Functional Programming

Coursework 1 - Power to the People

Introduction

In this coursework assignment, you will implement the well-known *power* function in two different ways. The *power* function takes two integer arguments, n and k, and computes n^k. Your implementations only have to work for non-negative k.

Downloading the skeleton code

1. Download the skeleton code zip file by going to this link.
2. Extract the contents of the zip to wherever you want to save it
   - How to extract zip files on Windows
   - How to extract zip files on Ubuntu
   - How to extract zip files on Mac

Compiling and running tests

If your code passes all the tests, you get all the marks! It's therefore essential that you can compile and run the tests. To do this:

1. In VSCode (in WSL mode if you're on Windows), open the folder containing the skeleton code
2. Open an integrated terminal in the VSCode window either by using the menu bar ("Terminal -> New Terminal") or by using the shortcut (Ctrl + Shift + `).
3. Run `cabal update` (this will update your package list for Haskell libraries)
4. Run `cabal build` (this will build your project and the tests, which may take a little while the first time)
5. Run `cabal test` (this will run the tests)

When you run this on the skeleton code all the tests should fail, because nothing has been implemented yet.

Every time you make a change, run `cabal test` to test it! You can also load `Power.hs` in GHCi/interactive mode using `cabal repl`

Write all your code in `Power.hs`, but don't change the name of the functions and don't change any other files. You are free to write any helper functions you like, as long as you write them in `Power.hs`.

Submitting your coursework

Submit your work via Blackboard. You should submit *only* `Power.hs`. Do not rename `Power.hs`. Do not submit any other files. Do not submit a zip file. You are responsible for early submission, thus submit *at least* an hour before the deadline to make sure there are no upload problems. The university systems will automatically apply quite severe penalties if the coursework is late by even a second.

Task 1

Look at the *power* implementation provided for you in `Power.hs`. In order to calculate *power n k*, for a given *n* and *k*, how many computational "steps" are required?

Write your answer in the form of a function definition:

stepsPower n k = ...

Your function *should not* be recursive.

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.

...

Task 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 the function *product* to multiply them all together.

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*. You could use the standard Haskell function *replicate* to do this. If you do, you might want to use the function *fromInteger* too! Use Hoogle to find out more about standard functions.

Task 3

A different approach to calculating the power function uses fewer computational 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:

$n^k = (n \cdot n)^{k/2}$ (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:

$n^k = n \cdot (n^{k-1})$ (k is odd)

In summary, to calculate *power n k*:

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

Implement this idea as a Haskell function *power2*.

Hints:

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

Task 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!

Implement two functions:

- A function *comparePower1* that, given *n* and *k*, compares the result of the *power* function with your *power1* function;
- A function *comparePower2* which does the same for *power* and *power2*.

These functions should return *True* when both implementations give the same result, and *False* if both implementations run without an error but do not give the same result.

Task 5

Complete the *comparisonList* function. It takes a list of *n*s and a list of *k*s to use as test cases. It returns a list with a test result for every combination of the given test values (a.k.a. the cartesian product of *n*s and *k*s). Each test result should be in the format: (n, k, result of *comparePower1* for that n and k, result of *comparePower2* for that n and k).

For example, if your *power1*, *power2*, *comparePower1*, and *comparePower2* functions all work correctly, then *comparisonList* $[2,3]$ $[0,1]$ should give you this result:

$$
\begin{aligned}
&\textit{comparisonList}\ [2,3]\ [0,1] \\
&\quad = [(2,0,\textit{True},\textit{True}) \\
&\quad\ \ ,(2,1,\textit{True},\textit{True}) \\
&\quad\ \ ,(3,0,\textit{True},\textit{True}) \\
&\quad\ \ ,(3,1,\textit{True},\textit{True})]
\end{aligned}
$$

If instead everything works correctly except your *power2* doesn't give the correct result when k = 0, then *comparisonList* $[2,3]$ $[0,1]$ should give you this result:

$$
\begin{aligned}
&\textit{comparisonList}\ [2,3]\ [0,1] \\
&\quad = [(2,0,\textit{True},\textit{False}) \\
&\quad\ \ ,(2,1,\textit{True},\textit{True}) \\
&\quad\ \ ,(3,0,\textit{True},\textit{False}) \\
&\quad\ \ ,(3,1,\textit{True},\textit{True})]
\end{aligned}
$$

This result would tell you *power2* fails for the test cases (n = 2, k = 0) and (n = 3, k = 0).

Hint: You can use a list comprehension to combine all cases *n* and *k*.