CIS*3190 Software for Legacy Systems

Assignment 1 - Fortran Programming (Word Jumble)

Maneesh Wijewardhana (1125828)

- **QUICK NOTE**
  - In the beginning of the assignment, I went about linear searching the system dictionary but after some performance issues, I decided to sort it in order to do a more efficient search later on. For this step, I initially implemented Merge Sort. **However**, we were provided a new dictionary that was sorted alphabetically but not by Ascii, I decided to remove Merge Sort and lowercase all the words instead in order to keep the efficient search the same.

1. - I decided to first build the word dictionary into a dynamically allocated array at the start of the program so it only happens once.
     - This was done by opening the file, reading the line count, allocating an array for the appropriate size, then reading in each line.
2. - I then prompt the user for a value for the amount of words they want which is then used to allocate my jumble pair data structure.
     - This structure contains the user inputted jumble as well as the first 'real' anagram that was found which is done using the generateAnagram subroutine.
     - This subroutine will generate permutations of the user inputted string using a recursion and backtracking technique and call upon another subroutine named findAnagram.
     - findAnagram's purpose is to find the first real word in the set of permutations which is done by another subroutine named findlex.
     - findlex's purpose is to search through our word dictionary for the permutations of our string but does this using binary search since our dictionary was sorted from before. This dramatically improves search speed especially when the user string gets longer as the # of permutations increase by an order of n!
3. - I make sure to sanitize the user inputted words at which point I call my generateAnagram subroutine.
4. - Once the first word has been found, we save this into our data structure for that specific jumble that the user inputted.
5. - I then give the user a choice to solve the final jumble or not and also make sure to error check the input given.
6. - If yes, then we simply allocate another copy of the jumble data structure except this time with size one as we know it will only be one jumble.
7. - Since the user chooses which letters of each jumble are in the final jumble, I let them enter characters and concatenate them into the final jumble data structure (I make sure to sanitize the input here as well).
8. - I then repeat the same steps above to generate the real anagram for this final jumble and return it to the user.
9. - At any point, if the generateAnagram subroutine exhausts all possible permutations which means none was found in our system dictionary, then we return 'No anagram found' for that particular word.

- Overall I felt as though using Fortran to solve this problem introduced a lot of tedious tasks that could have been abstracted away if using another language such as C, particular when it came to some helper subroutines I had to make and other small quirks. For example, I needed to make a custom toLowerCase subroutine for input sanitization whereas in another language, these functions would be built in. I also had some trouble printing to stdout in a formatted way. These problems forced me to introduce more code to fix an issue that would not be there in another language. Still, I feel as though writing this program in another low-level language would introduce some other problems, but the general implementation would have stayed the same. On the other hand, using Fortran was similar to using other languages such as C, especially due to how I worked with memory and initialized variables. Other than the specific syntax I needed to learn (which did not take long at all), I felt comfortable allocating and free'ing memory like I am used to and creating my own data structure using the type declaration felt easy. Also, being able to specify exactly what variables return and get passed into a subroutine allowed me to catch bugs early on. I appreciated how I can split certain functionalities into modules since it felt like a more cohesive program that can easily be iterated upon. For example, if I wanted the functionality for searching the dictionary to use a different algorithm, I know exactly where to change that and be certain that it will not affect any other code other than the ones in the module it is in. Given my knowledge of programming, Fortran was definitely easy to learn but just needed more code to do the things that I am used to in other languages which was not a huge problem. In any case, it allowed me to explore deeper on how these built-in functions actually operated so at the end of the day, I still learned more than I was expecting. Although Fortran is a little more verbose than other languages, it did not hinder the learning experience at all since quick documentation lookups usually solved any language specific problems I was having.