Reflection Report A3

For the third assignment, we had to refactor code that calculated the Babylonian square root in Cobol. This method of calculating square roots is significant from a historical perspective. This method of calculating square roots is perhaps the first algorithm ever created. The use of calculating a square root shows up in all aspects of mathematics, some of which helped shape the way we understand the world as it is. This method of calculating square roots uses a recursive method. We begin with the number X, which is an overestimate of the root, R. Since X is an overestimate, and S/X will be an underestimate of the root, we can assume the average of the two is closer to the actual root. Eventually, as the number of iterations increases the accuracy of the root will reach the value of the root. This is also known as "divide and average".

In order to refactor this old Cobol code to be compliant with Task 1, I performed the following steps to make sure any legacy structures were removed.

1. Convert all uppercase letters into lowercase to improve readability.
2. Refactor all if statements to use end-if. Also change the operands of all if statements (eg if temp is less than zero -> if temp < 0)
3. Refactor all while loops to have end-procedure at the end.
4. Remove all goto statements by restructuring the program
5. Remove anything to do with reading information from a file and all other file related structures
6. Add basic user input gathering that only runs once, then and modify program to use new user input variable
7. Put all user input into a while loop so that the user can calculate as many roots as they want
8. Update calculation "function" to set a variable when it has found the root, then print it out if it didn't reach the max number of iterations.
9. For Task 2, I put the calculation "function" in a separate file, created values to be passed into that file then created the linkage section.

Before starting the process of refactoring the program, I had to learn how Cobol functions. I found that I had a difficult time picking up the nuances of Cobol primarily because of the syntax. Cobol's syntax is a business-oriented language, and as someone who is accustomed to C like syntaxes it was very difficult to understand how the program worked. Understanding what each line of code did was difficult for me. Although the general program structure was like Ada, the actual language was much harder to read.

While refactoring the program, aside from the syntax the main thing that I found made Cobol challenging to program in was the extensive use of goto statements. This made the flow of the program very difficult to follow, and it seemed like using these goto statements were encouraged at the time. I also found simply storing data in variables was tedious, as rather than being able to

Although I generally disliked working in Cobol, there were a few things that I thought made Cobol a decent language. I liked how Cobol handled external functions. I found it much more intuitive and easier to grasp than in Fortran. I also liked how each part of the program is split up into different divisions. This felt like Ada in a way, and I found once I understood what each division did, it made the program more organized. However, my experience with Cobol was mostly negative. The primary thing that I did not enjoy about Cobol was the syntax. As explained above, it felt more verbose than Java and it was hard to organize properly. I also I disliked how Cobol stored variables. By explicitly declaring the number of digits an integer needs to be, it severely restricts the programmer and can introduce many unnecessary bugs.