

Prof. Dr. Jürgen Giesl  
René Thiemann

## Exercises *Functional Programming* – Sheet 1

Solutions will be collected until Thursday, Oct 24, 2002 in the exercise course.

The HASKELL interpreter HUGS can be download at [www.haskell.org/hugs](http://www.haskell.org/hugs).

You can use „:l filename“ to load a HASKELL file, „:r“ to reload a changed file and „:q“ to quit the interpreter. Your programs can be tested by typing arbitrary HASKELL-expressions at the prompt.

### Exercise 1 (1+1+1 points)

- (a) Suppose  $f$  and  $g$  have the following type:  $f, g :: \text{Int} \rightarrow (\text{Int} \rightarrow \text{Int})$   
Let  $h$  be defined by:  $h\ x\ y = f\ (g\ x\ y)$   
Fill in the correct type assignment for  $h$ .
- (b) Give examples of functions with the following types:
- (i)  $(\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int}$
  - (ii)  $(\text{Float} \rightarrow \text{Float}) \rightarrow (\text{Float} \rightarrow \text{Float})$
  - (iii)  $[\text{Int}] \rightarrow (\text{Int} \rightarrow \text{Int})$
- (c) Suppose we curry the arguments of some function  $f :: (\text{Int}, \text{Int} \rightarrow \text{Int}, [\text{Int}]) \rightarrow \text{Int}$ , so that we can write „f a b c“ rather than „f(a,b,c)“. What is the type of the curried version of  $f$ ?

### Exercise 2 (3 points)

Which of the following equations between two lists are correct? Why?

- (a)  $xs = [] : xs$     (b)  $xs : [] = xs$     (c)  $[] : xs = [[] , xs]$   
(d)  $[xs] = xs : []$     (e)  $x : y = [x, y]$     (f)  $(x : xs) ++ ys = x : (xs ++ ys)$

The operation  $++$  concatenates two lists. For example:  $[1,2,3] ++ [2,3] = [1,2,3,2,3]$ .

### Exercise 3 (3+3 points)

- (a) Write a HASKELL program to compute the median of a list of different integers. The median is a value  $x$  out of the list  $l$  such that  $|\{y \in l : y < x\}| = |\{y \in l : y > x\}|$ . For example  $\text{median } [1,200,2,3,400] = 3$  and  $\text{median } [1,2]$  is not defined.

To do this, you should define the functions

- (i) `iSort :: [Int] -> [Int]`, which performs the insertion sort algorithm. You can use the predefined functions `==, >, >=, <, ...`.
  - (ii) `nthElem :: [a] -> Int -> a`, which takes the  $n$ -th element out of a list.
  - (iii) `median :: [Int] -> Int`, which computes the median.
- (b) (i) Give the definitions of the boolean functions `and', or', xor' :: Bool -> Bool -> Bool` using pattern matching declarations.
- (ii) Implement the functions `ha` and `fa`, corresponding to half-adder and full-adder, in HASKELL using your defined boolean functions. The result should be of the form `(sum, carryover)`. If possible, use local declarations to avoid multiple evaluation of expressions.
- ```

ha :: Bool -> Bool -> (Bool, Bool)
fa :: Bool -> Bool -> Bool -> (Bool, Bool)

```

### Exercise 4: (1+1+1 points)

The function `length` which computes the length of a list of integers can be implemented as follows:

```

length :: [a] -> Int
length xs = length' 0 xs
  where length' :: Int -> [a] -> Int
        length' n []      = n
        length' n (x:xs) = length' (n+1) xs

```

This function uses the function `length'` that has an additional parameter to accumulate the result. The function `length'` is tail-recursive, because the recursive function call `length' (n+1) xs` in the right-hand side of the definition is not inside the arguments of other functions.

Using the technique of tail-recursion give a HASKELL implementation of the following functions:

- (a) `fac`, which computes the factorial of a natural number
- (b) `reverse`, which reverses a list
- (c) Which advantage do we get by using tail recursion in (a)? Which in (b)? Does it make a difference whether we use the strict or the non-strict evaluation strategy?

### Exercise 5 (1+2 points)

The Fibonacci numbers  $f_0, f_1, \dots$  are defined by the rule that  $f_0 = 0, f_1 = 1$  and  $f_{n+2} = f_n + f_{n+1}$  for all  $n \geq 0$ .

- (a) Give the simplest possible HASKELL implementation of the function `fib` that takes an integer `n` and returns  $f_n$ .
- (b) Modify the implementation from (a) so that the result is computed in linear time.