CIS*3190 - A3 Roman Numeral Converter
Name: Alex Lai
Student Number: 0920158
Email: alai02@uoguelph.ca

COBOL - Reflection Report

This is a report about my process of learning COBOL and re-engineering the roman numeral converting legacy program. The program now reads either a file full of roman numerals or allows the user to enter a roman numeral using the command line. The program consists of 2 files, one is the main program and the other is a specific subroutine for completing the core task of the program. The program was written in an older COBOL 1974 standard and has updated it to a newer 2014 standard.

In order to learn this new language and build the program, I decided to break the process up into smaller steps and design each part one by one to completely re-engineer the program. The first step I took was to read the intro the cobol and get an overview of the basic styling and syntax for the language. Next copied the legacy program code and converted the program to lower case characters. After that I ran the program and developed a better understanding of the control flow of the program. Furthermore, I removed all the read and write statements to standard out to the newer display and accept statements. I was then able to remove the standard input and output declarations. Beyond that, I removed all the go-to like statements and labels that are no longer necessary in the updated COBOL format. I then re-engineered the if and goto block in the conv.cob file to a more modern evalue when block which is a lot cleaner. This was a structural decision as I believe it is a much better way to find the corresponding weight of a letter of each character in the array. After that I changed all the variable names to be more informative and make the code more self documented. Next, I created a menu loop which lets the user choose which input type they wanted to use and save the input into a variable. I then looked into procedures more and began to modularize the code into separate functions. This allowed me to reduce duplicate code when it came to calling functions after the input was already stored in a single string. I also updated the UI to clean up the output and make it my own. Next, I changed the arithmetic operations to use the compute syntax which made the operations more obvious and readable. I continually tested the program throughout this process to prevent adding in new bugs by using the provided test file. Finally, I added in error handling and allowing for both upper and lower case input. I had to research how to check for a valid file input and was able to handle the error and display the proper

message. Beyond that, I added finishing touches like commenting my functions and organizing the file procedures. The result was a fully working, re-engineered COBOL program using better and more up to date functionality.

Overall, the process of completing this assignment was a lot different and more involved compared to the languages Fortran and Ada solely because of the extremely unique syntax. It made the other languages look much more like C in comparison. I found some similarities between the legacy langues too. All of them declared variables in a similar fashion. For example, they all declare variables at the beginning of the block and require you to declare which types for incoming parameters for subroutines. I believe the similarities were a result of the developers of the languages using standards of that time era to help with consistency while developing new languages to solve new tasks.

My knowledge of programming thus far was definitely an asset during the process but there was still a large learning curve to overcome. It helped in the beginning that lines of code were structured like sentences, making them easily readable and clear to understand but when it came to writing new code, it was a bit confusing with some of COBOLs standards. I also found the compiler error messages not to be very useful and the period ending statement structure was hard to get used to because there are certain circumstances where you are not supposed to have periods. After completing the program, I already feel used to the syntax enough to be comfortable with the language, just as I was with the past two assignment. It has reinforced that you can learn just any new language with a bit of determination and practice.

I believe the learning process was quite helpful and this was a very eye opening experience to the world of COBOL and its uniqueness compared to other languages. It seems that COBOL is a very powerful tool for specific problems but I don't believe COBOL was a good choice for an implementation of this program. I can see this being a perfect language for working with large amounts of data, specifically data organized into records. The structure of defining certain records and the members they contain are very well organized and easy to use. For this program I think a better choice of language would be python, it handles file I/O much easier and has a lot more functionality for strings. Loop structures are simpler and you wouldn't need to define as many things. If this a program actually used for an application, I think that it would be a good candidate program to completely rewrite but I see how re-engineering this program provides a lot of insight into the inner workings of the language.