Reflection Report

This assignment introduced me to Fortran, the first legacy programming language that I would learn this semester. I found Fortran to be easy to pick up, but it took me a while to get used to the syntax. I'm accustomed to C-like syntaxes, and I found this syntax to be like assembly-like languages which I found difficult to pick up at first. However, once I got the hang of it, I could easily add features to the program.

One thing that I liked about using Fortran was not having to deal with pointers directly. Although I still ran into segmentation faults occasionally, I didn't really have to deal with memory directly unlike C. Another thing I liked about Fortran was the ease of printing the same number in different bases. If you wanted to print out the same array in decimal, then again in hex, all I needed to do was change the format of the write statement.

Although writing this type of program was easy in Fortran, I found that the features I needed to implement would have been easier to do in something like C. Granted, this is probably just because I'm not too familiar with the language like I am in C. I found that dealing with strings in Fortran to be less intuitive than in C, and all the formatting needed in Fortran for simple output was hard to wrap my head around. In C all I needed to know was the type of what I was printing out, and it was more explicit.

I followed a linear approach to refactoring the old Fortran code to be the newer standard. I followed both the unit notes and the coding practice from the website. First, I changed the comments form c to !. I then changed all the variable declarations to use the double colon method of declaration, including removing any data statements and to simply initialize them on declaration. Then, I changed all ifs to use the new types of comparators (for example, .eq. to ==) and the dos to use end do rather than continues. Finally, I got rid of equivalence statements and either made new variables for them or reshaped them accordingly. I also added more whitespace and in line documentation to make the code more readable. I tried removing as many formats as I could because I found them difficult to follow the flow of the program.

Once I re-engineered the code to be usable, I tried adding in the ciphertext feature for Task 1. I did this easily, but I ran into some problems displaying the ciphertext properly. This was simply because I wasn't using the correct format for the hex text. When designing the readWord and word2hex functions, I decided to take user input as a char array since that would be easier to convert to ASCII later instead of having to do it while reading input. By passing in the word as a string, I just had to use iachar to convert the letter into ascii, and then iterate through that list converting the new decimal number into hex. With it being a string, I could also get the real length using len_trim rather than calculating it. I found I was running into a lot of pitfalls trying to convert the ascii to hex. This was the most trouble I had during the assignment. I initially thought I could use a write and format statement to give me the hex value and save that output to a variable, but trying that led me nowhere so I simply made an algorithm to manually convert the value by getting the remainder. Finally, I added an extra parameter to the hex2word function which returns the length of the new word in hex so that I can easily calculate the length for the printhex function.