Functional Programming

http://proglang.informatik.uni-freiburg.de/teaching/functional-programming/2017/

Exercise Sheet 1 – First steps

2017-10-25

Exercise 1 (Warming up)

1. Write two functions `maxi` and `mini` which computes the maximum and the minimum of two `Int`s. Provide type signatures for each functions. (Don't use the predefined `min` and `max`, obviously!)

2. Define `max3`, which compute the maximum of three `Int`s.

3. Define a function `med`, which computes the Median of three `Int`s.

4. Test your definitions with QuickCheck Properties. Try to use `Data.List.sort` as as reference.

Exercise 2 (Stack calculator)

We will now implement the core of a small stack-based calculator. A stack computer is capable of using `Int`s with the following operations: `push` n, `pop`, `dup`, `add`, `substract`, `multiply` and `neg`.

We represent the stack as a list of `Int`s: `[Int]`. The initial stack is infinitely deep and filled with zeros. This means the following sequence of operation succeeds and returns 8.

```
pop
push 8
add
```

1. Implement the stack operations as function that takes the initial stack as argument and return the updated stack.

2. Test your functions using QuickCheck. Be aware that QuickCheck can generate very long lists. If that is the case, use Properties instead.

3. In order to make our stack calculator convenient to use, we want users to be able to provide textual commands. Implement a function `readCommand :: String -> [Int] -> [Int]` which decodes the provided string and call the appropriate operations. An unrecognized operation should leave the stack unchanged (noop). The exact format of textual commands is up to you.

   Tip: Strings are list of `Char`s. You can use the functions presents in `Data.Char`.

Exercise 3 (List functions)

Implement the following functions: `head`, `tail`, `init`, `last`, `length`, `reverse`, `(++)`, `iterate`, `map`, `filter`, `intersperse`, `concat`, `zipWith`, `repeat`, `and`, `takeWhile`, `dropWhile`, `maximum`.

You can consult the type and the documentation for all these functions here:

https://hackage.haskell.org/package/base-4.10.0.0/docs/Data-List.html