lab1.md

Functional Programming - Lab Class Exercises 1

Frank C Langbein, frank@langbein.org

1. Getting Started

- Open a terminal and start the interactive version of the Glasgow Haskell Compiler, `ghci`.
  - Try some commands:
    - `reverse "test"`
    - `10 * 11 `div` 2`
    - `foldr (+) 0 [1..10]`
    - And more examples from the lecture...
  - Note:
    - Extension for Haskell module files is `.hs`
    - You can load a Haskell module by providing it as argument to `ghci` or use the command `:load FILE`.
    - To reload edited files run `:reload`, which reloads all loaded modules.
    - Type `:h` for help.
- With your preferred editor, create the sum of odd squares example from the lecture:

```
squares :: [Int] -> [Int]
squares xs = [ x*x | x <- xs ]

odds :: [Int] -> [Int]
odds xs = [x | x <- xs, odd x]

sumSqOdd :: [Int] -> Int
sumSqOdd xs = sum [x*x | x <- xs, odd x]

main :: IO () -- main function for compilation
              -- IO indicates this is an action to handle sideffects
main = do -- execute things in sequence
          putStrLn("Sum of odd Squares from 1 to ?:")
          inp <- getLine          -- read any stdin line
          let x = read inp :: Int  -- and convert to integer
          print (sum (squares (odds [1..x])))
          print (sumSqOdd [1..x])
```

  - Compile it with the `ghc` compiler and run the executable from the command line

2. Pythagorean Triples

A Pythagorean triple is a set of three integers `(a,b,c)` which satisfy the equation `a^2 + b^2 = c^2`. Write a Haskell function that verifies whether three integer inputs are a Pythagorean triple.

3. Half-Evens

Write a Haskell function that takes as input a list of integers and outputs a list of half of each of the even integers in the list.

4. Dot Product

Write a Haskell function that takes as input two lists of numbers, representing two vectors of equal length, and outputs the value of the dot-product of the two values (`x_1 * y_1 + .... + x_n * y_n` for a list `x = x_1 : x_2 : ... : x_n` and `y` equivalently).

5. Binary Sequence

Write a Haskell function that generates a sequence of binary number vectors with up to a given number of digits. E.g. `bin_seq 3` should produce `[[0,0,0],[1,0,0],[0,1,0],[1,1,0],[0,0,1],[1,0,1],[0,1,1],[1,1,1]]`.

Hints:

- How can you get from `bin_seq n` to `bin_seq (n+1)` ?
- What happens when you use list comprehension with more than one generator?
- Haskell's `iterate` may be useful.
- As always, there are multiple valid solutions.