1. Módulo Campus

Interfaz

se explica con: CAMPUS

géneros: `campus`

Operaciones básicas de campus

NUEVOCAMPUS(**in** al: `nat`, **in** an: `nat`) $\rightarrow$ res : `campus`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ crearCampus(al, an)}
Complejidad: $\Theta(1)$
Descripción: Crea un nuevo campus vacio de alto al x ancho an.

AGREGAROBSTACULO(**in** p: `pos`), **in/out** c: `campus`)
Pre $\equiv$ {posValida(p, c) $\wedge$ $\neg$ocupada?(p, c) $\wedge$ $c =_{\text{obs}} c_0$ }
Post $\equiv$ {$c =_{\text{obs}}$ agregarObstaculo(p, c_0)}
Descripción: Agrega un obstaculo al campus c en la posicion p.

FILAS(**in** c: `campus`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ filas(c}
Descripción: Devuelve el alto (filas) del campus c.

COLUMNAS(**in** c: `campus`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ columnas(c}
Descripción: Devuelve el ancho (columnas) del campus c.

OCUPADA?(**in** p: `pos`, **in** c: `campus`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ posValida?(p, c)}
Descripción: Devuelve $true$ si la posicion p es valida en el campus c, sino retorna $false$.

ESINGRESO?(**in** p: `pos`, **in** c: `campus`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ esIngreso?(p, c)}
Descripción: Verifica si la posicion p es una entrada del campus c.

INGRESOSUPERIOR?(**in** p: `pos`, **in** c: `campus`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ ingresoSuperior?(p, c)}
Descripción: Verifica si la posicion p es una entrada superior del campus c.

INGRESOINFERIOR?(**in** p: `pos`, **in** c: `campus`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ ingresoInferior?(p, c)}
Descripción: Verifica si la posicion p es una entrada inferior del campus c.

VECINOS(**in** p: `pos`, **in** c: `campus`) $\rightarrow$ res : `conj(`pos`)`
Pre $\equiv$ {posValida(p, c)}
Post $\equiv$ {$res =_{\text{obs}}$ vecinos(p, c)}
Descripción: Devuelve un conjunto de las posiciones que rodean a p en el campus c

()
Pre $\equiv$ {}
Post $\equiv$ {}

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: $\Theta(1)$
Descripción: 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 sólo si se elimina el elemento siguiente del iterador sin utilizar la función 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: $\Theta(1)$
Descripción: crea un iterador bidireccional de la lista, de forma tal que al pedir Anterior se obtenga el último elemento de l.
Aliasing: el iterador se invalida si y sólo si se elimina el elemento siguiente del iterador sin utilizar la función EliminarSiguiente.

Representación

Representación 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 $\iff$ (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**

Representación 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 $\iff$ 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 × puntero(nodo) p $\longrightarrow$ lst $\{\text{Rep}(\langle l, p\rangle)\}$

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 está bien, porque significa que el iterador no tiene siguiente.

Pos : lst l × puntero(nodo) p $\longrightarrow$ puntero(nodo) $\{\text{Rep}(\langle l, p\rangle)\}$

Pos(l,p) $\equiv$ **if** l.primero $= p \vee l$.longitud $= 0$ **then** 0 **else** 1 + Pos(FinLst(l), p) **fi**