CSE 341, Winter 2004, Assignment 4
Due: Monday, February 14 at 10:00pm

Last updated: 7 February

You will write several functions to compute binomial coefficients. (Problem 6 has nothing to do with binomial coefficients). A *binomial coefficient*, denoted as $C(n, k)$, is the number of ways of picking k distinct elements from a set of n distinct elements. For example, if I have a deck of 52 playing cards, then there are $C(52, 5) = 2,598,960$ different combinations of 5 cards (that's a lot of poker hands!). By definition:

- $C(n, 0) = 1$, and $C(n, k) = 1$ if $n = k$
- $C(n, k) = C(n-1, k-1) + C(n-1, k)$

We make the assumption that $n, k \geq 0$ and that $n \geq k$ for all of the problems below. This recursive definition of $C(n, k)$ is called *Pascal's Identity*, and is the basis for a geometric arrangement of the binomial coefficients in a triangle known famously as *Pascal's Triangle*. For $n \geq 0$, the *nth* row in this triangle consists of the binomial coefficients $C(n, k)$ for $k = 0, 1, 2, ...n$. For example, the first 4 rows (from top to bottom) of Pascal's Triangle, from $n = 0$ to $n = 3$, are shown below:

```
            1
        1       1
    1       2       1
1       3       3       1
```

Pascal's Identity shows that when two adjacent binomial coefficients in a row in this triangle are added, the binomial coefficient in the next row between these two coefficients is produced!

For the following problems, you may not use mutation (e.g., set!) except to implement memo tables in problems 2 and 3. Unless explicitly stated otherwise, no problem's solution may "cheat" by using another problem's solution as a helper function.

1. Define a function `binomial1` such that `(binomial1 n k)` computes the binomial coefficient `C(n,k)`. Your solution must compute `C(n,k)` recursively using Pascal's Identity, and may *not* use any helper functions. (It will be really inefficient (think why?), but that's ok.).

2. Define a function `binomial2` such that `(binomial2 n k)` computes the binomial coefficient `C(n,k)`. Your solution must maintain an *association list* (a list of triples) 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 `binomial1`, mutate the memo table to hold a new triple, and return the correct answer. Your memo table may *not* be a top-level binding. Hint: use Scheme's built-in `assoc` function.

3. Define a function `binomial3` such that `(binomial3 n k)` computes the binomial coefficient `C(n,k)`. 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. Your memo table may *not* be a top-level binding. Hint: make use of the helper function's closure.

4. Define a function `binomial4` such that `(binomial4 n k)` computes the binomial coefficient `C(n,k)`. Your solution *must* compute `C(n,k)` by computing Pascal's Triangle. Hints:

   - Write a helper function that, when given the ith row of Pascal's Triangle as a list, computes the (i+1)th row as a list.
   - Compute $C(n, k)$ by returning the kth element in the nth row of Pascal's Triangle. Use Scheme's built-in `(list-ref lst i)` function which returns the ith element in a list `lst`.

5. It turns out that there is a closed-form solution for computing `C(n,k)` directly. Formally:

$$C(n,k) = \frac{n!}{k!(n-k)!} \tag{1}$$

   where $k!$ means "k factorial." Define a function `(binomial5 n k)` that computes `C(n,k)` directly using this equation.

6. Define a function `cycle` that takes no arguments and produces a stream of the integers 1,2, and 3 such that the stream cycles as 1 2 3 1 2 3 1... In particular, `(cycle)` should return a pair of 1 and a stream of cycling integers starting with 2. So `(car ((cdr (cycle))))` should be 2. Your solution must *not* use mutation.

Turn-in Instructions

- Put all your solutions in one file, hw4.scm.
- Line 1 of your `.scm` file should include a Scheme comment with your name and the phrase `homework 4`.
- Use the turn-in form linked from the course website to turn your hw4.scm file in.