1) Indicare il tipo delle seguenti funzioni

```
let f g n m = m + g (n+2) ;;
let f g n m = g n + g m;;
let f g x = g (x+1) = 3;;
let f x y z = match (x y) with
        [] -> [] |
       n::ns -> z :: ns;;
let f g x n = if g (x+1) then n else n+1;;
```

2) Definire una funzione ricorsiva *check* con tipo
check : ('a -> bool) -> 'a list -> bool
tale che (*check p l*) vale true se e solo se il predicato *p* vale sull'ultimo elemento della lista *l*. Se la lista è vuota il valore calcolato deve essere false.

3) Definire una funzione ricorsiva *pair* con tipo
pair : 'a list -> 'a -> bool
tale che (*pair l n*) vale true se e solo se il valore *n* compare almeno due volte nella lista *l*.

4) Definire una funzione ricorsiva *cancdoppi* con tipo
cancdoppi : 'a list -> 'a list
che tutti gli elementi consecutivi ripetuti nella lista *l*, lasciandone solamente uno.
Esempio: *cancdoppi [1;2;2;2;2;5;4;23;4;4;5;6;7;7;7] = [1;2;5;4;23;4;5;6;7].*

5) Definire una funzione ricorsiva *deluntil* con tipo
deluntil : 'a list ->'a -> 'a list
tale che (*deluntil l n*) cancella da *l* tutti gli elementi che precedono l'elemento di valore *n*. Se non compare nessun elemento di valore *n* viene restituita la lista vuota.

6) Definire una funzione ricorsiva *delfrom* con tipo
delfrom : 'a list ->'a -> 'a list
tale che (*deluntil l n*) cancella da *l* tutti gli elementi che seguono l'elemento di valore *n*. Se non compare nessun elemento di valore *n* viene restituita la lista stessa.

f : (int → int) → int → int → int

f : ('a → int) → 'a → 'a → int

f : (int → int) → int → bool

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

f : (int → bool) → int → int → int

```
let rec check p l = match l with
    []  -> false
  | [x] -> p x
  | x::y::ys -> check y::ys ;;
```

```
let rec pair l n =
    let rec member x l = match l with
                []  -> false
              | y::ys when x=y -> true
              | y::ys when not(x=y) -> member x ys
    in match l with
        []  -> false
      | [x] -> false
      | x::y:ys -> if member x (y::ys) then true
                   else pair (y::ys) ;;
```

```
let rec concdoppi l = match l with
    [] -> []
  | [x] -> [x]
  | x::y::ys when x=y -> concdoppi (y::ys)
  | x::y::ys when x<>y -> x::concdoppi (y::ys);;

let rec deluntil l n = match l with
    [] -> []
  | x::xs when n=x -> x::xs
  | x::xs when n<>x -> deluntil xs n;;

let rec delfrom l n = match l with
    [] -> []
  | x::xs when n=x -> [x]
  | x::xs when n<>x -> x::delfrom xs n;;
```