

Prof. Dr. Jürgen Giesl  
Peter Schneider-Kamp

## Exercises *Functional Programming* – Sheet 1

Solutions will be collected until Friday, Apr 22, 2005 in the exercise course.

Exercises can be solved both in English and in German.

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 points)

(a) Give examples of functions with the following types:

- (i)  $(\text{Int} \rightarrow \text{Int}) \rightarrow \text{Bool}$
- (ii)  $(\text{Bool} \rightarrow \text{Bool}) \rightarrow \text{Bool} \rightarrow \text{Bool}$
- (iii)  $[\text{Int}] \rightarrow [\text{Int}] \rightarrow [\text{Int}]$

(b) Suppose that  $f$  and  $g$  have the type  $(\text{Bool} \rightarrow \text{Bool}) \rightarrow \text{Bool} \rightarrow \text{Bool}$ .

Let  $h$  be defined by  $h\ x\ y = f\ (g\ x)\ y$ . What is the type of  $h$ ?

### Exercise 2 (2 points)

Which of the following equations between two lists are correct? Explain your answers.

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

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

### Exercise 3 (1+1+2+2 points)

(a) Write a HASKELL function that deletes the first occurrence of an integer from a list:

`delete :: Int -> [Int] -> [Int]`

For example `delete 2 [1,2,3,2,1] = [1,3,2,1]`. If the integer does not occur in the list, the list should be returned unchanged, e.g., for the expression `delete 2 [1,3]` the result is `[1,3]`.

(b) Write a HASKELL function to compute the minimum of a list:

```
mini :: [Int] -> Int
```

For example `mini [3,2,1,2] = 1`. Note that the minimum of the empty list is not defined. To compute the minimum of two integers you may use the predefined `min` function:

```
min :: Int -> Int -> Int
```

(c) – Write a HASKELL function that implements the `minsort` algorithm as described below:

```
minsort :: [Int] -> [Int]
```

For an empty list, the algorithm returns the empty list. For a non-empty list, the `minsort` algorithm first computes the minimum of the list, deletes the minimum from the list, and then calls itself on the remaining list. From the sequence of minimums computed it constructs the sorted list.

– Give the sequence of expressions which results from evaluating `minsort [1]` using the lazy evaluation strategy.

(d) – Write a version of `minsort` with a local declaration (using `where`) that avoids the repeated evaluation of identical subexpressions:

```
minsortld :: [Int] -> [Int]
```

– How many evaluation steps does the lazy evaluation of `minsortld [1]` take? Explain the difference with regard to the lazy evaluation of `minsort [1]`.

## Exercise 4 (2 points)

Define a HASKELL function `/\` in infix notation such that the following holds:

- `x /\ y` evaluates to  $x^y$  for all  $x, y \in \mathbb{N}$ ,
- `x /\ y /\ z` is the same as `x /\ (y /\ z)`, and
- `x * y /\ z` is the same as `x * (y /\ z)`.

You may not use any predefined functions except for `(+)`, `(*)`, and `(-)`. Note that the binding priority of `(*)` is 7. Set the priority of `/\` accordingly.