- 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(s) should be compilable with ghci and be uploaded in OLAT.

Exercise 1 *Evaluation Strategies and Kinds of Recursion* **5 p.**

1. Given the four functions:

```
double x = x * 2
square x = x * x
add2times x y = x + double y
func x y = square x + add2times y x
```

   Evaluate each of the following expressions step-by-step under the three evaluation strategies call-by-value, call-by-name, and call-by-need. (3 points)

   (a) `add2times (5+2) 8`

   (b) `double (square 5)`

   (c) `func (2+2) 4`

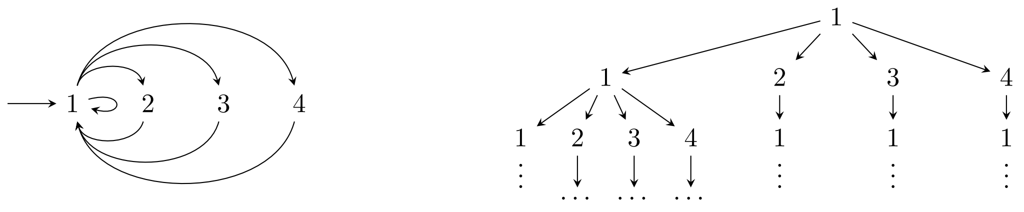
2. For each of the following functions, specify which kind of recursion they use: (1 point)

   (a)
```
squareList [] = []
squareList (x:xs) = x*x : squareList xs
```
   (b)
```
doubleTimes x 0 = x
doubleTimes x y = doubleTimes (x+x) (y-1)
```
   (c)
```
add2List [] = 0
add2List (x:xs) = x + add2List xs
```
   (d)
```
average :: [Double] -> Double
average xs = aux xs 0 (fromIntegral (length xs))
  where aux [] s c = s / c
        aux (x:xs) s c = aux xs (s+x) c
```

3. Implement two variants of a function that takes a string and produces an upper case version of it: `stringToUpperTail` using tail recursion and `stringToUpperGuarded` using guarded recursion. For example `stringToUpperTail "Hello" = stringToUpperGuarded "Hello" = "HELLO"`. (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

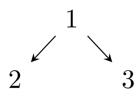
In this exercise graphs and (infinite) trees are represented by the following Haskell type definitions:

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

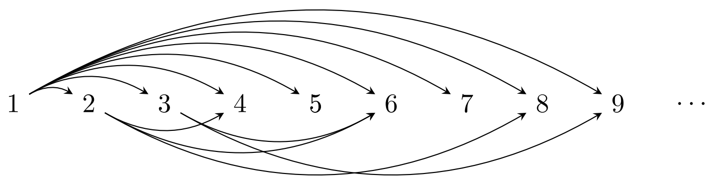
1. Implement a function `unwind :: Eq a => RootedGraph a -> Tree a` that converts a rooted graph into its tree representation. (1 point)

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)