

Programmier-Paradigmen

Tutorium – Gruppe 2 & 8

Henning Dieterichs

Blatt 1 – pow1, pow2, pow3

```
1  module Arithmetik where
2  ....
3  ....pow1 :: Int → Int → Int
4  ....pow1 b 0 = 1
5  ....pow1 b e = b * pow1 b (e - 1)
6  ....
7  ....pow2 :: Int → Int → Int
8  ....pow2 b 0 = 1
9  ....pow2 b e
10 .....| even e = pow2 (b*b) (e `div` 2)
11 .....| otherwise = b * pow2 (b*b) (e `div` 2)
12 .....
13 ....pow3 :: Int → Int → Int
14 ....pow3 = pow3Acc 1
15 .....where
16 .....pow3Acc acc b 0 = acc
17 .....pow3Acc acc b e
18 .....| e < 0 = error "Negative exponent"
19 .....| even e = pow3Acc acc (b*b) (e `div` 2)
20 .....| otherwise = pow3Acc (acc * b) (b*b) (e `div` 2)
21 .....
```

Blatt 1 – isPrime

```
1  module Arithmetik where
2  ....
3  ....isPrime :: Int → Bool
4  ....isPrime num
5  .....| num < 0 = error "Not a natural number"
6  .....| num == 1 = False
7  .....| otherwise = null [ x | x ← [2..root 2 num], num `mod` x == 0 ]
8
9  ....isPrime' :: Int → Bool
10 ....isPrime' num
11 .....| num < 0 = error "Not a natural number"
12 .....| num == 1 = True
13 .....| otherwise = not $ isNumDivisibleByAnyNumberIn 2 $ root 2 num
14 .....where
15 .....-- checks whether num is divisible by any nat. number in [a, b]
16 .....isNumDivisibleByAnyNumberIn a b
17 .....| a > b = False
18 .....| otherwise = num `mod` a == 0 || isNumDivisibleByAnyNumberIn (a+1) b
19
```

Blatt 1 – root

- Aufgabe: Schreibe Funktion *invert*, die eine ganzzahlige monotone Funktion auf einem Intervall invertiert

Blatt 1 – insertSort

- Aufgabe: Schreibe insert und insertSort mithilfe von insert

Blatt 1 – insertSort

```
insert :: [Int] → Int → [Int]
insert (x:xs) e
  .... | e < x = e:x:xs
  .... | otherwise = x:insert e xs
```

```
insertSort :: [Int] → [Int]
insertSort = foldl insert []
```

```
insertSort [5,3,7]
⇒ foldl insert [] [5,3,7]
⇒ foldl insert (insert [] 5) [3,7]
⇒ foldl insert [5] [3,7]
⇒ foldl insert (insert [5] 3) [7]
⇒ foldl insert [3, 5] [7]
⇒ foldl insert (insert [3,5] 7) []
⇒ foldl insert [3,5,7] []
⇒ [3,5,7]
```

Blatt 1 – mergeSort

```
1  module Sort where
2      ....
3      ....merge :: Ord a => [a] -> [a] -> [a]
4      ....merge [] [] = []
5      ....merge [] list = list
6      ....merge list [] = list
7      ....merge (head1:tail1) (head2:tail2)
8      ....    | head1 <= head2 = head1 : merge tail1 (head2:tail2)
9      ....    | otherwise = head2 : merge (head1:tail1) tail2
10     ....
11     ....mergeSort :: Ord a => [a] -> [a]
12     ....mergeSort [] = []
13     ....mergeSort [x] = [x]
14     ....mergeSort list = merge
15     ....    (mergeSort $ take half list)
16     ....    (mergeSort $ drop half list)
17     ....    where half = length list `div` 2
18
```