CSC 150 Program #3 – Simple Cipher

Due December 8

Assignment Overview:

Your third program will encrypt and decrypt secret messages using files.

Background:

Security and secrecy are serious concerns and popular research areas in computer science. While people enjoy the convenience of data communication, storage and computation, they also struggle with keeping their important information secret from unintended disclosure. In this assignment, you will be implementing a simple cipher to facilitate covert communication.

A typical cryptographic tool consists of both encryption and decryption functions. In order to prevent cryptanalysis attacks on the cipher, the mapping between plaintext and ciphertext needs to be very complex and change unpredictably. This can be done by either designing complicated algorithms or using a relatively simple algorithm with a secret key. Most cryptographic systems adopt the second option, since it keeps the encryption algorithm simple and stable. It is also possible to change the key text periodically for added security. Changing the key text will alter the mapping between the plaintext and the ciphertext without changing the algorithm.

For encryption, the user needs to provide the secret message in plaintext form and a cryptographic key which is another piece of text, and then the program will generate the ciphertext corresponding to the plaintext. For decryption the user needs to provide the ciphertext of the secret message and the same key used in encryption, and then the program will generate the deciphered plaintext that is the same as the original message.

A simple cipher can be achieved be replacing each character of the secret message by a code or number, which is the sequence number of an occurrence of that character in the key text.

For example, if the secret message is "**run**" and the key text contains all letters in alphabetical order as "**abcdefghijklmnopqrstuvwxyz** ", the ciphertext will be "**18 21 14**", where "**r**", "**u**", "**n**" are the 18th, 21st, and 14th characters in the key. If the key text is"**zyxwvutsrqponmlkjihgfedcba "**, then the same message will be encrypted as **"9 6 13"**. The key can be any text, not just the normal alphabet string, but you have to make sure the key contains all characters you want to encrypt. Decryption is achieved by finding the corresponding characters from the key text using the given sequence of numbers. You can change the key text from time to time which will cause the ciphertext of the same message to vary, but you need to make sure the key text contains all the characters that exist in the plaintext at least one time.

Problem Statement

Your program will be a fully functional cryptographic tool implementing the simple cipher algorithm explained above with encryption and decryption functionalities. However, different from programs 1 and 2 you did before, program 3 takes information from command-line arguments, so that it doesn't need any user-program interaction in the console during runtime.

Program Specifications:

1) Your program will need four command-line arguments in the following order:
    a) Mode of operation: **–e** for encryption or **–d** for decryption
    b) Input file: it contains plaintext if doing encryption, or ciphertext if doing decryption
    c) Key file: it contains the key text.
    d) Output file: it contains the encrypted ciphertext if doing encryption, or decrypted plaintext if doing decryption.
2) If –e is used, the program encrypts the contents of the input file using the key text in the key file, and saves the ciphertext to the output file.
3) If –d is used, the program decrypts the contents of the input file using the key text in the key file, and saves the plaintext to the output file.
4) The encryptable characters used in the plaintext message and key text are limited to 26 lowercase English letter and whitespace (excludes tab and newline), so that there is a total of 27 encryptable characters.
5) Error checking 1: The program should check if enough command-line arguments are provided. If not, print an error message and terminates the execution immediately.
6) Error checking 2: The program should check if mode of operation is one of "**–e**" and "**–d**". If other value entered, print an error message and terminates the execution immediately.
7) Error checking 3: If the program can't open any file, print an error message and terminates the execution immediately.
8) Error checking 4: The program should check if the key text contains all 27 required characters. If not, print an error message and terminates the execution immediately.
9) **Extra credit (5 points):** Encryption is less secure if the encryption algorithm always uses the first occurrence of each character in the key text. In order to address this, you can randomly select any occurrence of that character for use. This change will significantly enhance the robustness of your cipher against frequency analysis.

Implementation Requirements

1) You are guaranteed that neither the plaintext message file nor the key text file exceeds 100 characters per line, or exceeds 100 lines. (This is important for deciding the array size in the program). The maximum size of any number in the ciphertext will not exceed 10,000.
2) The output ciphertext should use whitespaces as separators to delimit numbers. No other formatting is required.
3) The output decrypted plaintext will only contain the 27 characters mentioned above. No other formatting is required.
4) Your program must be decomposed into the C++ functions whose prototypes appear in header.h. To use header.h in your program:
   a) After you have created your project and program files, right-click on "Header Files" within Solution Explorer, choose Add New Item, Header File, and name it header.h.
   b) Copy and paste the contents of the header.h file in the Program 3 folder on the website into this file.
   c) Go back to your program file and add the following line near the top of your file, with your other #include statements:

      #include "header.h"

   d) Your program will use arrays to store the plaintext, ciphertext (actually numbers) and key text. Since array sizes must be constant, you may use the included constant MAXSIZE and CHARSETSIZE for your array sizes.
   e) Since the function prototypes are already in header.h, you do not need to have them again in your program file.
   f) Programs will be graded using the provided header.h file. Do not change any part of the header.h file, unless you have already submitted your completed program and wish to add functionality.
5) (This is needed if you are pursuing extra credits.)
   To generate random numbers, you to put the following statement at the beginning of the main function to set the seed.

```
srand ((unsigned)time(NULL));
```

   At every place you need a random number, call **`rand()`** function**.**

   Example:

   For a random number between 0 and 49: r = rand() % 50;
   For a random number between 0 and 100: r = rand() % 101;
   For a random number between 1 and 100: r = rand() % 100 + 1;

6) You must use DoxyGen to document your code and format it according to the CSC150 coding standard. Be sure your code file is readable and neat. Do not allow lines to extend past 80 characters, use appropriate white space and make sure to use a consistent and attractive indentation scheme. The detailed coding standard is described in http://www.mcs.sdsmt.edu/csc150/Course/Coding_Style.pdf

7) Name your code file **prog3.cpp**. Points will be deducted if this naming convention is not followed.

8) Your program must correctly compile in Visual C++ 2012.

Timeline:

Nov 14: *readText* and *readNumber* functions completed.

Nov 21: *findLetter* function completed.

Nov 28: *encryption* and *decryption* functions completed.

Dec 5: All program should be completed, DoxyGen comments filled in.

Dec 8: Your program is due by midnight. Submit it following the instructions below. It should be fully commented and follow the coding standards that are posted on the class website

Comments and Suggestions:

- Do not delay, Start writing the program early. If you wait until the night before the due date, you will have a miserable night.
- Draw a flowchart that shows which functions will be called throughout the program, the order in which they will be called, and what pieces of information are passed between each function.
- Write detailed pseudocode for each of the individual functions, including main. Include a table of all variables used in each function and what those variables represent.
- Translate your pseudocode to C++ code and test your program.
- Use debugging cout statements and the Visual Studio debugger to find and fix issues. Refer to pre-lab 5 and book section 5.5 for reminders on how to do this.

Program Submission:

Submit your program code (the **prog3.cpp** file only) at http://www.mcs.sdsmt.edu/submit before midnight of the due date. (Your file gets time stamped, so late submissions will be noted and may be given a late penalty!) Be sure to submit to the correct lecture section!

DO YOUR OWN WORK.

Sample Output:

D:\prog3.exe -e p1.txt key1.txt c1.txt (Success. No console output)

D:\prog3.exe -d c1.txt key1.txt o1.txt (Success. No console output)

D:\prog3.exe (Error. Insufficient command-line augments)
Incorrect number of command-line arguments.

D:\prog3.exe *-x* c1.txt key1.txt o1.txt (Error. Incorrect mode)
Wrong mode entered. Enter -e for encryption or -d for decryption.

D:\prog3.exe -e *missing.txt* key1.txt c1.txt (Error. Incorrect file name or location)
Can't open input file.

D:\prog3.exe -e p1.txt key1.txt c1.txt (Error. Key0.txt doesn't have all 27 encryptable characters)
The key file doesn't have all 27 encryptable characters.

Need more test samples? Use the demo program on the course website to verify your solution.

1. Download http://www.mcs.sdsmt.edu/csc150/Assignments/Programs/Program%203/prog3.exe
2. Go to the parent directory of the directory that contains the downloaded **prog3.exe** file.
3. Hold **Shift** key and right click the icon of the directory with **prog3.exe,** then select **"Open command window here"** from the pop-up menu.
4. Type "**prog3.exe –e text.txt key.txt cipher.txt"** in the console and hit **Enter** key to execute the demo program. "**text.txt"**, as input file, contains secret message in plaintext. "**key.txt"**, as input file, contains key text. "**cipher.txt"**, as output file, contains corresponding ciphertext of the secret message.
5. Type "**prog3.exe –d cipher.txt key.txt deciphered.txt"** in the console and hit **Enter** key to execute the demo program. "**cipher.txt"**, as input file, contains secret message in ciphertext. "**key.txt"**, as input file, contains key text. "**deciphered.txt"**, as output file, contains recovered plaintext of the secret message.
6. Then, you can compare the contents of **text.txt** and **deciphered.txt** to confirm the correct functionality of the program. You can also compare the output files pf your program and the demo program to exam either encryption or decryption functionality.
7. Along with the demo program, two key files and two input files are provided for testing. You are welcome to test the program using your own key and input files.