Primo Compitino di Programmazione - 15 novembre 2013

Cognome ..... **Nome** ..... **Matricola** .....

Esercizio 1. Scrivere una funzione `eliminate` che dati un numero intero `k` e una lista `lst` restituisca la lista che si ottiene da `lst` rimuovendo il `k`-eseimo elemento, assumendo che il primo elemento della lista si trovi in posizione 0, il secondo in posizione 1, e così via.
Per esempio:

```
# eliminate 2 [``a'';``b'';``c'';``d''];;
- : string list = [``a''; ``b''; ``d'']
```

Infine si scriva il tipo della funzione `eliminate`.

Soluzione Esercizio 1.

```
# let rec eliminate k lst = match lst with
     | [] -> []
     | h :: t -> if k = 0 then t else h :: eliminate (k-1) t;;
 val eliminate : int -> 'a list -> 'a list = <fun>
```

Esercizio 2. La cogettura di Goldbach dice che ogni numero positivo pari maggiore di 2 è la somma di due numeri primi. Scrivere una funzione `goldbach` che dato un numero intero `n` restituisca una coppia di numeri primi che sommati danno `n`.
Per esempio:

```
# goldbach 16;;
- : int * int = (3, 13)
```

Infine si scriva il tipo della funzione `goldbach`.

Soluzione Esercizio 2.

```
# let prime n =
          let rec trynext i =
              (n = i) || ((n mod i !=0) & trynext (i+1))
          in trynext 2;;
prime : int -> bool = <fun>

 # let goldbach n =
     let rec aux d =
       if prime d && prime (n - d) then (d, n-d)
       else aux (d+1) in
     aux 2;;
 val goldbach : int -> int * int = <fun>
```

Esercizio 3. Scrivere una funzione `primes` che dato un numero intero `n` restituisca la lista dei fattori primi di `n` con le opportune ripetizioni.
Per esempio:

```
# primes 525;;
- : int list = [3; 5; 5; 7].
```

Infine si scriva il tipo della funzione `primes`.

Soluzione Esercizio 3.

```
# let primes n =
    let rec aux d n =
      if n = 1 then [] else
        if n mod d = 0 then d :: aux d (n / d) else aux (d+1) n
    in
    aux 2 n;;
primes : int -> int list = <fun>
```

Esercizio 4. Definire una funzione ricorsiva `zip: 'a list -> 'a list` che data una lista di elementi `lst` restituisca la lista ottenuta da `lst` sostituendo ogni sequenza di elementi uguali e contigui in `lst` con una sola occorrenza di tali elementi.
Per esempio:

```
# zip [1;1;2;2;2;3;3;4;3;3;3])
- : int list = [1;2;3;4;3].
```

Soluzione Esercizio 4.

```
# let rec zip lst = match lst with
                [] -> []
                | [x] -> [x]
                | x::y::xs when x<>y -> x::zip(y::xs)
                | x::y::xs when x=y -> zip(x::xs);;
```