Recursion

This document contains the first hand-in assignment ("lab") for the course. It's a small one just to get the ball rolling. See the course home page for the deadline (it is soon!).

We suggest that you find a lab partner and work together in pairs. (**When you are done, please submit your solution using the Fire system.**)

Please note that lab assignment 2 and onwards **must** be submitted by groups of exactly two people.

In this lab assignment, you will implement the well-known "power" function in two different new ways. The power function takes two arguments n and k and computes n^k. Your implementation only has to work for non-negative k.

You have possibly seen one implementation of this function in the lecture. If not here it is:

```
power :: Integer -> Integer -> Integer
power n k | k < 0 = error "power: negative argument"
power n 0 = 1
power n k = n * power n (k-1)
```

You will implement two more ways in this lab assignment.

Part 1

In order to calculate "power n k", for a given n and k, how many computing "steps" are being used?

Hint: "power n 0" takes 1 step.
"power n 1" takes 1 step, and then uses "power n 0".
"power n 2" takes 1 step, and then uses "power n 1".
"power n 3" takes 1 step, and then uses "power n 2".
And so forth.

Note: Please make sure you follow the **submission guidelines** when you write your code.

Part 2

A different way of computing the power function is to use the standard Haskell function "product", which calculates the product (multiplication) of all

elements in a list.

To calculate "power n k", first construct a list with k elements, all being n, and then use "product".

Implement this idea as a Haskell function "power1".

Hint: You have to come up with a way of producing a list with k elements, all being equal to n. Use a list comprehension, or use the standard Haskell function "replicate". If you use "replicate", you might want to use the function "fromInteger" too! Use **Hoogle** to find out more about standard functions (and also to search for standard functions by their type).

Part 3

A different approach to calculating the power function uses fewer computing steps.

We use the fact that, if k is even, we can calculate n^k as follows:

$n^k = (n^2)^{k/2}$ (k is even)

In other words:

$n^k = (n * n)^{k/2}$ (k is even)

So, instead of recursively using the case for k-1, we use the (much smaller) case for k/2.

If k is not even, we simply go one step down in order to arrive at an even k, just as in the original definition of power:

$n^k = n * (n^{k-1})$ (k is not even)

To sum up, to calculate "power n k":

- If k is even, we use $(n * n)^{k/2}$
- If k is odd, we use $n*(n^{k-1})$

Implement this idea as a Haskell function "power2".

Hints:

- Do not forget to add a base case (what do you do when k=0?)
- You need to find out when numbers are even or odd. Use the standard Haskell functions "even" and/or "odd".
- To divide integer numbers, use the function "div" (and not the function "/", which is used to divide floating point and rational numbers)

Part 4

We would like the three functions "power", "power1", and "power2" to

calculate the same thing. It is probably a good idea to test this!

A. Come up with a number of test cases (inputs you will test your functions on). Argue why you have chosen these test cases. (Think about for what inputs the functions are defined, and for what inputs the functions are not defined.)

B. Implement a function "prop_powers" which given n and k checks that "power n k", "power1 n k", and "power2 n k" all give the same answer.

C. Write all the test cases you suggested in part A as a Haskell function that performs all test cases. It is probably a good idea to use the function that you defined in part B.

Hint: You can use a list comprehension to combine all possible cases you would like to test for n and k. Use the standard Haskell function "and" to combine the results. If you are not familiar with list comprehensions, you do not have to use these.

D. Try running quickcheck on the function prop_powers. It will probably fail. If so, define a new version, prop_powers', for which quickcheck succeeds. The new version should of course still test the intended property!

Submission Guidelines

See the course home page for lab deadlines.

Write your answers in one file, called **Lab1.hs**. For each part, use Haskell comments to indicate what part of the file contains the answer to that part. For answers in natural language, use English; write your answers also in Haskell comments. Remove irrelevant things from the file.

Before you submit your code, Clean It Up! After you feel you are done, spend some time on cleaning your code; make it simpler, remove unnecessary things, etc. We will reject your solution if it is not clean. Clean code:

- Does not have long lines (< 78 characters)
- Has a consistent layout
- Has type signatures for all top-level functions
- Has good comments
- Has no junk (junk is unused code, commented code, unneccessary comments)
- Has no overly complicated function definitions
- Does not contain any repetitive code (copy-and-paste programming)

Feel free to use the hlint program to help with many of these issues and other haskell style issues. For future labs we will expect you to run your code through hlint before submitting. **When you are done, please submit it using the Fire system.**

(If you get a warning that the security certificate of the Fire server is out of date you will have to accept it anyway. Hopefully this will be fixed soon.)

Good luck!