type date = int × int × int

(* Change date format from (day, month, year) to (month, day, year) *)

fun dmy_to_mdy (d : date) = (#2 d, #1 d, #3 d)

val date1 : date = dmy_to_mdy (11, 1, 1900)
val date2 : int × int × int = dmy_to_mdy (22, 2, 1900)

(* We can call dmy_to_mdy on either c1 or c2 *)

fun append (xs, ys) = 
  if null xs 
  then ys 
  else hd xs :: append(tl xs, ys)

val ok1 = append(["hi", "bye"],["programming", "languages"])
val ok2 = append([1, 2], [3, 4])

(* Examples *)

1. 'a list * 'a list -> 'b list
   int list * int list -> string list
2. 'a list * 'a list -> 'b list
   int list * int list -> int list
3. 'a list * 'a list -> 'b list
   int list * string list -> int list
4. 'a list * 'a list -> 'a list
   'a list * 'a list -> 'b list
5. type foo = int * int
   int list * foo -> foo list
   int list * (int * int) -> (int * int) list

(* Equality Types *)

fun contains(x, xs) = 
  if null xs 
  then false 
  else (* if (hd x = xs) then true else contains(x, tl xs) ... notice 
         that this is exactly how we defined orelse using 'syntactic sugar' 
         using orelse is both more readable and more intuitive. *) 
    (hd xs = x) ∨ contains(x, tl xs)

fun same_thing(x, y) = if x = y then "yes" else "no" 
  (* "'a * 'a -> string" *)

fun is_three(x, y) = if x = 3 then "yes" else "no" 
  (* int * 'a -> string *)

(* if-then-else *)

if c0 then e0 else e1