```
(* ex.3.2.1 *)
- fun fact i = if i = 0 then 1 else i * fact (i-1);
val fact = fn : int -> int
-
fun cycle1 L = if null L then nil else tl(L) @ [hd(L)];
val cycle1 = fn : 'a list -> 'a list

fun  cyclen (L,n) = if n=0 then L else cyclen(cycle1(L), n-1);
val cyclen = fn : 'a list * int -> 'a list

fun dup L = if null L then nil else [hd(L),hd(L)]@dup(tl(L));
val dup = fn : 'a list -> 'a list

(* OR: *)
fun dup L = if null L then nil else hd(L)::hd(L)@dup(tl(L));


- fun len L = if null L then 0 else 1 + len (tl L);
val len = fn : 'a list -> int

- fun power (x: real, i) = if i = 0 then x else x * power (x, i-1);
val power = fn : real * int -> real

Exercise 3.2.1(f) [Ullman]
Note: Because real is not an equality type, and the function null to test
whether a list is empty even if the list is not of an equality type is not
introduced in Chapter 3, I have changed Exercise 3.2.1(f) to ask for the
maximum of a list of strings instead, and the code below reflects that change.
If you want to make the code work for lists of reals, you have to replace
tl(L)=nil in the second line by null(tl(L)), and of course change the type
declaration on line (1).
fun maxList(L: string list) =
        if tl(L)=nil (* L is a single element *)
                then hd(L) (* the single element is the maximum *)
                else (* assume there are at least 2 elements *)
                        if hd(L)>hd(tl(L)) (* the first element
                                        exceeds the second *)
                                then maxList(hd(L)::tl(tl(L))) (* eliminate
                                                        second element *)
                                else maxList(tl(L)) (* eliminate first
                                                element *);



(* 3.3.1 *)

- fun fact 0 = 1
= |   fact i = i * fact (i-1);
val fact = fn : int -> int
-

- fun cyclen (L, 0) = L
= |   cyclen (L, n) = cyclen (cycle1 (L), n-1);
GC #0.0.0.0.1.7:   (10 ms)
val cyclen = fn : 'a list * int -> 'a list
-

- fun dup [] = []
= |   dup (x::xs) = x::x::dup(xs);
val dup = fn : 'a list -> 'a list
-
- fun power (x:real, 0) = x
= |   power (x, i) = x * power (x, i-1);
val power = fn : real * int -> real

Exercise 3.3.1(f) [Ullman]
fun maxList([x:real]) = x
|   maxList(x::y::zs) =
       if x<y then maxList(y::zs)
              else maxList(x::zs);
```



```
(* Exercise 3.3.2 *)

- fun flip [] = []
= |   flip [x] = [x]
= |   flip (x::y::ys) = y::x::flip(ys);
val flip = fn : 'a list -> 'a list
-

(* Exercise 3.3.3 *)
- fun deli ([], i) = []
= |   deli (x::xs, 1) = xs
= |   deli (x::xs, i) = x::deli(xs,i-1);
val deli = fn : 'a list * int -> 'a list
-

(* Exercise 3.3.7 *)
val it = [1,2,4] : int list
- fun sq 0 = 0
= |   sq n = sq(n-1) + 2*n -1;
val sq = fn : int -> int
```