Introduction to Functional Programming

Zsók Viktória

Department of Programming Languages and Compilers
Faculty of Informatics
Eötvös Loránd University
Budapest, Hungary
zsv@elte.hu

Overview

Sorting lists

```
Start = sort [3,1,4,2,0] // [0,1,2,3,4]

// inserting in already sorted list
Insert :: a [a] → [a] | Ord a
Insert e [] = [e]
Insert e [x : xs]
| e ≤ x = [e , x : xs]
| otherwise = [x : Insert e xs]
Start = Insert 5 [2, 4 .. 10] // [2,4,5,6,8,10]

mysort :: [a] → [a] | Ord a
mysort [] = []
mysort [a:x] = Insert a (mysort x)
Start = mysort [3,1,4,2,0] // [0,1,2,3,4]

Insert 3 (Insert 1 (Insert 4 (Insert 2 (Insert 0 [] ))))
```

Mergesort

```
merge :: [a] [a] → [a] | Ord a
merge [] ys = ys
merge xs [] = xs
merge [x : xs] [y : ys]
| x ≤ y = [x : merge xs [y : ys]]
| otherwise = [y : merge [x : xs] ys]

Start = merge [2,5,7] [1,5,6,8] // [1,2,5,5,6,7,8]
Start = merge [] [1,2,3] // [1,2,3]
Start = merge [1,2,10] [] // [1,2,10]
Start = merge [2,1] [4,1] // [2,1,4,1]
Start = merge [1,2] [1,4] // [1,1,2,4]
```

Mergesort 2

```
msort :: [a] → [a] | Ord a
msort xs
| len ≤ 1 = xs
| otherwise = merge (msort ys) (msort zs)
where
    ys = take half xs
    zs = drop half xs
    half = len / 2
    len = length xs

Start = msort [2,9,5,1,3,8] // [1,2,3,5,8,9]
```

Quick sort

```
qsort :: [b] → [b] | Ord b
qsort [] = []
qsort [a : xs] = qsort [x \\ x ← xs | x <  a] ++ [a] ++
                 qsort [x \\ x ← xs | x >= a]

Start = qsort [2,1,5,3,6,9,0,1] // [0,1,1,2,3,5,6,9]
```

Quick sort

```
Start = qsort [2,1,5,3,6,9,0,1]

qsort [1,0,1] ++[2]++ qsort [5,6,9]

qsort [0]++[1]++qsort [1]++[2]++qsort []++[5]++qsort [6,9]

qsort []++[0]++qsort []++[1]++qsort []++[1]++qsort [] ++[2]++
[]++[5]++qsort []++[6]++qsort[9]

[]++[0]++[]++[]++[1]++[]++[]++[1]++[] ++[2]++
[]++[5]++[]++[6]++qsort []++[9]++qsort []

[]++[0]++[]++[]++[1]++[]++[]++[1]++[] ++[2]++
[]++[5]++[]++[6]++[]++[9]++[]

[0,1,2,5,6,9]
```

Prime numbers

```
divisible :: Int Int -> Bool
divisible x n = x rem n == 0

denominators :: Int -> [Int]
denominators x = filter (divisible x) [1..x]

prime :: Int -> Bool
prime x = denominators x == [1,x]

primes :: Int -> [Int]
primes x = filter prime [1..x]

Start = primes 100 // [2,3,5,7,...,97]
```

Sieve

```
sieve :: [Int] → [Int]
sieve [p:xs] = [p: sieve [ i \\ i ← xs | i rem p ≠ 0]]

Start = take 100 (sieve [2..])
```

Warm 4

```
// 1. exists x xs checks whether x exists as an element in the list xs
// logical operator or is ||
exists :: Int [Int] → Bool
exists x []      = False
exists x [y:ys] = x == y || exists x ys

Start = exists 3 [1, 2, 1, 1, 2, 3, 2, 1, 3] // True

// 2. write the function duplicates which checks if there are duplicates
// in the list xs
duplicates :: [Int] → Bool
duplicates []      = False
duplicates [x:xs] = exists x xs || duplicates xs

Start = duplicates [1, 2, 1, 1, 2, 3, 2, 1, 3] // True
```

Warm 4

// 3. remove x xs removes x from the list xs

```
remove :: Int [Int] → [Int]
remove x []                 = []
remove x [y:ys]
| x == y      = remove x ys
| otherwise = [y : remove x ys]
```

```
Start = remove 3 [1, 2, 1, 1, 2, 3, 2, 1, 3] // [1,2,1,1,2,2,1]
```

// 4. removeDuplicates l returns the list l with all duplicate elements removed

```
removeDuplicates :: [Int] → [Int]
removeDuplicates []     = []
removeDuplicates [x:xs] = [x : removeDuplicates (remove x xs)]
```

```
Start = removeDuplicates [1, 2, 1, 2, 3, 1, 2, 4, 2, 3] // [1,2,3,4]
```

Some more examples

```
qeq :: Real Real Real → (String,[Real])
qeq a b c
 | a == 0.0      = ("not quadratic",[])
 | delta < 0.0   = ("complex roots",[])
 | delta == 0.0  = ("one root",[¬b/(2.0*a)])
 | delta > 0.0   = ("two roots", [(¬b+radix)/(2.0*a),
                                  (¬b-radix)/(2.0*a)])
 where
     delta = b*b-4.0*a*c
     radix = sqrt delta

Start = qeq 1.0 2.0 1.0

Start = qeq 1.0 5.0 7.0
```