CSE 341, Spring 2004, Assignment 4
Due: Friday 7 May, 9:00AM

You will write several Scheme implementations of the fibonacci function. (Problem 6 has nothing
to do with fibonacci.) The fibonacci function is defined only for positive whole numbers; your functions may
assume they are passed such numbers. By definition:

- \( \text{fibonacci}(1) = \text{fibonacci}(2) = 1 \)
- For \( n > 2 \), \( \text{fibonacci}(n) = \text{fibonacci}(n - 1) + \text{fibonacci}(n - 2) \)

You may not use mutation (e.g., \texttt{set!}) except to implement “memo tables” in problems 4 and 5.
Note: Scheme’s “exact numbers” never overflow, so solutions 2 and 4, should be able to compute “really
big” fibonacci numbers. (Solutions 1 and 3 can too, but it takes too long.)

1. Define \texttt{fibonacci1} to compute fibonacci numbers. Your solution must not use any helper functions.
   (It will be really inefficient, but that’s okay.)

2. Define \texttt{fibonacci2} to compute fibonacci numbers. Your solution must use an accumulator-style helper
   function to be efficient. Hints:
   - Do not use the helper function when the input to \texttt{fibonacci2} is 1 or 2.
   - Have the helper function work from smaller numbers to larger ones.
   - Pass two accumulators, one for \( \text{fibonacci}(n - 1) \) and one for \( \text{fibonacci}(n - 2) \).

3. Define \texttt{fibonacci3} to compute fibonacci numbers. Your solution must maintain an \textit{association list}
   (a list of pairs) as a memo table of previously computed answers. If an answer has been previously
   computed, you must find it in the list. If not, you must call \texttt{fibonacci1}, mutate the memo table to
   hold a new pair, and return the correct answer. (Note: This function will be inefficient for a large
   \( n \) the first time it is called with \( n \).

4. Define \texttt{fibonacci4} to compute fibonacci numbers. Your function must use a recursive helper function
   that maintains an association list as a memo table. If the helper function has been previously called
   with the same argument, it must find the answer in the memo table. The helper function must not
   use an accumulator (nor call other functions that do). In particular, the helper function should take
   only one argument. (Note: This function will be efficient, even without accumulators!)

5. Define \texttt{fibonacci5} to compute fibonacci numbers \textit{approximately}: In math, it turns out we can compute
   fibonacci numbers without recursion: the \( n^{th} \) fibonacci number is
   \[
   \frac{1}{\sqrt{5}} \left( \left( \frac{1 + \sqrt{5}}{2} \right)^n - \left( \frac{1 - \sqrt{5}}{2} \right)^n \right)
   \]
   Scheme provides primitives \texttt{sqrt} and \texttt{expt} for square-roots and exponents. However, \texttt{sqrt}
   produces a floating-point number (called in Scheme an “inexact” number) and math operations with one or
   more floating-point operands convert the other operands to floating-point and produce a floating-point
   result. Therefore, your answer is subject to (1) rounding errors and (2) getting too big.

To deal with these problems, your solution must:

- Use the \texttt{floor} primitive to round the answer down to a whole number.
- If the result is \texttt{+inf.0}, return \texttt{+inf.0}.
- If the result is not \texttt{+inf.0}, use the \texttt{inexact->exact} primitive to convert the (rounded) answer
to a non-floating point number.

Note: The \texttt{+inf.0} possibility means your function may return an inexact number or an exact number.
6. Define a function alternate that takes no arguments and produces a stream of alternating booleans 
#t #f #t #f #t ... In particular, (alternate) should return a pair of #t and a stream of alternating 
booleans starting with #f. So (car ((cdr (alternate)))) should be #f. Your solution must not use 
mutation. Hints:
• Use mutual recursion.
• This is a different flavor of stream than we investigated in class because the second part of the 
pair is not given the previous stream element. So stream.sml from class doesn’t help.
• Sample solution is 4 lines.

7. (Extra Credit) Define a function exactness-table for computing the inaccuracy of fibonacci5.
Your function should take two positive integers, lo and hi and return a list of lists, where the outer 
list has hi-lo+1 elements and the n\textsuperscript{th} element has the inaccuracy data for lo+n-1. (In other words, 
it computes the data for every integer between lo and hi inclusive and the list is increasing order.) 
Each inner list should have 5 elements:
• The first element should be i, the number we’re computing fibonacci for.
• The second element should be the result of (fibonacci4 i).
• The third element should be the result of (fibonacci5 i).
• The fourth element should be the absolute-value of the difference between the second and third 
elements, or the string “n/a” if the third element is +inf.0.
• The fifth element should be the least j such that \textsuperscript\textdaggerdiff \cdot 10^j > exact where diff is the third 
element and exact is the first element. However, if diff is 0, the fifth element should be the 
string “inf” and if the third element is +inf.0, the fifth element should be the string “n/a”. (Note 
j is roughly how many significant digits the third element has.)

Turn-in Instructions
• Put all your solutions in one file, lastname_hw4.scm, where lastname is replaced with your last name.
• Line 1 of your .scm file should include a Scheme comment with your name and the phrase homework 
4.
• Email your solution to martine@cs.washington.edu.
• The subject of your email should be exactly [cse341-hw4].
• Your .scm file should be an attachment.