1) Indicare il tipo delle seguenti funzioni

```
let f g x = if g x = 3 then x else g (x+1) ;;
let f g x y = if g x = y then x else y;;
let f g x y= if g x y then y else y + x;;
let f g h x = g (h x) (h (x+1));;
let f g h x y = if x = y then h y else g x;;
```

2) Definire una funzione ricorsiva *incrnultimo* con tipo
incrnultimo: int list -> int list
che incrementa di uno l'ultimo elemento (se esiste) di una lista di interi.

3) Definire una funzione ricorsiva *insbefore* con tipo
insbefore : 'a -> 'a ->'a list -> 'a list
tale che (*insbefore x n l*) inserisce il valore *n* prima della prima occorrenza del valore *x*. Inserisce *n* alla fine se *x* non compare nella lista. Es. *insbefore 3 7 [1;2;3;4;5;9;3;8] = [1;2;7;3;4;5;9;3;8].*

4) Definire una funzione ricorsiva *split* con tipo
*split : 'a -> 'a list -> 'a list * 'a list*
che data un elemento *e* e una lista *l* restituisce le due liste che contengono rispettivamente gli elementi minori e maggiori o uguali di *e* in *l*. Es. *split 3 [1;5;9;2;3;4;7] = ([1;2],[5;9;3;4;7]).*

5) Definire una funzione ricorsiva *deleq* con tipo
deleq : 'a list -> 'a list -> 'a list
tale che (*deleq l1 l2*) cancella da *l1* tutti gli elementi che compaiono anche in *l2*.
Es. *deleq [1;2;3;7;3;8;3;4;5;3;9;4;7] [7;3;4;5;7] = [1;2;8;9].*

6) Definire una funzione ricorsiva *check* con tipo
check : int list -> bool
che controlla che tutti gli elementi della lista siano diversi tra di loro.

f : (int → int) → int → int

f : ('a → 'a) → 'a → 'a → 'a

f : (int → int → bool) → int → int → int

f : ('a → 'a → 'b) → (int → 'a) → int → 'b

f : ('a → 'b) → ('a → 'b) → 'a → 'a → 'b

```
let rec incultimo l = match l with
    [] -> []
  | [x] -> [x+1]
  | x::y::ys -> x :: incultimo (y::ys);;
```

```
let rec insbefore x n l =
        match l with
    [] -> [n]
  | y::ys when x=y -> n::y::ys
  | y::ys when x<>y -> y:: insbefore x n ys ;;
```

```
let rec split e l = match l with
    []  -> ([],[])
  | x::xs when x<e ->
        let (l1,l2) = split e xs
        in (x::l1, l2
  | x::xs when x>=e ->
        let (l1,l2) = split e xs
        in (l1, x::l2) ;;
```

Notare che si può dichiarare una coppia di nomi. Ordinamente l'espressione associata alla coppia di nomi dovrà restituire una coppia di valori

```
let rec deleq l1 l2 =
    let rec member n l =
        match l with
          [] -> false
        | x::xs when x = n -> true
        | x::xs when x<>n -> member n xs
    in math l1 with

      [] -> []
    | x::xs when member x l2
        -> deleq xs l2
    | x::xs when not member x l2
        -> x :: deleq xs l2 ;;
```

```
let rec check l =
    let rec member n l = come so/no
    in
      match l with
        [] -> true
      |x::xs when member x xs -> false
      |x::xs when not member x xs
          -> check xs ;;
```