CIS*3190 Software for Legacy Systems - The Trithemius Cipher

Geofferson Camp (0658817)

March 25, 2016

The general algorithm of my cipher program is quite simple:

1. Request file name of file containing text to encrypt/decrypt from user.
2. Request the user to choose operation mode (encrypt or decrypt).
3. Initialize resources such as files and variables.
4. Read supplied file and create string to encrypt/decrypt.
5. Encrypt or decrypt string.

Where the encrypt/decrypt algorithms are:

1. Count all characters in string.
2. Loop through string.
3. Foreach character of the string, loop through alphabet until a matching character is found.
4. Using the matched location as origin, traverse alphabet where the number of spaces traversed is equal to the index of the string letter matched, looping around to the beginning of the alphabet where necessary.
5. Append the character found via traversal to the output string being constructed.
6. Display output string as the requested encrypted or decrypted string.

Both the encryption and decryption traversal algorithms calculate the remainder of the string character count divided by 26, to ensure a range of 1-26; encryption then adds that value to the letter count of the alphabet loop and the decryption subtracts it from the letter count of the alphabet loop.

I chose to use the alphabet traversal method opposed to incrementing ASCII values as it was easier to implement given what I had already learned about Cobol and what I was already familiar with in other languages. I haven't worked extensively with ASCII values before, and I wasn't ready to mix the unknown of Cobol with the unfamiliarity of ASCII in Cobol. The downside to my chosen approach would be inefficiency, as it requires the use of a nested loop. I felt this wasn't a major issue given the scope of this assignment and that the inner loop can only iterate a maximum of 26 times. There was nothing overly complicated about the algorithm or its implementation. Most of the issues I had designing this program were a side effect of learning Cobol.

The greatest challenges I faced programing in Cobol were overcoming the different syntax (initialization, move, loops, etc) and learning how to open files. Opening a file to read from, in general, is straightforward as shown in Dr. Wirth's documentation. However, dynamically changing the name of the file at run time became an avoidable nuisance early in the development process. This was caused by a lack of understanding of cobol and some bad advice that I found searching the internet. For quite a while, I was under the impression that cobol opened the file specified in the "select….assign to…." statement before the procedural division even started. Which, in hindsight, doesn't make sense as there is a separate command which is used in the procedural division for opening the file. In any case, I was under the impression that the file path could not be changed after the program had started and ended up looking into JCL before finally realising the path could be changed by assigning to a variable and moving a new value to that variable before "open input" was called on the file variable. The other significant issue I faced while working with Cobol was dealing with the syntax in general. There were many times during programming where I questioned why anyone would design a language in a fashion such as Cobol. It seemed that writing even the simplest of Cobol statements would take twice as much typing as in any other language I have used. This lead to many typos and was very frustrating to deal with while debugging as implementing temporary print statements and other testing code took, in my opinion, twice as long as it should have.

Compared to opening the file, reading the contents of it went smoothly. After I learned how to implement the required "while loop" esq perform loop, reading line by line was similar to how I would have accomplished the task in C. The nature of strings in Cobol did result in a minor set back at this stage. Unlike C where the string would be trimmed at the end of the character list, in Cobol, the entire initialized character array presented itself when trying to work with the string. To get around this. I used the "trim" string function. This method worked well for my needs but I could imagine it becoming fairly cumbersome if more string manipulation was needed.

As I mentioned previously, I believe Cobol's biggest pitfall is its syntax. The sort of descriptive commands that Cobol uses or indicative of a language that has limited functionality, as they are very specific and imply they can only be used for a very small set of tasks. This results in a lot of cases of really different ways of accomplishing almost the same thing and therefore adds unnecessary time to the task of becoming comfortable working with the language. The only thing I liked about Cobol was its "pass by reference" way of transferring information between the main program and external modules. This strategy resulted in a refreshing notion of object oriented behaviour and made the language seem a little more progressive.

Given my knowledge of programming, Cobol was not easy to learn. Many times I found myself thinking that what I was trying to do could be done more efficiently or logically, based on what I have learned about programming in the past. I might have a different perspective if it was the first programming language I was learning. Many aspects of the language might seem intuitive to someone who was familiar working with only spoken languages. The most obvious example of this is the syntax, which as I mentioned earlier, is very descriptive. But, as new programmers become familiar with the language, this would undoubtedly become inefficient. Another indication of “newby friendliness” is the way the sections are split up. The segmentation between initialization and procedural sections lends itself to clarity and would no doubt aid a new programer in understanding how programs function. Touching again on the nature of strings in Cobol, this also makes working with the language challenging because tasks that would have been quite simple in C require a significant amount of research or “hacks” to accomplish. An example of this in my cipher program is where I append a “#” to the end of the string. As I was having trouble finding a more correct way to note the end of the string to be encrypted/decrypted I resorted to this method. It works well but the downside is a “#” in the actual text would result in the program not functioning properly.

In summary, I would not want to work in cobol in the future. I didn’t notice any significant benefits it has over other languages i’ve worked with and it’s implementation is counter intuitive to what I am used to.