universität innsbruck

Functional Programming	WS 2023/2024	LVA 703025
Exercise Sheet 11, 10 points		Deadline: Tuesday, January 16, 2024, 8pm

- Mark your completed exercises in the OLAT course of the PS.
- You can start from `template_11.hs` provided on the proseminar page.
- Your `*.hs` file must be compilable with `ghc`.
- Upload your solution to Exercise 1 in OLAT (`*.txt` or PDF or as part of `*.hs`)
- Upload your solution to Exercise 2 as `*.hs` file in OLAT.

Exercise 1 *Evaluation Strategies* **5 p.**

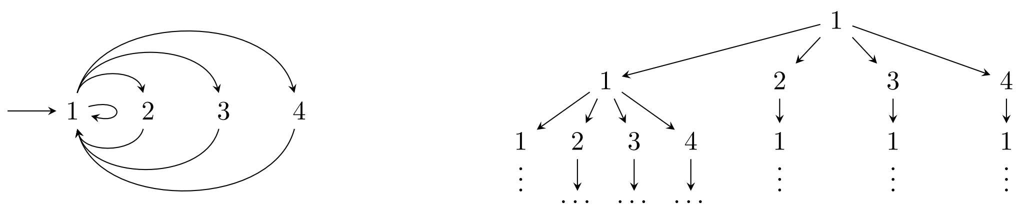
Consider the following functions.

```
-- program 1
[] ++ ys = ys
(x : xs) ++ ys = x : (xs ++ ys)

filter f [] = []
filter f (x : xs)
  | f x = x : filter f xs
  | otherwise = filter f xs

smaller p xs = filter (\x -> x < p) xs
bigger p xs = filter (\x -> x >= p) xs

qsort [] = []
qsort (x:[]) = [x]
qsort (x:xs) = qsort (smaller x xs) ++ x : qsort (bigger x xs)

-- program 2
double x = x + x

take 0 _ = []
take _ [] = []
take n (x : xs) = x : take (n - 1) xs

map f [] = []
map f (x : xs) = f x : map f xs
```

1. Evaluate the expression `qsort ([2] ++ [1])` step-by-step for two evaluation strategies, cf. slide 11/8.
   - (a) call-by-value (1 point) and (b) call-by-name (1 point)
2. Evaluate the expression `take 1 (map double [3 + 5, 7 + 8])` step-by-step for three evaluation strategies:
   - (a) call-by-value (1 point), (b) call-by-name (1 point), and (c) call-by-need (1 point)

Exercise 2 *Lazyness and Infinite Data Structures* **5 p.**

A rooted graph consists of a set of edges between nodes – of the form (source, target) – and additionally has a distinguished node called root. For instance, Figure 1a contains a rooted graph with distinguished node 1 and edges $\{(1,1),(1,2),(1,3),(1,4),(2,1),(3,1),(4,1)\}$.
One way of representing (possibly infinite) rooted graphs is to use (possibly infinite) trees, the so-called *unwinding* of a graph. For example the rooted graph of Figure 1a can be represented by the unwinding shown in Figure 1b.



(a) (b)

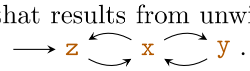
Figure 1: A graph and its unwinding

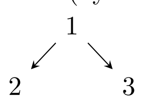
In this exercise graphs and (infinite) trees are represented by the following Haskell type definitions:

```haskell
type Graph a = [(a, a)]
type RootedGraph a = (a, Graph a)
data Tree a = Node a [Tree a] deriving (Eq, Show)
```

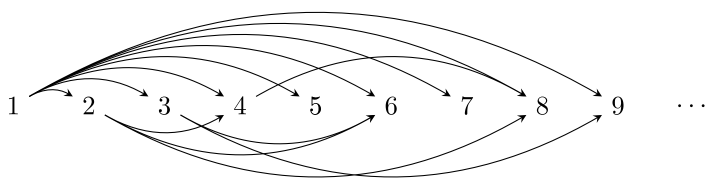
1. Implement a function `unwind :: Eq a => RootedGraph a -> Tree a` that converts a rooted graph into its tree representation. (1 point)

2. Implement a function `prune :: Int -> Tree a -> Tree a` such that `prune n t` results in a pruned tree where only the first `n` layers of the input tree are present. For example invoking `prune 2` on the infinite tree in Figure 1b drops all parts that are depicted by . . . and ⋮, and `prune 0` would return a tree that just contains the root node 1.

   Consider the tree that results from unwinding the rooted graph `(z, [(x,z), (z,x), (x,y), (y,x)])`, a figure of eight:  . What is the result of `prune 4` on this tree? (1 point)

3. Implement a function `narrow :: Int -> Tree a -> Tree a` that restricts the number of successors for each node of a tree to a given maximum (by dropping any surplus successors). For example, when calling the function `narrow 1` on the tree  , the result would be the tree  . (1 point)

4. Define an infinite tree `mults :: Tree Integer` that represents the graph where every natural number, starting from 1 points to all its multiples: (1 point)



5. Describe the results of evaluating each of the following three expressions: `narrow 4 $ prune 2 mults`, `narrow 1 mults`, and `prune 1 mults`. (1 point)