Programmazione Funzionale

Esercitazione 9 – Modulo List: Filtri, map, folding, init

Ricordiamo i effeti delle seguenti commandi del modulo `List`:

- `List.init` prende una funzione `f:int -> 'a` e un intero `n` e restituisce una lista di tipo `'a list` che corrisponde a `[f0;...;f(n-1)]`.
- `List.map` prende una funzione `f:'a -> 'b` e una lista `[a1;...;an] : 'a list` e restituisce `[f a1;...; f an]`.
- `List.filter` prende una funzione `f:'a -> bool` and a list `lst : 'a list` e restituisce la lista `lst` da cui cono stati tolti tutti i elementi tale che `f(a)=false` (oppure sono stati tenuti solo i elementi tale che `f(a)=true`).

  Ad esempio `List.filter (function x -> not(x=0)) [0;2;1;0;3]` restituisce `[2;1;3]`.
- `List.left_fold` ha un comportamento iterativo un po piu difficile da descrivere. La funzione prende tre argomenti, una funzione `f:'a -> 'b -> 'a` che puo essere pensata come un operazione binaria, un valore di partenza `start: 'a` e una lista `lst: 'b list`.

  L'espressione `E = List.left_fold f start [b1;...;bn]` si ridusce in `f(...(f(start,a1),a2)...)`. Cioé si puo definire per induzione su n; se $n = 0$ e quindi la lista e vuota l'espressione `E` restituisce `start`, se invece la lista ha un lunghezza $n+1 \neq 0$ cioè la lista e della forma `[b1;...;b(n+1)]` l'espressione restituisce `f( List.left_fold f start [b1;...;bn] , b(n+1) )`.

Esercizio 1. Vogliamo definire funzione che creano diverse liste.

1. Vogliamo definire la funzione `init` che prende due interi `i` e `lung` e restituisce la lista `[i;...;i+lung]`.
   (a) Definire `init` senza usare il commando `List.init`.
   (b) Definire `init` mediante il commando `List.init`.
2. Vogliamo definire una funzione `multiples` che prende due interi `n` e `k` e restituisce i primi `k` multipli di `n` cioè la lista `[0*k;...;n*k]`.
   (a) Definire `multiples` senza usare il commando `List.init`.
   (b) Definire `multiples` mediante il commando `List.init`.

Esercizio 2. La funzione `List.init` del modulo `List` puo essere definita mediante la funzione `List.map`;

1. Definire una funzione `firstn:int -> int list` che prende un intero `n` e restituisce la lista `[0;1;...;n-1]`.
2. Mediante la funzione `firstn` e la funzione `List.map` definire una funzione `init: int -> 'a list` che si comporta come `List.init`.

Esercizio 3. Dato un intero `k` la k–traslazione di una lista di interi `[a1;...;an]` e la lista `[a1+k;...;an+k]`.

1. Senza usare la funzione `List.map` definire una funzione `trasla` che prende un intero `k` e una lista `lst` e restituisce la `k`–traslazione di `lst`.
2. Definire `trasla` mediante `List.map`.
3. Implementare una funzione `mymap` che si comporta come la funzione `List.map`

Esercizio 4. Per calcore il massimo comune divisore di due interi n e m un metodo naturale puo essere di generare le liste i divisori di n e m e di costruire poi la loro intersezione, ottenando la lista dei divisori communi di n e m.

1. Senza usare la funzione `List.filter` definire la funzione `intersec` che prende due liste `la` e `lb` e costruisce la loro intersezione cioe la lista dei elementi di `la` che occorono in `lb`.
2. Definire `intersec` mediante `List.filter`.
3. Senza usare `List.filter`, definire una funzione `divisors:int -> int list` che restituisce la lista dei divisori di `n`.
4. Mediante `List.filter` e `List.init` definire la funzione `divisors`.
5. Definire la funzione `divisors2:int -> int -> int list` che prende due interi e restituisce la liste dei loro divisori comuni.

Esercizio 5. Vogliamo definire un metodo per filtrare una lista tale che ne teniamo solo i elementi pari, positivi, e inferiore o uguale a 10;

1. Definire una funzione `pari:int -> bool` che restituisce `true` se un intero e pari e `false` altrimenti.
2. Definire una funzione `dieci: int -> bool` che prende in entrata un intero `n` che restituisce `true` se e solo se $0 \leq n \leq 10$.

3. Definire una funzione `pariList` che prende una lista di interi e restituisce questa lista tenando solo i elementi pari.
4. Definire una funzione `dieciList` che prende una lista di interi e restituisce questa lista tenando solo i interi n tale che $0 \leq i \leq 10$.
5. Definire una funzione `filtro` che prende una lista di interi e restituisce questa lista tenando solo i interi n che sono pari e compreso fra 0 e n.
6. Definire `meet:(a' -> bool) -> (a' -> bool) -> (a' -> bool)` che prende due funzione `f` e `g` e restituisce la una funzione di tipo `'a -> bool` che applicata a un elemento `a` di tipo `'a` restituisce `true` se e solo se `f(a)=true` e `g(a)=true`.
7. Usando la funzione `meet`, definire la funzione `filtro` usando un unica volta la funzione `List.filter`.

Esercizio 6. Mediante le funzioni `List.fold_left` e `List.init` quando e utile, definire le funzione seguente:
1. `sumTo: int -> int` che prende un intero n e ritorna la somma $\sum_{1 \leq i \leq n} i$.
2. `sumFromTo: int -> int -> int` che prende due interi n e m e ritorna la somma $\sum_{n \leq i \leq m} i$.
3. la funzione fattoriale `fact:int -> int` che a un intero n restituisce $\prod_{1 \leq i \leq n} i$.

Esercizio 7. Vogliamo La funzione `listconcat : ('a list) list -> list` che prende una lista di liste e restituisce la loro concatenazione.
1. Definire `listconcat` senza usa la funzione `List.fold_left`.
2. Definire `binaryconcat: 'a list -> 'a list -> 'a list` che prende due liste `l1` e `l2` e restituisce la loro concatenazione.
3. Mediante `binaryconcat` e `List.fold_left` definire la funzione `listconcat`.

Esercizio 8. Vogliamo definire un concatenazione distributiva cioè una funzione `concatDist :'a -> ('a list) list -> ('a list) list` che prende un elemente `x` e una lista di liste `[l1;. . . ;ln]` e restituisce `[x::l1;. . . ;x::ln]`.
1. Definire `concatDist` senza usare la funzione `List.fold_left`.
2. Definire la funzione `add : 'a -> 'a list -> 'a list` che prende un elemento `x` e una lista `lst` e restituisce `x::lst`.
3. Mediante `add` e `List.fold_left` definire `concatDist`.

Esercizio 9. Definire la vostra versione delle funzioni del modulo `List`;
1. `List.init: int -> (int -> 'a) -> 'a list`.
2. `List.map: ('a -> 'b) -> ('a list) -> 'b list`.
3. `List.filter: ('a -> bool) -> 'a list -> 'a list`.
4. `List.fold_left:('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`.