Esercizi proposti – 4

1. Definire le funzioni seguenti (dandone, ove abbia senso, sia una versione non ricorsiva di coda che una ricorsiva di coda):

   (a) `length: 'a list -> int`, che riporta il numero di elementi in una lista (Notare che il modulo `List` contiene una funzione con questo nome, ma qui si chiede di ridefinirla per esercizio).

   (b) `sumof: int list -> int`, che riporta la somma degli elementi in una lista di interi.

   (c) `maxlist: 'a list -> 'a`, che riporta il massimo elemento in una lista (la lista vuota non ha elementi, quindi nemmeno un massimo; dunque se la lista è vuota deve essere sollevata un'eccezione).

   (d) `drop: int -> 'a list -> 'a list`, tale che `drop n lst` = lista che si ottiene da `lst` togliendone i primi n elementi. Se il numero di elementi di `lst` è minore di n (oppure uguale a n), allora `drop n lst = []`.

   (e) `append: 'a list -> 'a list -> 'a list`. Se `@` non fosse predefinito, come si potrebbe definire (utilizzando solo i costruttori delle liste)?

   (f) `reverse: 'a list -> 'a list`, che rovescia una lista, cioè riporta la lista che contiene gli stessi elementi di quella data, ma in ordine inverso (Notare che il modulo `List` contiene una funzione `rev` che rovescia una lista, ma qui si chiede di ridefinirla per esercizio).

   (g) `nth: int -> 'a list -> 'a`, tale che `nth n lst` = elemento di lst in posizione n, dove il primo elemento della lista è in posizione 0. La funzione solleverà un'eccezione se n è negativo o se la lista non contiene abbastanza elementi (Notare che il modulo `List` contiene una funzione con questo nome, ma qui si chiede di definirla per esercizio).

   (h) `remove: 'a -> 'a list -> 'a list`, tale che `remove x lst` elimina tutte le occorrenze di `x` da `lst`. Se non ve ne sono, viene riportata `lst` stessa.

2. Definire:

   (a) Una funzione `copy: int -> 'a -> 'a list` tale che `copy n x` riporti la lista di lunghezza `n` i cui elementi sono tutti uguali a `x`. Determinare il valore e il tipo di `copy 3 (copy 2 8)`.

   (b) Un predicato `nondec: int list -> bool` che, applicato a una lista `lst`, riporti `true` se gli elementi di `lst` sono in ordine non decrescente, `false` altrimenti.

   Ad esempio, `nondec [1;2;3;4] = true`, e `nondec [1;2;4;3] = false`.

   (c) Una funzione `pairwith: 'a -> 'b list -> ('a * 'b) list` che, applicata a un valore `y` e una lista `xs = [x1;x2;...;xn]`, riporti la lista `[(y,x1);(y,x2);....;(y,xn)]`.

   (d) Una funzione `duplica: 'a list -> 'a list` che, applicata a una lista `xs = [x1;x2;...;xn]`, riporti la lista `[x1;x1;x2;x2;...;xn;xn]`.

(e) Una funzione `enumera: 'a list -> (int * 'a) list` che, applicata a una lista `lst=[x0;x1;x2;...;xk]`, riporti la lista di coppie `[(0,x0);(1,x1);(2,x2);...;(k,xk)]`.

(f) Una funzione `position: 'a -> 'a list -> int` tale che `position x lst` riporti la posizione della prima occorrenza di `x` in `lst` (contando a partire da 0). Se `x` non occorre in `lst`, la funzione solleverà un'eccezione.

(g) Una funzione `alternate: 'a list -> 'a list` che, applicata a una lista `lst`, riporti la lista contentente tutti e soli gli elementi di `lst` che si trovano in posizione dispari. Ricordiamo che, per convenzione, il primo elemento di una lista si trova in posizione 0, il secondo in posizione 1, ecc. Quindi, ad esempio, `alternate [0;1;20;32;4;5] = [1;32;5]`.

(h) Una funzione `min_dei_max: int list list -> int` che, data una lista di liste di interi, riporti il valore minimo tra i massimi di ciascuna lista. Ad esempio, per la lista `[[3;100;1;9];[2;10;20];[80;65;4]]`, si otterrà il valore 20.

(i) `split2: 'a list -> 'a list * 'a list`, che suddivide una lista in due liste di lunghezza più o meno uguale, utilizzando le funzioni `take: int -> 'a list -> 'a list`, definita a lezione, e `drop`, dell'esercizio 1d. Come la funzione `split` definita a lezione, la `split2` si potrebbe utilizzare per implementare il *merge sort*.

A differenza di `split`, che mette gli elementi in posizione pari nella prima lista, quelli in posizione dispari nella seconda, `split2` metterà i primi elementi da una parte e gli ultimi dall'altra. Ad esempio, `split2 [1;2;3;4;5;6;7] = ([1;2;3], [4;5;6;7])`, mentre `split [1;2;3;4;5;6;7] = ([1;3;5;7], [2;4;6])`.