Jonas Bakelaar
0964977
April 8, 2018

Converting PL/I to Ada Flesch Index Reflection Report

When converting the PL/I program to Ada, most of my efforts were involved in observing the algorithm we were provided and not specifically converting the code provided to Ada. This was because a lot of the code was very difficult to decipher do to the old age and clearly different syntax of PL/I. The language in itself is extremely different from any other language I've ever written in. I found it incredibly confusing to try to take the code we were provided in PL/I and attempt to directly convert it to Ada.

While I didn't necessarily convert the direct code provided in PL/I to Ada, I did use a lot of the code provided as an inspiration for how I would attempt to finish the assignment and generate accurate numbers. The provided documentation gave me an excellent algorithm to use for counting the number of sentences, words and syllables in a passage of text. I used this algorithm, in conjunction with periodically observing how the old PL/I code was written, to determine how to count the specific numbers required to generate the Flesch Index and the Flesch Kincaid Grade Level.

PL/I is a difficult language to grasp, especially without being able to write in it/compile it. While there were many different resources that helped me decipher some of the PL/I code, I felt the best approach to reengineering this code was to take the essential bits (e.g. the algorithm) and rewrite the program in Ada, using more modern approaches to programming such as modularization in the form of subprograms and functions. This helped me in understanding how each part of the original PL/I program worked with the other parts, as I was able to split up each individual calculation (e.g. the word, syllable and sentence counters, along with the functions/subprograms to determine the Flesch Index and Flesch Kincaid Grade Level).

The main thing I changed about the old code was the process for how everything was calculated. From what I gathered, the old PL/I program did each calculation in one foul swoop. I felt this was inefficient and confusing, as calculations could be grouped much better to promote better modularization and more efficient code. For example, instead of dealing with ending "es", "ed" and "e" in it's own section of the code, I simply added that functionality to the syllable counting subprogram by applying a base syllable count of whatever the word count of the program is and initiating a flag at -1. Whenever the program found a vowel, and the vowel didn't have cases that made it invalid (e.g. the ending "ed", "es", and "e"), the vowel flag/counter would increase by 1. If the vowel counter was greater than 0, I would add the vowel counter to the base syllable number. This led to very accurate results for syllable counts, while grouping this functionality where it was meant to be (with the syllable counting subprogram, not in it's own section of the code).

Ultimately, I changed the code quite a lot compared to the old PL/I code. However, I feel that the way I completed the task was more efficient and made more sense when writing in a more "modern" language as it used the tools provided more effectively (subprograms, better use of variable definitions, etc.)

Q1: Knowing nothing about a “dead language”, how easy was it to decipher what the code did?
A1: In general, it wasn’t that bad. Most of the underlying algorithm was provided in the documentation, so the code itself was pretty much just a reference. However, I did ultimately decide to simply scrap a lot of the PL/I code provided as I did not like how some of the grouping of functionality and modularization was handled.

There were a lot of resources to help me attempt to decipher the code. With that, I felt it wasn’t too difficult to determine what the code did without using these resources as the comments were fairly helpful in determining what each section of the code was for. However, because this was a brand new and foreign language to me, I didn’t understand what most of the code was used for (e.g. the starting 20 lines or so where variables were seemingly initialized, some of the functions used, etc.). A lot of the code was fairly self explanatory as to what it did, however (e.g. looping, if statements, etc.).

PL/I is unique in how it does some accessing of different parts of strings, defining variables, and other basic concepts like this. It was slightly difficult to get my head around how some of this worked, but after determining the algorithm for the functionality of the program and writing the program out in Ada, I feel I have a better understanding of the language and would be able to decipher it more effectively in the future.

Q2: Were there any features of Ada which you found interesting from the perspective of writing code? E.g. the way it handled files?
A2: A lot of Ada is interesting to me. The way it handles basic coding concepts is a lot different than how other languages that I am more used to, like C, PHP and Javascript. For example, variables can only be defined in specific sections of a program. This tripped me up a lot in Assignment 2, however after working with Ada before I felt it wasn’t as hard to figure out in my second try with the language.

Other prime examples of how Ada is interesting in terms of writing code is how it handles strings. If I were honest, I’d say I despise how Ada handles string manipulations and the hurdles one must go through to figure out how to manipulate strings and work with strings in the language. The fact that there is a difference between unbounded and bounded strings, and that bounded strings are the basic form of strings in Ada, confuses me quite a lot and led to a lot of frustration when writing my program. I feel like any language should have just one set standard for dealing with strings, where strings can be dynamic and accessing specific elements of a string is easy to do. However in languages like Ada and C, there is a defined difference between dynamic strings and bounded strings, and it’s incredibly frustrating to work with, especially when trying to make the most efficient program possible and when working with memory.

Overall Ada is an incredibly interesting language. It’s not too unpleasant after working with it for 2 assignments and writing almost 1000 lines of functional code in the language. However, it is definitely not my go to language and I don’t think I would enjoy writing in this language every single day, especially due to its little intricacies that no other language I’ve ever worked with possesses. It’s at least more pleasant to work with than COBOL though!