

Prof. Dr. Jürgen Giesl
Peter Schneider-Kamp

Exercises *Functional Programming* – Sheet 2

Solutions will be collected until Wednesday, May 04, 2005 in the exercise course.

Exercises can be solved both in English and in German.

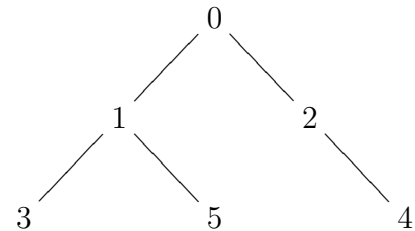
Exercise 1 (1+1+1+2+2 points)

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 Integer` in Haskell would be:

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



- (a) Give the most general types for the functions f, g, h defined as follows. Here we assume that `6.0` has type `Float`. Please explain your answers.

```
f (Node x y z) = Node x 6.0 (Node x y z)
```

```
g Nil          = Node Nil
```

```
h              = \w (Node x y z) -> Node x w z
```

- (b) Implement the following functions in Haskell.

- (i) The function `count` applied to a tree returns the number of nodes in that tree. For example, `count t` should compute `6`.

- (ii) The function `max` applied to a tree of integers (`t :: Tree Integer`) returns the largest integer that occurs in it. For example, `max t` should yield the integer 5.

Hint: You can write `deriving Show` behind your data declaration to look at your trees in HUGS (`data Tree a = ... deriving Show`).

Exercise 2 (2+1+1+1+4 points)

Define a data structure `Set` with the constructors `Insert` and `EmptySet` such that the set can contain elements of an arbitrary type `a` for which the constraint `Eq a` holds. Do not use any built-in types (such as `[a]`) in the definition.

Implement the following functions on this data structure `Set` and include their respective type declaration.

- (a) Make `Set a` an instance of the type class `Show`.
For example `show (Insert 1 (Insert 2 (Insert 3 EmptySet)))` should print `{1, 2, 3}` or a permutation thereof.
- (b) The `memberSet` function takes an element x and a set s and returns `True` if and only if $x \in s$.
- (c) The `subSet` function takes two sets s_1 and s_2 and returns `True` if and only if $s_1 \subseteq s_2$.
- (d) Make `Set a` an instance of the type class `Eq` such that two sets are equal if and only if they contain the same elements, i.e., `(==)` should be the usual equality on sets.
- (e) A graph can be represented by a set of edges, i.e, by an object of type `Set (a, a)`. The function `hasPath` applied to two nodes `m` and `n` and a graph `g` returns `true` if and only if there is a path from `m` to `n` in `g`.
For example, in the graph `Insert (1, 2) (Insert (2, 3) EmptySet)` there is a path from 1 to 3, but not from 3 to 1.