Team Project 1: Caesar Encoder and Decoder

Spring Semester 2023

1 Overview

As we discussed in the 2nd lecture, we will be working with a Caesar cipher to gain experience with C++. To make things *uninteresting*, we looked up Caesar ciphers on Wikipedia. You can probably ask ChatGPT to give you an initial version in any common language, especially one as common as C++. Although it is fun to write the code yourself, this is a common assignment, so I am just assuming everyone will start with a solution and then adapt it for the rest of this assignment, which will be harder to do with ChatGPT or StackOverflow.

To make things *more interesting* you are going to build two programs: an encoder and a decoder. This might even suggest a way to work as a team. Each member could focus on one of the programs and then make sure the entire system works as a coherent whole. I have written a reference implementation using the Bash scripting language. You can find this at `https://github.com/gkthiruvathukal/caesar-encode-decode`. I will be giving a demonstration on its usage during class.

We will also build on the tradition of Unix by taking advantage of the English dictionary distributed with the `aspell` project. The encoder will generate lines of text with random words from the dictionary. The decoder will read these lines of text and attempt to decrypt them by using the dictionary. The decoder will report the Caesar shift value for each line of input and show the decoded text.

2 Requirements

2.1 Encoder

The encoder has the following requirements.

Accepts the following command-line options, which are parameters to control encoder output:

- `-n,|-lines` number of lines of text to generate
- `-l,|-length` minimum word length (default 3)
- `-d,|-dict` location of dictionary containing the word list
- `-o,|-output output` filename (if absent, write to standard output)

Note that this options are different from my reference implementation in bash. You are also free to add/change your options as you see fit.

The encoder does the following:

- Loads the dictionary into a data structure (map/set). As words are loaded in, any words containing characters not in the alphabet will be discarded.
- Words are all converted to upper case.
- Generates "sentences". You may generate sentences with whitespace or punctuation. Just keep in mind that the decoder must be able to exclude any text not in the input alphabet.
- Each line of text must be encrypted using the Caesar cipher with a *randomly chosen* shift key. No information about the key is written to the output.
- The output lines may either be directed to standard output or to a file. (The idea is to make sure you learn how to do both in C++!

2.2 Decoder

The encoder has the following requirements.

Accepts the following command-line options, which are parameters to control encoder output:

- `-d,|-dict` location of dictionary containing the word list
- `-i,|-input filename` input text from filename (if absent, write to standard output)
- `-o,|-output filename` output text (of shift values) to filename (if absent, write to standard output)

Note that this options are different from my reference implementation in bash. You are also free to add/change your options as you see fit.

The decoder does the following:

- Reads one line of input at a time.
- Parses the line of input into words. A word is any continuous sequence of letters from the alphabet.
- Characters not in a word should be preserved to the fullest extent possible (e.g. whitespace and any punctuation.)
- Try all shift values with your Caesar cipher to see if all words in a line of input can be decoded. You will know if they can be decoded by looking up the words in the dictionary. Thus the decoder must be able to load the dictionary as well.
- Output the shift value on success. If for any reason an input line cannot be decoded, output a -1. This would never happen if your encoder and decoder are both working, but it is entirely possible a file generated by someone else could present problems.
- You can also write the decoder to succeed if *most* of the words are decoded. This could be an additional command line option, e.g. `--threshold=percentage`.

3 Expectations

This first project is done in teams of 2.

A logical way to organize this project is for one person to work on the encoder and the other on the decoder. However, some building blocks should be common between both tools. I can see the dictionary of words (map) and Caesar cipher code being in a separate module, perhaps called `util.{h,cc}`.

While you might find it useful to create C++ classes, strictly speaking, this code can be written almost entirely as functions, since you will be able to make use of generic C++ classes (via STL, the Standard Template Library) to do most of the work. Because we need the fluency of Modern C++ and STL for working with OneAPI, make sure you put your energy into using proper STL classes (e.g. vector, set, map) and C++ `string` as opposed to `char *`.

4 Rubric

You will be graded using the following criteria.

1. 1 point for a `cmake` file to compile your code. You must have separate commands for the encoder and decoder (i.e. separate build targets)
2. 2 points for proper software engineering practices (modular structure, comprehensible code, modern C++ usage, and overall organization)
3. 3 points for unit tests and their thoroughness.
4. 3 points for your data structure implementation and algorithm for decoding the key.
5. 1 point for user docs.