Recommended exercises with hints to Chapters 1-4

– D7012E Declarative programming –

Håkan Jonsson

March 27, 2018

1 How to work

- Sit by a computer with your book, pen, and paper.
- Work on the exercises one at a time and in order.
- For each exercise:

    1. Read about the exercise in the course book. Make sure you (at least) sort of understand it. Start sketching a solution, on paper if you like[1].
    2. Read my hints and (hopefully) clarifying comments. Sketch some more but no need to finalize the solution. It is enough that you come so far in thinking out a solution that you *could* write down the whole solution, if you were asked to do so.
    3. Implement the solution and test it. My advice is that you always write type signatures of functions. (Trust me, you really want to avoid the error messages `ghci`, or your Haskell system, produces when it detects something strange with your types...).
    4. Be happy! :)

[1] Note: The final test is pen and paper only so it is wise to start practicing...

2 Chapter 3

Exercise 3.7 on page 37

A simple practice exercise to start with. Write two different programs `threeDifferent1` and `threeDifferent2` one of which makes use of

- guards while the other makes use of
- conditional expressions.

Exercise 3.8 on page 37

What is a good exercise here is to first define `twoEqual :: Int -> Int -> Int` and use it to define `threeEqual :: Int -> Int -> Int -> Int`. Then use (any combination of) these two to define `fourEqual :: Int -> Int -> Int -> Int -> Int`.

Exercise 3.17 on page 46

This is a classical exercise. As written in the book it asks for two functions each of which returns one of the two roots a quadratic equation has (the $\pm$ in the equations gives the different roots).

It is fine to solve the exercise as given *but* if you know tuples a better version would be just one function (not two) that returns a pair `((Float,Float), (Float,Float))` with the two roots as *complex numbers*. Each component (a pair `(Float,Float)` then has a real part and imaginary part. Then, it is always possible to return the true roots.

3 Chapter 4

Exercise 4.7 on page 64

For natural numbers, $a \times b = b + (a - 1) \times b$, where $\times$ denote multiplication. For what numbers a should the function be defined? What is the base case?

Exercise 4.8 on page 64

The function will recursively call itself with a number that is increased by one in every call (so it is $1, 2, 3, \ldots$) until the square of the number is too large. At that moment, the number has been increased one time too many.

Exercise 4.9 on page 64

`f :: Int -> Int` is a global function that has previously been written. We need not bother with how it is defined, by who etc. Our task is to write (another) function that computes the maximum of `f 0`, `f 1`, ..., `f n`.

Find a way to generate the list of numbers `[0, 1, 2, ..., n]`. Use either a list comprehension or a (primitive) linear recursive function you write yourself. This gives us the list `[f 0, f 1, f 2, ..., f n]`. Left to do is to find the maximum value in this list, which can be done by a separate linear recursive function.

Exercise 4.14 on page 67

The function typically uses guards to separate two cases: One when `n` is even and one when it is odd. We also need to do integer division. You have all the operators and functions available on values of type `Int` in Chapter 3.

The name of the mathematical function is implicit, so you need to find a name of the function yourself, say `powerOfTwo`, so 2^n can be written `powerOfTwo n`.