Fortran — Jason Nguyen (XXXXXXX)

Learning Fortran was quite the experience. There were convenient parts of the language, but there were also annoying parts. Re-engineering the code took this one step further. I didn't think that F95 code could look that different from F77, but it did. For starters, the most extreme difference between the two was implicit variables. These implicit variables made understanding the code difficult. It's no wonder that so many people dislike JavaScript's messiness for similar reasons. I made sure getting rid of these implicit variables was my first non-cosmetic change. Of course, fixing the comments, indenting, and compiling was before all.

My next biggest gripe with the old code was the labelling. Print statements as simple as ' key ' had entire labels dedicated to them. These labels would all appear at the bottom of the program, a design decision that bothered me. What also bothered me was having to trace through the archaic labelled do loops. It wasn't immediately clear if you had to nest a do loop inside another or not.

One thing that was frustrating, but fascinated me, later, was 'equivalence'. It was an intriguing operator that I hadn't encountered in the C programming language. Mapping memory from one variable to another was unheard of at the time and I wanted to learn how it worked. For developers living in the age of F77, every byte mattered, so it made sense. In discussing this with my colleagues (later confirmed by the prof. through an email), there was a mistake in the original code. The author of this variant of Lucifer (Sorkin) mismapped the variables' equivalences. To my understanding, his intent was to map key to k and message to m. People noticed that instead of starting at 0, the equivalence started at 1. This was a problem because 'message' and 'key' started at 0 and ended at 127. What this meant was the equivalence would start one element later (from 1, and not from 0). In investigating this issue with my friend, we changed the 1s in the statement to 0s. As a result, the code ended up matching the example output. Nonetheless, I had already reshape()'d the arrays by the time we made this discovery.

On a positive note, I like the bounds safety of strings in Fortran. These built-in string variables (character(len = x)) are safer than C's strings. For starters, Fortran's arrays beats C in buffer overflow safety. If you define a string of length n in Fortran, the read(*,*) statement only reads the first n characters. The rest are ignored. Meaning, exploits like gets() in C aren't as apparent. Furthermore, I like how you can create substrings by doing string(n:m). In C, you would have to do a lot more busywork to achieve the same result. To top it off, you can use len() and len_trim(), very convenient functions. len() lets you find the capacity of the string, and len_trim() lets you find the actual length of the string. In C, where buffer overflow exists, you wouldn't have this "built-in" luxury.

It would be easier and *better* to write this program in Fortran. As stated earlier, C's lack of array bounds checking makes things a lot more annoying. As C lacks the reshape() function, a C Lucifer project would have to use nested loops. This is where bounds checking comes back to bite you in the butt. As Fortran is a mathy language, it is much more suited for stuff like Lucifer, hence the arrays and reshape(). That said, the changes are not that polarizing. These are minor setbacks that would only slightly graze development. This holds especially if the developer prefers a certain language. Fortran would likely result in a faster program, though, due to lack of pointer aliasing.

Given my programming knowledge hitherto, Fortran was not a difficult language to learn. Reengineering the code was difficult due to the lack of structure, but doable. Such difficulties are inherent to every language, not Fortran alone. From a big-picture point of view, C and Fortran are very similar, especially the F95 variant. Despite being from different times, I was able to take advantage of such similarities. That being said, I maintain that Fortran is the superior language for a mathy program such as Sorkin's Lucifer.