CSCI 365 Problem Set 2: Functions & Haskell

due Friday, 2 February 2024

Specification

To receive credit for this problem set:

- You must complete 7 out of 10 exercises.
- Your code must adhere to the Haskell style guide linked from the course web page.

Guidelines

When solving the homework, strive to create not just code that works, but code that is stylish and concise. See the style guide on the website for some guidelines. Try to write small functions which perform just a single task, and then combine those smaller pieces to create more complex functions. Don't repeat yourself: write one function for each logical task, and reuse functions as necessary.

Be sure to write functions with exactly the specified name and type signature for each exercise (to help in testing your code). However, you may (should!) create additional helper functions with whatever names and type signatures you wish.

"HCFP" is an abbreviation for your textbook, "Haskell: the Craft of Functional Programming".

Installing Haskell

You will need to install a Haskell toolchain on your computer. Follow the instructions found on the course website. Please try this early and come ask for help if you are stuck!

Haskell Basics

Consider the following Haskell definitions:

```haskell
f :: Integer -> Integer
f 0 = 11
f n = 5*n - 3

g :: Integer -> Integer -> Integer
g x y = (x + 3) `div` f (y - 2)
```

Exercise 1 Demonstrate (on paper, via a series of rewrite steps) the evaluation of `(g 33 4)`.

Exercise 2 Demonstrate (on paper, via a series of rewrite steps) the evaluation of `(f (g (f 7) (f 1)))`.

Exercise 3 Implement a Haskell function `nand :: Bool -> Bool -> Bool` corresponding to logical NAND (negated conjunction).

Exercise 4 Complete HCFP Exercise 3.16. Do not use the standard library `toUpper` function.

Exercise 5 Implement each of the following Haskell functions:

- `dup :: Integer -> (Integer, Integer)`, which duplicates its input
- `add :: (Integer, Integer) -> Integer`, which adds the two components of a pair of integers
- `doubleFst :: (Integer, Integer) -> (Integer, Integer)`, which doubles the first component of a pair and leaves the second component unchanged

Haskell Fun

Exercise 6 Complete HCFP Exercises 3.22, 3.23, and 3.24.

Exercise 7 Write a Haskell function `roman :: Integer -> String` which converts any positive integer up to 3999 to a Roman numeral. You will probably find helpful the built-in operator ++ which concatenates two `String` values into one, for example,

```
"hi" ++ "there" == "hithere"
```

Exercise 8 Write a Haskell function `isqrt :: Integer -> Integer` which computes the *integer square root* of a given nonnegative integer n, defined as the largest integer whose square is less than or equal to n. For example, `isqrt 4 = isqrt 8 = 2`, `isqrt 9 = 3`, and `isqrt 604 = 24`. Note that your function must work correctly on integers of *any* size, so converting to a floating-point number, taking a square root, then rounding down will not work, since it will be inaccurate for inputs which are too big to be represented exactly as floating-point numbers.

One way you could accomplish this is via binary search. You could also use Newton's method, which works by starting with a guess x and repeatedly updating x to the average of x and n/x.

Function composition

Exercise 9 Using the definition of function composition,

```
(f . g) x = f (g x),
```

show that function composition is *associative*, by showing that both `f . (g . h)` and `(f . g) . h` yield the same result when applied to the same input. Note that this justifies us in writing `f . g . h` (or even longer function composition chains) with no parentheses, since the result is unambiguous.

Exercise 10 Using the functions from Exercise 5, along with the `swap :: (Integer, Integer) -> (Integer, Integer)` function from class, implement each of the following functions, using *only* the given functions and the function composition operator.

(a) `double :: Integer -> Integer`, which doubles its input

(b) `dectuple :: Integer -> Integer`, which multiplies its input by 10