Assign-01 : Getting to Know the Linux Development Environment

Description

This assignment has you writing an encrypting / decrypting utility for Linux. This utility will take any ASCII file and encrypt it in such a way that its contents are not readable – until they are decrypted by the utility.

Objectives

- Practice creating a software project in Linux in the required development directory structure
- Practice creating a makefile for a Linux software project
- Reinforce C Programming techniques - File I/O using ASCII files, Command-Line Processing

Requirements

1. The utility needs to be called **cryptoMagic** and needs to be written in C
    a. The utility has 2 command-line switches – they are `-encrypt` and `-decrypt`
    b. If none of these switches is specified, then `-encrypt` is assumed
    c. The utility also takes the name of an **ASCII input file** to encrypt/decrypt as an argument.
    d. For example:

```
cryptoMagic -encrypt myFile.txt   ← will encrypt the contents of the myFile.txt file
cryptoMagic myFile.txt            ← will encrypt the contents of the myFile.txt file
cryptoMagic -decrypt myFile.crp   ← will decrypt the contents of the myFile.crp file
```

2. When the utility is asked to `-encrypt` an ASCII file, it will take the input filename and produce the encrypted file with the same base filename and an `.crp` file extension
    a. For example:

```
cryptoMagic -encrypt myFile.txt   ← will produce an encrypted file called myFile.crp
```

3. When the utility is asked to `-decrypt` an encrypted file, it will take the input filename and produce the decrypted file with the same base filename and an `.txt` file extension
    a. For example:

```
cryptoMagic -decrypt myFile.crp   ← will produce a decrypted file called myFile.txt
```

4. It should be noted that the input file can have any file extension. When asked to *encrypt*, you need to replace the existing file extension (if any) with .crp. Similarly when asked to *decrypt*, you need to replace the existing file extension (if any) with .txt
    a. Encrypting always produces a file with a .crp extension, decrypting always produces a file with a .txt extension

b. Also note that you may be asked to encrypt or decrypt a file that has no extension! In this case, you have no existing extension to replace – you only need to append the .txt (if decrypting) and .crp (if encrypting)

5. Each line (up to and including the carriage return (noted as **<CR>** below)) in the unencrypted ASCII file is guaranteed of being 120 characters of less. While processing the input ASCII file you need to process one line at a time. Continue to process the input file until you reach the end of the file.
   a. Please note that it is not guaranteed that each line actually ends in a carriage return … how will you handle that?
6. The encryption scheme is applied to each character in the line:
   a. If the character is a **<tab>** (ASCII value 9) then simply transform it into the output character sequence **TT**.
   b. The **carriage return** characters are **not** to be **encrypted** – they are left as is in the resultant output file (the **<CR>** in the example below is meant for illustration **only** – do not output "<CR>")

      NOTE: The term "carriage return" in this document does not apply to any specific ASCII code. It applies to the typical end-of-line character that exists in TEXT files within the Linux OS. You may want to investigate what constitutes a *carriage return* (i.e. end-of-line character in a TEXT file) in the Linux OS
   c. If is not a tab or a carriage return character, then apply the encryption scheme in steps d through f below
   d. Take the ASCII code for the input character and subtract a value of 16 from it. Let's call this resultant value "outChar"
   e. If the resulting outChar value is less than 32, then another step must be taken: outChar = (outChar – 32) + 144
   f. You need to write the ASCII value of the new encrypted character (i.e. outChar) to the destination file as a 2 digit hexadecimal value. Note that this will effectively double the length of the input line in terms of size …

      For example – if the input (unencrypted) file is:

      ```
      Hello There how are you?<CR>
      My name is Sean Clarke.<tab>I like software!<CR>
      ```

      Then the encrypted file is:

      ```
      38555C5C5F80445855625580585F678051625580695F652F<CR>
      3D69805E515D55805963804355515E80335C51625B558ETT39805C595B5580635F56646751625581<CR>
      ```
7. Each line (up to an including the carriage return (if it exists)) in the encrypted ASCII file is guaranteed of being less 255 characters or less. Remember - while processing the input ASCII file you need to process one line at a time.
8. The decryption scheme is applied to each pair of characters in the input line:
   a. You need to see if the pair of characters you are processing is the sequence **TT** – if so, then simply transform this pair of characters into a **<tab>** character (ASCII value 9) in the output file.

b. If the pair of characters is not the sequence TT, then translate the first character of the pair by multiplying its *face value* by 16. Remember that hex values of A through F take on the *face values* of 10 through 15. Then add the *face value* of the second character in the pair. Let's call the resulting value "outChar". For example:
   - Reading the pair of characters "38" from the encrypted file will translate into an outChar value of 56 decimal.
   - Reading the pair of characters "5C" from the encrypted file will translate into an outChar value of 92 decimal.

c. Now you need to add 16 to outChar.

d. If the resulting outChar value is greater than 127, then another step must be taken: outChar = (outChar - 144) + 32

e. The outChar value now contains the decrypted ASCII code for the character that you have just decoded. So take this decrypted character value (i.e. outChar) and write it to the destination file as a character.

f. The **carriage return** characters are **not** to be **decrypted** – they are left as is in the resultant file.

For example – if the input (encrypted) file is:

```
4458596380555E5362696064595F5E80635358555D55805963806062556464698067555962548E<CR>
39635E87648059642F812F<CR>
```

Then the decrypted file is:

```
This encryption scheme is pretty weird. <CR>
Isn't it?!? <CR>
```

9. It is expected that your cryptoMagic utility has been tested (perhaps by using the above examples) as well as with any other encrypted / decrypted examples you can think of. Don't forget about the extreme / boundary test cases!
10. It is expected that your cryptoMagic utility has been designed using modular techniques (i.e. the program has been written using functions that you've created).
    a. The solution must have at least 2 source files and 1 include file
    b. There should be no debugging messages present in your final submitted utility
11. Your solution structure must include a makefile and also follow the recommended Linux development directory structure as outlined in the Linux-Development-Project-Code-Structure document within eConestoga.

Hand in

1. Please clean and hand in your cryptoMagic solution (entire development directory structure) – tar'd up into a file titled `lastName-firstInitial.tar` (e.g. John Smith would submit `smith-j.tar`)
   - Ensure that you follow the recommended Linux Development Directory Structure guidelines
2. Make sure you comment your source appropriately, especially the file and function header comments