MATHFUN
Discrete Mathematics and Functional Programming

Worksheet 6: Higher-Order Functions

Introduction

This worksheet aims to give you practice in writing functions that take other functions as arguments. Use of the higher-order functions `map`, `filter` and `foldr` is emphasised, as is the use of function composition. Begin by downloading the Week6.hs file from the unit web-site which includes some functions from the lecture - experiment with these definitions before moving onto the exercises.

Use of map, filter and foldr

Use the higher-order functions `map`, `filter` and `foldr` from the Prelude to write the following functions. You should aim for your solutions to be as concise as possible. In particular:

- where possible, give function-level definitions (see page 6 from the lecture);
- don't declare the types of the functions (let Haskell determine the most general type). Try to work out what the type of each of your solutions is, and use the `:type` command to check your understanding.

Many of the solutions will require the application of more than one higher-order function, although all answers should involve only one line of code.

1. Write a function `mult10` that multiplies each element of a list by 10. (For example, `mult10 [5,7,2,4] = [50,70,20,40]`.)
2. Write a function `onlyLowerCase` which removes any character from a string that is not a lower-case letter. (For example, `onlyLowerCase "Port 15" = "ort"`.)
3. Write a function `orAll` which finds the disjunction (or) of the elements in a Boolean list. (E.g., `orAll [True,False,True] = True`, `orAll [False,False] = False` and `orAll [] = False`.)
4. Write a function `sumSquares` that returns the sum of the squares of the elements of a list. (For example, `sumSquares [3,2,4] = 29`.)
5. Write a function `zeroToTen` that keeps only those values that are between 0 and 10 in a list. (For example, `zeroToTen [7,-3,0,15,10,2] = [7,0,10,2]`.)
6. Write a function `squareRoots` that finds the square roots of all the non-negative values in a list. (For example, `squareRoots [4,-8,10] = [2.0, 3.16]`.)
7. Write a function `countBetween` that counts the number of items in a list that are between specified lower- and upper bounds. (For example, `countBetween 3 6 [5, 9, 2, 4, 6, 3, 1, 4] = 5` since 5, 4, 6, 3 & 4 are between the bounds 3 and 6.)
8. Write a function `alwaysPositive` that tests whether applying a given function to all the elements of a list results only in positive values. (For example, `alwaysPositive (+10) [-1,-8,2] = True` and `alwaysPositive (*2) [-1,-8,2] = False`.)

9. Write a function `productSquareRoots` function that finds the product of the square roots of all the non-negative values in a list. (E.g., `productSquareRoots [4,-8,10] = 6.32`.)

Other higher-order functions

Complete the following questions without using `map`, `filter` or `foldr`.

10. Write a function `removeFirst` that removes the first element of a list that has a given property. (E.g., `removeFirst (<0) [3,-1,4,-8,2] = [3,4,-8,2]`.)

11. Write a function `removeLast` that removes the last element of a list that has a given property. (E.g., `removeLast (<0) [3,-1,4,-8,2] = [3,-1,4,2]`.)

Using lambda expressions

Lambda expressions are nameless (i.e. anonymous) functions. (The term lambda expression comes from the lambda calculus, which underlies all functional languages.) Lambda expressions are often used as arguments to higher-order functions such as `map`, `filter` and `foldr`. To define a lambda expression, we use a backslash (intended to look like the letter lambda - λ), function arguments, an arrow and a return expression. For example:

```
\x -> 2 * x       -- this function doubles its argument
\x y -> x + y     -- this function adds its two arguments
```

We can apply a lambda expression as follows:

```
ghci> (\x -> 2 * x) 4
8
```

12. Using `filter` and a single lambda expression, give an alternative solution to exercise 5.

13. [harder] Using only lambda expressions and `foldr` (i.e. not `map` or `filter`), write new versions of (i) the `mult10` function from exercise 1, (ii) `reverse` (to reverse a list), and (iii) `onlyLowerCase` from exercise 2.