1. Mï¿½dulo Lista Enlazada(α)

Interfaz

parï¿½metros formales
gï¿½neros α
funciï¿½n COPIAR(**in** a: α) $\rightarrow$ res : α
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} a$}
Complejidad: $\Theta(copy(a))$
Descripciï¿½n: funciï¿½n de copia de α's

se explica con: SECUENCIA(α), ITERADOR BIDIRECCIONAL(α).

gï¿½neros: `lista`(α), `itLista`(α).

Operaciones bï¿½sicas de lista

VACIï¿½A() $\rightarrow$ res : `lista`(α)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} <>$}
Complejidad: $\Theta(1)$
Descripciï¿½n: genera una lista vacï¿½a.

AGREGARADELANTE(**in/out** l: `lista`(α), **in** a: α) $\rightarrow$ res : `itLista`(α
Pre $\equiv$ {$l =_{\text{obs}} l_0$}
Post $\equiv$ {$l =_{\text{obs}} a \bullet l_0 \wedge res =$ CrearItBi(<>, l) $\wedge$ alias(SecuSuby(res) = l)}
Complejidad: $\Theta(copy(a))$
Descripciï¿½n: agrega el elemento a como primer elemento de la lista. Retorna un iterador a l, de forma tal que Siguiente devuelva a.
Aliasing: el elemento a agrega por copia. El iterador se invalida si y sï¿½lo si se elimina el elemento siguiente del iterador sin utilizar la funciï¿½n ELIMINARSIGUIENTE.

Operaciones del iterador

CREARIT(**in** l: `lista`(α)) $\rightarrow$ res : `itLista`(α)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ crearItBi(<>, l) $\wedge$ alias(SecuSuby(it) = l)}
Complejidad: $\Theta(1)$
Descripciï¿½n: crea un iterador bidireccional de la lista, de forma tal que al pedir SIGUIENTE se obtenga el primer elemento de l.
Aliasing: el iterador se invalida si y sï¿½lo si se elimina el elemento siguiente del iterador sin utilizar la funciï¿½n ELIMINARSIGUIENTE.

CREARITULT(**in** l: `lista`(α)) $\rightarrow$ res : `itLista`(α)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ crearItBi(l, <>) $\wedge$ alias(SecuSuby(it) = l)}
Complejidad: $\Theta(1)$
Descripciï¿½n: crea un iterador bidireccional de la lista, de forma tal que al pedir ANTERIOR se obtenga el ï¿½ltimo elemento de l.
Aliasing: el iterador se invalida si y sï¿½lo si se elimina el elemento siguiente del iterador sin utilizar la funciï¿½n

EliminarSiguiente.

Representaciï¿½n

Representaciï¿½n de la lista

`lista`(α) **se representa con** `lst`

donde `lst` es `tupla`(*primero*: `puntero(nodo)`, *longitud*: `nat`)
donde `nodo` es `tupla`(*dato*: α, *anterior*: `puntero(nodo)`, *siguiente*: `puntero(nodo)`)

Rep : lst $\longrightarrow$ bool
Rep(l) $\equiv$ true $\Longleftrightarrow$ (l.primero = NULL) = (l.longitud = 0) $\wedge_{\text{L}}$ (l.longitud $\neq$ 0 $\Rightarrow_{\text{L}}$
Nodo(l, l.longitud) = l.primero $\wedge$
($\forall i$: nat)(Nodo(l,i)$\rightarrow$siguiente = Nodo(l,$i+1$)$\rightarrow$anterior) $\wedge$
($\forall i$: nat)($1 \leq i < l$.longitud $\Rightarrow$ Nodo(l,i) $\neq$ l.primero)

Nodo : lst l $\times$ nat $\longrightarrow$ puntero(nodo) {l.primero $\neq$ NULL}
Nodo(l,i) $\equiv$ **if** $i = 0$ **then** l.primero **else** Nodo(FinLst(l), $i - 1$) **fi**

FinLst : lst $\longrightarrow$ lst
FinLst(l) $\equiv$ Lst(l.primero$\rightarrow$siguiente, l.longitud $-$ mín{l.longitud, 1})

Lst : puntero(nodo) $\times$ nat $\longrightarrow$ lst
Lst(p, n) $\equiv$ $\langle p, n \rangle$

Abs : lst l $\longrightarrow$ secu(α) {Rep(l)}
Abs(l) $\equiv$ **if** l.longitud = 0 **then** <> **else** l.primero$\rightarrow$dato $\bullet$ Abs(FinLst(l)) **fi**

Representaciï¿½n del iterador

`itLista`(α) **se representa con** `iter`

donde `iter` es `tupla`(*siguiente*: `puntero(nodo)`, *lista*: `puntero(lst)`)

Rep : iter $\longrightarrow$ bool
Rep(it) $\equiv$ true $\Longleftrightarrow$ Rep($*$(it.lista)) $\wedge_{\text{L}}$ (it.siguiente = NULL $\vee_{\text{L}}$ ($\exists i$: nat)(Nodo($*it$.lista, i) = it.siguiente)

Abs : iter it $\longrightarrow$ itBi(α) {Rep(it)}
Abs(it) $=_{\text{obs}}$ b: itBi(α) | Siguientes(b) = Abs(Sig(it.lista, it.siguiente)) $\wedge$
Anteriores(b) = Abs(Ant(it.lista, it.siguiente))

Sig : puntero(lst) l $\times$ puntero(nodo) p $\longrightarrow$ lst {Rep($\langle l, p \rangle$)}
Sig(i,p) $\equiv$ Lst(p, $l\rightarrow$longitud $-$ Pos($*l$, p))

Ant : puntero(lst) l $\times$ puntero(nodo) p $\longrightarrow$ lst {Rep($\langle l, p \rangle$)}
Ant(i,p) $\equiv$ Lst(**if** $p = l\rightarrow$primero **then** NULL **else** $l\rightarrow$primero **fi**, Pos($*l$, p))

Nota: cuando p = NULL, Pos devuelve la longitud de la lista, lo cual estï¿½ bien, porque significa que el iterador no tiene siguiente.

Pos : lst l $\times$ puntero(nodo) p $\longrightarrow$ puntero(nodo) {Rep($\langle l, p \rangle$)}
Pos(l,p) $\equiv$ **if** l.primero = $p \vee l$.longitud = 0 **then** 0 **else** 1 + Pos(FinLst(l), p) **fi**