CSE1729 - Introduction to Programming January 29, 2017

Problem Set 2

Remark 1. *When you are asked to hand in* Scheme *code for a function, you may cut-and-paste the definition from your interpreter window into your solution set.* ***Always include with your code a number of illustrative examples.***

1. Recall from class the definition of `number-sum`, which computes the sum of the first n numbers:

```
(define (number-sum n)
  (if (= n 0)
      0
      (+ n (number-sum (- n 1)))))
```

   (a) Adapt the function so that it computes the sum of the first n positive squares. (So your function, when evaluated at 4, should return the sum of the first 4 positive perfect squares: $1+4+9+16 = 30$.)

   (b) Adapt the function so that it computes the sum of the first n even numbers. (So your function, when evaluated at 4, should return the sum of the first 4 even numbers: $2+4+6+8 = 20$.)
   Incidentally, evaluate your function at 1, 2, 3, 4, 5, 6, and 7. Does this sequence of numbers look familiar?

2. Write a recursive function that, given a positive integer k, computes the product

$$\underbrace{\left(1-\frac{1}{2}\right)\left(1-\frac{1}{3}\right)\cdots\left(1-\frac{1}{k}\right)}_{k-1}.$$

   (Experiment with the results for some various values of k; this might suggest a simple non-recursive way to formulate this function.)

3. Consider the problem of determining how many divisors a positive integer has. For example:

   - The number 4 has three divisors: 1, 2, and 4;
   - The number 5 has two divisors: 1 and 5;
   - The number 10 has four divisors: 1, 2, 5, and 10.

   In this problem you will write a Scheme function `(divisors n)` that computes the number of divisors of a given number n.

   The first tool you will need is a way to figure out if a given whole number ℓ divides another whole number n evenly. We provide the code for this, which you can just use as-is in your solution (it involves a function that we haven't talked about in class yet):

```
(define (divides a b) (= 0 (modulo b a)))
```

   Once you have defined this function, `(divides a b)` will be `#t` if a divides b evenly, and `#f` if not. For example:

```
> (divides 2 4)
#t
> (divides 3 5)
#f
> (divides 6 3)
#f
```

At first glance, the problem of defining `(divisors n)` appears a little challenging, because it's not at all obvious how to express `(divisors n)` in terms of, for example, `(divisors (- n 1))`; in particular, it's not really clear how to express this function recursively.

To solve the problem, you need to introduce some new structure! Here's the idea. Focus, instead, on the function `(divisors-upto n k)` which computes the number of divisors n has between 1 and k (so it computes the number of divisors of n upto the value k). Now you will find that there is a straightforward way to compute `(divisors-upto n k)` in terms of `(divisors-upto n (- k 1))`. Specifically, notice that

$$\texttt{(divisors-upto n k)} = \begin{cases} 0 & \text{if } k = 0; \\ 0 & \text{if } n = 0; \quad \text{(otherwise, } n \geq 1\text{, and)} \\ 1 & \text{if } k = 1; \\ 1 + \texttt{(divisors-upto n (- k 1))} & \text{if } k \text{ divides } n; \\ \texttt{(divisors-upto n (- k 1))} & \text{if } k \text{ does not divide } n. \end{cases}$$

Write the Scheme code for the function `divisors-upto`; notice then that you can define

```
(define (divisors n) (divisors-upto n n))
```

In this case, we call `divisors-upto` a "helper" function. What did it do? It let us "re-structure" the problem we wish to solve in such a way that we can recursively decompose it.

4. Write a function that, given a positive integer k, returns the sum of the first k terms of the infinite series:
$$\frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \cdots .$$
Use your function to sum the first 100 terms of this series.

   Observe that signs alternate in this series; one easy way to implement an alternating sign is to use the function $(-1)^\ell$, which is 1 when ℓ is even, and -1 when ℓ is odd. You may wish to use the built-in Scheme function `(expt x t)`, which returns x^k (and so provides a straightforward way to compute $(-1)^\ell$).

   Depending on how you wrote your code, Scheme may have produced *exact* output of the form $^a/_b$. To coerce Scheme to give you an approximation in decimal form, change the constant `4` in your code to `4.0`.

   Now compute the sum of the first 100,000 terms. Does this number look (roughly) familiar?

5. Consider the definition of your last function.

   (a) To compute the 300 terms, how many calls to `expt` were made? What are the actual values passed to each call?

   (b) Revise the function you just wrote to eliminate the repeated invocations of `expt`. (Your function should not use `expt` at all but somehow compute the signs on its own.) You can introduce a helper function with more arguments if you wish!

6. (cf. SICP problem 1.4) Suppose we designed a new `if` function as follows:

```
(define (new-if predicate then-clause else-clause)
        (if predicate then-clause else-clause))
```

   Check that `new-if` works as you might expect by evaluating:

```
> (new-if (= 0 0) 4 5)
4
> (new-if (= 0 1) 4 5)
5
```

Recall now the `factorial` function that we defined and discussed in class:

```
(define (factorial n)
    (if (= n 0)
        1
        (* n (factorial (- n 1)))
    )
  )
```

How does `factorial` behave if you replace the usage of `if` with `new-if`? Explain.

7. Recall from high-school trigonometry the sin function: If T is a right triangle whose hypotenuese has length 1 and interior angles x and $\pi/2 - x$, $\sin(x)$ denotes the length of the edge opposite to the angle x (here x is measured in radians). You won't need any fancy trigonometry to solve this problem.

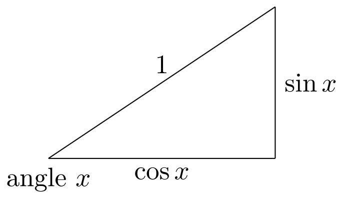

Figure 1: A right triangle.

It is a remarkable fact that for all real x,

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

(a) Write a Scheme function `new-sin` so that `(new-sin x n)` returns the sum of the first $(n+1)$ terms of this power series evaluated at x. Specifically, `(new-sin x 3)`, should return

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

and, in general, `(new-sin x n)` should return

$$\sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{(2k+1)!} .$$

You may use the built-in function `(expt x k)`, which returns x^k. It might make sense, also, to define `factorial` as a separate function for use inside your `new-sin` function. (Aesthetic hint: Note that the value $2k$ is used several times in the definition of the kth term of this sum. Perhaps you can use a `let` statement to avoid computing this quantity more than once?)