CSE 232 Spring 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, February 27th 2017. That's two weeks because of the midterm on Wed, Feb 15th .

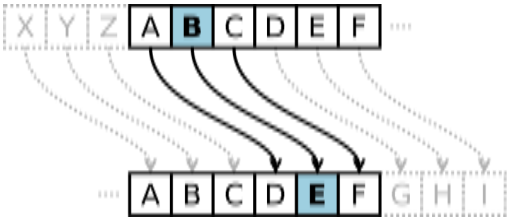
Background

Cryptography

Cryptography (https://en.wikipedia.org/wiki/Cryptography) is the process of encoding a "secret message" in such a way that the message can be sent in a public way, but only those who know how the encoding works can actually understand the underlying message.

Caesar cipher, an example

The Caesar cipher is named after Julius Caesar who used this type of encryption to keep his military communications secret. A Caesar cipher replaces each plain-text letter with one that is a fixed number of places down the alphabet. The fixed number is called the ***shift***. The ***plain-text*** is your original message; the ***cipher-text*** is the encrypted message. The example shown below uses a shift of three so that "B" in the plain-text becomes "E" in the cipher-text, a "C" becomes "E", and so on. The mapping wraps around so that "X" maps to "A" and so on.



Here is the complete mapping for a shift of three:

Plain-text: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher-text: DEFGHIJKLMNOPQRSTUVWXYZABC

To encrypt a message simply substitute the plain-text letters with the corresponding cipher-text letter. For example, here is an encryption of "the quick brown fox jumps over the lazy dog" using our shift-three cipher (case is ignored and spaces are preserved):

Plaintext: the quick brown fox jumps over the lazy dog
Ciphertext: WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ

To decrypt the message simply reverse the process.

A Caesar cipher is easy to describe and easy to break. It is subject to statistical analysis, looking at letter use frequencies in English (or whatever the language the message is encoded in) and deducing a map back to the encoded letters. We can do better than that.

Pangram key encryption
A Pangram (also called a holoalphabetic sentence) is a well-formed English sentence that contains every letter of the English (or whatever the underlying language of the message) alphabet. The best known was is probably (though there are many others):

"The quick brown fox jumped over the lazy dog"

We can use a pangram as an easy to remember key to encode and decode a secret message.

The process
Let's say our secret message is `"help"`. Pretty simple. Let's also say that our pangram key is going to be `"abcdefghijklmnopqrstuvwxyz "` (all the lower case letters with a space at the end). We show the pangram with indices below.

```
                      1 1 1 1 1 1 1 1 1 2 2 2 2 2 2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
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
```

Encoding
- assuming that we have a marker index into our key called `start`, initialized to 0.
- take the first letter of our secret message, `'h'`.
    - search for the letter `'h'` beginning from `start`
    - if we find `'h'`, determine the ***distance from start to the letter*** (in this case, 7);
    - record 7 as our encrypted letter, update `start` to 7
    - encrypted message so far: 7, `start` at 7
- repeat for the letter `'e'`.
    - That letter does not occur in a search from `start` to the end, so we allow for "wrap around" behavior.
    - The distance is 18 (from `start` to the last letter) + 5 (the distance from the beginning of the pangram to `'e'`), 23
    - encrypted message so far: 7 23, `start` at 4
- `'l'`: distance from `start` is 7, encrypted message so far 7 23 7, `start` at 11.
- `'p'`: distance from start is 4, encrypted message is 7 23 7 4, start at 16.

Decoding
The numbers are offsets, including wrap around. Knowing the key, you can re-create the message

Program Specifications

`string lower_and_strip(string s)`

- returns the lower case version of the input string `s`, that is all alphabetic characters are converted to lower case
- removes any elements that are not alphabetic characters ***~~or the space character~~***. We will not be encoding anything but alphabetic characters and the space character.
- returns the updated string

```
int return_encoded_char(string key,
                        string::size_type &start, char C)
```

- if the character `C` is found in `key`:
  - returns an integer that represents the distance required to find the provided character `C` in the key (including wrap around).
  - otherwise it returns `key.size()` + 1.
- updates `start` to the new index (passed by reference)
  - if the character is not found, `start` is updated to 0.
- returns the integer

`string encode (string message, string key)`

- converts `message` using `lower_and_strip`
  - ***key is not changed***, but is used as is!
- for every character in the message
  - call `return_encoded_char` with that character
  - concatenates the number into a space separated string
- returns the encoded message

```
char return_decoded_char(string key,
                         string::size_type &start, int num)
```

- if `num < key.size()+1`
  - returns character that is `num` characters away from `start` in the key (including wrap around).
  - else returns the null character `'_'` (the underline character)
- updates `start` to the new index (passed by reference)
  - if `num >= key.size()+1`, `start` is set to 0
- returns the char

`string decode (string encoded_text, string key)`

- for every character in `encoded_text`
  - call `return_decoded_char` with that character
  - concatenates the number into a string
- returns the encoded message

Deliverables

You must use handin to turn in a file called `proj05.cpp` . Please be sure to use the specified file name, and save a copy of your `proj05.cpp` file to your H drive as a backup.

Assignment Notes

1. Testing will be done using provided test cases. There are 11 test cases, described below:
    a. 1, test `lower_and_strip`
    b. 2-4, test `return_encoded_char`
    c. 5, test `encode`
    d. 6-8, test `return_decoded_char`
    e. 9, test `decode`
    f. 10, harder `encode`
    g. 11, harder `decode`
2. We will provide a main that will test the functions. Do not modify it.
    a. `skeleton.cpp` is the main to start with
3. You may write other functions as well. They can be in `proj05.cpp` to be called by other functions in the same file.