Haskell

Esercizi Semplici

Prodotto

- Si scriva una funzione

  `prodotto :: Num a ⇒ [a] → a`

  che, data una lista di valori numerici, calcola il prodotto degli elementi
- Es:

```
> prodotto [1,2,3]
6
```

1

Lunghezza

- Si scriva una funzione

  `lunghezza:: [a] → Int`

  che calcola la lunghezza di una lista
- Es:

```
> lunghezza [1,2,3]
3
```

2

Inversa

- Si scriva una funzione

  `inversa:: [a] -> [a]`

  che data una lista fornisce la lista inversa
- Es:

```
> inversa [1,2,3]
[3,2,1]
```

3

zip'

- Si scriva una funzione

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

  che date due liste, fornisce una lista di coppie. La coppia in posizione i-esima della lista è costituita da
  - l'elemento i-esimo della prima lista
  - l'elemento i-esimo della seconda lista
- Se le due liste hanno lunghezza diversa, `zip'` restituisce una lista con la lunghezza minima delle due
- Es:

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

4

Elimina Primi

- Si scriva una funzione

  `eliminaPrimi:: [a] -> Int -> [a]`

  che dati una lista *l* e un numero *n*, fornisce la lista costituita dagli elementi della lista *l* tolti i primi *n* elementi
- Es:

```
> eliminaPrimi 3 [1,1,2,1,3,4]
[1,3,4]
```

5

concatena

- Si scriva una funzione

  `concatena:: [a] -> [a] -> [a]`

  che date due liste, fornisce la concatenazione delle due liste
- Es:

```
> concatena [1,4,3] [7,5,6]
[1,4,3,7,5,6]
```

6

Exercises

(1) Without looking at the standard prelude, define the following library functions using recursion:

⌘Decide if all logical values in a list are true:

`and :: [Bool] → Bool`

⌘Concatenate a list of lists:

`concat :: [[a]] → [a]`

7

⌘Produce a list with n identical elements:

```
replicate :: Int → a → [a]
```

⌘Select the nth element of a list:

```
(!!) :: [a] → Int → a
```

⌘Decide if a value is an element of a list:

```
elem :: Eq a ⇒ a → [a] → Bool
```

8

(2) Define a recursive function

```
merge :: Ord a ⇒ [a] → [a] → [a]
```

that merges two sorted lists of values to give a single sorted list. For example:

```
> merge [2,5,6] [1,3,4]

[1,2,3,4,5,6]
```

9

(3) Define a recursive function

```
msort :: Ord a ⇒ [a] → [a]
```

that implements merge sort, which can be specified by the following two rules:

⌘Lists of length ≤ 1 are already sorted;

⌘Other lists can be sorted by sorting the two halves and merging the resulting lists.

10

Part of these slides were adapted from the material of the book

Graham Hutton, Programming in Haskell, Cambridge University Press, 2nd edition, 2016

11