Logistics

This homework is officially due in class on **Wednesday, March 5**. However, it comes with an automatic extension: anything submitted by **5pm on Friday, March 7** will be accepted as being on time.

You should work singly or in pairs on this assignment.

You should turn in a hard copy of your source code and a transcript demonstrating convincingly that your code is correct. You should also submit your code electronically.

Exercises (due Wednesday, March 5)

1. Use `foldr` to write a Haskell function

   ```haskell
events :: (a -> Bool) -> [a] -> Bool
```

   such that `exists pred lst` returns `True` if there is some element of `lst` that satisfies the predicate `pred`; otherwise, the function should return `False`. For example, `exists even [1..3]` should return `True`, but `exists (\ x -> x>10) [1..3]` should return `False`.

   Specifically, what I’m looking for is an answer of the following form, where you fill in the blanks:

   ```haskell
   exists pred lst = foldr ___________ ___________ ___________
   ```

2. Use `foldr` to write a one-equation Haskell function

   ```haskell
   rev :: [a] -> [a]
   ```

   such that `rev lst` reverses the list `lst`. For example, `rev "abcde"` should return "edcba".

   Specifically, what I’m looking for is an answer of the following form, where you fill in the blanks:

   ```haskell
   rev lst = foldr ___________ ___________ ___________
   ```

3. Write a Haskell function

   ```haskell
   cartProd :: [a] -> [b] -> [(a,b)]
   ```

   such that `cartProd xs ys` returns the list of pairs obtained by combining all elements of `xs` with all elements of `ys`. For example, `cartProd [1..3] "abc"` should return the list `[(1,'a'),(1,'b'),(1,'c'),(2,'a'),(2,'b'),(2,'c'),(3,'a'),(3,'b'),(3,'c')]`.

   To receive maximal credit for this problem, your code should be a one-equation function, with no helper functions. (*Hint: map* is very useful here; a more complex solution uses `foldr`.) However, you will receive substantial credit for any solution that works correctly.

4. Recall the polynomials of Assignment 4.

   Use `foldr` to write a one-equation Haskell function
evalPoly :: Int -> Poly -> Int

such that evalPoly m p returns the value of the polynomial p when x is m.
Specifically, what I'm looking for is an answer of the following form, where you fill in the blanks:

    evalPoly m p = foldr _______________ ___________ __________________

Note: This question is more difficult than the others. You may also find it useful to use some built-in functions we’ve talked about in class, such as length and zip.