Programming Project 05

Background
Steganography
Steganography (https://en.wikipedia.org/wiki/Steganography) is the process of hiding a "secret message" in another text file, image or even sound file. It differs from cryptography in that the overall file/video/audio looks reasonably normal and still conveys information, making it hard to tell that there is a secret hidden inside. We are going to write a steganographic encoder/decoder for text.

A Simple Steganography
We are going to take a plaintext message, one that anyone can read, and embed in that message a secret message which someone, who knows the code, can decipher.

The process is this. We are going to take the plaintext message, reduce the plaintext message to all lower case, then encode the secret message in the plaintext based on combinations of UPPER or lower case. Thus it is the ***sequence*** of upper and lower case letters in the plaintext message that encode our secret message.

We do this as follows. Each individual letter of the secret message is turned into a binary string of 5 0's and 1's. Exactly 5 for each letter. We will only recognize letters and will ignore any other characters in the secret message. For each of these binary strings, we take the next 5 letters of the plaintext and modify them as follows: for every 0 in the secret message binary string we lower case the plaintext letter, for every 1 we upper case the letter. We do this only for letters, ignoring all other characters in the plaintext. Let's consider the following example. The secret message is "help " and the original text is "Mom please send more money!" The index of each letter is its position in the alphabet with 'a' at 0, 'z' at 25.

Secret letter	'h'	'e'	'l'	'p'
Letter index	7	4	11	15
binary	00111	00100	01011	01111
Encoded 5 letters	moM PL	eaSe s	eNd MO	rE MON

The new message, with the encoded secret message would be (hard to write with autocorrect) "moM PLeaSe seNd MOrE MONey!". As we said, we can only capitalize ***letters*** so we ignore (don't count as one of the 5 letters) any other character in the plaintext message which just gets passed through unaltered.

Reverse the process for decoding: take 5 letters from the encoded plaintext, ignoring any other characters, determine the binary string the capitalization indicates, and add the new letter that binary string represents as the next letter of the secret message.

Rules of the Process.

- we ignore non-alphabetic characters in the plaintext and pass them through to the encoded text as is.
- we also ignore any non-alphabetic characters in the secret message. Those non-alphabetic characters will not be encoded. Thus spaces will be lost in decoding the secret message, as will any numbers of punctuation marks.
- if there are "left over" letters in the plaintext, letters we do not require to encode a portion of the secret message) we just pass them through into the encoded text unchanged.
- if there are not enough letters in the plaintext to encode the secret message, that is an error condition and we indicate as such and quit.
- We mentioned that, if there are more letters than necessary to encode the secret message in the plaintext, then we just pass the "extra" letters through. On decoding that may create garbage at the end of our decoded message. That’s OK

ASCII

As a note, you don't need a string to turn a letter into an index number. 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.

Program Specifications

`string lower_case(string s)`

- returns the lower case version of the input string `s`, that is all alphabetic characters are converted to lower case.

`string to_binary(char c)`

- returns 5 bit string that is the index of the character argument.
    - if the provided character is not a lower-case alphabetic character, return the empty string.

`char from_binary(string bit_str)`

- returns the character that the 5 bit binary string `bit_str` represents.
    - if any of the following conditions are not true:
        - the size of `bit_str` is 5
        - every element of `bit_str` must be a ‘1’ or a ‘0’
        - the character produced must be a lower case letter

      return the character 0 (the NULL char).

`bool check_message(string plaintext, string secret_message)`

- returns `true` if there are at least 5x the count of characters in `secret_message` as in `plaintext`, `false` otherwise

- Remember, only alphabetic characters matter, all other characters in the `plaintext` are ignored. We are counting whether there are enough useable characters in `plaintext`.

`string encode(string plaintext, string secret_message)`

- `plaintext` and `secret_message` should be converted to lower case by `lower_case`
- the `plaintext` string should have been checked by `check_message`
    - if `check_message` is false, return the string `"Error"`.
- otherwise returns `encoded_text` encoded (as described) with the `secret_message`.

`string decode(string to_decode)`

- returns the original `secret_message` as a string.
    - if there were more characters in `to_decode` than 5*characters in the `secret_message,` you will get extra 'a' characters at the end of the decoded message.
    - if the number of characters in `to_decode` is not a multiple of 5, it will be ragged. By that I mean that you will get 4 or less characters at the end of `to_decode`. Since we cannot turn those characters into a `secret_message` character (we need 5), they should be ignored.

Assignment Notes

1. You turn in only the proj05/proj05_functions.cpp
2. The proj05_functions.h will be provided by Mimir as well as in the project directory. Mimir will compile against the provided .h file, even if you provide one in the directory.