1 Prelude

1.1 List operations

`map`: applies a function to each element of a list.

`(++)`: appends two lists.

`filter`: list of elements that satisfy the predicate.

`head`: first element of a list.

`last`: last element of finite, non-empty list.

`tail`: elements after the head of a list.

`init`: all elements except the last one.

`null`: tests whether a structure is empty.

`length`: size of a finite structure.

`(!!)`: list index (subscript) operator.

`reverse`: elements of a finite list in reverse order.

1.1.1 Special folds

`and`: the conjunction ($\wedge$) of a container of Bools.

`or`: the disjunction ($\vee$) of a container of Bools.

`any`: does any element satisfies the predicate?

`all`: do all elements satisfy the predicate?

`concat`: concatenates elements of a container of lists.

`concatMap`: maps a function over all the elements of a container and concatenate the resulting lists.

1.1.2 Building lists with scans

`scanl`: list of successive reduced values from the left, similar to `foldl`: $[z, z * x_1, (z * x_1) * x_2, \ldots]$.

`scanl1`: variant of `scanl` with no starting value.

`scanr`: right-to-left dual of `scanl`

`scanr1`: variant of `scanr` with no starting value.

1.1.3 Building infinite lists

`iterate`: ∞ list of repeated function applications.

`repeat`: ∞ list from a single element.

`replicate`: finite list from a single element.

`cycle`: ties a finite list into a circular one.

1.1.4 Sublists

`take`: prefix of fixed length.

`drop`: suffix after prefix of fixed length.

`splitAt`: a tuple equivalent to `take` and `drop`.

`takeWhile`: the longest prefix satisfying a predicate.

`dropWhile`: the suffix remaining after `takeWhile`.

`span`: a tuple from `takeWhile` and `dropWhile`.

`break`: `break p == span (not . p)`

1.1.5 Searching lists

`notElem`: negation of `elem`.

`lookup`: looks up a key in an association list.

1.1.6 Zipping and unzipping lists

`zip`: list of corresponding pairs from two lists.

`zip3`: analogous to `zip`, takes three lists.

`zipWith`: is a `zip` with custom zupping function.

`zipWith3`: analogous to `zipWith`, takes three lists.

`unzip`: transforms a list of pairs into two lists.

`unzip3`: analogous to `unzip`, returns three lists.

1.1.7 Functions on strings

`lines`: breaks a string up into a list of lines.

`words`: breaks a string up into a list of words.

`unlines`: an operation inverse to `lines`.

`unwords`: an operation inverse to `words`.

author: Remigiusz Suwalski,
date: May 15, 2017