Assignment 7

Text file I/O, iterations, pointers, and functions

Purpose of the exercise

This exercise will help you do the following:

1. Develop familiarity with basic input and output of data through text files.
2. Develops skills in C programming with iteration statements and functions.

Overview

In cryptography, the Caesar's cipher is a basic encryption technique that for each letter of input that belongs to some alphabet produces an output letter that belongs to the same alphabet but with its position shifted by a known, fixed number of positions. This cryptographic method belongs to a family of *symmetric cryptographic algorithms*: the same **key** - the number of position shifts - is used to encrypt plain text into a cipher and to decrypt a cipher into plain text.

The Vigenère cipher improves upon Caesar's cipher by encrypting messages using a sequence of keys which can be represented as a **keyword**. While a Caesar's cipher adds a fixed value - the key - to each input character during encryption, and requires subtraction of the same value during decryption, a Vigenère cipher's adds the first fixed value - the first letter of a keyword - to the first input character, the second fixed value - the second letter of a keyword - to the second character, and so on, and requires subtraction of the same corresponding values during decryption.

If we wanted to say `"HELLO"` (ASCII character set) to someone confidentially using a Vigenère cipher with a keyword `"01"`, the encryption process would look as shown below:

Input text	**'H'**	**'E'**	**'L'**	**'L'**	**'O'**
Input ASCII code	72	69	76	76	79
Key text	**'0'**	**'1'**	**'0'**	**'1'**	**'0'**
Key ASCII code	48	49	48	49	48
Cipher text	x	v	|	}	*[DEL]*
Cipher ASCII code	=72+48=120	118	124	125	127

As you have known, the basic ASCII character set includes codes 0 through 127. If the ciphered code goes outside of this range, it should **wrap around**. For example, if we add characters with a code 1 to a character 127, the result should be a character with the code 0; if we subtract, the result should be 127 again.

In this assignment, you must write a program that:

1. **Encrypts** a plain text input file by writing the ciphered text into a ciphered text output file.
2. **Decrypts** a ciphered text input file into a deciphered text output file.
3. **Counts** the number of words in the deciphered text file; words in the text are delimited by *whitespace characters* (including a space, a new line `\n`, a carriage return `\r`, a tab `\t`); any punctuation is considered a part of a word.

A single executable should be able to perform **either** the first or the last two tasks depending on how it has been compiled. If it has been compiled with the macro definition `ENCRYPT` provided from the command-line only the first functionality should be provided.

Task

1. Download the assignment source code files:

   Go to the course page in *Moodle* - DigiPen (Singapore) online learning management system and download a zipped archive that contains source code files and extract them into a Microsoft Windows directory *c:\sandbox*. You may adjust the path as suitable for your system. The included files are:

   - *main.c*
   - *q.h*
   - *plain.txt*
   - *expected-plain.txt*
   - *expected-cipher.txt*
   - *expected-output.txt*
2. Using the Microsoft Windows command prompt navigate to the *sandbox* folder. Then type `wsl` to open Linux bash in the same current directory.
3. From Linux open *q.h* in Microsoft Visual Studio Code:

   ```
   code q.h
   ```

   Study declarations of functions `encrypt()` and `decrypt()` and related type declarations. Then create a file `q.c` and implement both function definitions following the Vigenère cipher's algorithm presented above. Both functions can be defined with a single statement.
4. From Linux open *main.c* and investigate the skeleton of the code provided there. You will find the following preprocessor directives:

   ```
   #ifdef ENCRYPT

   // TODO: encrypt plain.txt into cipher.txt

   #else

   // TODO: decrypt cipher.txt into out_plain.txt
   // TODO: write count of words into stderr

   #endif
   ```

   The meaning of the above code is the following: if a macro `ENCRYPT` is defined during compilation, only the code for encrypting is included in the translation unit; otherwise, only the code for decrypting and counting words is included in the translation unit.

   You may notice that this macro has not been defined anywhere is the code; do **not** include it in any of the files. Instead, you will use an addition compilation option `-DENCRYPT` (`-D` followed by a name of a preprocessor macro you want to define; in Microsoft Visual C++ compiler use `/D`).
5. Think about the solution and implement the function `main()`.

   1. Use file I/O related functions to open files, fopen(), read and write data using fgets() or fgetc(), and fputs() or fputc() for line-by-line or character-by-character operations, and

do not forget to close files with fclose().
2. Use your `encrypt()` and `decrypt()` functions to convert read characters before writing them into an output file. Read characters, convert them, write and repeat the process; do not read the entire input file first before converting characters and writing them into an output file - assume that the input file can have an arbitrarily large size.
3. Write the word count during decryption to the standard errors stream stderr with fprintf().

6. Compile and run both versions (first for encryption, then for decryption) of an executable file in the following way:

```
gcc -Wall -Werror -Wextra -Wconversion -Wstrict-prototypes -pedantic-errors -std=c11 -c -o q.o q.c

gcc -Wall -Werror -Wextra -Wconversion -Wstrict-prototypes -pedantic-errors -std=c11 -DENCRYPT -c -o main.o main.c
gcc q.o main.o -o encrypter
./encrypter 1> actual-output.txt 2>> actual-output.txt

gcc -Wall -Werror -Wextra -Wconversion -Wstrict-prototypes -pedantic-errors -std=c11 -c -o main.o main.c
gcc q.o main.o -o decrypter
./decrypter 1>> actual-output.txt 2>> actual-output.txt

cat out_plain.txt >> actual-output.txt
```

The first executable, *encrypter*, should produce a file identical with *expected-cipher.txt*. The second executable, *decrypter*, should produce a file identical with *expected-plain.txt*, and write a message to the `stderr` stream. Combined output is written into *actual-output.txt* and it can be compared with *expected-output.txt*.

7. Use *diff* to compare all three files:

```
diff --strip-trailing-cr expected-plain.txt plain.txt
diff --strip-trailing-cr expected-plain.txt out_plain.txt
diff --strip-trailing-cr expected-cipher.txt cipher.txt
diff --strip-trailing-cr expected-output.txt actual-output.txt
```

Make sure that the output matches **exactly**.

Submitting the deliverables

You have to upload a complete files *q.c* and *main.c* to *Moodle* - DigiPen (Singapore) online learning management system, where the file will be automatically evaluated.