CSE1729 - Introduction to Programming October 15, 2018

Laboratory Assignment 7

## Objectives

- Work with "tail recursion"
- Work with higher order functions

## Activities

1. Masahiko Fujiwara showed that 1458 is one of four positive integers (with the trivial case 1) which, when its digits are added together, produces a sum which, when multiplied by its reversal, yields the original number:

$$\begin{aligned} 1+4+5+8 &= 18 \\ 18\times 81 &= 1458 \end{aligned}$$

   You may have completed parts (a) and (b) since you will need them for this week's problem set.

   (a) Define a function named `(explode x)` which converts any positive integer $x$ to a list of single-digit integers (use division by 10 and the floor function). For instance, the expression `(explode 12345)` would return `'(1 2 3 4 5)`. You may only use `cons`, `car`, `cdr`, functions you wrote in lab or are contained in the lecture slides. You may not use string or char functions.

   (b) Define a SCHEME function named `(implode l)` which takes a list of digits in base 10 and converts them to an integer (the inverse operation of `explode`). You may only use `cons`, `car`, `cdr`, functions you wrote in lab or are contained in the lecture slides. You may not use string or char functions.

   (c) Write a function named `(has-property x)` which accepts an integer $x$ and returns `#t` if $x$ has this property and `#f` otherwise. Test your function with the integer 1458. Feel free to use the `sum-list` and `reverse` functions from lecture.

   (d) So far this semester, we have written functions which find the $n^{th}$ number in a sequence which satisfies some property (e.g. the "$n^{th}$ prime number from ..." function from Problem Set 3 ). It would be helpful if we wrote a higher order function which could take a function which generated the `sequence` and a function which `test`ed for the property in which we are interested as well as the integer $n$ and returned the $n^{th}$ value in that `sequence` which satisfied the property (i.e. the `test` function returns true when passed that value).
   Write a function, named `find`, which takes three parameters (`sequence`, `test`, and `n`) and returns the $n^{th}$ value in `sequence` for which the `test` function returns true (`#t`) when passed a value in `sequence`. For example, given a function `prime?` which tests for primality, we can find the 100th prime.

```
>(find (lambda (x) x) prime? 100)
541
```

   (e) Use the `find` function to identify two other integers with Fujiwara's discovered property.
   *Hint: 1 is the first integer with this property and 1458 is the third. You are looking for the second and fourth integers with this property.* You don't need to write a separate function to test these, they will simply be some of the test cases for this part of the lab.

2. **Bubble Sort** is often used to introduce students of Computer Science to sorting algorithms. It's simplicity makes it fairly easy to understand, even though it is not an efficient algorithm for sorting. The idea is to crawl along the list swapping adjacent pairs of elements if the previous of the adjacent elements is larger than the

later adjacent element. In this way, the largest element of the list "bubbles" up to the top. If we repeat this process until no swaps are needed in a single pass through the list, then the list is sorted. We will implement a simpler version of Bubble Sort which makes one pass for each element of the array. This implementation may make more passes than necessary, but it is easier to implement.

(a) Define a SCHEME function to swap the first two elements in a list named `swap-first-two`.

(b) Modify your `swap-first-two` function so it works if there are less than two items in the list.

(c) Define a recursive SCHEME function named `bubble-up` that uses `swap-first-two` to bubble the largest item in a list to the end of the list. This function should crawl along the list, swapping the current element and the next element only if the current element is larger than the next element.

(d) Use `bubble-up` as a helper function to define a SCHEME function named `bubble-sort-aux` that will invoke `bubble-up` once for each element of the list. It should take two parameters, the list and the number of elements in the list.

(e) Define a SCHEME function named `bubble-sort` which takes one parameter, a list to sort, and invokes `bubble-sort-aux` passing it the list and the length of the list. You can use the SCHEME function `length` to find the length of a list. Note: after the $n^{th}$ pass, the top $n$ largest elements have already "bubbled" up and are already at the end of the list. So, the `bubble-up` function can stop after crawling past (length $- n$) elements.