Exercises (& some assorted code) from Chapter 1

We use LD(n) for the least natural number greater than 1 that divides n.

$$
\begin{array}{l}
\mathit{divides}\ d\ n = \\
\quad \mathit{rem}\ n\ d \equiv 0
\end{array}
$$

It is useful to define LD in terms of a second function that calculates the least divisor starting from a given threshold k, with $k <= n$.

$$\mathit{ld}\ n = \mathit{ldf}\ 2\ n$$

$$
\begin{array}{lll}
\mathit{ldf}\ k\ n & \mid \mathit{divides}\ k\ n & = k \\
 & \mid k \uparrow 2 > n & = n \\
 & \mid \mathit{otherwise} & = \mathit{ldf}\ (k+1)\ n
\end{array}
$$

Problem 1.4. If *ldf* used $k^2 \geqslant n$ how would that change the function?

It wouldn't, because otherwise *divides* $k\ n$ would have been *True*.

$$
\begin{array}{ll}
\mathit{prime0}\ n & \\
\quad \mid n > 1 & = \mathit{error}\ \texttt{"not a positive integer"} \\
\quad \mid n \equiv 1 & = \mathit{False} \\
\quad \mid \mathit{otherwise} & = \mathit{ld}\ n \equiv n
\end{array}
$$

Problem 1.6. Can you gather from the definition of *divides* what the type declaration for *rem* would look like?

$$\mathit{rem} :: \mathit{Integer} \to \mathit{Integer} \to \mathit{Integer}$$

Problem 1.9. Define a function that given the maximum of a list of integers. Use the predefined function *max*.

$$
\begin{array}{ll}
\mathit{maxIn} :: [\mathit{Int}] \to \mathit{Int} & \\
\mathit{maxIn}\ [\,] & = \mathit{error}\ \texttt{"UNDEFINED: empty list"} \\
\mathit{maxIn}\ [x] & = x \\
\mathit{maxIn}\ (x : \mathit{xs}) & = \mathit{max}\ x\ (\mathit{maxIn}\ \mathit{xs})
\end{array}
$$

Problem 1.10. Define a function `removeFst` that removes the first occurrence of an integer m from a lst of integers. if m does not occur n the list, the list remains unchanged.

$$
\begin{array}{lll}
\mathit{removeFst} :: \mathit{Int} \to [\mathit{Int}] \to [\mathit{Int}] & & \\
\mathit{removeFst}\ \_\ [\,] & & = [\,] \\
\mathit{removeFst}\ m\ (x : \mathit{xs}) & \mid m \equiv x & = \mathit{xs} \\
 & \mid \mathit{otherwise} & = x : \mathit{removeFst}\ m\ \mathit{xs}
\end{array}
$$

Problem 1.13. Write a function *count* for counting the number of occurrences of a character in a string.

$$
\begin{array}{lll}
\mathit{count} :: \mathit{Char} \to \mathit{String} \to \mathit{Int} & & \\
\mathit{count}\ \_\ [\,] = 0 & & \\
\mathit{count}\ c\ (y : \mathit{ys}) & \mid c \equiv y & = 1 + \mathit{count}\ c\ \mathit{ys} \\
 & \mid \mathit{otherwise} & = \mathit{count}\ c\ \mathit{ys}
\end{array}
$$

Problem 1.14. Write a function `blowup` such that *blowup* `"bang!"` should yield `"baannngggg!!!!!"`

$$
\begin{aligned}
&\mathit{blowup} :: \mathit{String} \to \mathit{String}\\
&\mathit{blowup} = \mathit{concat} \circ \mathit{zipWith}\ \mathit{replicate}\ [1\,..]
\end{aligned}
$$

The above solution is in point-free form because I'm under the impression that point-free form is what Haskellers aim for by default. Like, to write in point-free form often is an achievement. Check out the slight difference (the readability in particular) in the following definition:

$$
\mathit{blowup}\ \mathit{chrs} = \mathit{concat}\ (\mathit{zipWith}\ \mathit{replicate}\ [1\,..]\ \mathit{chrs})
$$

Problem 1.15. Write a function `srtString :: [String] -> [String]` that sorts a list of strings in alphabetical order

Problem 1.17. Write a function $\mathit{substring} :: \mathit{String} \to \mathit{String} \to \mathit{Bool}$ that checks whether $\mathit{str1}$ is a substring of $\mathit{str2}$.

The `prefix` function was given in Example 1.16:

$$
\begin{aligned}
&\mathit{prefix} :: \mathit{String} \to \mathit{String} \to \mathit{Bool}\\
&\mathit{prefix}\ [\,]\quad \_\quad = \mathit{True}\\
&\mathit{prefix}\ \_\quad [\,]\quad = \mathit{False}\\
&\mathit{prefix}\ (x : xs)\ (y : ys) = (x \equiv y) \wedge \mathit{prefix}\ xs\ ys
\end{aligned}
$$

The actual definition for `substring` is here:

$$
\begin{aligned}
&\mathit{substring} :: \mathit{String} \to \mathit{String} \to \mathit{Bool}\\
&\mathit{substring}\ \mathit{str1}\ \mathit{str2}@(\_ : \mathit{restOfStr2})\\
&\quad \mid \mathit{prefix}\ \mathit{str1}\ \mathit{str2} = \mathit{True}\\
&\quad \mid \mathit{otherwise} \quad = \mathit{prefix}\ \mathit{str1}\ \mathit{restOfStr2}
\end{aligned}
$$

Problem 1.20. Use *map* to write a function *lengths* that takes a list of lists and returns a list of the corresponding lengths.

$$
\begin{aligned}
&\mathit{lengths} :: [[a]] \to [\mathit{Int}]\\
&\mathit{lengths} = \mathit{map}\ \mathit{length}
\end{aligned}
$$

Problem 1.21. Use *map* to write a function *sumLengths* that takes a list of lists and returns the sum of their lengths.

$$
\begin{aligned}
&\mathit{sumLengths} :: [[a]] \to \mathit{Int}\\
&\mathit{sumLengths} = \mathit{sum} \circ \mathit{map}\ \mathit{length}\\
&\mathit{sumLengths}' = \mathit{sum} \circ \mathit{lengths}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{factors} :: \mathit{Integer} \to [\mathit{Integer}]\\
&\mathit{factors}\ n \mid n < 1 \quad = \mathit{error}\ \texttt{"argument not positive"}\\
&\qquad\quad \mid n \equiv 1 \quad = [\,]\\
&\qquad\quad \mid \mathit{otherwise} = p : \mathit{factors}\ (n\ `\mathit{div}`\ p)\\
&\qquad\qquad \textbf{where}\ p = \mathit{ld}\ n
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{primes0} :: [\mathit{Integer}]\\
&\mathit{primes0} = \mathit{filter}\ \mathit{prime0}\ [2\,..]
\end{aligned}
$$

$$
\mathit{ldp} :: \mathit{Integer} \to \mathit{Integer}
$$

$$
\begin{aligned}
&\mathit{ldp}\ n = \mathit{ldpf}\ \mathit{primes1}\ n\\
&\mathit{ldpf} :: [\mathit{Integer}] \to \mathit{Integer} \to \mathit{Integer}\\
&\mathit{ldpf}\ (p : ps)\ n \mid \mathit{rem}\ n\ p \equiv 0 = p\\
&\qquad\qquad\quad\ \mid p \uparrow 2 > n \qquad = n\\
&\qquad\qquad\quad\ \mid \mathit{otherwise} \quad\ = \mathit{ldpf}\ ps\ n\\
&\mathit{primes1} :: [\mathit{Integer}]\\
&\mathit{primes1} = 2 : \mathit{filter}\ \mathit{prime}\ [3\,..]\\
&\mathit{prime} :: \mathit{Integer} \to \mathit{Bool}\\
&\mathit{prime}\ n \mid n < 1 \qquad = \mathit{error}\ \texttt{"not a positive integer"}\\
&\qquad\quad\ \mid n \equiv 1 \qquad = \mathit{False}\\
&\qquad\quad\ \mid \mathit{otherwise} = \mathit{ldp}\ n \equiv n
\end{aligned}
$$

Problem 1.24. What happens when you modify the defining equation of `ldp` to `ldp = ldpf primes1`? Nothing. It's just in point-free form.