CIS 3190
Fortran to Ada Reflection Report

James Singaram
0758172

Feb. 27, 2021

Using Ada to reengineer an early Fortran version of hangman offered insight into both the strengths of Ada as a programming language and the challenges that must be overcome in reading and understanding old code. While the solutions used closely resemble those found in the previous assignment, greater effort was required to understand the original program's execution flow.

The original Fortran code appeared to be written with the express purpose of being inscrutable. It contained frequent goto statements and flowed in a way that was nearly impossible to follow. In order to replicate the program it was easier to compile and run the game than to attempt to salvage the original code. A handwritten flowchart was created to follow the game's flow, and then expanded to create a skeleton on which to build the Ada version. Much of the challenge rested on properly structuring nested loops to ensure the game ran in the correct order. Nearly every step in the flowchart would later become either a function or a procedure. It was decided that everything beyond the first character on the user-inputted strings would be truncated and the remaining character would be converted to lowercase in order to eliminate case sensitivity from the game.

Ada possessed several characteristics that made it easier to work with than other languages. Its strong typed design caught many errors that would otherwise have required extensive review to catch. One of Ada's greatest strengths was its simple and self explanatory function names compared to other languages. Reviewing code was fast because everything did exactly what would be expected based on common definitions. The 'put("text");' function puts text on the screen. The 'get(text);' function gets text from the user. In comparison to C and Java, loops and if statements are all clearly ended with 'end loop;' or 'end if;' rather than a single } brace that can easily be overlooked. Being able to compare strings using the '=' operator made string comparison far more streamlined, especially in contrast to C's strcmp function that returns an integer rather than a boolean value. Ada's ability to use longer lines of text lends itself to superior readability than Fortran. Indentation can be used liberally without fear of running out of space and the ability to use lower case text makes code far more legible.

That's not to say that Ada was without faults. Functions and procedures required care to ensure that correct values were being input and output (however that may have been a consequence of inexperience). Online communities for Ada were significantly smaller than for other languages, making it challenging to find how other people had resolved compiler errors and logic bugs. The official documentation provided function names, parameter types, and the return type, but rarely explained what the returned value meant. For example, the delete() function for unbounded strings takes in an unbounded string and two positive whole numbers as parameters and returns an unbounded string. Documentation did not explain the significance of the input numbers or what the returned unbounded string was. While past experience made it possible to infer their meanings, it complicates the learning process and can frustrate the inexperienced.

Despite these limitations, I would not hesitate to use Ada in the future if working on projects that already use it. My preference for other languages stems from experience and access to robust documentation, however these would become less of an issue with more practice.