Lab Worksheet 8. Haskell.

This week, the exercises are in Haskell. For an introduction into functional programming and Haskell, see lecture slides.

Exercise 1: Preparation

In the CSEE labs, a recent version of the Haskell Platform including GHCi should be installed. This is the recommended environment for the exercises below.

- Please create a folder for holding your Haskell source files, say `M:\ce303\haskell`, or similar.
- In that folder, create a new file `labs.hs` and open it in a text editor. My recommendation would be to use Notepad++.
- Warning: **Layout is significant in Haskell programs**. Tabs can cause problems due to editors treating them differently. My recommendation is to avoid tabs in Haskell programs or to configure your editor so that it automatically converts tabs into spaces.

Exercise 2: Program "Hello World" in GHCi

In file `labs.hs`, add the following line:

```
hello s = "Hello " ++ s ++ "!"
```

Start WinGHCi from the Windows Start menu under the Haskell folder. WinGHCi is a simple console-based environment that allows loading and running Haskell programs. Load file `labs.hs` in WinGHCi by using commands `:cd` and `:l`, e.g.

```
:cd M:\CE303\haskell
:l labs.hs
```

Alternatively, you can simply use the GUI and open the file.

Note how the Haskell prompt has changed to `*Main`. Run your program by entering:

```
hello "World"
```

Exercise 3: Miscellaneous Function Exercises

In File `labs.hs`, code Haskell functions which solve the tasks described below. Re-load the file into GHCi every time you change it. You can use command `:r` for that. You can also scroll through the recent commands using the up and down arrows.

After you have coded a function, test it interactively in GHCi for some sample arguments.

1. Write a function `square` that calculates the square on integers.
2. Write a function `fib` that calculates the Fibonacci numbers, e.g.

```
fib 0 = 0
fib 1 = 1
fib n = fib(n-2) + fib(n-1)
```

3. In Haskell, you can define functions on pairs (and other tuples) by pattern matching. For example, here is a `swap` function that swaps the two components of a pair:

   ```
   swap:: (a,b) -> (b,a)
   swap (x,y) = (y,x)
   ```

   Following this example, write a function `permute3` which performs a cyclic permutation of the three components of a triple. For example, the result of `permute3(1, "a", [])` should be the triple `([], 1, "a")`.
4. Write a recursive function `sqList` that returns a list with the first n square numbers starting at 1, e.g. `sqList 3` should return `[1, 4, 9]`.
5. Write a non-recursive version `sqList1`. Your solution should use of the `map` function and an integer range of the form `[a..b]`.
6. Write a recursive function `minList1` which returns the smallest element of a non-empty list of integers.
7. Write a recursive function `minList` which returns the smallest element of a possibly empty list of integers. The solution should return a value of type `Maybe`, e.g. `minList` applied to an empty list should return the value `Nothing`. You will need to add

   ```
   import Data.Maybe
   ```

   at the start of the source file.
8. Write a (higher-order) function `deriv` which calculates the "discrete derivative" of an integer function, e.g.:

   ```
   deriv f n = f(n+1) - f(n)
   ```

9. Test this function by calculating the discrete derivatives of the square function on the interval `1..10`.
10. Write a function `forall p xs` that tests whether a predicate p is true for all elements of a list `xs`. The function should return `True` if the list is empty.
11. Write a function `isPrime` which decides whether an integer n is prime by checking that n it not divisible by any number between 2 and n/2. In your solution, please make use of function `forall` from above. For divisibility checking, use the remainder operator `` `mod` ``.
12. Write a function `primesUpto` which lists all prime numbers between 2 and some number n. You may want to compare your solution with the corresponding exercise for streams in Lab 7.

Exercise 4: Higher-order Functions foldl1 and foldl

Rewrite function `minList1` from above using `foldl1` instead of explicit recursion.

Exercise 5: Recursive Data Type Example: Terms

Add the `Term` data type from the lecture to file `labs.hs`.

```
data Term = Number Int | Plus Term Term | Minus Term Term
            deriving (Eq, Ord, Show)

eval:: Term -> Int
eval (Number i) = i
eval (Plus t1 t2) = eval t1 + eval t2
eval (Minus t1 t2) = eval t1 - eval t2
```

1. Test the `eval` function on a couple of sample expressions.
2. Extend the data type `Term` with constructors that denote multiplication and division. Modify function `eval` accordingly.
3. Test your code by evaluating some suitable sample expressions.