School of Computer Science, University of Nottingham
G51FUN Functional Programming, Spring 2015
Graham Hutton

Coursework I, Part 3/5

Thursday 5th February 2015

Deadline: Monday 23rd February 2015, 11am

This exercise sheet covers material from Lecture 4 (Defining Functions), and is worth 20% of the first coursework. Each question is worth a quarter of the marks. You should attempt to complete the exercises in your own time, but if you get stuck you can ask for help during the lab sessions.

Assessment will be carried out by oral examination during the lab sessions (nothing needs to be handed in). When you have completed the exercises you should ask a tutor to examine your solution. The tutor will then ask you some questions to test your understanding. You will not receive any marks if you cannot explain your solution. Note that tutors will only be available to assess your solution during the official lab session (Mondays, 9am to 11am, A32).

Ensure that your script type checks before submitting it to a tutor for assessment. If you are unable to complete some of the definitions then comment out those that are incomplete.

1. Simplify the following definitions as much as possible:

$$
\begin{aligned}
&\mathit{verbose} :: \mathit{Bool} \to \mathit{Bool} \\
&\mathit{verbose}\ b = \textbf{if}\ b == \mathit{True}\ \textbf{then}\ \mathit{True}\ \textbf{else}\ \mathit{False}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{complex} :: x \to y \to (\mathit{Int}, y) \\
&\mathit{complex}\ x\ y = (\mathit{fst}\ p, \mathit{snd}\ q) \\
&\qquad\qquad \textbf{where}\ p = \mathit{head}\ [(2,3),(8,4),(1,5)] \\
&\qquad\qquad\quad\ q = (x, y)
\end{aligned}
$$

2. The "exclusive or" operator takes two logical values and returns *True* when exactly one of its arguments is *True*, and *False* otherwise.

$$\mathit{xor} :: \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool}$$

   Define the *xor* function:

   (a) using pattern matching;

   (b) using **if then else** (hint: use two nested conditionals);

   (c) using /= (which decides if two values are different)

   Do not use any other library functions or operators. Name your functions *xor1*, *xor2* and *xor3*. Try and simplify your definitions as much as possible.

3. The function *third* returns the third element in a list.

$$\mathit{third} :: [a] \to a$$

   Define the *third* function:

   (a) using *head* and *tail*;

   (b) using pattern matching;

   (c) using list indexing !!.

4. **Bonus Exercise.** (Optional, but will earn you additional marks if you complete it.)

   The *Luhn Algorithm* is a simple approach to validating bank card numbers by computing a checksum from their digits. The algorithm proceeds as follows:

   i. Consider each digit as a separate number;

   ii. Double every second number starting from the second last and working backwards;

   iii. Subtract 9 from each number that is greater than 9;

   iv. Add all sixteen numbers together;

   v. If the result is divisible by 10, the card number is valid.

   Try it on your own 16-digit bank card number and see!

   Your task is to encode the Luhn Algorithm in Haskell. This can be divided into two steps:

   (a) Define a function that doubles a number and subtracts 9 if the result is greater than 9:

$$\mathit{luhnDouble} :: \mathit{Int} \to \mathit{Int}$$

   (b) Using *luhnDouble* and the *mod* library function, define a function *luhn* that checks if a bank card number is valid. For simplicity, only consider four digit card numbers.

$$\mathit{luhn} :: \mathit{Int} \to \mathit{Int} \to \mathit{Int} \to \mathit{Int} \to \mathit{Bool}$$

   For example, 1784 is a valid card number, whereas 4783 is not.