CSE 1729 - Introduction to Principles of Programming | September 26, 2017

Laboratory Assignment 4

Objectives

- Work with functions that take functions as parameters
- Work with functions that evaluate to functions
- Work with higher order functions

Activities

1. In Problem Set 3 you wrote a function `(harmonic n)` that calculates the n^{th} harmonic number, $H_n = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$. It probably looked something like this:

```
(define (harmonic n)
  (if (= n 1)
      1.0
      (+ (/ 1 n) (harmonic (- n 1)))))
```

   This is a "recursive" process, as it has a pending operation (the addition) that is deferred until the recursive call is finished.

   Write an iterative version of this function `(harmonic-iter n)` that calculates the n^{th} harmonic number. As it is iterative, it should not have pending operations, rather it should carry along the necessary information so you can report the result when you hit the base case. (You saw an example of an iterative function in Problem Set 3, `fast-lucas-help`.

2. Recall the example code from the lecture slides which introduced higher-order functions (functions that take other functions as parameters). You may notice that the base case in the code below has been changed to $n = 1$.

```
(define (sum f n)
    (if (= n 1)
        (f 1)
        (+ (f n) (sum f (- n 1)))))
```

   This code computes $f(1) + f(2) + ... + f(n)$ with f and n passed as parameters.

   Use `sum` to write a new function `(harm-sum n)` that calculates the n^{th} harmonic number, using `sum` to define `harm-sum` by passing it a function `(harm-term k)` that can calculate the kth term in the harmonic series. Use your new function to compute a few harmonic numbers, and use your old function to verify that your answers are correct.

3. As you noticed, the `sum` function in the course lecture slides sums from 0 to n and the function in question 2 sums from 1 to n.

(a) Write a new, more general, summation function named `g-sum`, which takes three parameters, `f` - the function, `a` - the starting index of the summation and `b` the ending index of the summation. For a given function f, and integers a and b,

$$\text{g-sum}(f, a, b) = \sum_{i=a}^{b} f(i)$$

In the above, the sum should be 0 if $a > b$.

(b) Test your generalized version of `sum` with the harmonic numbers in question 2. You should be able to reuse `harm-term`.

(c) We can define a geometric series of the negative powers of 2:

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = \frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \cdots$$

Use `g-sum` to define a function `(geom-series-np2 n)` to calculate the sum of the first $n + 1$ elements of this series. As part of this, define a function `(geometric-term k)` that calculates the $(k + 1)^{th}$ term in this series, that is, $\frac{1}{2^k}$. Do not make `(geometric-term k)` internal to `(geom-series-np2)`.

(d) It is possible to use an *unnamed* function (that is, a lambda form) as a parameter whenever a function is expected. Use an unnamed function with `g-sum` to define a function `(convergent-series a b)`, which calculates the sum of the inverse-squares from a to b, that is

$$\text{convergent-series}(a, b) = \sum_{k=a}^{b} \frac{1}{k^2}$$

4. So far this semester, we have written several functions which find the n^{th} number in a sequence which satisfies some property (e.g. the n^{th} even number). 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).

(a) Write a function, named `find`, which takes three parameters (`sequence`, a function; `test`, a function and `n`, a positive integer) and returns the n^{th} value in `sequence` for which the `test` function returns true (`#t`) when passed a value in `sequence`. `(sequence k)` should return the k^{th} element of the sequence (where k is a positive integer); `(test x)` should return `#t` if the property holds for `x`, `#f` otherwise.

(b) Test your program with by finding the 15^{th} even non-negative integer and the 15^{th} odd non-negative integer. Hint: the first non-negative integer is 0.

(c) A Fibonacci prime is a Fibonacci number which is prime. Use your higher-order function to define a function `(nth-fib-prime n)`, which evaluates to the n^{th} Fibonacci prime. Remember the "smooth" code from the lecture slides.

Note: recall that calculating Fibonacci numbers can be computationally intensive so `(nth-fib-prime n)` can take a long time to calculate for large n. Two options: test

on relatively small n values, or write an iterative routine to calculate Fibonacci numbers. In either case, your function should be able to calculate `(nth-fib-prime 6)`.

5. The composition of two functions f and g, denoted $f \circ g$, is the function defined as

$$(f \circ g)(x) = f(g(x))$$

   (a) Write a Scheme function `(comp f g)` that returns the function $f \circ g$. Once `comp` is defined you would see the following behavior:

```
> (define (double x) (* 2 x))
> (define (add-one x)(+ x 1))
> (define com (comp add-one double))
> (com 3)
7
> ((comp double add-one) 3)
8
```

   Carefully note and understand the syntax!

   (b) Use `comp` to define the `pos-cos` function:

$$\text{pos-cos}(x) = \begin{cases} \cos(x) & \text{if } \cos(x) \geq 0 \\ -\cos(x) & \text{if } \cos(x) < 0 \end{cases}$$

   Key question: what two functions can be composed to produce `pos-cos`?

   (c) Define `square` as follows: `(define (square x)(* x x))`.
   Use `comp` to define the composition of `square` and `sqrt`. How does it behave? Does the order of the functions in the composition matter?

1. Save your work to a file named lab4.rkt
2. Submit your solution file for grading via the Mimir site.