CS515 Fall 2014

Assignment 2

Due on Monday, September 22

The purpose of this assignment is to give you practice working with C++ arrays, strings and basic ascii file input/ouput.

Download all starter files from `~cs515/public/2P`

A Vigenere cipher encrypts a message using a keyword, or password. The letters of the key each represent an amount to add (encrypting) or subtract (decrypting) from the input text: a=0, b=1, c=2, ... y=24, z=25. The numeric value of the first letter of the key is used to encrypt (or decrypt) the first letter of the text, the second for the second, and so on. When the end of the key is reached, the first letter is reused, and the pattern repeats cyclically, with the (*i* mod *N*)th letter of the key being used to encrypt the *i*th letter of the text (where *N* is the length of the key).

Part 1 (40%)

Implement a Vigenere cipher to encrypt and decrypt texts using a known key. Your program will be invoked with 4 command line arguments in the following order. (If the number of arguments is not 4, it is an error.)

1. Encryption or decryption mode: **e** is for encryption, and **d** is for decryption. Any other value is an error.
2. The key. This should be all lower-case alphabetic letters and the key length should be less than 10. If it is not, it is an error.
3. The name of the input file.
4. The name of the output file.

Your program should open the input file and read its content. For each alphabetic character (a–z, A–Z), it should convert it to lowercase, encrypt (or decrypt) it and print it to the output file (the output should all be lower case). For any other character, it should be printed un-modified, and the position in the password should be unchanged. For example, if the input file contained `Hello World!` and the password were `abc` (0,1,2) then the program should write `hfnlp yosnd!` to the output file.

Here is a sample run:

The program should provide at least the following error checking:

```
$ ./crypto e mystery
wrong number of arguments
$ ./crypto en mystery in out
must enter e for encryption or d for decryption
$ ./crypto e mysterY in out
key must have less than 10 all lower case letters
$ ./crypto e mystery2 in out
key must have less than 10 all lower case letters
```

Next, run the program with proper arguments. The program is run in encryption mode and uses mystery to encode the existing input file **in** and output the result to **out**.

```
$ ./crypto e mystery in out
```

Then, examine both files to confirm the results:

```
$ more in
The Vigenere encryption was the creation of the French diplomat, Blaise de
Vigenere, 1523-1596. Like Caesar and all the cryptographers that followed, he
did not visualize the cipher in modular arithmetical terms. Rather he viewed the
cypher as a substitution cipher where a different alphabet was used for the next
letter of the message, with the alphabets repeating periodically --- according
to some key.

$ more out
ffw omxczcjx ieadwhmmfl iyk mlv adcsmmfl ad lai wpqlua hznxmetx, sjmgkx hv
tuewgiic, 1523-1596. xgcx grceyj tru yxj lai tpknlhkiybfwkw kfmr xhpcmicv, ai
ugp lgm zzqgydbdv rtc ubtycd gf fsusxyj tvzrtkwmmtyx rwkqj. pmrzxv yc hgwpiu rtc
urtycd yk t wlzeramykgal ubtycd uzxvv y pgxyiiczr setyyncl pej secv ysi rtc fxbk
jqrlxv fd ffw fijqmew, pmkf ffw tpgfmzwmw icbcsmmee bcjbsugoydec --- raomjwmee
fm khqv iqw.
```

Run the program in decryption mode. Use **out** as the input file and output to another file **newin**

```
$./crypto d mystery out newin
$ more newin
the vigenere encryption was the creation of the french diplomat, blaise de
vigenere, 1523-1596. like caesar and all the cryptographers that followed, he
did not visualize the cipher in modular arithmetical terms. rather he viewed the
cypher as a substitution cipher where a different alphabet was used for the next
letter of the message, with the alphabets repeating periodically --- according
to some key.
```

Note that your program should print all error messages to standard output (stdout) not standard error (stderr) in order for us to grade your program correctly.

Hint: Use an `ifstream` object for input file and an `ofstream` object for output file. Also, you can read in (and write to) the file one character at a time while processing the file.

http://www.cplusplus.com/reference/fstream/ifstream/
http://www.cplusplus.com/reference/fstream/ofstream/

Part 2 (60%)

Write a program to guess (break) the key word of an encrypted file. We assume the length of the key *L* is known and it will be passed to your program as command line arguments. The algorithm for breaking the program is described in `VigenereCipher.pdf.` Since the key length is given. You only need to focus on the last two pages of the document.

You may use the letter frequency array ***A*** provided in the paper. To break the first letter in the key, you need to follow the calculation steps below:

1. Arrays ***A(i)*** which is ***A*** cyclic right shifted ***i*** positions. (***i*** = 0, 1, ..., 25). That is, ***A(i+1)*** arrays is created by cyclically right shifting an element from ***A(i).***
2. Array ***W*** which has the probabilities of letter a, b, ..., z in positions *0, L, 2L, ..., (q−1)L* of the encrypted file, where *q* = file length / key length *L*.
3. The array dot products ***W*** • ***A(i)*** where i = 0, 1, ..., 25. The maximum of the dot product results would give us the best guess for the key letter.

You should repeat steps 2 and 3 to recalculate ***W*** and the dot products ***W*** • ***A(i)*** in order to break other letters in the key. For example, to break the second key, array ***W*** should now have the probabilities of letter a, b, ..., z in positions *1, L+1, 2L+1, ..., (q−1)L+1* of the encrypted file.

Two encrypted files are given to test your program. The files are encoded with keys of length 9 and 8 respectively. You need to find out what is the key for each file by running the following command:

```
$ mybreak 9 encoded1
?????????
$ mybreak 8 encoded2
????????
```

Requirements

- Submit two source code **`crypto.cpp mybreak.cpp.`** The file names have to be exactly as indicated or your program won't be compiled by our grading program. You should also fill out the README file and submit it as well. To submit your files, use the following command on agate:

  `~cs515/sub515 2P` **`crypto.cpp mybreak.cpp README`**

- Do not turn in executables or object code. Do **not** submit the provided makefile.

- Make sure your submission compiles successfully on Agate. Programs that produce compile time errors or warnings will receive a zero mark (even if it might work perfectly on your home computer.) To check this, use the make command with the provided makefile. You just need to type in **`make`** in your working directory on agate.

- Be sure to provide comments for your program. You must include the information as the section of comments below:

```
/**    CS515 Assignment X
File: XXX.cpp
Name: XXX
Section: X
Date: XXX
       Collaboration Declaration:
       assistance received from TA, PAC and online resources etc.
       …
 */
```

- Programs are graded for correctness (output results and code details), following directions and using specified features, documentation and style, efficiency, appropriateness of algorithms. Included below is a tentative grading scheme.

crypto command line arguments, 9 test cases	10
crypto test run 1	5
crypto test run 2	5
crypto test run 3	5
crypto test run 4	5
crypto test run 5	5
crypto test run 6	5
mybreak test 1	10
mybreak test 2	10
mybreak test 3	10
mybreak test 4	10
mybreak test 5	10
mybreak test 6	10
others	-20
Programming Style	