COSC 1437 Graded Exercise #4 Due 11:45PM, Sunday, November 3, 2013

Internet References:
Information on Vigenere cipher: http://en.wikipedia.org/wiki/Vigenere_cipher

Objective: Write a program that can either encrypt or decrypt the contents of a text file containing upper and lower-case alphabetical characters, other displayable ASCII characters, and whitespace characters. The encryption and decryption algorithm to use is the Vigenere cipher. Only alphabetical characters will be encrypted or decrypted. Other characters will be passed from the input file to the output file unchanged.

If a key is used to encrypt a text file into a cipher text file and then that cipher text file is decrypted using the same key, we should get back the original file with alphabetical case preserved. Note: When the same key is used for encryption and decryption, we call this symmetric encryption.

The windows fc command (file compare) is useful for comparing to see if two files have the same content.

The program will be run using command line arguments as shown in the examples below: The following are just a few examples of how the program can be run.

```
GE4  –e  –p  PlainText.txt –c  CipherText.txt –k password.txt
GE4  -d  - p  PlainText.txt –c  CipherText.txt –k password.txt
GE4  -e
GE4  -d
```

The – e argument indicates the program should encrypt the file indicated by the –p argument and save the cipher text in a file specified by the –c argument.

The – d argument indicates the program should decrypt the file indicated by the –c argument and save the plain text in a file specified by the –p argument.

The –k argument specifies the name of the file containing the password.

The e, d, p, c, and k arguments can occur in any order on the command line. Exactly one of the arguments e or d must occur on the command line. All other arguments are optional. If either the –c or –p arguments is not specified, the plaintext file is assumed to be plaintext.txt and the cipher text file is assumed to be ciphertext.txt. If the –k argument is not specified, it is assumed that a file named password.txt will be used. All default files are assumed to be in the current directory.

If an invalid command line is entered, the program should abort after displaying a helpful message to the user. Also the program should display help if the program is executed without any command line arguments, invalid command line arguments, or if files cannot be opened.

Implementation Details:

The program should use a two-dimensional 26 by 26 array of char to store the Vigenere table. See the reference above for the contents of this array. This array can be populated using a nested loop. Or you can initialize it using hard coding if you don't mind typing in 676 values. It turns out that the array is actually not necessary to do the encryption and decryption, but you must use the array for at least one of these operations. You must use the array in order to get practice using two-dimensional arrays in a program.

The maximum key size is 1024 characters. The key is stored in a file. The key must be all upper case alphabetical characters, just like the characters in the Vigenere array.No other characters are allowed. If the key file contains more than 1024 characters, ignore all characters after the 1024th character.

Use an enumerated data type to give names to potential sources of error:

```
enum errorSource {keyFile, plainTextFile, cipherTextFile, badCommandLine, noError};
```

This will help you to display a meaningful error message to the user.

Use the following structures to encapsulate related data:

The following structure encapsulates data parsed from the command line (or contains default values). This structure contains information that will be useful in sending file names to a file open function and for reporting errors generated by parsing the command line.

```
struct CommandLineInfo
{
  errorSource error; // holds error code of first error encountered or noError
  string keyFile;
  string plainTextFile;
  string cipherTextFile;
  char mode; // e = encrypt or  d = decrypt
};
```

The following structure encapsulates data about open files and encrypt/decrypt mode. This structure contains information will be useful in reporting errors and for storing file object information.

```
struct OpenFileInfo
{
  errorSource error; // holds error code of first error encountered or noError
  char mode; // encrypt or decrypt
  ifstream keyFile;
  ifstream inFile;   // could be plain text file or cipher text file
  ofstream outFile; // could be plain text file or cipher text file
};
```