CSE1729 - Introduction to Programming April 11, 2017

Laboratory Assignment 10 Solutions

Objectives

- Work with trees
- Use Binary Search Trees

Activities

1. Define a SCHEME function named `num->list` which converts any integer to a list of single-digit integers (use division by 10 and the floor function). For instance, the expression `(num->list 12345)` would return `'(1 2 3 4 5)`.

Solution:

```
(define (num->list num)
  (if (< num 10)
      (list num)
      (append (num->list (floor (/ num 10)))
              (list (- num (* 10 (floor (/ num 10))))))))
```

2. Define a SCHEME function named `(list->num l)` which takes a list of digits in base 10 and converts them to an integer (the reverse of `num->list`).

Solution:

```
(define (list->num l)
  (define (add-digits l place)
    (if (null? l)
        0
        (+ (* (car l) (expt 10 place)) (add-digits (cdr l) (+ place 1)))))
  (add-digits (reverse l) 0))
```

3. "Don't they teach recreational mathematics anymore"?

Enjoy this clip of the BBC television show *Doctor Who*: http://www.youtube.com/watch?v=ee2If8jSxUo. Ok, now get to work. Your job, should you choose to accept it, is to write a function (and any helper functions necessary) to find the next happy prime in the sequence $\ldots, 313, 331, 367, 379, \ldots$.

Now, what did he say?

> "Any number that reduces to one when you take the sum of the square of it's digits and continue iterating until it yields 1 is a happy number. Any number that doesn't, isn't. A happy prime is a number that's both happy and prime, now TYPE IT IN."

(a) We already have a function from question 1 that explodes an integer into a list of digits.

Solution:

```
(define (num->list num)
  (if (< num 10)
      (list num)
      (append (num->list (floor (/ num 10)))
              (list (- num (* 10 (floor (/ num 10))))))))
```

(b) To compute the sum of the squares of the digits, define a SCHEME function which takes a list of integers as a parameter and returns the sum of the squares of the integers in the list.

Solution:

```
(define (square x) (* x x))
(define (sum-square-of-digits d)
  (if (= 0
         (length d))
      0
      (+ (square (car d))
         (sum-square-of-digits (cdr d)))))
```

(c) If we ever sum the squares of the digits of a number and it produces a number we've already computed, we will always loop through the numbers we've already tried. We'll need to keep track of the numbers we've tried so that we can stop if we come to a number we've already seen before. Define SCHEME functions `insert` and `element?` which implement a set ADT with a binary search tree.

Solution:

```
(define (make-tree value left right)
  (list value left right))

(define (value T) (car T))
(define (right T) (caddr T))
(define (left T)  (cadr T))

(define (element? x T)
  (cond ((null? T) #f)
        ((eq? x (value T)) #t)
        ((< x (value T)) (element? x (left T)))
        ((> x (value T)) (element? x (right T)))))

(define (insert x T)
  (cond ((null? T) (make-tree x '() '()))
        ((eq? x (value T)) T)
        ((< x (value T)) (make-tree (value T)
                                    (insert x (left T))
                                    (right T)))
        ((> x (value T)) (make-tree (value T)
                                               (left T)
                                               (insert x (right T))))))
```

(d) Use all of these helper functions to define a SCHEME function named `is-happy?` which, given a positive integer as a parameter, returns true if the number is a happy number and false otherwise. It might be useful to define a helper function that also takes the set, S, as a parameter and adds the sum of the squares of the digits of a number to that set if it isn't there already.

Solution:

```
(define (is-happy? x)
  (define (is-happy-aux x s)
    (let ((sum (sum-square-of-digits (num->list x))))
      (cond ((= sum 1) #t)
            ((element? sum s) #f)
            (else (is-happy-aux sum (insert sum s))))))
(is-happy-aux x '()))
```

(e) Use your new `is-happy?` function and a function that tests for primality to define a SCHEME function named `happy-prime?` which, given a positive integer as a parameter, returns true if the number is a prime number that is also a happy number and false otherwise.
Recall,

```
(define (prime n)
  (define (divides a b) (= (modulo b a) 0))
  (define (smooth k)
    (and (>= k 2)
         (or (divides k n)
             (smooth (- k 1)))))
  (not (smooth (floor (sqrt n)))))
```

Solution:

```
(define (happy-prime? x)
  (and (is-happy? x)
       (prime x)))
```

(f) Define a SCHEME function named `happy-primes` which takes an integer n as a parameter and returns a list of the first n happy primes. See the following code for a `filter` function. You should see that the happy primes mentioned in the clip are the 18^{th}, 19^{th} and 20^{th}. What is the 21^{st}?

```
(define (find sequence test n)
   (define (find-aux x found)
      (let* ((fx ( sequence x))
             (satisfies-test (test fx)))
        (cond ((and satisfies-test
                    (= (+ found 1) n))
               fx)
```

```
              (satisfies-test (find-aux (+ x 1) (+ found 1)))
              (else (find-aux (+ x 1) found)))))
  (find-aux 1 0))
```

Solution:

```
(define (happy-primes n)
  (define (happy-primes-aux x)
    (if (> x n)
        '()
        (cons (find identity happy-prime? x)
              (happy-primes-aux (+ x 1)))))
  (happy-primes-aux 1))
(happy-primes 21)
```