Esercizi di Haskell

Esercizio last

- Date le funzioni di libreria:
- Seleziona il primo elemento di una lista
  Prelude> head [1,2,3]
  1
- Rimuovi il primo elemento di una lista
  Prelude> tail [1,2,3]
  [2,3]
- Seleziona l'n-esimo elemento di una lista
  Prelude> [1,2,3,4,5] !! 2
  3
- Seleziona i primi n elementi di una lista
  Prelude> take 5 [1,2,3,4,5,6,7,8]
  [1,2,3,4,5]

Esercizio last

- Rimuovi i primi n elementi di una lista
  Prelude> drop 5 [1,2,3,4,5,6,7,8]
  [6,7,8]
- Calcola la lunghezza di una lista
  Prelude> length [1,2,3,4,5]
  5
- Calcola la somma di una lista di numeri
  Prelude> sum [1,2,3,4,5]
  15
- Calcola il prodotto di una lista di numeri
  Prelude> product [1,2,3,4,5]
  120
- Inverti una lista
  Prelude> reverse [1,2,3,4,5]
  [5,4,3,2,1]

Esercizio last

- Scrivere la definizione della funzione last che selezione l'ultimo elemento di una lista usando le funzioni di libreria
  last :: [a] -> a

Soluzione Esercizio last

last xs = head (reverse xs)

- Oppure

last xs = xs !! ((length xs)-1)

Esercizio init

- Scrivere la definizione della funzione init che rimuove l'ultimo elemento di una lista usando le funzioni di libreria

Soluzione Esercizio init

```
init xs = reverse ( tail (reverse xs))
```

- Oppure

```
init xs = take ((length xs)-1) xs
```

Esercizio inversa

- Si scriva una funzione
  inversa :: [a] -> [a]
- Che data una lista fornisce la lista inversa
- Es
  Prelude> inversa [1,2,3]
  [3,2,1]
- Senza usare la funzione di libreria reverse

Soluzione Esercizio inversa

```
inversa [] = []
inversa (x:xs) = (inversa xs) ++ [x]
```

- Oppure meglio

```
inversa xs = inv xs []
inv []  ys = ys
inv (x:xs) ys = inv xs (x:ys)
```

- La prima soluzione ha complessità quadratica, la seconda lineare

Esercizio zip

- Si scriva una funzione
  zip' :: [a] -> [b] -> [(a,b)]
- che, date due liste, restituisca una lista di coppie.
- La coppia in posizione i-esima è cosituita 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 liste
- Es:
  Prelude> zip' [1,2,3] ['a','b']
  [(1,'a'),(2,'b')]

Soluzione Esercizio zip

```
zip' (x:xs) (y:ys) = (x,y):zip' xs ys
zip' _ _ = []
```

Esercizio eliminaPrimi

- Scrivere una funzione

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

- che, dati una lista l e un intero n, fornisce la lista costituita dagli elementi della lista l tolti i primi n elementi, senza usare la funzione di libreria drop
- Es:
  Prelude> eliminaPrimi [1,1,2,1,3,4] 3
  [1,3,4]

Soluzione Esercizio eliminaPrimi

```
eliminaPrimi xs 0 = xs
eliminaPrimi (x:xs) n = eliminaPrimi xs (n-1)
```

Esercizio concatena

- Scrivere una funzione

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

- che, date due liste, le concateni senza usare la funzione di libreria (++)
- Es:
  Prelude> concatena [1,2,3,4] [5,6,7]
  [1,2,3,4,5,6,7]

Soluzione Esercizio concatena

```
concatena [] ys = ys
concatena (x:xs) ys = x:(concatena xs ys)
```

Esercizio replicate

- Scrivere una funzione

replicate :: Int -> a -> [a]

- che, date un intero n e un valore v, produca un lista di n ripetizioni di v
- Es:
  Prelude> replicate 4 True
  [True,True,True,True]

Soluzione Esercizio replicate

```
replicate 0 x = []
replicate n x = x:replicate (n-1) x
```

Esercizio merge

- Scrivere una funzione

merge :: Ord a=> [a] -> [a] -> [a]

- che, date due liste ordinate, le fonde generando una lista ordinata
- Es:
  Prelude> merge [2,5,6] [1,3,4]
  [1,2,3,4,5,6]

Soluzione Esercizio merge

```
merge (x:xs) (y:ys)
  | x>y = y:merge (x:xs) ys
  | otherwise = x:merge xs (y:ys)
merge [] ys = ys
merge xs [] = xs
```

Esercizio msort

- Scrivere una funzione

msort :: Ord a=> [a] -> [a]

- che ordina una lista usando il seguente algoritmo ricorsivo:
  - Le liste di lunghezza <= 1 sono già ordinate
  - Le altre liste possono essere ordinate dividendole in due metà, ordinando le due metà e fondendo con merge i le due metà ordinate
- Es:
  Prelude> msort [5,2,6,1,4,3]
  [1,2,3,4,5,6]
- Usare merge, take, drop, e div (divisione intera)

Soluzione Esercizio msort

```
msort [] = []
msort [x] = [x]
msort xs = merge (msort (take (div (length xs) 2) xs)) (msort (drop (div (length xs) 2) xs))
```

Esercizio filterDiv3

- Scrivere una funzione

filterDiv3 :: [a] -> [a]

- che, data una lista, elimina tutti gli elementi non divisibili per 3
- Si usi la funzione a mod b che calcola il resto della divisione intera di a per b
- Es:
  Prelude> filterDiv3 [1..10]
  [3,6,9]

Soluzione Esercizio filterDiv3

```
div3 :: Int -> Bool
div3 n = (n mod 3)==0
filterDiv3 xs = filter div3 xs
```

Esercizio factorial

- Scrivere una funzione

factorial :: Int -> Int

- che calcola il fattoriale
- Es:
  Prelude> factorial 6
  720

Soluzione Esercizio factorial

```
factorial n = foldr (*) 1 [1..n]
```