Eric Morse (1141504)
Fortran 95 Reflection
Febuary 3rd, 2023

Fortran was a relatively easy language to learn. Thanks to prior experience using Assembly language, as well as C and other more “modern” languages. The hardest part was probably re-learning the types for variables. However even that wasn’t difficult.

Overall I would say that fortran was a middle of the road language for this task. Maybe it was because I was thinking in a “mordern” sense, however I felt it was difficult to do things that would have been quite easy in C or python. That being said, I have used python to parse through larger datasets, and I believe it is very likely that this task would have ran a lot slower in a more modern language like python. I did however, like how easy it was to use the lexicon module in the solveJumble program. Being able to just throw a “USE lexicon” anywhere where those subroutines where needed was a lot easier than trying to set something up in C or python.

My design approach was a simple one. I started with the lexicon and went from there. The very first thing I wanted to do, was find a good method to taking in a storing the dictionary. This consisted of me inputting each word in one at a time, then creating a tolower subroutine so that I was always working with lowercase strings. At first I had put this in a basic array, but that caused challenges having to go through the file twice, as well as not meeting the task requirements. I then switched over to using a linked list. My type had 2 variables. A pointer to the next node, as well as a character array to store the dictionary word. After implementing this, it made it quite easy to take in the dictionary, as for each word I would just allocate space for it, make it all lower, and add it to the end of the linked list. After that, it was a simple task creating the findlexicon subroutine. That consisted of me going through the linked list, and setting a variable true then exiting if we found a match.

After lexicon, I worked on getting input from the users for the jumbled words, which consisted of return a lowercase word from inputJumbe n times, and then making it lower and putting it into another type. This type consisted of 2 strings, a list of strings for anagrams, and a integer counter. The 2 string where for the pre-solved word, and a solved word. The counter was to keep track of how many anagrams I had created so far, and where to put the next one in the list. Then I created the generate anagrams which would recursively generate a new anagram, and put it at the counters index in the anagram list. And finally, findAnagram, which would go through the anagram list, and check if the anagram was in the dict using findlexicon. After that it would set solved word to that anagram and exit. At the end of the program, I get the circled words from the user, and use the generateAnagram and findAnagram subroutines one more time to solve the final jumble, and print that to the user.