CSE 232 Fall 2017

Programming Project 05

Assignment Overview
This assignment is worth 40 points (4.0% of the course grade) and must be completed and turned in before 11:59pm on Monday, October 16th , 2017. That's two weeks because of the midterm on ***Tuesday, Oct 3rd in the evening, 7pm, 1281 ANH*** .

Background
Caesar cipher
The Caesar cipher is named after Julius Caesar who used this type of encryption to keep his military communications secret. There are a couple of ways to think about how this works, but one is the idea of a *string rotation*. A single left rotation of a string moves all the letters down (to the left) one index, and the letter at the front is moved to the end of the string. A rotation of 3 does this three times. For example:

```
           left rotation        left rotation       left rotation
abcdefg      →      bcdefga    →    cdefgab    →     defgabc
```

A Caesar cipher left rotates the alphabet being used by some number, say a rotation of 3, and aligns the rotated alphabet with the original alphabet. The number of rotations is a fixed number is called the ***key***. The ***plain-text*** is your original message; the ***cipher-text*** is the encrypted message. We create a rotation of the original alphabet, below a left-3-rotation, and align the letters of the original alphabet and the rotated alphabet

Here is the complete mapping for a left rotation of three:

```
   Original alphabet:        abcdefghijklmnopqrstuvwxyz
  left rotation 3 alphabet:  defghijklmnopqrstuvwxyzabc
```

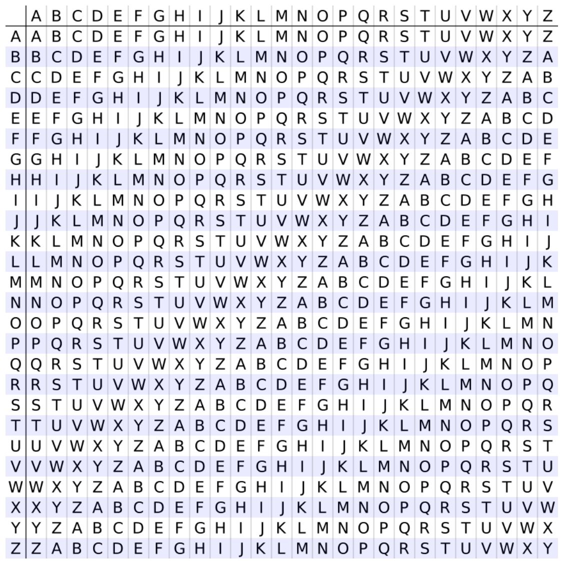
To encrypt a message simply substitute the plain-text letters with the corresponding rotated letters. For example, here is an encryption of "the quick brown fox jumps over the lazy dog" using our three-rot cipher (case is ignored and spaces are preserved, not the case for the project!!):

```
Plaintext:  the quick brown fox jumps over the lazy dog
Ciphertext: wkh txlfn eurzq ira mxpsv ryhu wkh odbc grj
```

To decrypt the message simply reverse the process.

Beaufort and Vigenère cipher
It turns out the Caesar cypher can be cracked pretty easily. Two improved versions of a Caesar cipher are the Beaufort http://practicalcryptography.com/ciphers/classical-era/beaufort/ and

Vigenère http://practicalcryptography.com/ciphers/classical-era/vigenere-gronsfeld-and-autokey/ ciphers.
Both require a key (a string) and both are classically described as using a tabula recta of letters, shown below.



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

Both align the key with the plaintext to create a cipher text, but their approach is slightly different. Let us use "`fred`" as the key. Note that if the key is shorter than the plaintext, we reuse the letters of the key to repeatedly to cover all the letters in the plaintext. For example:

Key:	`fredfr`
Plain:	`cipher`
Vigenère :	`hztkji`
Beaufort :	`djpwba`

For Vigenère the process is:

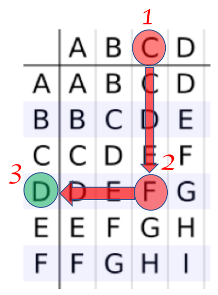
- take a letter from the plaintext, let's start with *c*
- find *c* in the top row of the tabula recta
- find the corresponding key letter in the column to the far left, here the letter *f*.
- find the intersection of the *c* column and the *f* row, the letter *h*, which would be the letter for the cipher text.
- repeat

For Beaufort, the process is very similar:

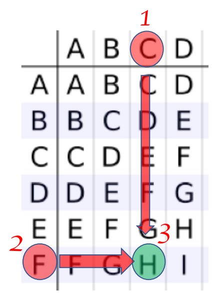
- take a letter from the plaintext, let's start with *c*
- find *c* in the top row of the tabula recta
- find the corresponding key letter in that column, here the letter *f*

- find the column letter in the far left column on the same row, *d*, which is the letter for the cipher text
- repeat

Graphically, the difference is:



Beaufort



Vigenère

ASCII
The index order of an ascii letter can be found by subtracting the character 'a' from any other lower-case letter. Thus the letter `'f'` is index 5, found by `'f' - 'a'`. You did this in lab last week. We can use this.

Without Tabula Recta
Describing the process as using a tabula recta (tr) is convenient but inefficient. We can do better. Here's what you can notice:

- Each column of the tr is actually a predictable left rotation of the lower-case alphabet. The amount of left rotation is the column letter - `'a'`. Thus, the c-column-alphabet is a left rotation of 2 (`'c' - 'a'== 2`).
- For Vigenère:
  - creates a left rotated alphabet based on the plaintext letter
  - finds the index number of the key letter in the normal alphabet (far left column).
  - Uses that index in the rotated alphabet to find the cipher letter.

- For Beaufort:
  - creates a left rotated alphabet based on the plaintext letter
  - finds the appropriate key letter in that rotated alphabet, an index
  - you can use that index to select the letter in the normal alphabet (far left column) which is the cipher letter.

Program Specifications
As before, you provide the functions listed below.

function `rotate_left`:

- two arguments: positive integer (left rotation amount) and string to rotate
- returns the left rotated string.

The rotation can be longer than the length of the string, but modulus would make that faster than actually doing all the rotations.

Error check: If the rotation amount is not positive (greater than 0), the original string is returned.

function `beaufort_letter`:

- two arguments: a `char` plaintext letter and a `char` key letter
- returns a `char`, the beaufort cipher letter.

Uses `rotate_left`, string `find` and `char` math to solve the problem as described above.
Error check: if either the plaintext or key letter are not lower case alphabetic characters, the function returns the original plaintext letter

function `vigenere_letter`:

- two arguments: a `char` plaintext letter and a `char` key letter
- returns a `char`, the Vigenère cipher letter.

Uses `rotate_left`, `char` math and string indexing to solve the problem as described above.
Error check: if either the plaintext or key letter are not lower case alphabetic characters, the function returns the original plaintext letter

function `encode_beaufort`

- two arguments: a `string` plaintext and a `string` key
- returns a `string`, the encoded cipher
- uses `beaufort_letter`

Error check: For any letter in the plaintext that is not a lower case alphabetic letter, that letter is ***ignored*** and not placed in the resulting cipher string. Thus "flee now" would create a cipher 7 long (the space would be skipped).

function `encode_vigenere`

- two arguments: a `string` plaintext and a `string` key
- returns a `string`, the encoded cipher
- uses `vigenere_letter`

Error check: For any letter in the plaintext that is not a lower case alphabetic letter, that letter is ***ignored*** and not placed in the resulting cipher string. Thus "flee now" would create a cipher 7 long (the space would be skipped, i.e. “fleenow”).

Deliverables
As with the previous project, we give you a main program as part of the starter code. When you submit, please use the original main. This main is a little more complicated, but it gives you an idea how you might easily test your code

You turn in proj05/proj05.cpp (with the provided main) to Mimir.

Assignment Notes

1. As before, write the functions individually and test them with your main (we certainly will).
2. Test your own code. Tests on Mimir won’t always exist, so it’s important to test your code against possible edge cases.

3. Notice in the provided main program that I read 3 lines for the test cases on encode_vigenere and encode_beaufort. That is so I can read in an entire cipher text that includes spaces. Look at this week's videos for more details.