Daniel Domalik
CIS3190
Michael Wirth
03/04/2018

Assignment 2, Ada Word Jumble

To start, I will run through a description of how the program operates. When you run the program, it will display a short introduction as well as some rules. A check will then be performed to verify that the dictionary file exists. If the file does not exist, the program will terminate with an error message. If the file does exist, the buildLEXICON function is called and each string in the file under 7 characters is added to the data structure, which is an array of strings. The user is only allowed to enter jumbles up to 6 characters in length, and only 25 per iteration. After the user enters a jumble, the program verifies that it is in between 1 and 6 characters in length. The jumbles will then be passed through the program to solve for anagrams. The algorithm I decided to go with was to count the frequency of each individual character and store that in an integer array. In order for the program to find a solution, all values in the integer array must be the same between words from the jumble and the lexicon. After solutions are displayed, the user can enter more jumbles and continue doing so as s/he desires.

A simpler version can be displayed like this:

1. Check if dictionary file exists
2. Load dictionary file into data structure/lexicon
3. Prompt user for input
4. Verify user input
5. Pass user input to algorithm
6. Algorithm counts character frequency for jumble
7. Algorithm loops through lexicon counting character frequency
8. Two frequencies are compared
9. Algorithm prints out matches

My rationale behind picking a string array was because it was the simplest solution in my head. The very first thing I looked at was parsing the dictionary file. I figured I could just parse through the file until the end and load strings into an array as I go. I use two string arrays, one for the lexicon and one for the user inputted jumbles. These two string arrays are then passed into another function which counts the frequency of each character and puts it into an integer array. These two integer arrays are then compared to see if there is a match or not.

The problem was obviously doable in Ada, although I would have preferred a different language to create a solution. In Java, this problem could easily be solved by a couple lines of code. You could load all the words into a hash-map, count frequencies, and then compare. Similar to what I did, but it would be much easier.

In terms of advantages/disadvantages that come to learning and/or programming in Ada, the compiler is kind of a double-edged sword. Someone who is proficient in Ada would not have these problems, I'm sure, but as someone who is beginning to learn it it is quite frustrating sometimes. For starters, you need to make sure everything in the code is perfect before going to test it. The compiler will not compile your program unless it is perfect. It is very strict and unforgiving, although it does give you some pointers as to what you need to fix. That could be both good and bad. It promotes good programming and understanding of code in general, which is good. At the same time, it can quite infuriating when bringing practices over from other languages. Some things I ran into trouble with were array indexing, Ada starting at 1 instead of at 0. Something as simple as printing an integer is quite different in Ada than it is in Python. You can manipulate strings easy, they are character arrays. This made the program easier to complete.

My biggest gripe with Ada is strings. The different string types, bounded, fixed-length, and unbounded, seemed quite backwards at first. Learning the language for the first time, and the problem being heavily string centrered was quite a bad combo. Starting out, I kept fumbling around trying to assign the dictionary contents to a string array, and then manipulating that string array gave me issues as well. This string is unbounded, you can't do that! This string is bounded, you have to do it like this! Looking back at it now, it is much easier to dynamically allocate memory for a string than it is in a language like C. In C you have to malloc every time the string changes size, in Ada, you can have an unbound string and just convert it every time you would like to perform operations on it. At the beginning it was annoying, but now I understand it better and find it much simpler and easier.

In conclusion, Ada was quite enjoyable (except for the first three hours fumbling about string operations). It was a small little program, but was challenging nonetheless. It didn't help that I started it last minute. Ada was more complex, personally, than Fortran was. Fortran was straightforward for me and I understood how to do everything I needed to do in a short amount of time. Ada, on the other hand, threw some punches at me and told me I couldn't do things like I have in other languages.