CS314
Lab 4
Winter 2014

Introduction

The purpose of this lab is to give you exposure to a programming paradigm that is different from what you're used to. If you've already used Haskell or another language that supports functional programming, then you have a head start. If not, then I recommend you read one or more of the various introductions and tutorials that you can find from the Haskell web page (*especially the Learn You a Haskell for Great Good.*) Do not expect to be able to complete this lab without studying the Haskell language, to at least an introductory level. Your task is to write a number of functions in Haskell that solve various simple problems. To do this you'll need access the Glasgow Haskell Compiler (GHC) and it's interpreter (ghci). The best installation is to install `haskell-platform`. See www.haskell.org for more information.

Nothing required here will be any more difficult than you'll see in class during lecture. I have no doubt that you could search the web for answers to some of these problems. Please do not do this. This lab needs to be your own work. You will learn much more about functional programming this way, and you'll also be able to answer the questions about Haskell on the quiz.

For this lab it is helpful to have a good editor – one that performs syntax highlighting for Haskell. Notepad++ seems to do an OK job and Textmate on the Mac has a bundle plug-in. GEdit on Linux works well. Since Haskell uses two dimensional layout (no braces or semicolons) it is essential that your editor uses spaces for tabs or always assigns 8 spaces to a tab.

For each of these problems, write your Haskell code in a single file. Name it Lab4.hs Please start with something like this, with your name of course.

```
{-          Lab 4  Scot Morse  -}

-- Example
square :: Integer -> Integer
square x = x^2

{- ANSWER: The square of 93240823948293048 is 8693851250556578380656712885130304 -}

-- Problem #1

{- ANSWER:  -}

-- Problem #2

{- ANSWER:  -}

-- Problem #3

{- ANSWER:  -}
```

Each problem will go in this file in the appropriate place. To get started, please begin with the interpreter. From the command line, navigate to where you have written this file, then start your `ghci` session. Go ahead and try out the square function. First you must :load (or :l) the file containing your code. After working on your code, you need to tell the interpreter to use the new code. You can do this by typing :reload (or :r). It will then reload the last module and you can be off and running. This will be the typical procedure for this lab. Write code, reload, try it out, repeat. There will be one part that will require a `main` function.

For most of the questions you'll write the function or functions and then be asked to use it to provide a resulting value for an example input. Put that answer in comments just after the function, as illustrated above.

Important note: For every function you write, make sure and write a type signature for it. Haskell doesn't force you to do it, but it is good practice. It forces you to think about types and that is a good thing, especially when you're first learning.

Questions: A Bunch of Random Things

1. As a good scientist should, I use SI units in everyday life. For example my thermometers are set to Celsius and I record my fuel efficiency in liters per 100km. Some misguided individuals persist in using archaic units. To help them when traveling to sensible countries, write functions necessary to answer the following questions. How many US gallons are there in a certain number of liters? How much have you spent in USD if you spent $x CAD? Now, use these two functions to answer this question: you filled up your gas tank in Canada. You bought 62.3 liters of fuel and paid 78.4 Canadian dollars. What price was that in US dollars per gallon? To get you started on this one, call your first function `gallons`, the second one `usd` and the last `price`.

2. Write a function called `flightDistance` to compute the approximate distance between two locations on the Earth (in nautical miles[1]) given the latitude and longitude of each coordinate as `Double` values. The needed function is

$$d = a\ cos^{-1}\Big(cos(\delta_1)cos(\delta_2)cos(\lambda_1 - \lambda_2) + sin(\delta_1)sin(\delta_2)\Big)$$

   where d is the flight distance[2] in miles between two points on a sphere of radius a (in miles), roughly 3963 miles for the Earth. Point one has latitude δ_1 and longitude λ_1 while point 2 has latitude δ_2 and longitude λ_2. The latitude and longitude need to be in decimal form, i.e. +45.58 latitude and -122.6 longitude is basically 45 degrees North latitude and 122.6 West longitude. Note that these values are in *degrees* and that trigonometry functions usually assume values in *radians*. Also, in case you are unfamiliar with the notation, cos^{-1} means an inverse, or arc cosine. Important: Please write your function using tuples to represent each coordinate; write a helper function to convert from degrees to radians. *Also you must use a where clause.* What is the flight distance between the point at $45°N$,$122°W$ to the point $21°N$,$158°W$?

[1] Ok, so nautical miles aren't SI units – but they're standard for navigation purposes on a curved earth, so work with me here.

[2] `http://mathworld.wolfram.com/GreatCircle.html`

3. Write a function called `factorial` that computes the factorial of an `Integer`. Your function must use the `foldl` or `foldr` function. Also, use a guard to alert the user of an error when the parameter is negative. What is the factorial of 99?

4. Write a function called `isEven` that tells you if a number is even or not. Please use a if-then-else expression.

5. Use a list comprehension and the sum function to determine the sum of the cubes of all the odd numbers between 1000 and 2000?

6. Write a function that confirms the following closed form solution to the sum

$$\sum_{i=0}^{N} i^2 = \frac{N(N+1)(2N+1)}{6}$$

for a given value of N.

7. Write a function with this type signature:

```
count :: (Eq a, Num b) => [a] -> a -> b
```

or `count :: Eq a => [a] -> a -> Int`

This function takes, say, a list of numbers and a value. It tells you how many times the value appears in the list. So

```
count [1,2,3,2,5,1,4] 1 == 2
```

since the number 1 appears two times. *Write this function twice: once using a list comprehension and another time explicitly using recursion.* Use it to count the number of double-ues in 'western oregon wolves (wow) win winter wrestling', as well as the number of things in something of another type (besides Int's or Char's).

8. Write a function called `maxList` that uses recursion to compute the maximum element of a list. Write an explicit type declaration so that it can be used with numbers, letters, or anything that can be compared with $>$. Hints: look up the `Ord` type class and use the `max` function.

9. Write expressions using `map`, `filter`, `any` or `all` to

   - remove all spaces from a string
   - filter out all even numbers from a list (use the `even` function)
   - double every value in a list
   - tell you True or False if a list contains the number 55
   - tell you True or False if all the values in a list are odd (use the `odd` function)

10. Write a function called `isPrime` that determines if an `Integer` is a prime number (evenly divisible only by itself and one). For reference, here's a list of the primes less than 100:

```
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]
```

What are the 1000^{th} through 1020^{th} prime numbers? (starting at 2)

11. Write a function called `factor` that will take an `Integer` and determine its prime factors (called prime factorization). The function should take an `Integer` and return a list of `Integers`. This list should be the prime factors of the number, not including multiplicity. For example, the prime factors of 65 are 2 and 7 (even though the full multiplication is $56 = 2^3 \cdot 7 = 2 \cdot 2 \cdot 2 \cdot 7$). What are the prime factors of 175561 and 62451532000?

12. The previous function can be sped up considerably by compiling it and running via a main function. To do this add the following at the top of your file:

```
module Main where
import System.Environment
```

and then add the main function (please put it at the bottom):

```
main :: IO ()
main = do
        args <- getArgs
        print $ factor (read (head args)::Integer)
```

Compile like this:

```
ghc --make PartI.hs -O -o PartI
```

where the O and o are both the letter oh, the first one capitalized for 'optimize' and the second just the output name. And then run like this

```
./PartI 5698
```

which should give you

```
[2,7,11,37]
```