```
-- AUTOR: Javier Campos. Modificado por Elvira Mayordomo Cámara
-- PROYECTO: módulo de declaración del TAD listas con acceso por posición
-- FICHERO: listasap.ads
-- FECHA: 31-X-96; 16-10-03; 11-11-03

-- especificación listas_acceso_por_posición
--   usa booleanos, naturales
--   parámetro formal
--     género elemento
--   fpf
--   género lista
--   operaciones
--     listaVacía: →lista
--     +izq: elemento lista → lista
--     longitud: lista → nat
--     está: elemento lista → bool
--     esvacía: lista → bool
--     _&_: lista lista → lista
--     insertar: lista nat elemento → lista
--     borrar: lista nat → lista
```

-- **parcial** _[_]: lista nat $\rightarrow$ elemento
-- posición: elemento lista $\rightarrow$ nat
-- **dominios de definición** e:elemento; l:lista; i:nat
-- l[i] **está definido sólo si** (1<=i)$\wedge$(i<=longitud(l))
-- **ecuaciones** e,e1,e2:elemento; l,l1,l2:lista; i:nat
-- longitud(listaVacía) = 0
-- longitud(+izq(e,l)) = longitud(l)+1
-- está(e,listaVacía) = falso
-- está(e1,+izq(e2,l)) = (e1=e2) $\vee$ está(e1,l)
-- esvacía(listaVacía) = verdad
-- esvacía(+izq(e,l)) = falso
-- listaVacía & l = l
-- +izq(e,l1) & l2 = +izq(e,l1 & l2)
-- insertar(l,1,e) = +izq(e,l)
-- (i>1)$\wedge$(i<=longitud(+izq(e,l))+1) $\rightarrow$
-- insertar(+izq(e1,l),i,e2) = +izq(e1,insertar(l,i-1,e2))
-- (1>i)$\vee$(i>longitud(l)+1) $\rightarrow$ insertar(l,i,e) = l
-- borrar(+izq(e,l),1) = l
-- (i>1) $\wedge$(i<=longitud(+izq(e,l)) $\rightarrow$ borrar(+izq(e,l),i) = +izq(e,borrar(l,i-1))
-- (1>i)$\vee$(i>longitud(l)) $\rightarrow$ borrar(l,i,e) = l

```
--      +izq(e,l)[1] = e
--      i>1 → +izq(e,l)[i] = l[i-1]
--      posición(e,+izq(e,l)) = 1
--      está(e1,+izq(e2,l)) ∧ (e1≠e2) → posición(e1,+izq(e2,l)) = posición(e1,l)+1
--      ¬está(e,l) → posición(e,l) = 0
-- fespec

generic
      type elemento is private;
package listasap is
      type lista is limited private;
      -- coste en memoria: una lista de n datos ocupa O(n)
      procedure creaVacia(l:out lista);
      -- Post: l=listaVacía
      -- coste en tiempo O(1)
      procedure aniadirIzq(e:in elemento; l:in out lista);
      -- Pre: l=l0
      -- Post: l=+izq(e,l0)
      -- coste en tiempo O(1)
      procedure aniadirDch(l:in out lista; e:in elemento);
```

```
-- Pre: l=l0
-- Post: l=+dcha(l0, e)
-- coste en tiempo O(1)
procedure concatenar(l1:in out lista; l2:in lista);
-- Pre: l1=l0
-- Post: l1=l0 & l2
-- coste en tiempo O(1)
-- Atención: No se realiza copia, la lista l2 ya no se puede usar ya que forma parte
-- de l2
function longitud(l:lista) return natural;
-- Post: longitud(l)= longitud(l)
-- coste en tiempo O(1)
function esta(e:elemento; l:lista) return boolean;
-- Post: esta(l)= está(l)
-- coste en tiempo: para una lista de n datos tarda O(n)
function esVacia(l:lista) return boolean;
-- Post: esVacia(l)= esvacía(l)
-- coste en tiempo O(1)
procedure insertar(l:in out lista; i:in natural; e:in elemento);
-- Pre: l=l0
```

```
        -- Post: l=insertar(l0,i,e)
        -- coste en tiempo O(i)
        procedure borrar(l:in out lista; i:in natural);
        -- Pre: l=l0
        -- Post: l=borrar(l0,i)
        -- coste en tiempo O(i)
        function observar(l:lista; i:positive) return elemento;
        -- Pre: (1<=i)∧(i<=longitud(l))
        -- Post: observar(l,i)=l[i]
        -- coste en tiempo O(i)
        function posicion(e:elemento; l:lista) return natural;
        -- Post: posicion(e,l)=posición(e,l)
        -- coste en tiempo: para una lista de n datos tarda O(n)
private
        type unDato;
        type ptUnDato is access unDato;
        type unDato is
                record
                        dato: elemento;
                        sig: ptUnDato;
```

```
        end record;
    type lista is
        record
            prim,ult: ptUnDato;
            n:natural;
        end record;
end listasap;
```