Honors Homework 2

Discrete Structures, CS 173, Spring 2019

Due Friday, March 29

This assignment has you build some simple functions using lists and mapping operations, strings and characters, and more recursive numerical functions. This will get you set up to do the third assignment, which implements a simple version of RSA encoding.

As with the first assignment, you will use moodle to submit your work. Submit two files:

- A file (extension rkt) containing all your functions. Include enough comments that I can easily understand what you did.
- A file showing sample inputs and outputs for your functions. Find inputs and outputs that clearly illustrate that the code is working right.

Please submit the two files separately. Do not bundle them into a tar/zip archive.

Please submit all your input/output examples in one file. If necessary, combine individual items (e.g. screenshots) into one document.

This is a solo assignment. Do not share code or detailed designs with other students.

1 Basic list operations

There is information on basic list operations in section 8 of the Quick introduction from the first assignment, and in the following part of the Racket Guide for some basic list operations.

`http://docs.racket-lang.org/guide/Lists__Iteration__and_Recursion.html`

Experiment with the map function from the Quick tutorial and the filter function from the Racket Guide.

Also check out the Racket documentation on strings and characters:

```
https://docs.racket-lang.org/reference/strings.html
https://docs.racket-lang.org/reference/characters.html
```

Notice that the function string->list converts a string to a list of characters. Use this function, and filter and map, to write a function that normalizes a string:

- Non-alphabetic characters are removed.
- All letters are converted to uppercase.

The function should take a string as input and output a new string.

2 Character encoding

Build a function (perhaps calling other functions) that takes a string as input, normalizes the string as in the previous section, and returns a new version of the string as encoded by the ROT13 cipher.

Some hints:

- Look up the ROT13 substitution cipher on Wikipedia.
- In the Racket documentation on characters and strings (see previous section), find the function char->integer.
- First build functions that do only part of the task, e.g. convert alphabetic characters into numbers in the range 1-26.

3 Recursive powers

Build a recursive function that takes integers a and n as input and produces a list containing the first n powers of a. Your function should create the larger powers by multiplying smaller ones by a. Do not call an exponentiation function (built-in or your own). You can assume that n is a non-negative integer.

When building this kind of list-based function, you should rely on primitives such as car, cdr, cons and derived functions such as append. Do not use functions such as list-ref which effectively treat the list as an array. Also, do not supply your recursive function with extra input parameters.

You should submit two functions. The first version should output the list in reverse numerical order (largest element first). The second version should output the list in numerical order (smallest element first). Both versions should create the list in the right order. Do not write one version and then use a list reversal function to create the other.

4 Modular arithmetic

Recall the "repeated squaring" method used in class to raise an integer a to selected large powers, specifically the powers that are themselves powers of 2. We started with $a^1 = a$, a^2 and then, for each i, computed $a^{2^{i+1}}$ as the square of a^{2^i}.

In racket, you can use (remainder a k) to find the remainder of a divided by k. Build a recursive function that computes a^{2^i} for the first n values of i. Output them in a list, in increasing numerical order. Then modify it to take an additional input and compute a list of the first n values of remainder(a^{2^i}, k). Your submission should include both versions.

For the second (remainder mod k) version, you should keep intermediate results small by taking the remainder mod k after each major arithmetic step (e.g. after multiplying one number by another). Do not compute all the powers a^{2^i} and then map remainder across the list of results.