CIS241 Project One

Automatic Caesar Cipher Breaker

(Due Oct. 11, 2019)

Caesar ciphers are a basic encryption technique in which each letter in the original text is replaced by a letter some fixed number of positions down the alphabet. Here, the fixed number of positions is referred to as the key for the cipher. For example, if the key is 4, then the mapping between the original and cipher alphabets will be as follows:

```
Original: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Cipher:   E F G H I J K L M N O P Q R S T U V W X Y Z A B C D
```

Caesar ciphers can be automatically broken by using the known letter frequencies of the language of the encrypted text. The frequencies for English as given on Wikipedia are available in the file *letFreq.txt* that can be found from the instructor's website.

In this project, you will write a program that uses this data to crack the code for arbitrary Caesar ciphers. The input for your program is an encrypted file produced by using the Caesar Cipher with a key between 0 and 25. Note that in the file, only the letters are changed; all other characters remain intact, and capitalization is preserved.

You need to define the following functions and use them in your program. Each function represents a step in the procedure that solves this problem.

```
// Load array given with the letter frequencies for English from file letFreq.txt
void readFreq ( float given[] ,    FILE * letFreq )

// Read the encoded text from an input file and accumulate the letter frequency
// data for the encoded text. Store the frequency data in array found.
void calcFreq ( float found[] ,    FILE * datafile )

// Compare the data in array found with the frequency data in array given, looking
// for a key that will give you the best match. To do this, try each of the 26 rotations,
// and remember which gives the smallest difference between the frequencies you
// observed and the frequencies given. Return the key.
int findKey ( float given[], float found[] )

// Decrypt the encoded text in the input file using the key and write the decoded text
// in the output file
void decrypt ( int key ,    FILE * datafile, FILE* outfile )
```

One question here is how to compare two lists of letter frequencies to find the best match. You can measure the difference between the two lists; for this, use the sum of the squares of the differences of the corresponding elements in the lists. If you minimize this sum by trying different rotations, you are doing what is called a least squares fit. Note that each letter frequency is the quotient of the number of times that letter was seen divided by the total number of letters examined.

Since each function above represents a step for the program to do its work, you can develop your program in an incremental fashion; that is, define one function, test it, and if it works correctly, move on to the next function. A minor error may produce a quite different result for a program like this. Incremental development is an effective way to ensure the correctness of a program.

In addition to the file *letFreq.txt*, you can find the program *cipher.c* from the instructor's website, which is provided for you to prepare your test data. You should use it to generate an encrypted file from a file that you create for testing purposes, run your program with the encrypted file as input, and then compare the output file of your program to the original file. Since such a process involves multiple command lines and you have to run them as often as you need, it would be much convenient to define a *makefile* and use command *make* to automate your testing process. A *makefile* is attached, which can be used to build an executable for cipher.c and run it against data file data.txt (it is not provided so you need to create it).

When your program is done, test it twice, one with a file that contains a small paragraph and the other with a file that contains more than a page of text. Use Linux command *diff –s originalTextFile resultingTextFile* to see if they are identical. Also add this command line to your *makefile* so as to perform your test and compare the result automatically.

When your program works correctly, make a Word or PDF file with your source code, makefile, and two sets of test data as well as program output (including the original file, the encrypted file, and the output file of your program) and submit via Blackboard.