CSE341 – Programming Languages (Fall 2019)

Homework #1 – LISP Programming

Handed out: 3:00pm Thursday October 10, 2019.

Due: 11:55pm Sunday October 27, 2019.

Hand-in Policy: Source code should be handed in via Moodle.
Collaboration Policy: No collaboration is permitted. Any cheating (copying someone else's work in any form) will result in a grade of -100 for the first offense and -200 for the subsequent ones.
Grading: Each homework will be graded on the scale 100. Unless otherwise noted, the questions will be weighed equal.

Programming in Common Lisp (100 points): In this project, you will implement various functions for encoding and decoding a sequence of words using a cipher alphabet.

Cipher Alphabet

A cipher alphabet is a one to one mapping to a plain text alphabet. An example of such mapping is given below:

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

Encoding and decoding is simply done by replacing the letter in the source alphabet with the corresponding one in the target alphabet using the provided mapping. For example:

```
Original: testing this
Encoded : mqjmlgb nklj
```

Project Description

For this project, words are represented as lists of lower case symbols, e.g., the word "class" is represented as '(c l a s s). Paragraphs are lists of words, e.g.,'((c l a s s) (i s) (r e a d y) (f o r) (w o r k)). A document is a list of paragraphs.

A document encoded with a cipher alphabet is not easy to break without the help of a computer.

You are asked to implement two different methods to break the cipher,

1. a brute force version that uses a spell checker, and
2. a version that uses knowledge about the distribution of particular letters in the English language (frequency analysis).

You will need to write two functions **`Gen-Decoder-A`** and **`Gen-Decoder-B`** that take as input a paragraph of encoded words, and return as output a function that takes as input an encoded word,

and returns the plain text of the decoded word. This function can then be used to decode the entire document which may consist of multiple paragraphs. The function **Code-Breaker** takes an encoded document and a decoding function as input, and returns the entire document in plain text.

The two decoder functions implement different strategies to break the encrypted code. Both have advantages and disadvantages. Your Clojure program will provide an infrastructure to assess the effectiveness of these decoding approaches for different document types.

Brute Force with Spell Checker Gen_Decoder_A

The algorithm for the brute force code breaker is simple. The input words are encoded for each possible mapping. There are 28! (! means factorial) such mappings. For each mapping, a spell checker determines whether the resulting words are words in the English language. The mapping that generates the most correctly spelled words is assumed to be correct one.

For this method, you will need to implement a spell checker. You can implement your spell checker using the dictionary of words in file *dictionary.cl*. You will need to implemented the spell checker **spell_checker** that takes as input a word, and returns the truth value T or NIL. You have two options for this:

- **spell_checker_0:** A brute force version of the spell checker that just checks whether the word occurs in the dictionary or not.
- **spell_checker_1:** Implements a faster search strategy using hash mapping.

Frequency Analysis Gen_Decoder_B

This code breaker is based on the fact, that in the English language some letters occur more often than others. Knowing the distribution of the different letters, this method just counts the number of occurrences of each letter in the encoded words. If there are enough words to be statistically relevant, the letter distribution can be used to identify the most common letters in English, namely 'e', 't', 'a', ‘o’, ‘i’ and ‘n’. In other words, the assumption is that the most frequent encoded letter has to correspond to the letter 'e', etc. This means that one character in the mapping is decided. One can continue on the second and third letters similarly. You are asked to implement two types of this decoder.

- **Gen_Decoder_B_0:** Assume that the most frequent six letters are (in the correct order) 'e', 't', 'a', ‘o’, ‘i’ and ‘n’. After doing the frequency analysis for these six letters and establishing the mapping, you are asked to determine the rest of the letters using a brute force method.
- **Gen_Decoder_B_1:** Implements a faster strategy. It assumes that the first six most frequent letters are the same as above. But this time, we decide on the other letters by looking at the co-occurrence of letters. For example, “is” occurs much more frequent than “ig”.

Implementation

For the implementation of these functions, you can use standard built-in Clojure functions such as *first* etc. You must not use any global variables.

How to Get Started

Copy the files *decode.lisp*, *include.lisp*, *dictionary1.txt*, *dictionary2.txt, document1.txt* and *document2.txt* into your own subdirectory. You will implement your code breaker in file *decode.lisp*. File *dictionary1.txt* contains a small dictionary consisting of only a few words. Use it to debug your program. File *dictionary2.txt* contains a list of over 45,000 words allowing you to generate a realistic spell checker. You must not change this file. Finally, file *document1.txt* and *document2.txt* contains two sample documents to test your encoder, decoder, etc. You should use your own test cases as well.

Submission and Grading

You will submit your version of file *decode.lisp* via Moodle.