actually much simpler than even my earlier method, if only a different way of thinking

- I initially was struggling with understanding the recursive returns, eventually came up with different ideas on how to tackle the indexes for dict and word, ended up removing letter by letter using substr, having a loop that I was trying to make into a boolean

- was not successful (after several hours of trying) to create a boolean helper function as a standalone, created a secondary helper with the sole purpose of passing a boolean reference to a void function that simply looped through and processed characters, which made the rest of it go by in about 15 or so minutes after I finalized, and tested on both Xcode and g32

**5-6 hours total**

**Total of 12 or so hours spent on project**

Synopsis: Rather than use the professor's pseudocode for permutation, which seemed confusing to me, I developed two distinct recursive methods/algorithms for my own use which had the added advantage of processing in milliseconds, compared to what my classmates reported as taking over an hour for longer words with their method. My code is about 100 lines in length. Overall, I am extremely happy with my algorithm, the project, and feel that I have learned a lot, just as I did at the end of Project 2.

- Arteen Abrishami (July 25, 2022)

Test Data

*Handles all correctly*

<empty string>

anaconda

regardless

rat

abc

look

wow

on

never