Project 5: Square Cipher

Due: Monday June 11, 11:59 pm

1 Assignment Overview

This project focuses on using arrays. You will write functions that encrypt strings.

2 Assignment Deliverables

You must turn in completed versions of the following files:

- proj05/cipher.cpp

Be sure to use the specified file name and to submit your files for grading via **Mimir** before the project deadline.

3 Background

In Project 4, we used a variation on the Caesar cipher to encrypt one letter at a time. Unfortunately, this leaves it somewhat open to frequency analysis as in practice different letters will appear at different rates. To make this type of analysis more difficult we will encrypt two characters at once.

The key for our method is a 5×5 square that contains each letter (except for 'j') exactly once ('i' and 'j' are treated as the same letter). This square is constructed from a keyword or phrase by placing each letter in the first open space (going left to right then top to bottom), skipping letters that have already been seen. Any letters not in the original keyword are then placed in alphabetical order. For example, the key "squarecipher" will generate the following square:

s	q	u	a	r
e	c	i	p	h
b	d	f	g	k
l	m	n	o	t
v	w	x	y	z

It seems like we would need to use a 2D `vector` but we don't. We can represent the above table with the string "`squarecipherdfgklmnotvwxyz`". If we look at the letter 'd', we see that it is at index 11 in the string (remember the first index is 0) or at row 2, column 1 in the square. We can convert between the two by using the conversions

$$row = index/width$$
$$col = index\%width$$
$$index = row * width + col$$

To encrypt a message we break it into pairs of letters (known as bigrams) and encrypt each bigram. If the message has an odd number of characters, we append an extra 'x' to then end. Thus, the plaintext "runfast" is represented by the bigrams `ru nf as tx`.

There are four cases for a particular bigram.

1. Both letters are the same: replace the second letter with an 'x' and move to the appropriate other case.
2. Both letters appear in the same row: each letter is represented by the letter directly to its right (wrapping around to the left as necessary)
3. Both letters appear in the same column: each letter is represented by the letter directly below it (wrapping around to the top as necessary)
4. The letters are not in the same row or column: each letter is represented by the letter in the same *column* but in the same row as the other letter.

For example, `ru` appears in one row so these are shifted rightward to `sa`. Similarly, `nf` appears in one column so these are shifted downward to `xn`. `tx` doesn't appear in a single row or column so it instead forms two corners of a smaller square $\begin{matrix} n & t \\ x & z \end{matrix}$ We thus use `zn` by selecting the other letter in the same column of smaller square.

To decrypt a message we perform the steps in reverse: going up or left instead of down or right.

4 Assignment Specifications

In this and in future projects we will provide *exactly* our function specifications: the function name, its return type, its arguments, and each arguments type. The functions will be tested individually in Mimir using these exact function specifications. If you do not follow the function specifications, these independent tests of your functions will fail. Do not change the function declarations!

What you test on Mimir is a file that contains only the functions. You do not turn in a main program. We can test the functions individually on Mimir. However, you should write your own main program to test your functions separate from Mimir. It is more flexible and you can debug more easily.

function `CellToIndex`
params: x : size_t, y : size_t
returns: size_t

function `IndexToCellX`
params: index : size_t
returns: size_t

function `IndexToCellY`
params: index : size_t
returns: size_t

function `CreateSquare`
params: keyword: const string&
returns: string

function `EncodeString`
params: plaintext: const string&, keyword: const string&
returns: string

function `DecodeString`
params: ciphertext: const string&, keyword: const string&
returns: string

5 Assignment Notes

- You will receive no points if your solution does not compile on Mimir.

- Points will be deducted if your solution has any compiler warnings.
- Points will be deducted if you include `using namespace std`.
- The `<string>` library contains a number of useful methods.
- You *do not* need to check for bad input values for this project. In future projects we will explicitly indicate the errors we are looking for.
- You do need to be able to handle the letter 'j' and strings of odd length.
- `size_t` is the return type of the `size` function of collections like `string` and `vector` and is the same as `string::size_type`. It is an **unsigned** integer.
- You may assume that the string is all lowercase.
- The encoding of "runfast" with the keyword "squarecipher" should yield the ciphertext "saxnrqzn".
- If you decode the string that you encode using the same keyword you may not get the same string back; strings of odd length, containing double letters, or the letter 'j' will be transformed in a consistent and recognizable way.