Cancellare gli elementi che seguono l'ultima occorrenza del valore $\emptyset$.
Se il valore $\emptyset$ non occorre il risultato è la lista vuota

```
let rec member el l =
        match l with
          [] -> false
        | x::xs -> if x = el then true
                   else member el xs;;

let rec canc l =
        match l with
          [] -> []
        | x::xs -> if not member ∅ (x::xs)
                   then []
                   else x:: canc xs;;
```

```
canc [∅; 2; ∅; 3]
= {def canc}
  if not mmbr ∅
= 0:: canc [2; ∅; 3]
```

$= \emptyset :: \text{conc } [2; \emptyset; 3]$

$= \{\text{def conc}\}$

$\emptyset :: \text{if not member } \emptyset\ [2; \emptyset; 3]$

$\emptyset :: 2 :: \text{conc } [\emptyset; 3]$

$= \{ \text{“} \}$

$\emptyset :: 2 :: \emptyset :: \text{conc } [3]$

$= \{ \text{“} \}$

$\emptyset :: 2 :: \emptyset :: [] = [\emptyset; 2; \emptyset]$

Voglio definire la stessa funzione senza usare member

```
let canc l =
    let rec canc_b l =
        match l with
        [] -> ([], false)
      | x::xs -> let (l1, b) = canc_b xs
                 in if b then (x::l1, b)
                    else if x=a
                         then ([x], true)
                         else (x::l1, b)
    in let (l1, b) = canc_b l
       in if b then l1
          else [] ;;
```

Se l'esercizio dicesse:

in caso in cui $\emptyset$ non compare restituire l'intera lista

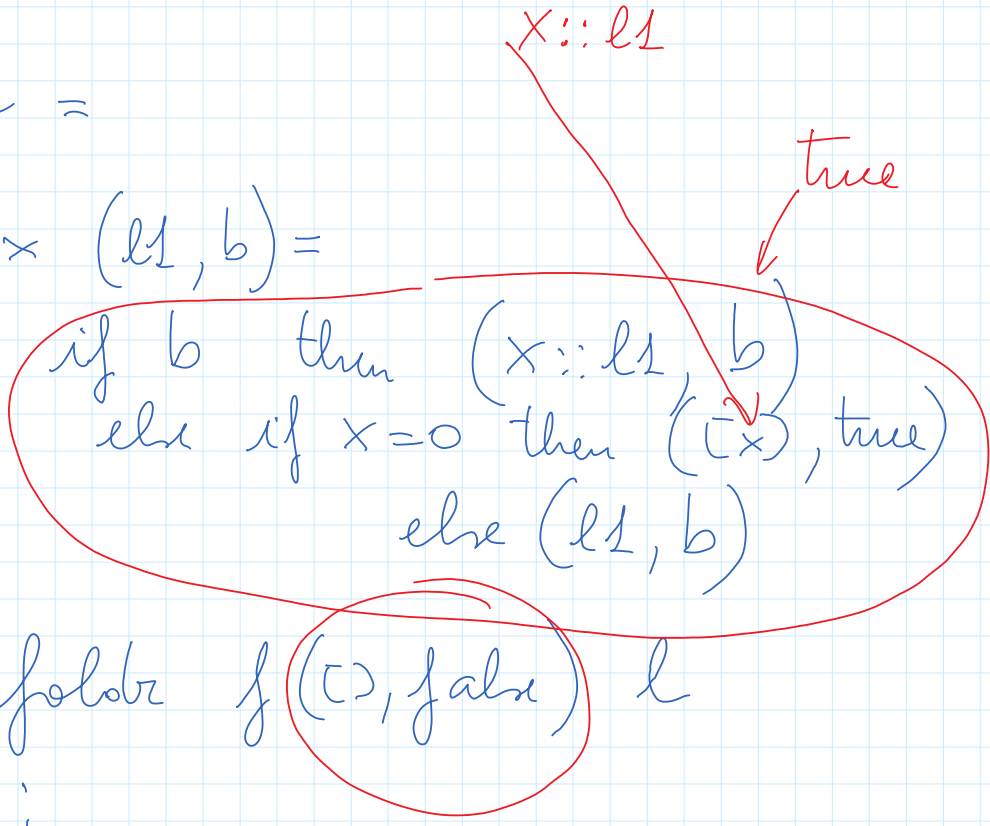
```
let conc l =
    let rec conc_b l = ...
    in let (l1, b) = conc_b l
    in l1;
```

Se 0 non occorre la funzione restituisce le liste vuote.

```
let conc l =
    let rec conc_b l =
        match l with
          [] -> ([], false)
        | x::xs -> let (l1, b) = conc_b xs
                   in
                   if b then (x::l1, b)
                   else if x=0
                        then ([x], true)
                        else ([], false)
    in let (l1, b) = conc_b l
       in l1 ;;
```

Meccanismo folobr

Se $\emptyset$ non occorre restituiamo la lista vuota.




```
let conc l =
    let f x (l1, b) =
        if b then (x::l1, b)
        else if x=0 then ([x], true)
             else (l1, b)
    in
    let (l1,b) = folobr f ([], false) l
    in l1 ;;
```

folobr

se ∅ non occorre ni restituisce l'intera lista

```
let conc l =
  let f x (l1,b) =
    if b then (x::l1, b)
    else if x=∅ then ([x], true)
    else (x::l1, b)
  in let (l1,b) = folobr f ([],false) l
  in l1;
```

x :: l1

Usando foldr si definisca la funzione split.

split : 'a list → 'a list * int list

che data una lista l, restituisca la coppia (l1, l2) tale che:

- l1 @ l2 = l
- l2 è la sottolista finale più lunga possibile i cui elementi sono in ordine crescente

split [2;3;-1;4;5;10] = ([2;3],[-1;4;5;10])

```
let split l =
    let f x (l1,l2,b) =
        if l2 = [] then (l1, x::l2, b)
        else if b then
            if x < hd l2
            then (l1, x::l2, b)
            else (x::l1, l2, false)
        else (x::l1, l2, b)
```

[x]

```
                else (x::l1, l2, b)
in let (l1, l2, b) = folobr f (_,_,true) l
                                  a
   in (l1, l2) ;;
```

foldr

prec : 'a list → 'a list
che data una lista l, restituisce la lista degli elementi che sono preceduti da un elemento minore. Il primo elemento di una lista NON è preceduto da un elemento minore.

prec [3;4;5;2;-1;7;3;8] = [4;5;7;8]

```
let prec l =
  let f x (y, l1) =
      if x < y then (x, y :: l1)
               else (x, l1)
  in let (n, l2) = foldr f
                   (tail l, []) (butlast l)

let rec last l =
    match l with
    [x] → x
  | x :: y :: ys → last (y :: ys) ;;
```

```
| x::y::ys -> last (y::ys);;

let rec canclast l =
    match l with
    [] -> []
  | [x] -> []
  | x::y::ys -> x:: canclast (y::ys);;

let last l = hd (rev l);;

let canclast l = match rev l with
    [] -> []
  | x::xs -> rev xs;;
```

folobr

split : 'a list → 'a → 'a list * 'a list

data una lista l e un valore x restituisce la coppia (l1, l2) tale che l1 contiene tutti i valori di l che precedono l'ultimo occorrenza di x e l2 gli altri.

Se l non contiene il valore x, l1 sarà la lista vuota.

$[x_1; x_2; \dots\dots x_n]$

$\longleftarrow$

(l1, l2, b)

```
let split l x =
    let f x (l1, l2, b) =
        if b then (x :: l1, l2, b)
        else if x = x then (l1, x :: l2, true)
        else (l1, x :: l2, b)
    in let (l1, l2, b) = folobr f ([], [], false) l
```

```
in let (l1, l2, b) = folobr f ([], [], false) l
in (l1, l2);;
```

split : 'a list -> 'a -> 'a list * 'a list

split l x = (l1, l2)

l1 contiene gli elementi che PRECEDONO la PRIMA occorrenza di x

soluzione

-

[x, l3 x, l2]

([], x::l2, true)

(l3, x::l2, true)

([]; x :: l3 @ (x::l2), true)

let split l n =

let f x (l1, l2, b) =

```
if x = n then ([], x :: l1 @ l2, true)
else
  if b then (x :: l1, l2, b)
       else (l1, x :: l2, b)

in
let (l1, l2, b) = foldbr f ([], [], false) l
in (l1, l2) ;;
```