CSE341 – Programming Languages (Fall 2014)
Homework #1

Handed out: 3:00pm Tuesday September 30, 2014.

Due: 11:50pm Sunday October 19, 2014.

Hand-in Policy: Source code should be handed in via Moodle.
Collaboration Policy: No collaboration is permitted. Any cheating (copying someone elses 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 Scheme (100 points): In this project, you will implement various functions for encoding and decoding a sequence of words using Caesar's Cipher. The homework is adapted from Ulrich Kremer of Rutgers University.

Caesar's Cipher

The cipher translates each letter in a word independently by "shifting" the letter up or down the alphabet by a fixed distance. We assume that we are using the Turkish alphabet with 23 letters in their "usual" order, all lower case: a, b, c, … z. Five of the letters in the alphabet will use their replacements as follows: c=ç, g=ğ, i=ı, o=ö, s=ş and u=ü. The function `ctv` ("character-to-value") maps each character to its value, and the function `vtc` ("value-to-character") maps each value to its corresponding character. For example, ctv(c) = 2 and vtc(23) = z.

The encryption function has the following form:

$$\text{Encript}_n(x) = \text{vtc}((\text{ctv}(x) + n) \bmod 23)$$

where "n" represents the amount by which the letters are "shifted" up the alphabet. For this project, we assume the values of "n" to be in the range of 0 and 28, i.e., $0 \leq n \leq 22$.

As part of the project, you will first need to implement the encoding function `encode-n` which takes as input the shift value n and returns a function that takes as input a word w, and returns its encoded version:

```
(define encode-n
      (lambda (n)
            (lambda (w)
                  ...
)))
```

Project Description

For this project, words are represented as lists of lower case symbols, e.g., the word “sinif" is represented as '(s i n i f). Paragraphs are lists of words, e.g.,'((s i n i f) (d e r s e) (v e) (o d e v e) (h a z i r)). A document is a list of paragraphs.

A document encoded with the Caesar’s cipher is easy to break since there are only 23 possible “values” n. 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 knowedge about the distribution of particular letters in the English language (frequency analysis).

Both methods try to find the value of n’ such that (n + n’) = 23.

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 Scheme 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 value of n’. For each value, a spell checker determines whether the resulting words are words in the Turkish language. The value of n’ for which most words are spelled correctly is assumed to be decoding value.

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.ss*. You will need to implemented the spell checker **`spell-checker`** that takes as input a word, and returns the truth value #t or #f. A brute force version of the spell checker may just see whether the word occurs in the dictionary or not.

Frequency Analysis Gen-Decoder-B

This code breaker is based on the fact, that in the Turkish 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 Turkish, namely 'e', 't', and 'a'. In other words, the assumption is that the most frequent encoded letter has to correspond to the letter 'e', etc. This means that the shift between the encoded letter and 'e' has to be the value n. The same n has to work for the other frequent letters as well. From that, we can determine the decoding value n’. Note that this method needs a large enough set of words in order to be successful. In other words, it may not work well for very short paragraphs.

Implementation

For the implementation of these functions, you can use standard built-in Scheme functions such as append. You should use the `reduce` function at least once in your implementation. The definition of `reduce` is given in file *include.ss*. You must not use assignment (set!) in Scheme.

How To Get Started

Copy the files *decode.ss*, *include.ss*, *test-dictionary.ss*, *dictionary.ss* and *document.ss* into your own subdirectory. You will implement your code breaker in file *decode.ss*. File *test-dictionary.ss* contains a small dictionary consisting of only six words. Use it to debug your program. File *dictionary.ss* contains a list of over 45,000 words allowing you to generate a realistic spell checker. You must not change this file. Finally, file *document.ss* 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.ss* via Moodle.