Peter Hudel 1012673

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 with a few caveats. I'm accustomed to C-like syntaxes, and I found Fortran to more akin to assembly-like languages, which I found difficult to pick up at first. I also had some trouble getting used to format statements, as the descriptors don't make much sense to a beginner (why is F for real numbers? Shouldn't it be something like R?)

Once I got the hang of it, I really enjoyed writing in Fortran. After getting used to the format statements, I liked the ease of printing out the saem value in a different way. 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. I also found it interesting how arrays could begin and end at any index desired. This made iterating through lists strange at first, but giving the programmer more flexibility is appreciated.

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 a higher level language. I found that dealing with strings in Fortran to be less intuitive than in something like Python, so trying to convert a string to hex was a bit challenging. I also wish the documentation for Fortran was better, as when I wanted to search for something outside of the lecture notes, I found it difficult to get the results I wanted.

When designing my program, I followed a linear approach to refactoring the old Fortran code to follow the F95+ standard. I followed both the unit notes and the coding practice from the website. First, I changed the comments from c to !. I then changed all the variable declarations to use the double colon method of declaration, including removing any data statements as well as removing implicits. 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. I then changed all subroutines to have intent statements. 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 string since that would be easier to convert to ASCII later. To do this, 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. After some fiddling with format statements, I simply made an algorithm to manually convert the value by getting the remainder. I then finished the printhex function and implemented them into my main program, adjusting it to use my new functions to pass it into lucifer().