Soluzioni Primo Compitino di Programmazione - 20 gennaio 2011
Tema A

Esercizio 1. Scrivere il tipo della seguente funzione `f`:

```
let rec f x =
   match x with
   | [] -> 0
   | y::z -> if (y mod 2 <>0) then y + (f z) else (f z);;
```

Che cosa restituisce ?

Soluzione. Il tipo della funzione è:

```
f : int list -> int = <fun>
```

Si tratta della funzione ricorsiva che calcola la somma degli elementi dispari di una lista di interi.

Esercizio 2. In matematica, dati due numeri interi m e n, un intervallo $[m,n]$ rappresenta l'insieme di tutti i numeri interi x tali che $m \leq x \leq n$. Scrivere una funzione `interval` tale che dati due numeri m e n restituisca l'intervallo corrispondente. Per esempio:

```
(interval 0 5)= [0; 1; 2; 3; 4; 5]
(interval 6 9) = [6; 7; 8; 9].
```

Si dica se la funzione definita è iterativa o ricorsiva. Infine si scriva il tipo della funzione `interval`.

Soluzione.

FUNZIONE RICORSIVA

```
# let rec interval m n =
        if m > n then []
        else m::interval (m+1) n;;
interval : int -> int -> int list = <fun>
```

FUNZIONE ITERATIVA

```
# let interval (m,n) =
         let rec aux (m,n,acc) =
             if m>n then acc
             else aux (m, n-1, n::acc)
         in aux (m,n,[]);;
interval : int * int -> int list = <fun>
```

Esercizio 3. Definire il tipo albero e scrivere la funzione `nodes` che conta il numero di nodi presenti nell'albero. Inoltre si scriva il tipo della funzione `nodes`.

Soluzione.

```
type a btree = Empty | Node of a * a btree * a btree;;
Type btree defined.
```

```
# let rec nodes bt = match bt with
           Empty -> 0
           | Node (x, lt, rt) -> 1 + (nodes lt) + (nodes rt);;
nodes : 'a btree -> int = <fun>
```

Esercizio 4. Scrivere una funzione `minMax` che, data una lista non vuota di liste non vuote di interi, restituisca il valore minimo tra i massimi di ciascuna lista. Per esempio:

`minMax([[3;100;1;9]; [2;10;20]; [80;65;4]]) = 20.`

Infine si scriva il tipo della funzione `minMax`.

Soluzione.

```
# let rec max lst =
        match lst with
        | [x] -> x
        | x::y::ys -> if (x>y) then max (x::ys) else max (y::ys);;
max : 'a list -> 'a = <fun>
# let rec min lst =
        match lst with
        | [x] -> x
        | x::y::ys -> if (x<y) then min (x::ys) else min (y::ys);;
min : 'a list -> 'a = <fun>
#let rec minMax lst =
        match lst with
        | [x] -> min x
        | x::y::ys -> min [(max x); (minMax (y::ys))];;
minMax : 'a list list -> 'a = <fun>
```

Esercizio 5. Si definisca una funzione `swap` che, data una lista `l` e un indice intero `k` con $k \geq 0$, restituisce la lista che si ottiene da `l` scambiando l'ordine degli elementi di posizione `k` e `k+1` (e solo di questi due elementi). L'indice della prima posizione della lista è `0`; inoltre, se `k` o `k+1` non sono indici validi, allora `swap` restituisce la lista `l` immutata. Per esempio:

```
swap([1;2;3;4;5;6;7;8],0) = [2;1;3;4;5;6;7;8]
swap([1;2;3;4;5;6;7;8],3) = [1;2;3;5;4;6;7;8]
swap([1;2;3;4;5;6;7;8],6) = [1;2;3;4;5;6;8;7]
swap([1;2;3;4;5;6;7;8],7) = [1;2;3;4;5;6;7;8]
```

Inoltre si scriva il tipo della funzione `swap`.

Soluzione.

```
#let rec swap (l,k) = match (l,k) with
         ([],k) | ([x],k)  -> l
         | (x::y::xs, 0) -> y::x::xs
         | (x::xs, k) -> x:: swap(xs,k-1);;
swap : 'a list * int -> 'a list = <fun>
```