Software and Legacy Systems

A3: Cobol Re-engineering
Roman Numeral Converter
Dominic Lee - 0823091

Task: To re-engineer an older version of a Cobol program that does Roman numeral conversions, in a newer version.

Overview

Upon some research in the Cobol language, the program in the older version of Cobol was successfully translated to the newer version. This included the deletion of certain structures that were dominantly used in the older version, as well as the addition of new commands and coding paradigms that are present in the newer version.

Approach Taken

Upon gaining a good understanding of writing simple programs in Cobol, my translation process followed the general heuristic of looking at each code segment within the program and comparing it to how I would write the same code in the simplest way. The overall logic behind the re-engineered program did not have to deviate much from the initial algorithm in the older program.

New Vs. Old

Some of the structures that were taken out included 'GO TO' statements with their respective labels and the 'filler picture' initializing statements. The 'MOVE TO' statements were replaced with a number of other loop statements in the newer version. Improvements were also made in the scope of each data type, in that the newer version had allowed for new data types, which are more suited for specific tasks.

'GO TO' statements and labels were used numerously in this program, and are generally known to be used extensively in older Cobol programs. This was thought to be dangerous, however, as there could be a typo in the label number, which would result in an illogical loop. These 'Go To' statements were replaced with 'perform…times' control loops. These loops are essentially better suited for specific tasks, and also allow for cleaner looking code, as there would not be a clutter of labels throughout the code. The newer version of Cobol made use of the 'paragraphing' concept, which allowed specific blocks of the program to be controlled by its own loop. The program also made use of 'GO TO' statements alongside 'If' statements at respective labels. These were replaced with 'if…else if" logical statements, which carried out the same logic, which again looked cleaner than having numerous labels.

The 'Filler Picture initializing statement was omitted from the new program. These were replaced with simple 'pic' statements.
The re-engineered program now provided users to add any number of roman numerals to convert, in a linear fashion; rather than a numeral per line.
Users are also given the option to read in their roman numerals to be converted form a file. These can either be in upper case or lower case. This program however, does this properly if there is only one Roman numeral sum per line.

Modernizing vs. Re-implementing

One could be faced with the choice of either taking the path of porting the legacy code to a more modernized version or re-implementing it in a more widely used language today, such as C or Java. I believe as the code becomes more complex, it might be easier to re – implement in a language you are more proficient in or one with a wide variety of resources and optimized libraries. Cobol makes use of paragraphing and modularity, which made the task relatively reasonable to tackle.

Main Obstacles

The main difficulty in this migration was following the logic of scattered 'GO TO' statements. However once this was deciphered, it was fairly simple in restructuring the respective code with simpler constructs such as paragraphing. Cobol is also said to be really good at file handling, though it did take me some time in figuring out the best approach in doing this.

Conclusion

As legacy systems such as 'Cobol' are still the backbone of many complex and critical systems used today, I definitely see the importance of acquiring a good grasp on how the languages are written and how they have been modernized over the years.

Compiling:

On command line, enter 'make'
Then enter ' ./roman'