Programming Assignment 1: Beyond the Warm-up Assignments

Jack West and George K. Thiruvathukal
Loyola University Chicago
Department of Computer Science

October 5, 2021

1 Caesar Ciphers Explained

Caesar Ciphers work on the idea that two people Alice and Bob want to send data to each other and they want to hide their message by shifting the letters over to the right by a secret amount (known as a displacement) only known by Alice and Bob. It is one of the earliest encryption systems, developed in Ancient Rome and named for Julius Caesar, who used it for encoding military messages.

True to the original tradition, We will be working with a fixed alphabet. The English alphabet, `ABCDEFGHIJKLMNOPQRSTUVWXYZ`, will be used. To keep things simple, all letters will be capitalized and we will not be encoding any other symbols. For example, a right shift of five positions in our alphabet would look like this `GHIJKLMNOPQRSTUVWXYZABCDEF`.

Notice how all of the letters are shifted over by five positions. So as an example of how you can use the shifted alphabet to encode a word "HELLO" you translate the letter by looking up its normal position in the shifted alphabet. Since `H` is the 8th letter (7 if you start from 0) you would output the 8'th letter in the shifted alphabet, which is the letter `M`. The process is repeated for each subsequent letter. `E` (the 5th letter) is translated as `M`, `L` as `Q`, etc.

The receiver would receive the message `MJQQT`.

To decode the message, you do the reverse and look at the key (6) and perform the reverse encoding to get HELLO.

While this cipher might appear to be straightforward, it remains one of the interesting ones from a historical point of view and was even used to build many variations that are hard to crack (e.g. the Viginere Cipher).

2 Assignment

Below are the requirements for the assignment:

- Your job is to decode a text file where every **sentence** has a *different* Caesar Cipher shift.

- You will have to read from `stdin` (standard input) dynamically, split the input line into words, and then figure out the Caesar cipher shift for that specific sentence.
- You may not use arrays. You must use linked lists as your data structure to hold and examine your data.

There are **26 possibilities** for each sentence.

2.1 How to Decode a Caeser Cipher

You must apply a shift value to each letter in a sentence. Do not apply a shift to the whitespace. To know if you got the shift correct, check to see if all the words are in the dictionary file. If all of the words are found in the file your code more than likely works. Once you find the shift value, save the shift to a file. The format in which to save the values is exaplained below:

```
Write your sentence shift to a file called shifts.txt
The output will look like this:

#Below is the first line in the file
1 #What this tells me is the first sentence has a Caesar shift of 1
2 #sentence two had a Caesar shift of 2
5 #sentence three had a Caesar shift of 5
.
.
. Processes n sentences
.
.
14 #sentence n had a Caesar shift of 14.
```

3 Grading

You will be graded using the following criteria.

1. Not working on the assignment at the last minute. (Yes, we always know.)
2. Able to parse a sentence from `stdin` (1 point).
3. Code is readable and commented (1 point).
4. Linked Lists are properly implemented (1 point).
5. Code has modular structure with separate files and great use of functions (1 point).
6. Code is able to write results to a file (1 points).
7. Code is able to correctly determine the Caesar cipher of each sentence (5 points).

4 Enrichment and Challenge Opportunities

While we don't necessarily award extra credit in this class, there are opportunities to distinguish yourself. Here are some possibilities.

1. Allow for upper and lowercase input, but use the uppercase translation table to handle both. Use case-insensitive comparison to identify words when determining whether you have "cracked the key" so to speak.
2. Allow for symbols not in the alphabet by gently ignoring them. Since a word is defined as being anything in the alphabet, assume anything not in the alphabet was not encoded (or to be decoded). This would allow punctuated text to pass through during processing.
3. Have an option to write the decoded text to a line, with its shift value (e.g. shift value:decoded text) to the standard output. Write a second program that takes the colon-separated shift:decoded-text and recreates the encoded text file.
4. Use GitHub to demonstrate that you are making daily progress. This will require you to learn `git` basics and be able to use `git` within `https://repl.it`.