**Matthew Blatz 0843623**

**Cobol Write-up, Legacy Assignment 3**

How To Run My Program

* make all
  + Compiles program with -Wall
* make clean
  + Removes files for rebuild
* ./romanA3\_1 runs file

Limitations & Assumptions

Similar to the original program the roman numeral length is limited to 30 characters long and if an error character is encountered the program will translate the rest ignoring the error character.

Reengineering Synopsis

Steps I Took:

1. Read through the content on Cobol
2. Looked at examples of file reading and string handling
3. Implemented single line input
4. Implemented file reading
5. Modernized the rest of the code removing gotos etc
6. Implemented the ability to use lower case
7. Ran tests to ensure it was working properly
8. Documented

The process I took to re-engineer the program was somewhat trial and error. Due to my limited time spent with Cobol due to other assignments I was going in slightly blind. I began by working on the implementation of the single line input, the first steps was to run through as many examples of Cobol programs as I could to begin to understand the string handling and input operations of Cobol. After reading through these examples I began to work on the implementation of the single line input and file reading. The file reading was the harder of the two as until it was working it was hard to tell what was going wrong, whether it was my handling of the output of the file or whether the file was even properly opening all. It also didn’t help that the file name was dynamic and I had to try several different formats of entry adding another variable of uncertainty into the mix.

The major rework needed to modernize the Cobol code was the removal of the go to statements. These were riddled throughout the logic of the program and many seemed completely unnecessary as there were better ways to complete the program even with legacy Cobol. I began by building a pseudocode structure of the “subprograms” involving the go to statements, after the pseudo code was fleshed out I constructed if statements that were able to complete the original tasks without the use of the goto statements. These changes were relatively easy as I very infrequently code in languages that make use of goto statements.

Cobol Analysis

Surprisingly I found Cobol much easier to learn than the last two languages we used, while the assignment given was much simpler I blew through it with ease in comparison to the others. This shocked me as many students who have taken the course in the past warned me that Cobol would be the most time consuming and frustrating part of the course. However due to my experience with machine code and other simplified programming languages I easily adapted to the structures of Cobol and was able to complete the assignment.

The most difficult parts of the program for me to understand, in particular going in blindly with little to no knowledge of Cobol’s structure, was the contents of the headers. Setting up variables in a completely global environment was a new experience for me that flew in the face of everything I have learned up until this point, getting myself out of the habit of attempting to locally declare variables was one of the larger challenges of this assignment.

I really enjoyed how strings are handled in Cobol, while limiting and making the use of dynamics difficult in comparison to something like Java or Python, the string handling was very straight forward allowing you to easily know what would cause an overflow unlike C. In addition, once you had the string initialized you would not be dealing with segfaults or other simm

I did not like how Cobol handles loops and if statements, their structure and syntax was frustrating to get the hang of to say the least and I found myself constantly making mistakes and having to go back after received multiple errors on compilation. While this would be improved upon with further use of the language, and I began to understand more as well as I proceeded through the program, it was something that was difficult to get the hang of and was in stark contrast to other programming languages I use on a regular basis.