

Prof. Dr. Jürgen Giesl
Peter Schneider-Kamp

Exercises *Functional Programming* – Sheet 3

Solutions will be collected until Wednesday, May 11, 2005 in the exercise course.

Exercises can be solved both in English and in German.

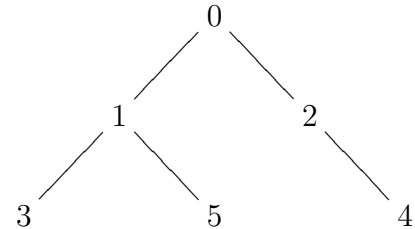
Exercise 1 (2+2+1 points)

The following data structure represents binary trees:

```
data Tree a = Node (Tree a) a (Tree a) | Nil
```

Consider the tree t of integers on the right-hand side. The representation of t as an object of type `Tree Int` in Haskell would be:

```
Node (Node (Node Nil 3 Nil) 1 (Node Nil 5 Nil)) 0 (Node Nil 2 (Node Nil 4 Nil))
```



Implement the following functions in Haskell.

- The function `foldTree` of type `(b -> a -> b -> b) -> b -> Tree a -> b` works as follows: `foldTree g e t` replaces all occurrences of the constructor `Node` in the tree t by `g` and it replaces all occurrences of the constructor `Nil` in t by `e`. So for the tree t above, `foldTree (\x y z -> x + y + z) 0 t` should return 15. Here, `Node` is replaced by `(\x y z -> x + y + z)` and `Nil` is replaced by 0.
- The function `mapTree` of type `mapTree :: (a -> b) -> Tree a -> Tree b` applies a function to all values in the nodes of a tree. For example, `mapTree (*2) t` should return the following tree: `Node (Node (Node Nil 6 Nil) 2 (Node Nil 10 Nil)) 0 (Node Nil 4 (Node Nil 8 Nil))`
- Use the `foldTree` function from (a) to implement the `maxTree` function which returns the largest (w.r.t. `>`) element of the tree. Apart from the function declaration, also give the type declaration for `maxTree`. For the empty tree (`Nil`), the function should return an error message using the `error` function:

```
maxTree Nil = error "maxTree undefined for the empty tree"
```

Exercise 2 (1+2 points)

Implement the following higher-order functions in Haskell and also give their type declarations.

- (a) Given a list of functions $[f_1, f_2, \dots, f_k]$ and a value x , the function `mapF` returns the list $[f_1x, f_2x, \dots, f_kx]$.
For example, `mapF [(*)2, \x -> div x 2] 14` should return the list `[28, 7]`.
You should use pre-defined higher-order functions (e.g., `map`) when appropriate, but you may not use recursion in your declarations.
- (b) Given a list of functions $[f_0, f_1, \dots, f_{k-1}]$ and a start value v , the function `generate` returns the infinite list $[x_0, x_1, x_2, \dots]$ where $x_0 = v$ and $x_{i+1} = f_j x_i$ for $j = i \bmod k$. For example, `generate [(+3), (*2), \x -> x-1] 1` should return `[1, 4, 8, 7, 10, 20, 19, 22, 44, 43, \dots]`.

Exercise 3 (1+1+1 points)

Implement the following function only with the help of list comprehensions, i.e., if you have to implement a function `f` then you should use only two declarations of the following form (where `v` is a variable):

```
f :: ...  
f ... = [v | ...]
```

- (a) Given a natural number $n > 0$, the function `divisors` computes the list of all natural numbers m that divide n .
For example, `divisors 12` should return `[1, 2, 3, 4, 6, 12]`.
- (b) Given two lists `xs` and `ys`, the function `intersect` computes a list `zs` such that $x \in zs$ if and only if $x \in xs$ and $x \in ys$.
For example, `intersect [1,2,3] [5,1,3]` should return a list containing only 1 and 3 (possibly multiple times).
- (c) Given a list of lists $[l_1, \dots, l_k]$, the function `getHeads` returns the list of all first elements of all non-empty lists l_i . For example, `getHeads [[], [1], [2,3]]` should return the list `[1,2]`.