Reading
You should read (if you haven’t already) the first two chapters of *Introduction to Functional Programming using Haskell* (IFPH).

Logistics
This homework is officially due in class on **Wednesday, February 5**. However, it comes with an automatic extension: anything submitted by **noon on Thursday, January 23** will be accepted as being on time.

You should work **alone** on this assignment. You should turn in a hard copy of your source code and a transcript demonstrating convincingly that your code is correct. For this assignment, you do not need to submit your code electronically.

Exercises (due Wednesday, February 5)
Note that Questions 1, 2, and 4 involve no writing of code.

1. IFPH, Exercise 1.4.1
   Give the type declaration for $h$. Also, for each of the three statements, either give a proof of its validity or explain why the validity fails (a counterexample suffices).

2. IFPH, Exercise 1.6.1

3. IFPH, Exercise 1.6.2
   In addition to the Haskell code for `swap`, give a proof of the stated equality.

4. IFPH, Exercise 2.4.1
   Note that the definition of `cross` appears on page 42.

5. IFPH, Exercise 2.4.3

6. Don’t be fooled by the long description that follows: this question doesn’t require much code (only two or three lines are necessary). Instead, it just requires a little thought about what is being asked.

   **Background terminology:** In the description that follows, the word *function* refers to mathematical functions over tuples of nonnegative integers.

   Let $g$ be a $n$-argument function and let $h$ be a $(n+2)$-argument function. Now, consider the $(n+1)$-argument function $f$ defined as follows:
   
   (a) $f(x_1, x_2, \ldots, x_n, 0) = g(x_1, \ldots, x_n)$
   
   (b) $f(x_1, x_2, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y))$ (for $y \geq 0$)

   The function $f$ is said to be **obtained from $g$ and $h$ by primitive recursion**.

   **An example:** Consider the case where $n = 1$ and $g$ and $h$ are the functions

   \[ g(x) = x, \quad h(x, y, z) = z + 1. \]

   The function obtained from $g$ and $h$ by primitive recursion is `add`, where:
(a) \( \text{add}(x,0) = g(x) = x \)
(b) \( \text{add}(x, y+1) = h(x, y, \text{add}(x, y)) = \text{add}(x, y) + 1 \)

This function is nothing other than standard nonnegative-integer addition.

**The problem:** Write a Haskell function

\[
\text{primRec} :: \text{(Int -> Int) -> ((Int,Int,Int) -> Int) -> ((Int,Int) -> Int)}
\]

such that \( \text{primRec} \ g \ h \) returns the function obtained from \( g \) and \( h \) from primitive recursion.

For example, \( \text{primRec} (\ \lambda x \to x) (\ \lambda (x,y,z) \to z+1) \) returns the function that accepts a pair of integers and adds them together (i.e., it’s the function \( \text{add} \) defined above).