CSE 1729 - Introduction to Principles of Programming	September 19, 2017
Laboratory Assignment 3	

Objectives

- Work with recursive functions
- Learn to work with others

Your code should be saved as file lab3.rkt, and submitted to Mimir. Be careful with the names, and good luck!

Activities

1. Young Jeanie knows she has two parents, four grandparents, eight great grandparents, and so on.

   (a) Write a recursive function `(num-in-gen n)` to compute the number of Jeanie's ancestors in the n^{th} previous generation without using the `expt` function. The number of ancestors in each generation back produces a sequence that may look familiar:

   $$2, 4, 8, 16, \ldots$$

   For each generation back, there are twice the number of ancestors than in the previous generation back. That is, $a_n = 2a_{n-1}$. Of course, Jeanie knows she has two ancestors, her parents, one generation back.

   (b) Write a recursive function to compute Jeanie's total number of ancestors if we go back n generations. Specifically, `(num-ancestors n)` should return:

   $$2 + 4 + 8 + \cdots + a_n$$

   Use your function in part (a) as a "helper" function in the definition of `(num-ancestors n)`[1].

2. Perhaps you remember learning at some point that $\frac{22}{7}$ is an approximation for π, which is an irrational number. In fact, in number theory, there is a field of study named Diophantine approximation, which deals with rational approximation of irrational numbers. The Pell numbers are an infinite sequence of integers which correspond to the *denominators* of the closest rational approximations of $\sqrt{2}$. The Pell numbers are defined by the following recurrence relation (which looks very similar to the Fibonacci sequence):

   $$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ 2P_{n-1} + P_{n-2} & \text{otherwise} \end{cases}$$

[1] Of course, we can use the closed-form solution for the geometric progression to compute `num-ancestors` ($ancestors(n) = 2^{n+1} - 2$) but that doesn't give us any experience with recursive functions. However, this is a useful fact we can use when testing our functions to ensure they are correct.

(a) Use this recurrence relation to write a recursive function, `pell-num`, which takes one parameter, n, and returns the n^{th} Pell number.

(b) The *numerator* for the rational approximation of $\sqrt{2}$ corresponding to a particular Pell number is half of the corresponding number in the sequence referred to as the *companion Pell numbers* (or Pell-Lucas numbers). The companion Pell numbers are defined by the recurrence relation:

$$Q_n = \begin{cases} 2 & \text{if } n = 0 \\ 2 & \text{if } n = 1 \\ 2Q_{n-1} + Q_{n-2} & \text{otherwise} \end{cases}$$

Use this recurrence relation to write a function, named `comp-pell-num`, which returns the n^{th} companion Pell number.

(c) Finally write a function `(sqrt-2-approx n)` that uses the Pell number and companion Pell number functions to compute the n^{th} approximation for $\sqrt{2}$. Use your new function to compute the approximation for $\sqrt{2}$ for the sixth Pell and companion Pell numbers.

3. **Binary exponentiation** Consider the following function, `(power base exp)`, that raises a number (`base`) to a power (`exp`):

```
(define (power base exp)
  (cond ((= exp 0) 1)
        (else
         (* base (power base (- exp 1))))))
```

(a) There are more efficient means of exponentiation. Design a Scheme function `fastexp` which calculates b^e for any integer $e \geq 0$ by the rule:

$$b^e = \begin{cases} 1 & \text{if } e = 0, \\ (b^{\frac{e}{2}})^2 & \text{if } e \text{ is even}, \\ b * (b^{\frac{e-1}{2}})^2 & \text{if } e \text{ is odd}. \end{cases}$$

You may find it useful to define `square` as a separate function for use inside your `fastexp` function. You may also want to try the `even?` and `odd?` functions defined in Scheme.

(b) Show that the `fastexp` function is indeed faster than the power function by comparing the number of multiplications that must be done for some exponent e in both functions. (You can assume the exponent e is of the form 2^k.)

4. It is an interesting fact the the square-root of any number may be expressed as a *continued fraction.* For example,

$$\sqrt{x} = 1 + \cfrac{x-1}{2 + \cfrac{x-1}{2 + \cfrac{x-1}{\ddots}}}$$

(a) We can rewrite the above equation as

$$\sqrt{x} - 1 = \cfrac{x-1}{2 + \cfrac{x-1}{2 + \cfrac{x-1}{\ddots}}}$$

and let its right-hand-side be the continued fraction we want, then the following recurrence relation describes our approximation:

$$\text{cont-frac}(k, x) = \begin{cases} 0 & \text{if } k = 0, \\ \frac{x-1}{2+\text{cont-frac}(k-1,x)} & \text{otherwise} \end{cases}$$

Given this recurrence, write a function `(cont-frac k x)` that computes it.

(b) Next, write a Scheme function called `new-sqrt` which takes two formal parameters x and n, where x is the number we wish to find the square root of and n is the depth of the fraction computed (that is, the k in the `cont-frac` function). Demonstrate that for large n, `new-sqrt` is very close to the builtin `sqrt` function. Use `cont-frac` (as you defined above) as a helper function, but do not define it within new-sqrt.