Introduction to Functional Programming

More list functions and accumulating parameters

Some slides are based on Graham Hutton's public slides

Recap previous lecture

GÖTEBORGS UNIVERSITET
CHALMERS

- Import declarations
- Polymorphic functions
- Strings are lists!
- Common type classes:
  - `Show`, `Eq`, `Ord`, `Num`

- Announcements:
  - Deadline lab 2 at 18:00 today!

Today

- Building an executable
- Sorting and showing of with QuickCheck
- More list functions
  - With multiple arguments
- Accumulating parameters

List functions with multiple arguments

- Functions with more than one argument can also be defined using recursion.

```
drop :: Int -> [a] -> [a]
drop 0 xs     = xs
drop _ []     = []
drop n (_:xs) = drop (n-1) xs
```

Remove the first n elements from a list

```
(++) :: [a] -> [a] -> [a]
[]     ++ ys = ys
(x:xs) ++ ys = x : (xs ++ ys)
```

Appending two lists

The `zip` function

- A useful library function is `zip`, which maps two lists to a list of pairs of their corresponding elements.

```
zip :: [a] -> [b] -> [(a, b)]

ghci> zip ['a','b','c'] [1,2,3,4]
[('a',1),('b',2),('c',3)]
```

- Using `zip` we can define a function returns the list of all *pairs* of adjacent elements from a list

```
pairs :: [a] -> [(a, a)]
pairs xs = zip xs (tail xs)

ghci> pairs [1,2,3,4]
[(1,2),(2,3),(3,4)]
```

The `zip` function

- Using `pairs` we can define a function that decides if the elements in a list are sorted

```
sorted :: Ord a => [a] -> Bool
sorted xs = and [ x =< y
                | (x, y) <- pairs xs]

ghci> sorted [1,2,3,4]
True

ghci> sorted [1,3,2,4]
False
```

The `zip` function

- Using `zip` we can define a function that returns the list of all positions of a value in a list

```
positions :: Eq a => a -> [a] -> [Int]
positions x xs =
  [i | (y, i) <- zip xs [0..], x == y]


ghci> positions 0 [1,0,0,1,0,1,1,0]
[1,2,4,7]
```

GÖTEBORGS
UNIVERSITET

CHALMERS