

Project 6 (50 points)
Due Monday, April 15 by midnight

Background:

In cryptography, *plaintext* is the original message and *ciphertext* is the encrypted message. A *cipher* is an algorithm that converts plaintext into ciphertext.

The ***four-square cipher*** is an encryption method that uses four matrices to convert pairs of plaintext. This technique involves two *keys* that are used to encrypt messages and to decrypt messages. We use the two keys to construct the 5x5 matrices *plain1*, *plain2*, *cipher1*, and *cipher2* as follows:

- 1) We construct *plain1* and *plain2* by filling in all the letters in the alphabet one row at a time. (We skip Q since we only have 25 spaces). So, *plain1* and *plain2* always look like this:

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

- 2) We construct *cipher1* by first filling in the matrix with the unique letters from the first key. For example, if the key is “testkey”, then we would start by filling in the letters T, E, S, K, Y. (Notice that we do not fill in repeated characters from the key.) We then fill the matrix with the remaining characters of the alphabet (again, skipping Q). We do not repeat any characters that appeared in the key. So, for “testkey”, *cipher1* looks like this:

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

- 3) We construct *cipher2* in the same manner as *cipher1*, except we use the second key. So if the second key is “secondkey”, then *cipher2* looks like this:

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

Encryption

Suppose we want to encrypt the string “HELLO WORLD”. The first thing we do is divide the message up into groups of two: HE LL OW OR LD. We will encrypt each group of two separately.

We will first encrypt “HE”. We find the row and column of “H” in *plain1*: **row 1, column 2**. We then find the row and column of “E” in *plain2*: **row 0, column 4**.

The first letter in the encrypted string is at **row 1** and **column 4** in *cipher1* – F. The second letter in the encrypted string is at **row 0** and **column 2** in *cipher2* – C. Thus “HE” becomes “FC”.

When we repeat this process for every pair of characters in the “HELLO WORLD”, we get the encrypted message “FCHGHZHTJE”.

Decryption

Decryption works in the same manner as encryption. Suppose we are decrypting “FCHGHZHTJE”. We first break the string up into groups of two – FC HG HZ HT JE – and decrypt each group separately.

We will first decrypt “FC”. We find the row and column of “F” in *cipher1*: **row 1, column 4**. We then find the row and column of “C” in *cipher2*: **row 0, column 2**.

The first letter in the decrypted string is at **row 1** and **column 2** in *plain1* – H. The second letter in the decrypted string is at **row 0** and **column 4** in *plain2* – E. Thus “FC” becomes “HE”.

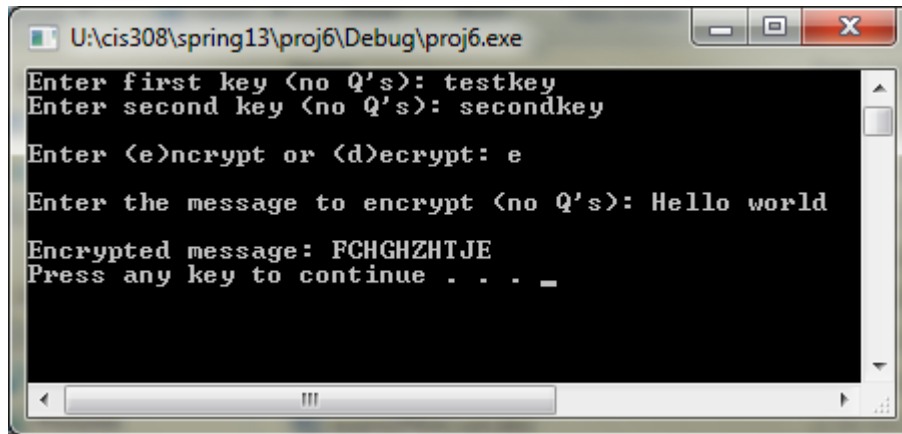
When we repeat this process for every pair of characters in the “FCHGHZHTJE”, we get the encrypted message “HELLOWORLD”.

Notice that it would be difficult to decrypt this message without having the two keys to build *cipher1* and *cipher2*. This prevents outsiders who do not know the keys from decrypting messages.

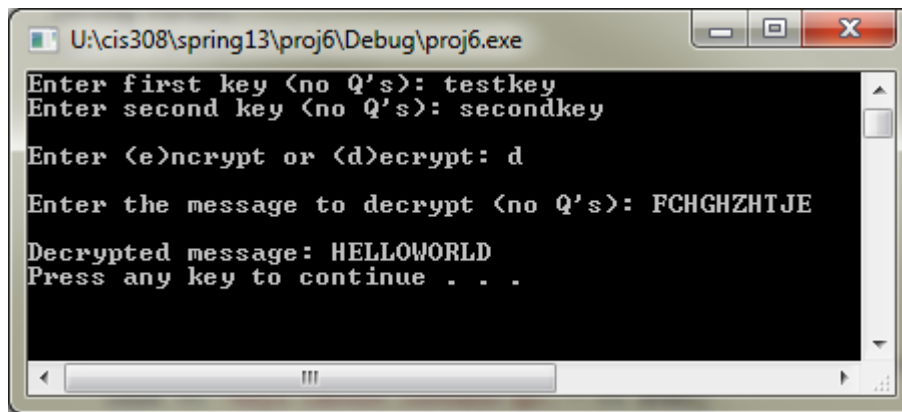
Assignment Description:

In this project, you will implement the four-square cipher algorithm. You will ask the user for both keys, and whether they want to encrypt or decrypt. If they want to encrypt, you should ask for the original message, and then print the encrypted result. If they want to decrypt, you should ask for the encrypted message and then print the decrypted result. When you read the message, you should ignore all spaces (remove them from the input string).

Here are examples of encrypting and decrypting with the keys “*testkey*” and “*secondkey*” and the original message “*Hello world*”:



```
U:\cis308\spring13\proj6\Debug\proj6.exe
Enter first key (no Q's): testkey
Enter second key (no Q's): secondkey
Enter (e)ncrypt or (d)ecrypt: e
Enter the message to encrypt (no Q's): Hello world
Encrypted message: FCHGHZHTJE
Press any key to continue . . . _
```



```
U:\cis308\spring13\proj6\Debug\proj6.exe
Enter first key (no Q's): testkey
Enter second key (no Q's): secondkey
Enter (e)ncrypt or (d)ecrypt: d
Enter the message to decrypt (no Q's): FCHGHZHTJE
Decrypted message: HELLOWORLD
Press any key to continue . . .
```

Implementation Requirements:

Your program must include the following files:

- square.h and square.cpp
- cipher.h and cipher.cpp
- proj6.cpp

square.h should contain the definition for class Square. It will represent one of the 5x5 matrices. It should have a 5x5 array of characters as an **instance variable**. It should also have two **constructors** – one for creating plain1 and plain2 that fills the matrix with all the letters of the alphabet, and one for creating cipher1 and cipher2 that takes the key and initializes the matrix as shown in the background section. It should have a **destructor** that releases any dynamic memory allocated in the class. Square should also contain the following functions:

- **getChar** – take a row and column and return the character at that position in the matrix
- **getPos** – take a character and return an integer array that holds the row and column positions of that character in the matrix

square.cpp should implement the constructors, destructor, and each function in square.h.

cipher.h should contain the definition for class `Cipher`. The `Cipher` class will handle the encryption and decryption. It should have four `Square` objects **as instance variables** – one each for `plain1`, `plain2`, `cipher1`, and `cipher2`. It should also have a **constructor** that takes the two keys and creates the four `Square` objects. It should have a **destructor** that releases any dynamic memory allocated in the class. Finally, it should have the following functions:

- **encrypt** – takes a string message and returns the encrypted string
- **decrypt** – takes an encrypted string and returns the decrypted string

Both `encrypt` and `decrypt` should use the four-square cipher algorithm.

cipher.cpp should implement the constructor, destructor, and each function defined in `cipher.h`.

proj6.cpp will contain the main function. It will create a class will create `Cipher` object, get appropriate user input, and call functions in `Cipher` to do the encryption and decryption.

Unexpected Cases:

The algorithm will not work in certain cases, such as when:

- The user includes the letter ‘Q’ in either key or the original message
- The original message has an odd number of characters (not including spaces)

If the user either includes a ‘Q’ or uses an odd number of characters, you should print an appropriate error message and exit without performing the encryption/decryption. You do not need to check any other error conditions, but you are welcome to add additional error checks if you want.

Your program should not make any assumptions about the capitalizations of characters in the keys or messages. The easiest way to handle this is to convert all words to upper case, like this:

```
char orig = 'h';
char caps = toupper(orig);
```

In the example, `caps` is now ‘H’. You will want to call `toupper` for each character in the string.

Documentation:

Your program must include a comment block at the top of every file and every function. The function comments should include a brief description of what the function does, and explain any function arguments and return values. You may use the comment block below as a template:

```
/******
* Name: (YOUR NAME)                               *
* Date: (THE DUE DATE)                             *
* Assignment: Project 6: Four Square Cipher         *
```

 * (WRITE A DESCRIPTION OF THE PROGRAM) *
 *****/

Submission:

Your project must be submitted using the Project Submission Link on K-State Online. Submit a zip file of your project using this submission page before midnight.

Grading:

Programs that do not compile will receive a grade of 0. Programs that do compile will be graded according to the following point breakdown:

Requirement	Points
Used given class/method structure	5
Correctly gets input keys, (e) or (d), message	3
Encryption	16
Decryption	16
Works with both upper and lower-case keys and messages	4
Error handling (doesn't allow Q, odd length)	4
Documentation, submission, file naming	2
Total:	50