CIS 3190
Legacy Fortran Reflection Report

James Singaram
0758172

Jan. 31, 2021

Reengineering William Main's DANGER subroutine into Fortran 95 presented both an intriguing learning opportunity and a chance to peer into the programming design decisions of the late 1960s. Modernizing and tidying up the original subroutine presented multiple challenges that had to be overcome. Despite the initial learning curve, the overall experience would best be described as time consuming as the solutions discovered early on were able to be reused for almost the entire exercise.

The format of the original subroutine leads to a very slow, dense reading experience. It seems to have been designed in a single, additive process with no thought given to future use or changes. Very early into our review process we decided that it would be expedient to redesign the subroutine from scratch rather than struggle to manipulate the existing code into something comprehensible. Divining the functionality of the original subroutine required several hours and multiple sheets of paper in order to fully illustrate its branching structure. While background documentation provided insight into some of the equations used in the subroutine, others, such as the drying factor, were undocumented and needed to be retrieved from the provided code.

In order to enhance readability and make future modifications easier,we decided to break DANGER into several subroutines and functions. As input and output of variables was necessary to testing the program, these were designed and implemented in a similar fashion to ensure that it was possible. Data input relies on calling a subroutine GET_INPUT, which in turn calls smaller functions to prompt and collect each required variable from the user. The updated version of DANGER runs similarly, with subroutines being used to perform each block of calculations. With the exception of some lines containing exact formulas none of the original code was used in the updated version.

The largest challenge associated with relying on nested subroutines was the names used for variables. The program behaved strangely when using the same variable names as elsewhere in the program, so the variables in subroutines and functions often relied on prefixes to distinguish them from their other versions. The program contained a real number called dry to store the dry bulb temperature. It was renamed DRY when passed into DANGER, and then renamed again into FFM_DRY when passed into the CALCULATE_FFM subroutine. In comparison to functions in C and methods in Java, this need to rename variables heavily obfuscates the purpose and functionality of code in Fortran.

The formatting used in Fortran slowed reading, review, and bug catching compared to other languages. Use of full capital letters makes the code appear contiguous and hinders legibility. The short line size (starting at column 7 and ending before column 80) breaks a lengthy line of code into multiple lines and restricts the user's ability to differentiate portions of code by indentation. Fortran compensates for this by using incredibly simple statements and syntax. C's *printf("text %d", x);* is significantly more error-prone than Fortran's simple *WRITE (*,*) 'text ', x*. The difference is even more pronounced when receiving input from the user. Despite Fortran's apparent simplicity, DANGER's lack of depth makes it possible that many solutions used in assignment were poor-practice shortcuts that would need to be avoided when performing more complex work. Without a more detailed task it is difficult for a novice to see the shortcomings of their approach.

Learning Fortran presented an engaging puzzle coming from C, Java, and Python. Its layout presented unique challenges that reflect its ancient origins. Simple statements and formatting make proficiency easy to acquire. Unfortunately, the way it manages variables and passes them into functions and subroutines leaves much to be desired. Given a choice, other languages offer superior tools for everyday use.