This week, I completed my first CIS*3190 assignment. Before I dove deep into the process of re-engineering the code we were provided, I read the document that accompanied the code to better understand the purpose of the program. I read more on the input the program takes, how that input is formatted, and how the output is calculated. Understanding the format of the input allowed me to (1) understand why the code used the algorithm it did, and (2) gave me hints on what certain variables represented as they stored values read from the command line in them.

I grasped a firm understanding of the code when I read the equations in the document provided and compared the values used in those equations with values in the code. Though comments in the original Fortran program were sparse, a handful of them helped me find the code associated with the equations in the document faster, as the code broke the program into components based on what was being calculated (FFMC, DMC, DC, or indices).

Still, one of the hardest parts of the assignment was understanding the flow of the program. The ambiguous variable names made it harder to understand the purpose of many parts of the code. The lack of indentation/spacing which could have clarified which block of code depended on an outer block also made it difficult to understand the algorithm and develop a plan on how to re-engineer it.

After analyzing the original program, I copied the code into a new file with a .f95 extension and began making changes required for the code to run in the newer version of Fortran (F95). I replaced all "C" comment delimiters with "!" to have a program that compiles. I explicitly declared all variables by adding an "implicit none" at the top of my program and giving each variable a type based on what the documentation suggested and how the variables were being used in the code. For example, the PDF provided explained all codes and indices are rounded to the nearest whole number, but all intermediate quantities must be floating point values to ensure precision.

I changed variable names to accurately communicate what each variable was being used for. I then restructured if statements and loops. For both, I first tackled the simpler blocks of code to ensure I had a solid understanding of the simple parts of the code before I worked on changing the more advanced bits (e.g. arithmetic if-statements). Modifying the do loops also helped remove some labels, which reduced how "busy" the code was. I then removed go to statements, because I knew with my newfound knowledge of F95 dos and don'ts, as well as previous programming experience, that go to statements make program flow confusing and can cause unexpected results if not extremely careful.

With every change I made, I recompiled my code and ran it to ensure the output was not changing. I saved previous working versions of my code to ensure working code was not lost in case I needed to undo recent changes. I added comments along the way to ensure thorough documentation.

Next, I cleaned up my code by breaking it into subroutines. My aim was to make each subroutine serve only one purpose. For example, my main "wrapper" program *ffwi* contained a subroutine for reading the names of the input and output files, and another subroutine in my Fortran module *FFWIndices* was responsible for rounding the calculated values before writing them to the user's desired output file.

In the process of learning Fortran to complete this assignment, I came to develop an appreciation for it. Fortran95 does not allow go to statements, which helps programmers avoid writing misleading code. Variable types and purposes of functions are communicated through "intent" statements. I didn't have to worry about how I was going to return multiple values from a subroutine, because I could simply add to a variable's declaration that the intent was to receive a value from the calling subroutine and return an updated value through intent(inout). These are all features that make Fortran a good language.

Given my knowledge of programming, Fortran was relatively easy to learn. Some aspects of the language, however, work differently than I would expect them to as a more experienced C-programmer. Fortran array declarations look similar to function calls in C. real :: a(100) in Fortran means declaring a real array consisting of 100 elements, while my experience with C leads me to think I am looking at a function call to a function named a, which is being passed the value 100. I appreciate that I have the option to declare variables more explicitly as arrays though, using dimension. Fortran also indexes from 1, unlike C, which indexes from 0. Fortran's starting index of 1 made it easier for me to introduce boundary errors, especially when trying to loop through an array's values – this is something I caught myself doing a few times as I worked on the assignment, which also resulted in runtime errors.