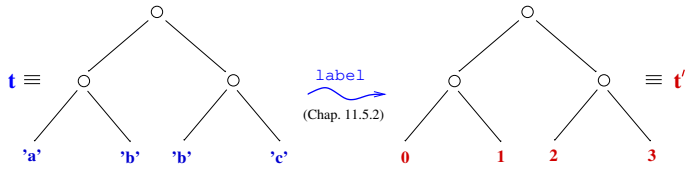
Advanced Functional Programming: Assignment 6 (Wed, 05/29/2019)
Topic: Monadic vs. Non-Monadic Programming
Submission deadline: Wed, 06/05/2019 (3pm)

Regarding the deadline for the second submission: Please, refer to „Hinweise zu Organisation und Ablauf der Übung“ available at the homepage of the course.

Store all functions to be written for this assignment in a top-level file `assignment6.hs` of your group directory. Comment your program meaningfully; use auxiliary functions and constants, where reasonable.

1. *Numbering leafs:* We consider the problem of leaf numbering of Chapter 11.5.2:



   however, with respect to the tree type `Tree1 a`:

```
data Tree1 a = Leave a | Node [Tree1 a] deriving Show
```

   1.1 *Monadic programming:* Following the model of the monadic implementation of function `label :: Tree a -> Tree Int` of Chapter 11.5.2, implement a function:

```
label1 :: Tree1 a -> Tree1 Int
```

   solving the same problem as function `label`, however, for trees of type `Tree1 a` instead of `Tree a`. Like `label`, also `label1` shall number leafs from ‘left to right,’ and rely on monadic programming.

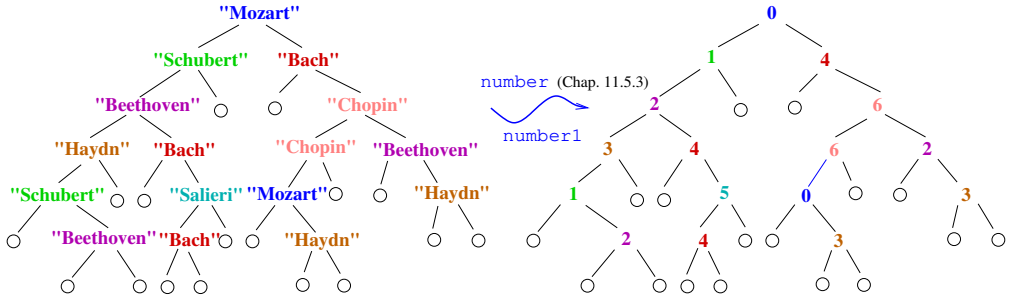
   1.2 *Non-monadic programming:* Implement a function `label2 :: Tree1 a -> Tree1 Int` which is functionally equivalent to `label1` but does not make use of monadic programming.

   1.3 **Without submission:** Comparing your implementations of `label1` and `label2`, do you consider one of them easier to obtain or more comprehensible? What is the reasoning underlying your assessment?

2. *Renaming node labels:* We consider the problem of node label renaming of Chapter 11.5.3, however, rename the tree type of Chapter 11.5.3 as follows:

```
data Tree2 a = Nil | Node a (Tree a) (Tree a) deriving Show
```

   Note that function `number` of Chapter 11.5.3 replaces a node label by the smallest free number, i.e., not yet used number when the label is first reached in the course of a prefix traversal of the tree as illustrated in the below figure:

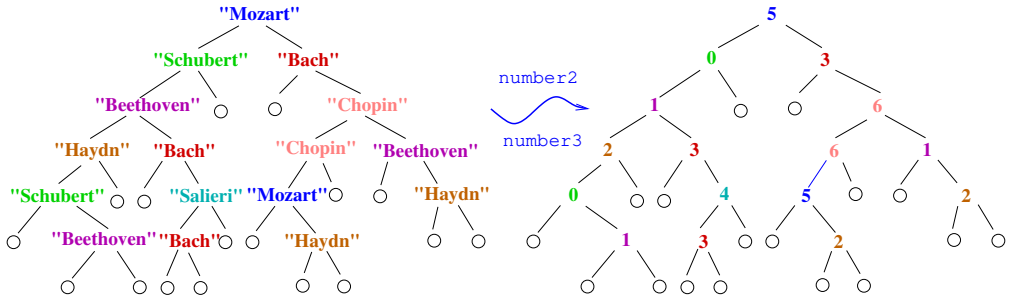


2.1 *Non-monadic programming:* Implement a function

```
number1 :: Eq a => Tree2 a -> Tree2 Int
```

that is functionally equivalent to function `number :: Eq a => Tree a -> Tree Int` of Chapter 11.5.3 but does not make use of monadic programming.

Next, we consider a variant of the node label renaming problem of Chapter 11.5.3. In this variant, node labels shall be replaced by the smallest number not yet used when a label is first reached in the course of an infix (instead of a prefix) traversal of the tree as illustrated in the figure below:



2.2 *Monadic programming:* Adapt the monadic implementation of function `number :: Eq a => Tree a -> Tree Int` of Chapter 11.5.3 to a function

```
number2 :: Eq a => Tree2 a -> Tree2 Int
```

solving the modified renaming task using monadic programming.

2.3 *Non-monadic programming:* Adapt the implementation of function `number1 :: Eq a => Tree2 a -> Tree2 Int` to a function

```
number3 :: Eq a => Tree2 a -> Tree2 Int
```

that is functionally equivalent to `number2` but does not make use of monadic programming.

2.4 **Without submission:** If you compare the conceptual and implementational effort of adapting the implementation of `number` to `number2` and of `number1` to `number3`, do you consider them roughly the same? Are there differences making one of them easier to adapt? If so, why?

Important: *Do not use self-defined modules!* If you want to re-use functions (written for earlier assignments), copy these functions to the new submission file. An `import` declaration for self-defined modules will fail, since only the submission file `assignment`i`.hs` , where i, $1 \leq i \leq 8$ (*tentatively*), denotes the running number of the assignment, willl be copied for the (semi-automatic) evaluation. No other file in addition to `assignment`i`.hs` will be copied.