Exercise sheet for lecture 04— Laziness

1 Old friends

There exist many functions for `LazyList` that you already know from before.

Implement `map`, `filter`, `append` and `flatMap`, each using `foldRight` inside the enum. Part of this exercise is to write the signatures yourself. The implementations of these methods works very similar to the ones you already know. The signature for `append` is provided:

```
def append[B >: A](b: => LazyList[B]): LazyList[B] = ???
```

Hint: Use the *smart constructors* from the companion object.

2 takeWhile

In this exercise you will implement the function `takeWhile` in three different ways. `takeWhile` is basically the opposite of `dropWhile`, it returns all elements from the start of a `LazyList`, for which a given predicate returns `true`. As soon as the function find's the first element, for which the predicate returns `false`, it stops.

You can use the following provided signature to implement it in the `LazyList` enum.

```
def takeWhile(p: A => Boolean): LazyList[A] = ???
```

a) Implement `takeWhile` on LazyList using explicit pattern matching!

b) Implement `takeWhile` using `foldRight`!

c) Implement `takeWhile` using `unfold`!

3 tails

Implement the function `tails` via `unfold` (and `append` if necessary)!

For a given `LazyList`, `tails` returns a `LazyList` of all suffixes, i.e. all sublists we can get by removing elements from the beginning, starting with the original `LazyList`. For example, if we have `LazyList(1,2,3)`, `tails` would produce the resul `LazyList(LazyList(1,2,3), LazyList(2,3), LazyList(3), LazyList())`.

The signature of `tails` on the `LazyList` enum looks like this:

```
def tails: LazyList[LazyList[A]] = ???
```