



# Algoritmos
# Estructuras de Datos I

Facultad de Ciencias Exactas y Tecnología
Universidad Nacional de Tucumán
2024

UNIVERSIDAD NACIONAL DE TUCUMÁN
facet
FACULTAD DE CIENCIAS EXACTAS Y TECNOLOGÍA

# LISTA CIRCULAR - LISTA

# LISTA CIRCULAR

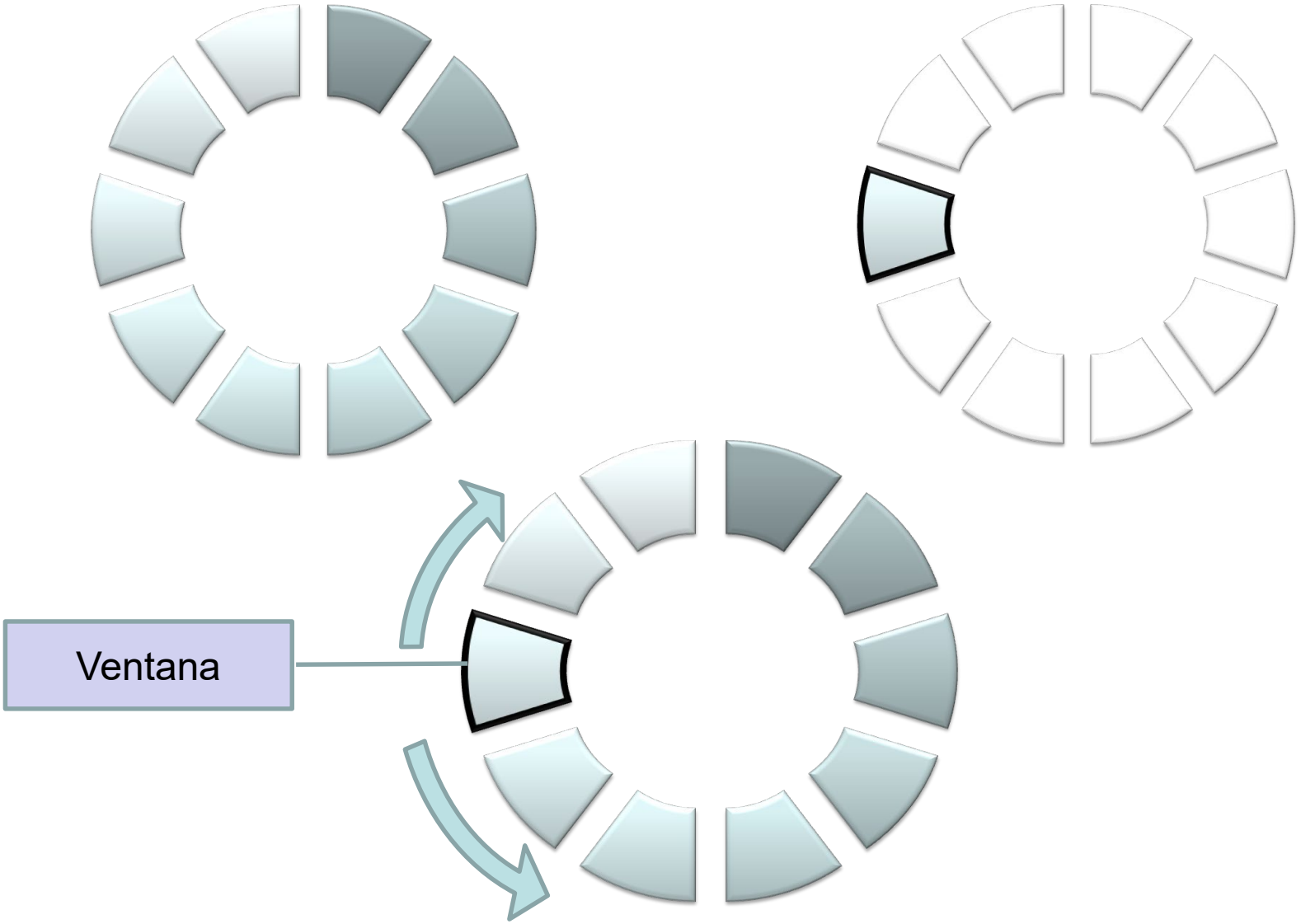
Una lista circular es una estructura de datos de tipo lineal donde el ultimo nodo esta ligado al primero y permite una operación de rotación entre sus elementos.

El acceso a sus componentes se hace con como en una LIFO, pero la operación de rotación permite pasar un elemento del tope de la pila al fondo de la pila.

Ventaja:

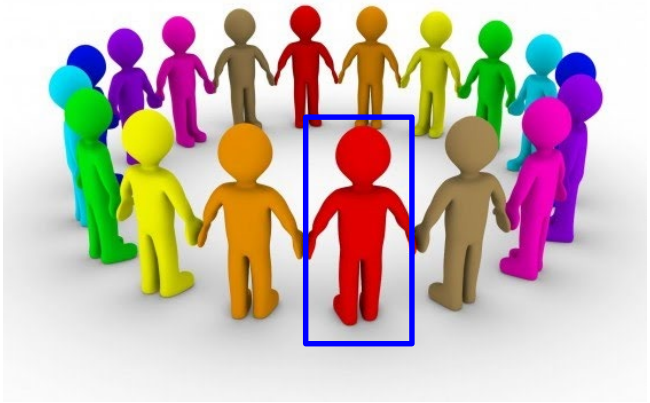
Con la operación de rotación se puede acceder a todos sus elementos sin necesidad de borrarlos de la lista.

# Esquema LC



# LISTA CIRCULAR (ITEM)

## Especificación Algebraica



**OPERACIONES**
**A) Sintaxis:**

LCVACIA : → LISTACIRCULAR

LCINSERTAR : LISTACIRCULAR x ITEM → LISTACIRCULAR

LCBORRAR : LISTACIRCULAR → LISTACIRCULAR

LCVALOR : LISTACIRCULAR → ITEM U {indefinido}

ESLCVACIA : LISTACIRCULAR → BOOLEAN

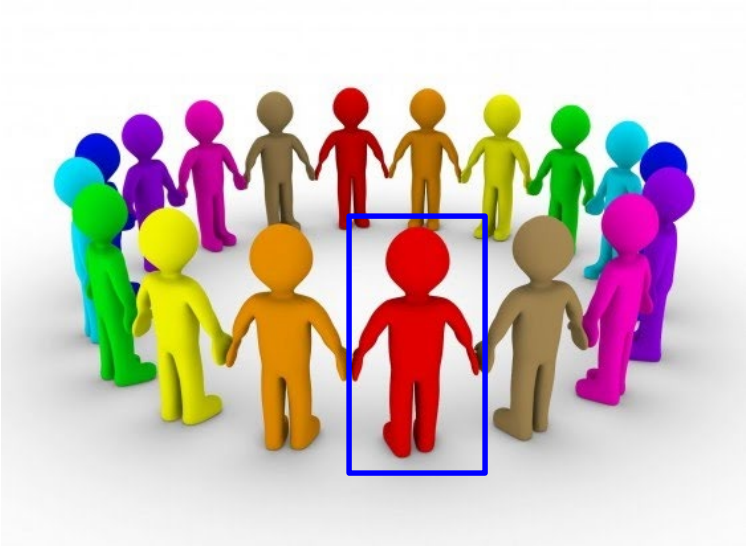
LCROTAR : LISTACIRCULAR → LISTACIRCULAR

# LISTA CIRCULAR (ITEM)
## Especificación Algebraica

**B) Semántica:** Para todo $L \in$ LISTACIRCULAR, $\forall$ $i,j \in$ ITEM

ESLCVACIA(LCVACIA) $\equiv$ TRUE

ESLCVACIA(LCINSERTAR(L,i)) $\equiv$ FALSE



LCBORRAR(LCVACIA) $\equiv$ LCVACIA

LCBORRAR(LCINSERTAR(L,i)) $\equiv$ L

LCVALOR(LCVACIA) $\equiv$ indefinido

LCVALOR(LCINSERTAR(L,i)) $\equiv$ i

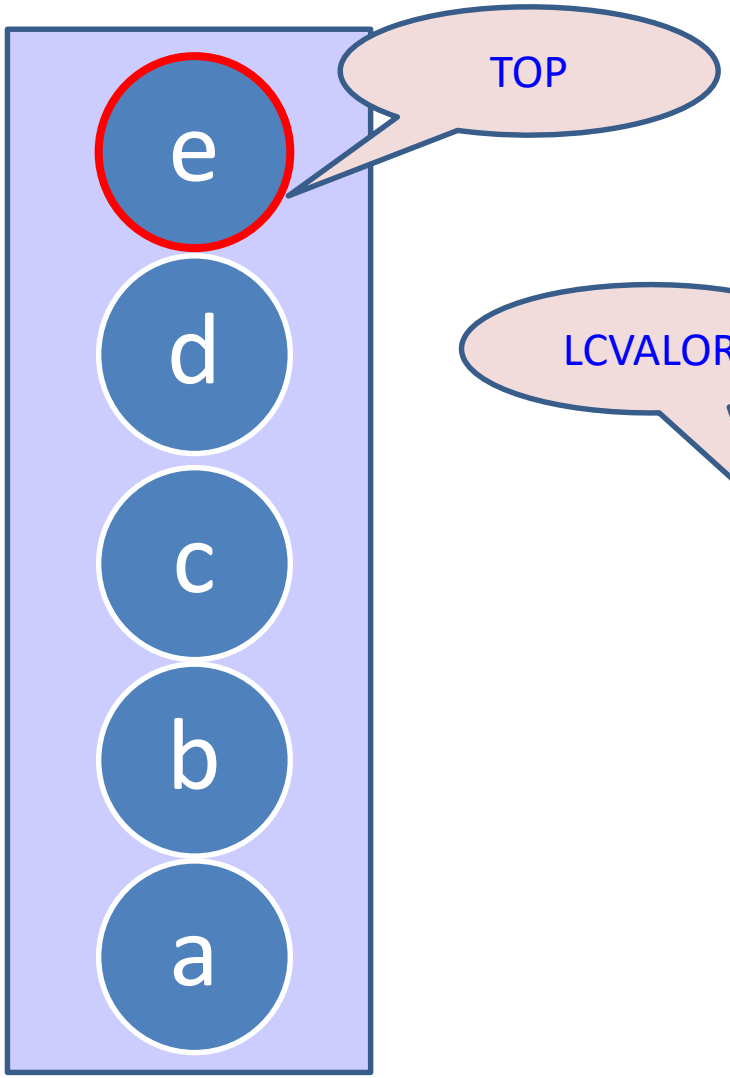
LCROTAR(LCVACIA) $\equiv$ LCVACIA

LCROTAR(LCINSERTAR(LCVACIA,i)) $\equiv$ LCINSERTAR(LCVACIA,i)

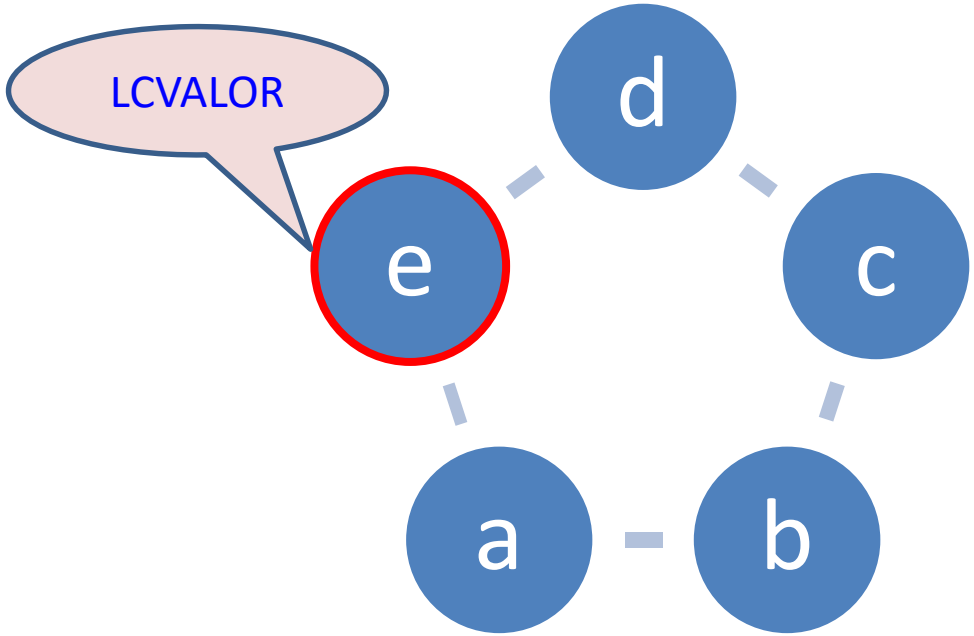
LCROTAR(LCINSERTAR(LCINSERTAR(L,i),j)) $\equiv$
LCINSERTAR(LCROTAR(LCINSERTAR(L,j)),i)

# Comparación PILA con LISTACIRCULAR

**PILA:**



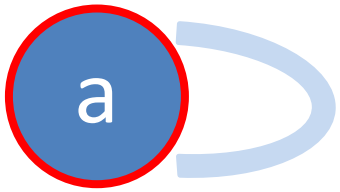

**LISTACIRCULAR:**



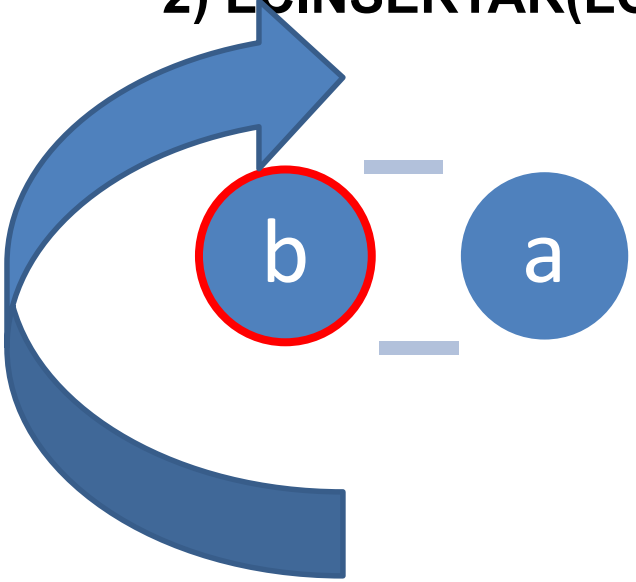
# Esquema LCINSERTAR

**0) LCVACIA(LC1):**

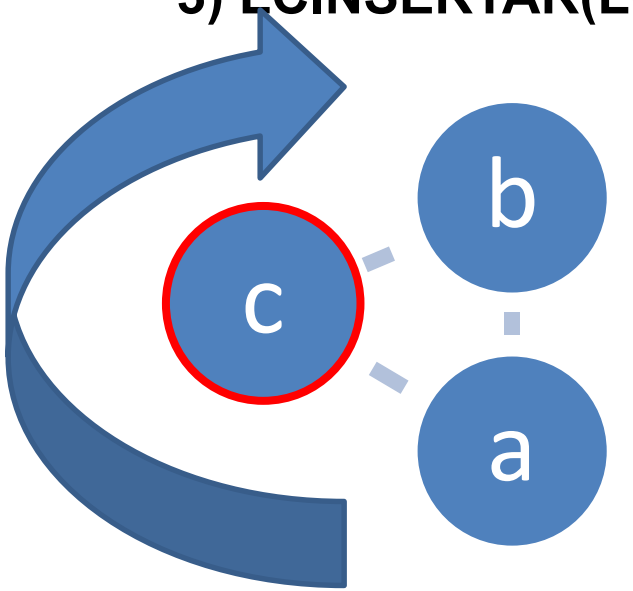
**1) LCINSERTAR(LC1,a):**




**2) LCINSERTAR(LC1,b):**

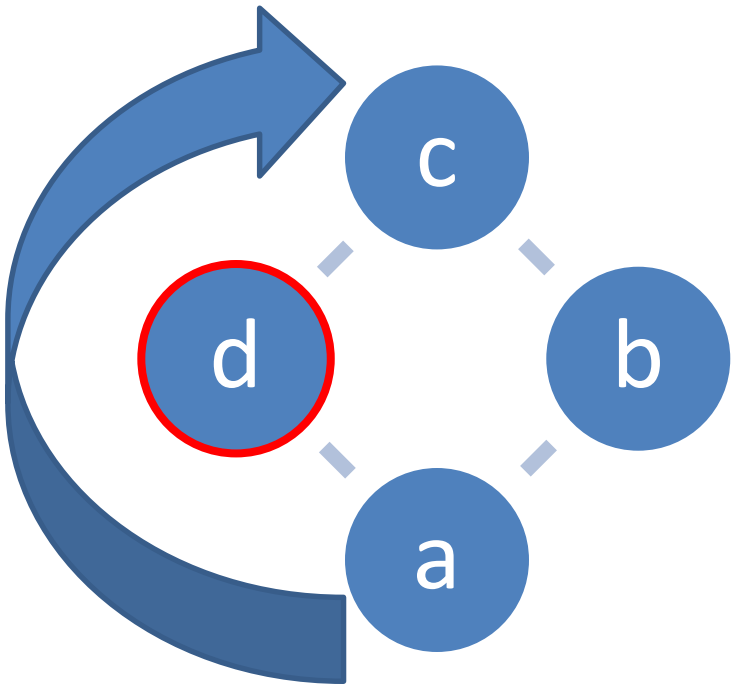



**3) LCINSERTAR(LC1,c):**

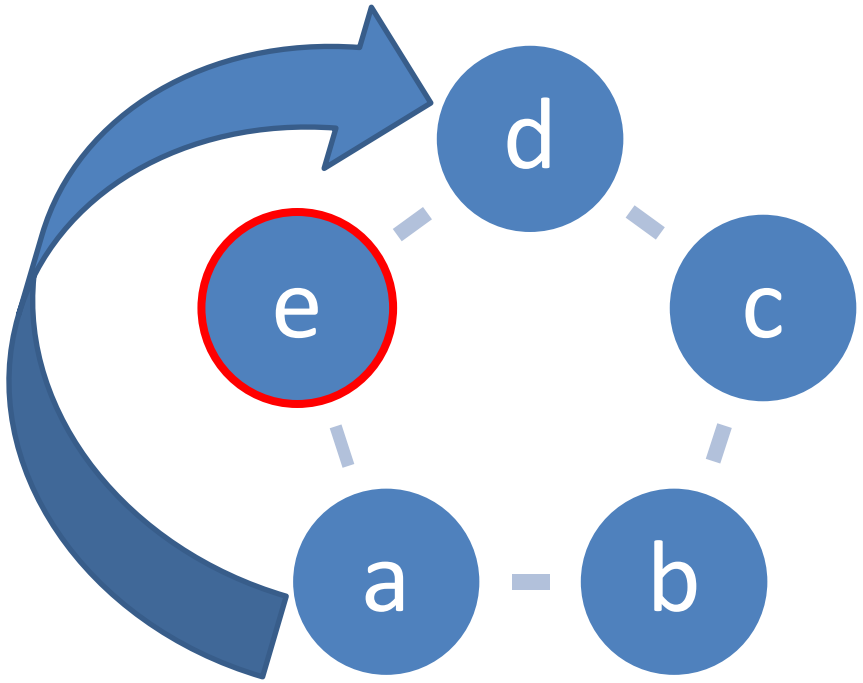


# Esquema LCINSERTAR

**4) LCINSERTAR(LC1,d):**

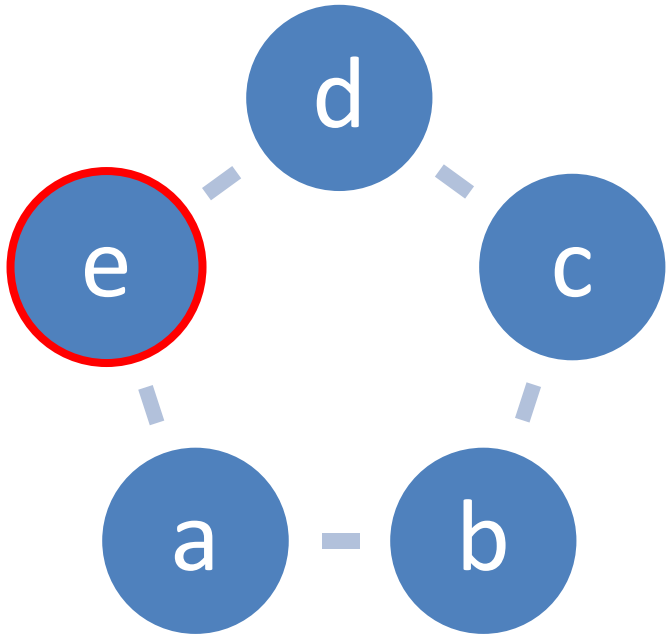



**5) LCINSERTAR(LC1,e):**

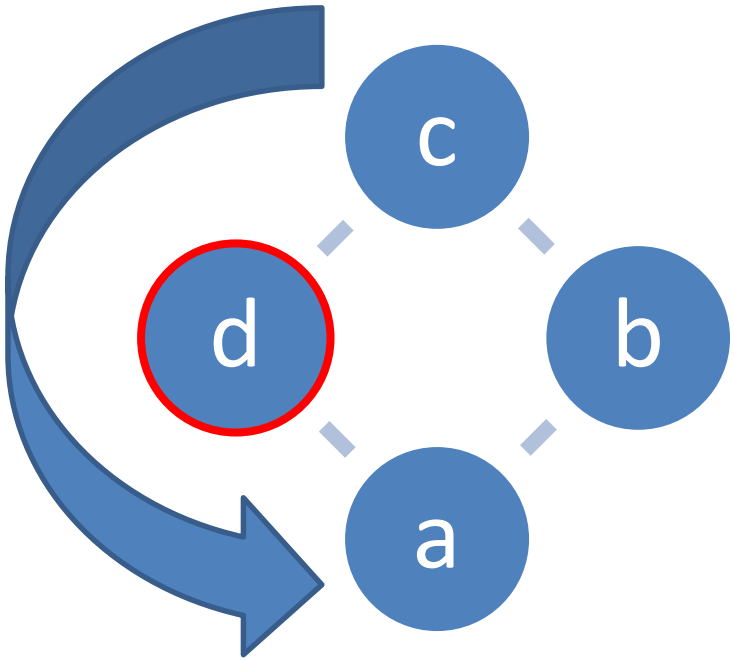


# Esquema LCBORRAR

**0) LC1:**



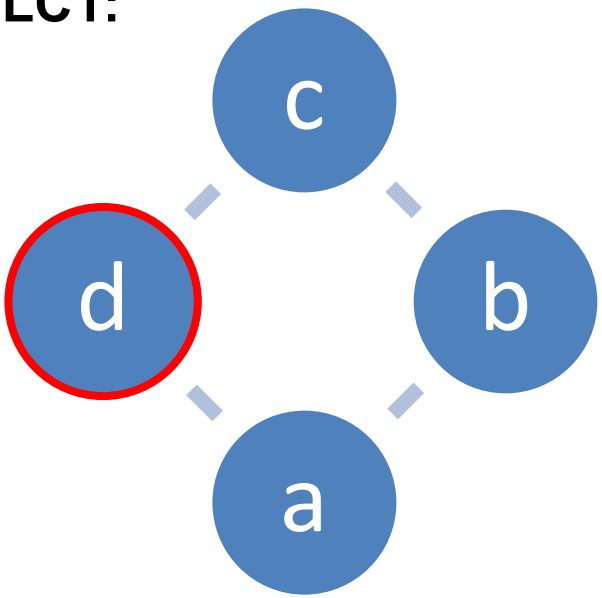

**1) LCBORRAR(LC1):**



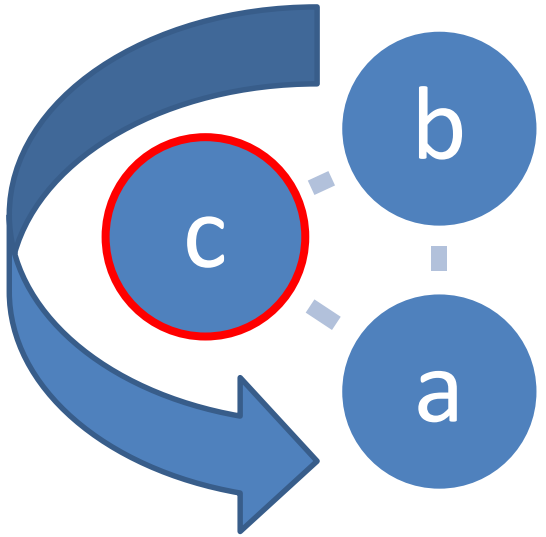

LCBORRAR(LCINSERTAR(L,i)) ≡ L

# Esquema LCBORRAR

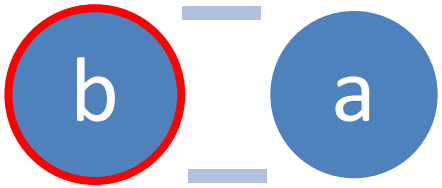
0) LC1:



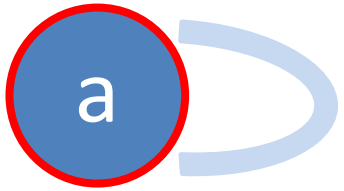

1) LCBORRAR(LC1):




2) LCBORRAR(LC1):

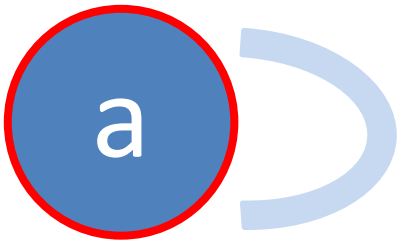



3) LCBORRAR(LC1):

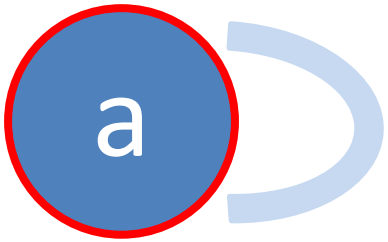


# Esquema LCROTAR

**LC2:**

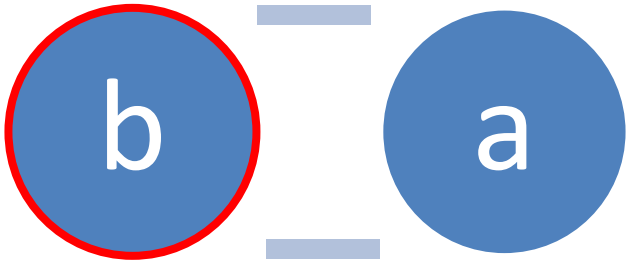



**LCROTAR(LC2):**

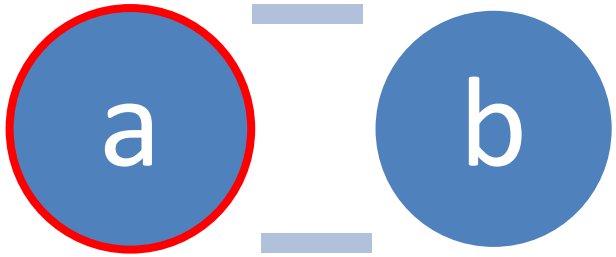



LCROTAR(LCINSERTAR(LCVACIA,i)) ≡ LCINSERTAR(LCVACIA,i)

**LC3:**




**LCROTAR(LC3):**

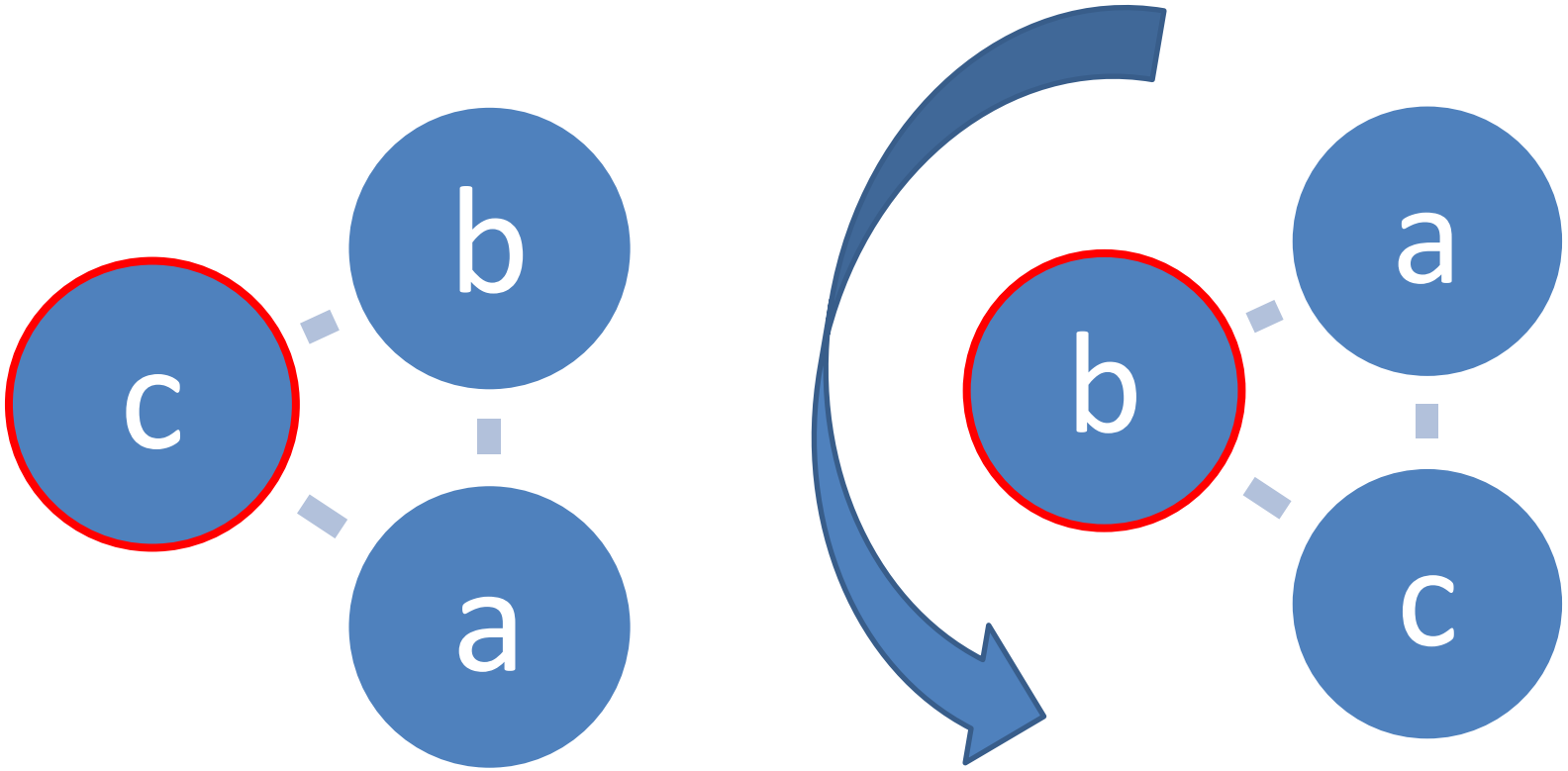



LCROTAR(LCINSERTAR(LCINSERTAR(L,i),j)) ≡
LCINSERTAR(LCROTAR(LCINSERTAR(L,j)),i)

# Esquema LCROTAR

**LC4:**

**LCROTAR(LC4):**

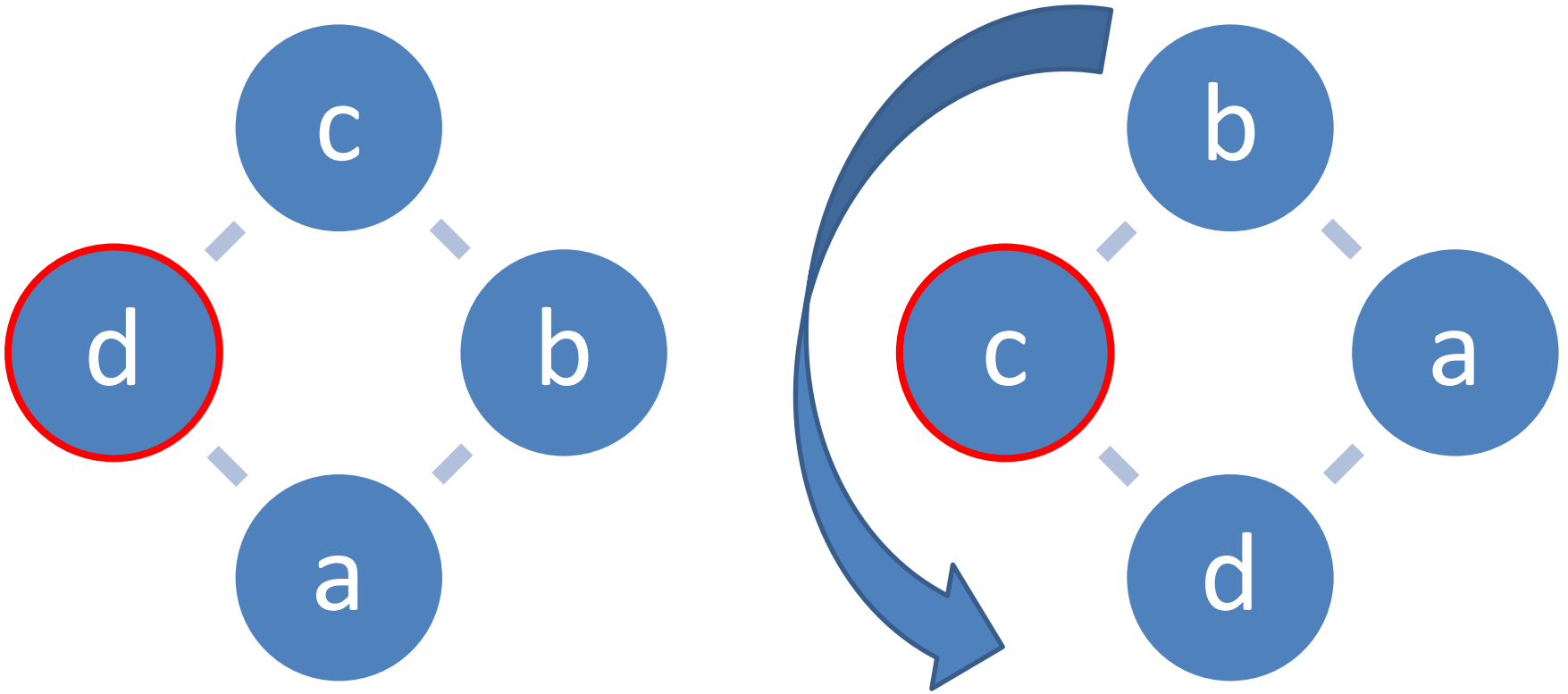



LCROTAR(LCINSERTAR(LCINSERTAR(L,i),j)) ≡
LCINSERTAR(LCROTAR(LCINSERTAR(L,j)),i)

# Esquema LCROTAR

**LC5:**

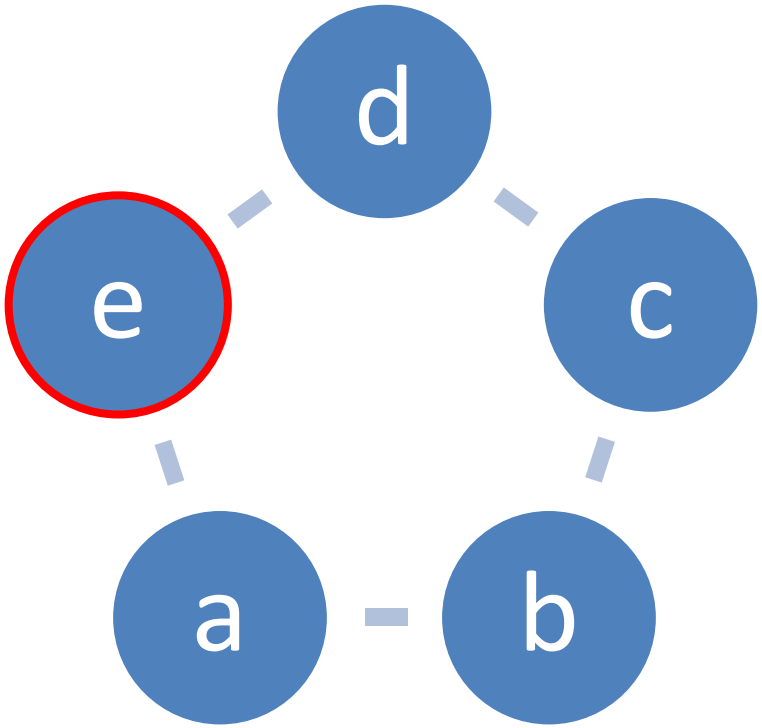
**LCROTAR(LC5):**



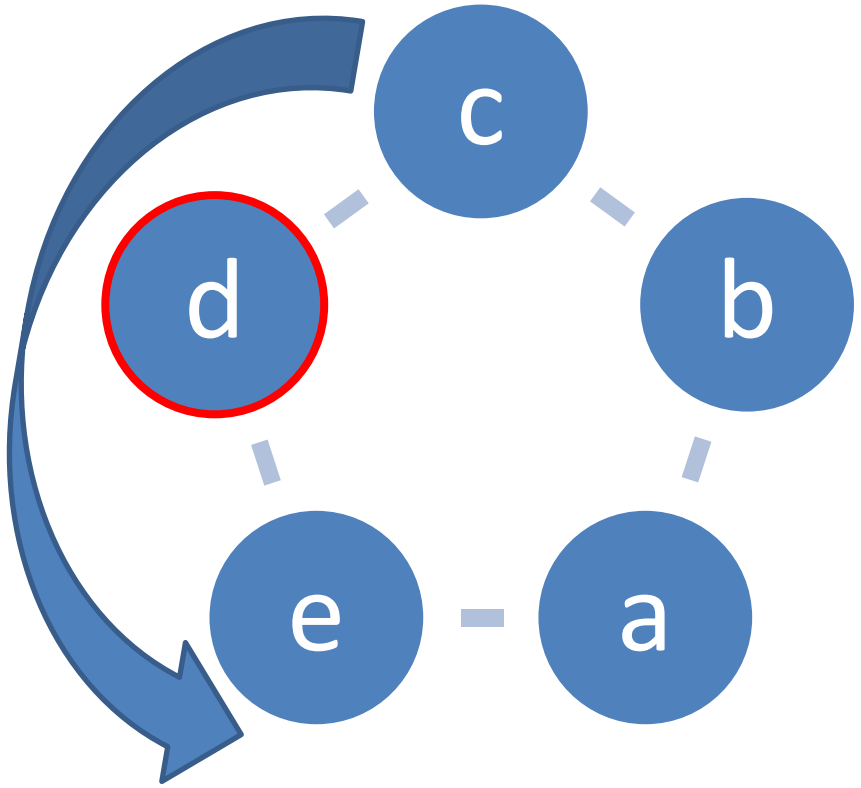

LCROTAR(LCINSERTAR(LCINSERTAR(L,i),j)) ≡
LCINSERTAR(LCROTAR(LCINSERTAR(L,j)),i)

# Esquema LCROTAR

**LC6:**

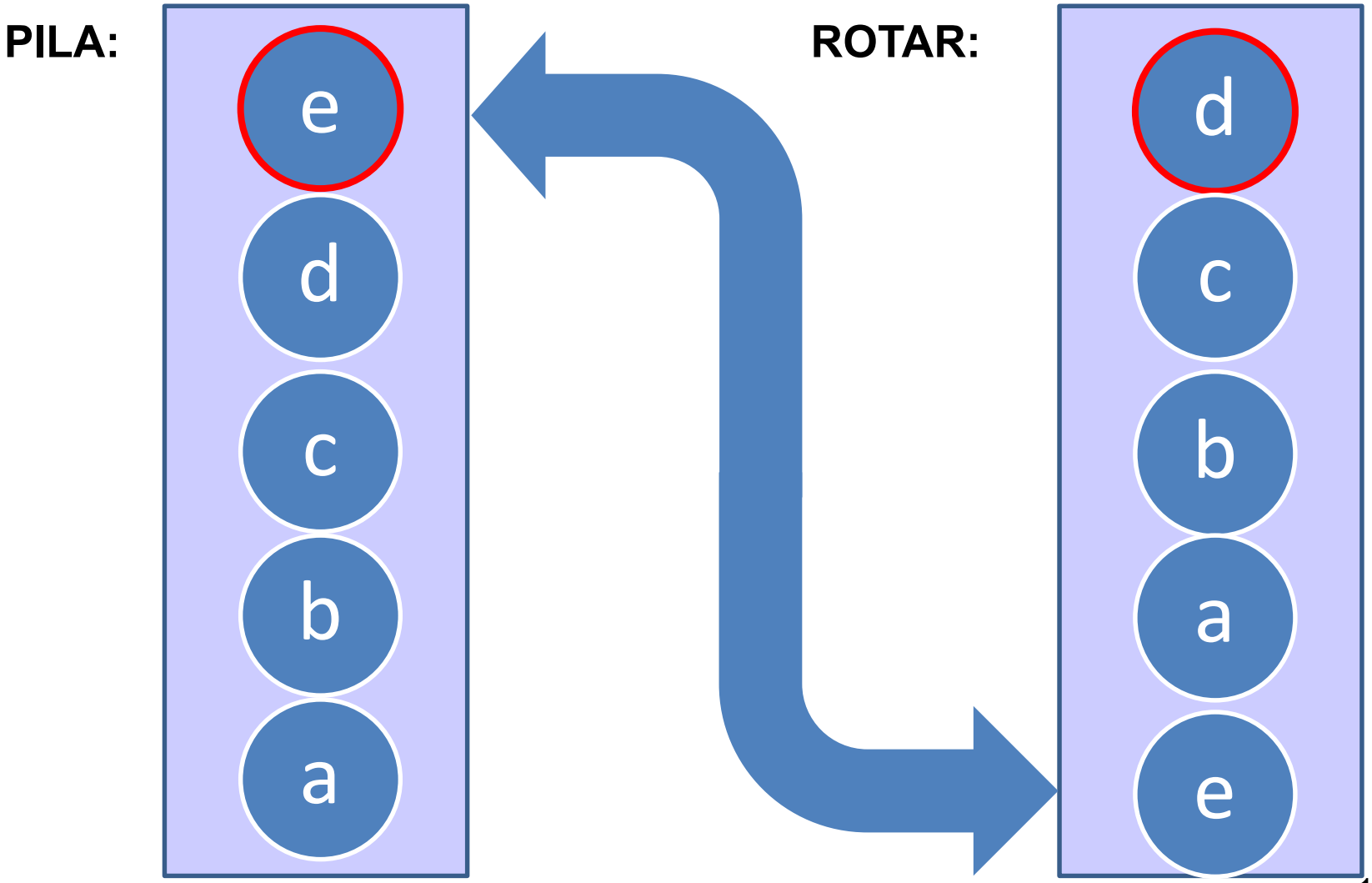



**LCROTAR(LC6):**



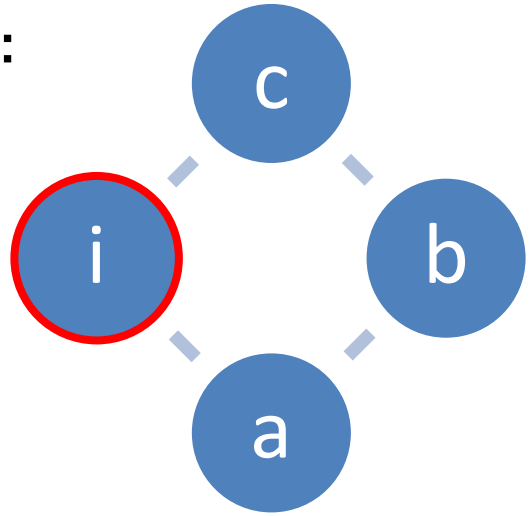

LCROTAR(LCINSERTAR(LCINSERTAR(L,i),j)) ≡
LCINSERTAR(LCROTAR(LCINSERTAR(L,j)),i)

# Esquema LCROTAR sobre una PILA

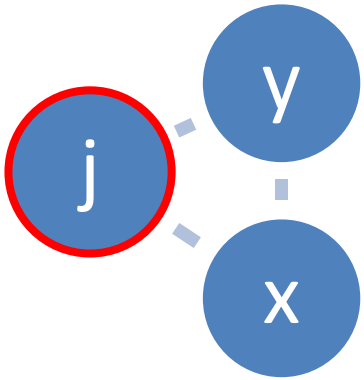


# Esquema operación LCUNIR

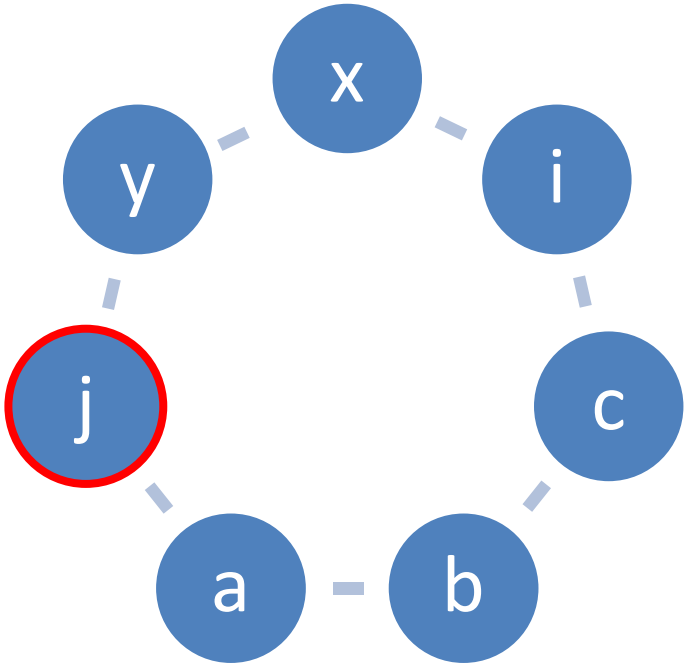
**LC1:**




**LC2:**




**LCUNIR(LC1,LC2):**



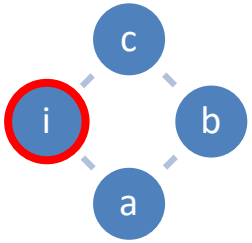
# LISTA CIRCULAR

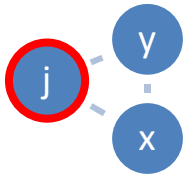
## Aplicación: unir 2 listas circulares

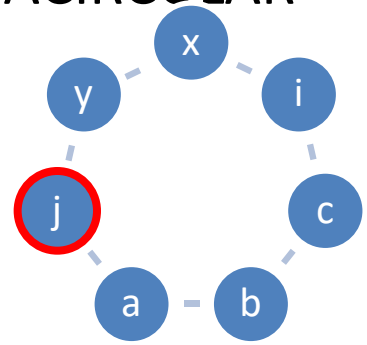
**1) Dentro de la Especificación Algebraica, 4 axiomas**

Sintaxis:

LCUNIR : LISTACIRCULAR x LISTACIRCULAR → LISTACIRCULAR




Semántica: ∀ L1,L2 ∈ LISTACIRCULAR, ∀ i,j ∈ ITEM

LCUNIR (LCVACIA , LCVACIA) ≡ LCVACIA

LCUNIR (LCINSERTAR(L1,i) , LCVACIA) ≡ LCINSERTAR(L1,i)

LCUNIR (LCVACIA , LCINSERTAR(L2,j) ) ≡ LCINSERTAR(L2,j)

LCUNIR (LCINSERTAR(L1,i) , LCINSERTAR(L2,j)) ≡

LCINSERTAR (LCUNIR (LCINSERTAR(L1,i),L2) , j)

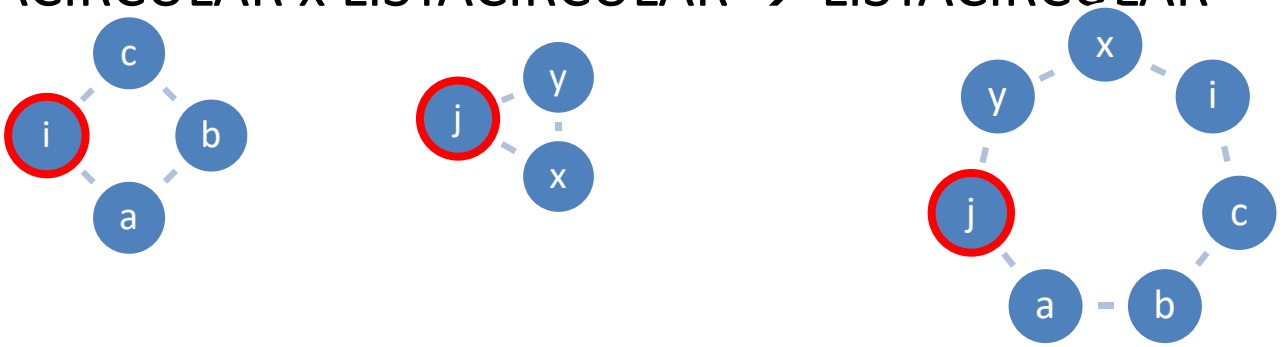
# LISTA CIRCULAR

## Aplicación: unir 2 listas circulares

**2) Dentro de la Especificación Algebraica, 2 axiomas**

Sintaxis:

LCUNIR : LISTACIRCULAR x LISTACIRCULAR → LISTACIRCULAR




Semántica: ∀ L1,L2 ∈ LISTACIRCULAR, ∀ j ∈ ITEM

LCUNIR (L1, LCVACIA) ≡ L1

LCUNIR (L1, LCINSERTAR(L2,j)) ≡

LCINSERTAR (LCUNIR (L1,L2) , j)

# LISTA CIRCULAR

## Aplicación: unir 2 listas circulares

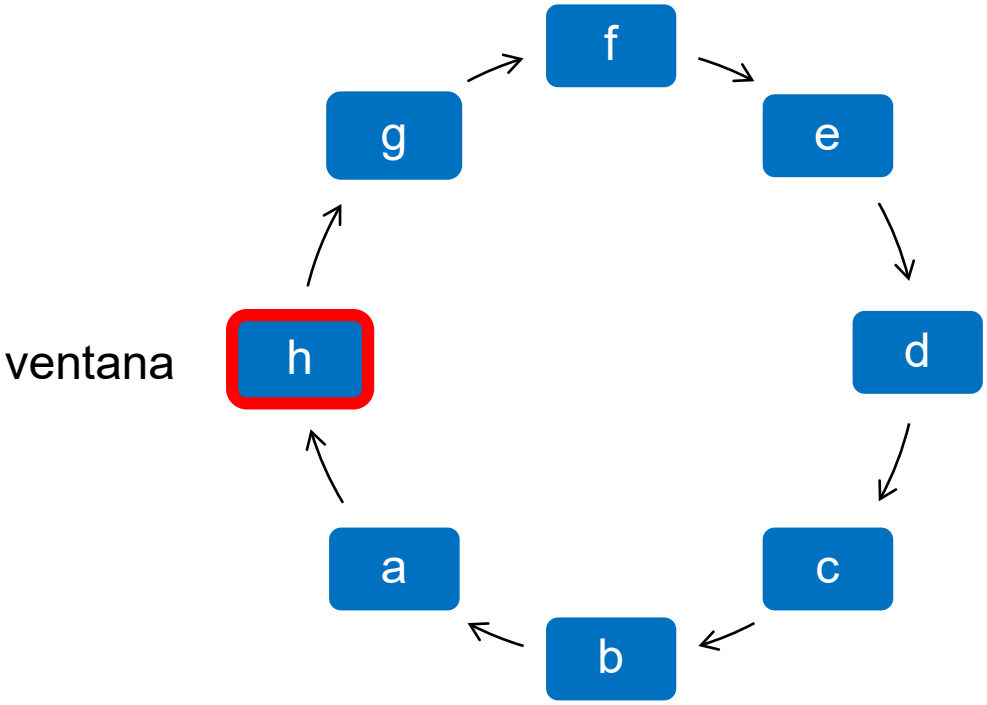
**Ejercitacion:**

**3) Como usuario del ADT LC,** función recursiva

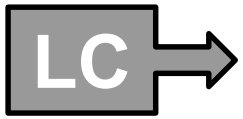
**4) Como usuario del ADT LC,** función iterativa

# Implementación LC

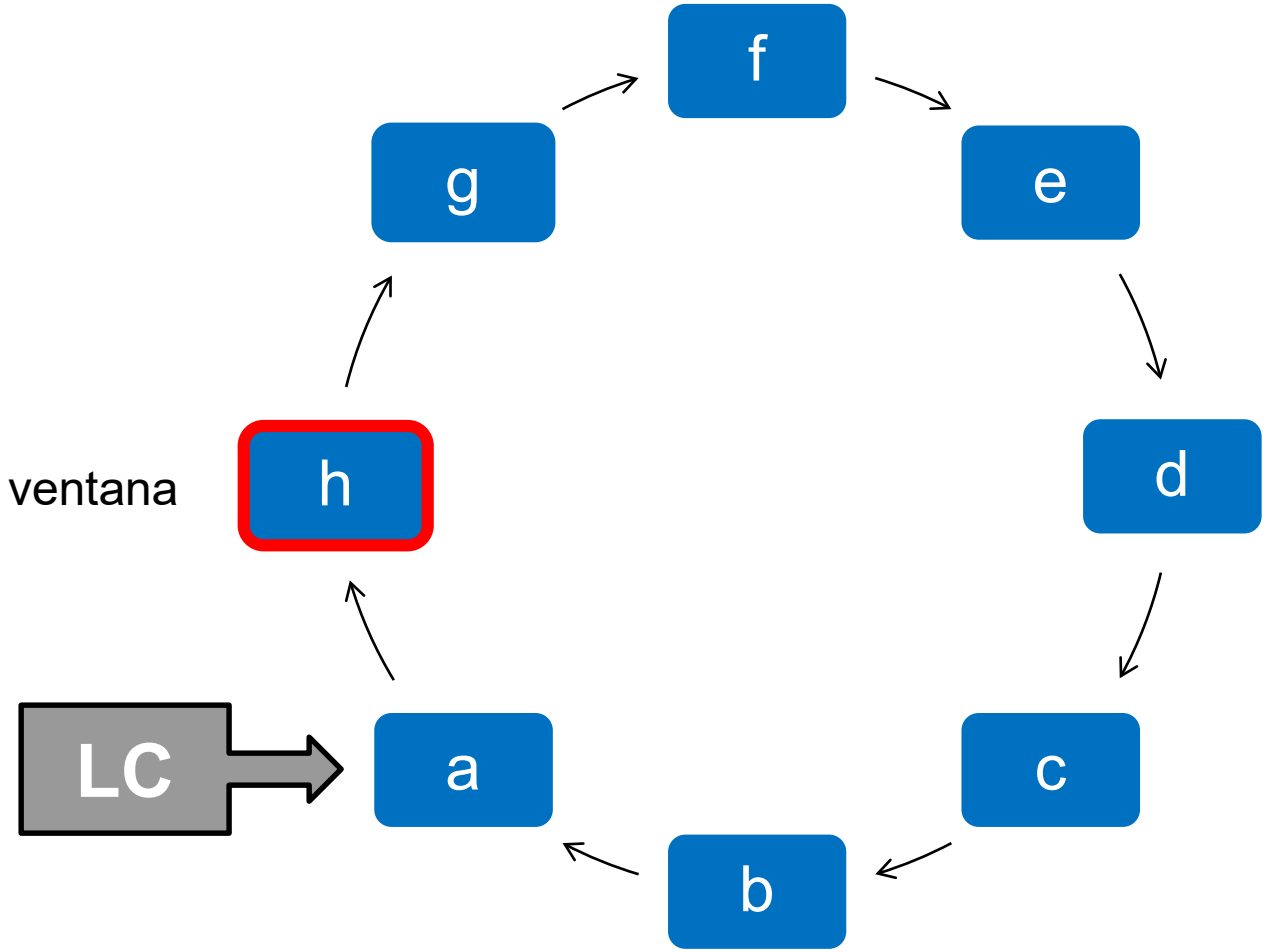
Con lista enlazada circular:




Donde se pone el puntero a la lista enlazada circular?:

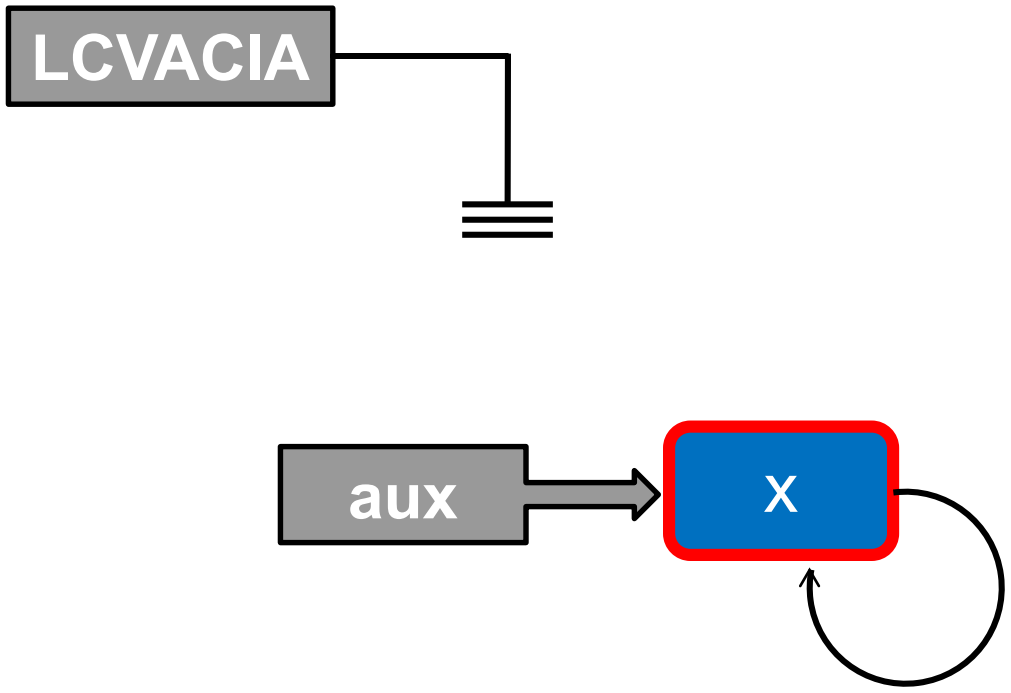


# Implementación LC

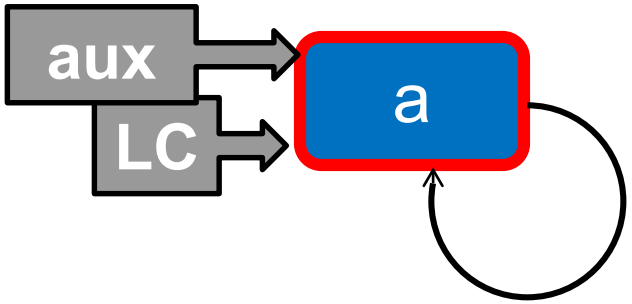


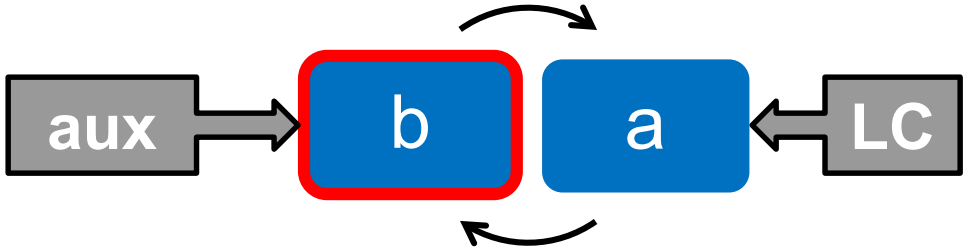

*// NOTA: todas las operaciones son O(1) en esta implementación*

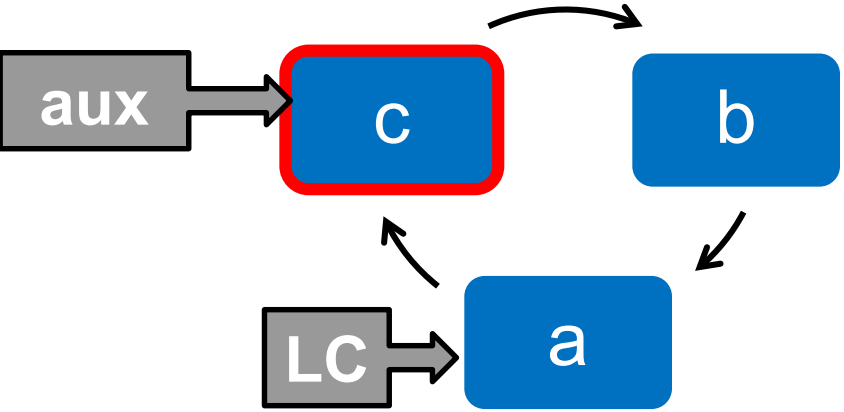
# Implementación LCINSERTAR

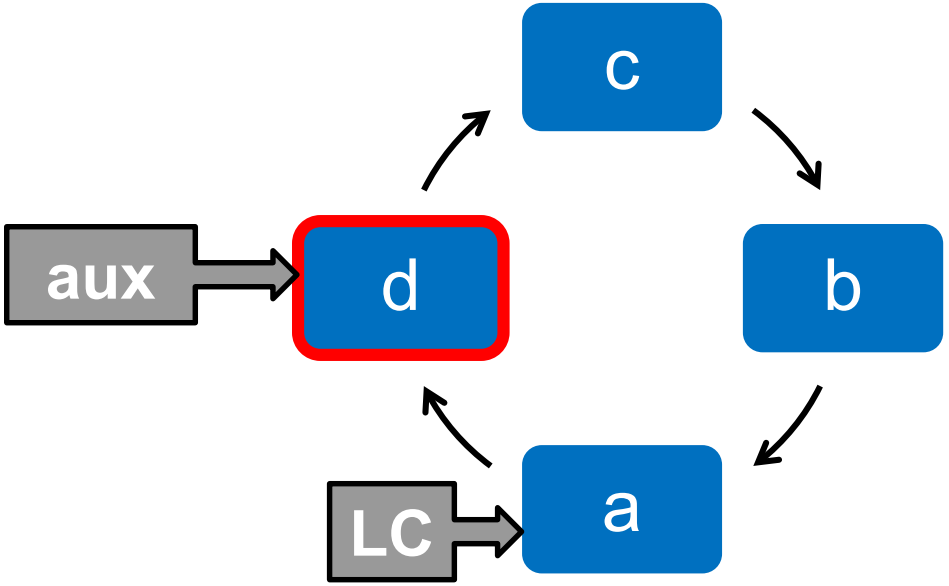


# Implementación LCINSERTAR

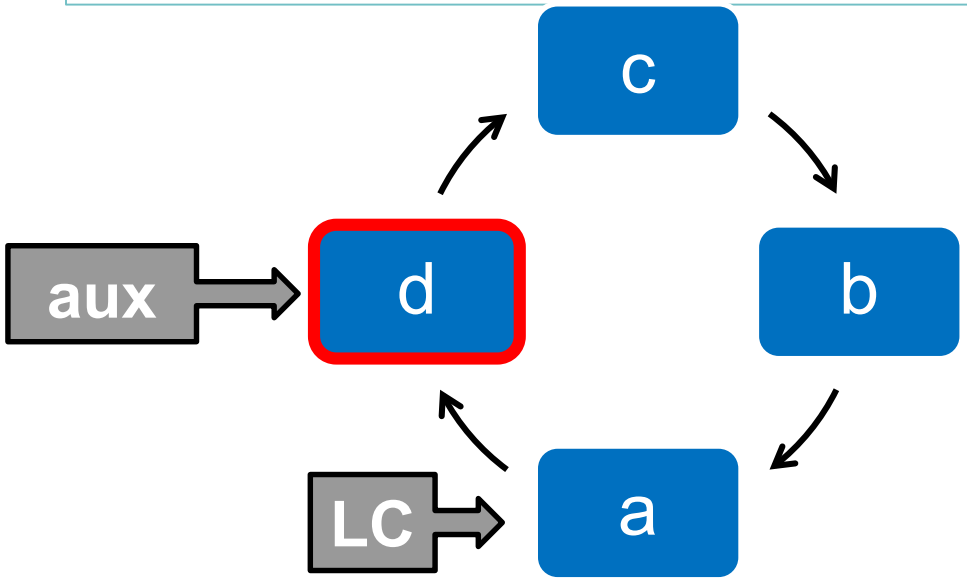


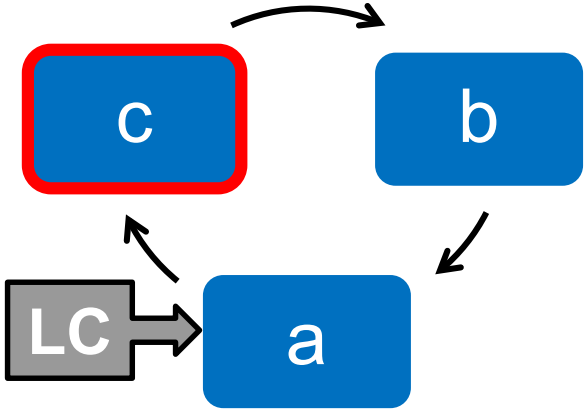



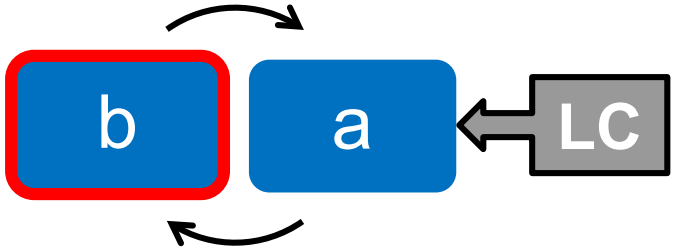



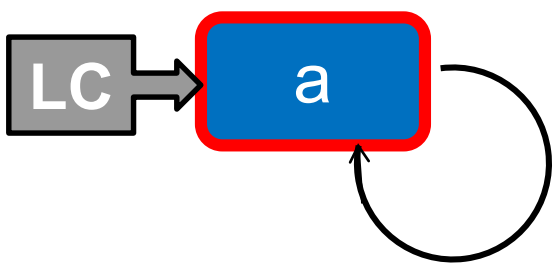



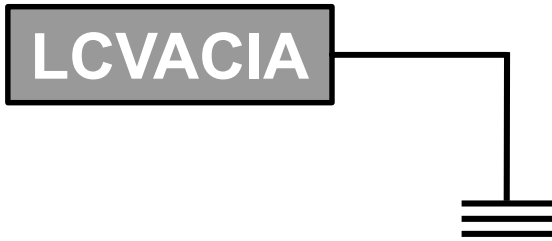
# Implementación LCBORRAR



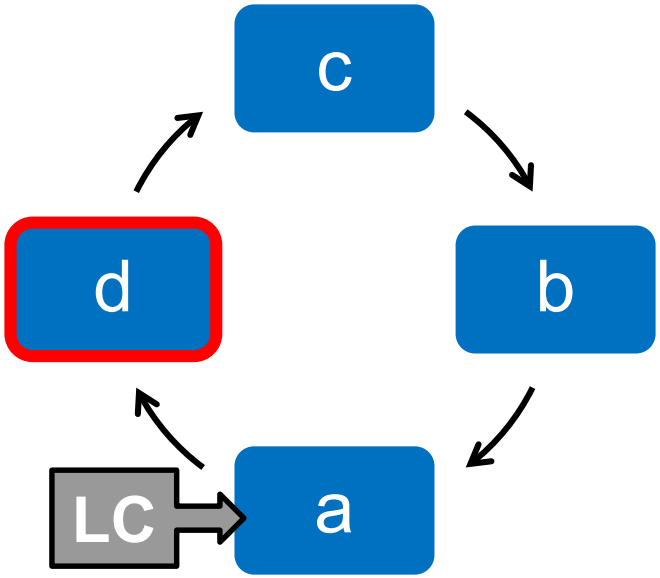



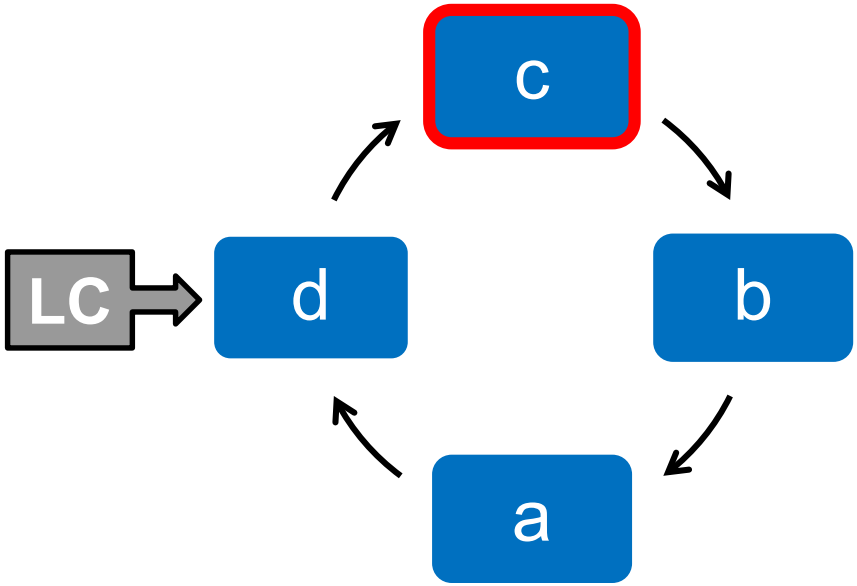



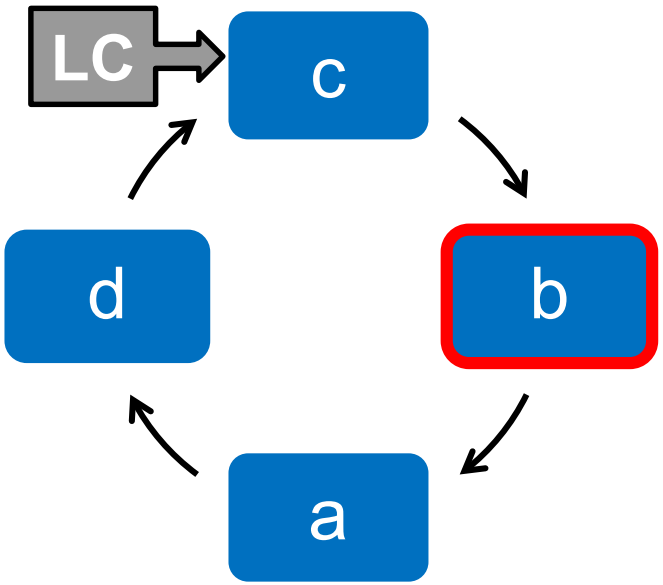



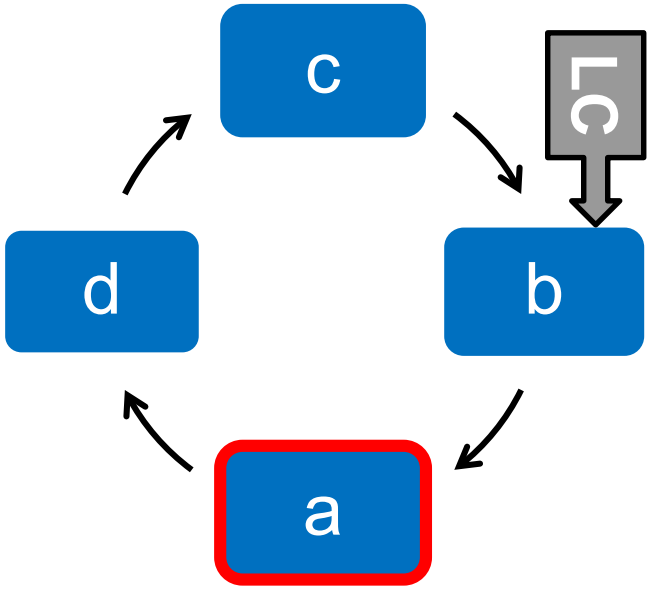
# Implementación LCROTAR



# Implementación LCROTAR



# ADT LISTA

La lista es una secuencia de cero o mas elementos de un tipo dado. Una lista es una estructura para almacenar información con la característica de que puede contener cualquier número de elementos, y que estos están representados en un cierto orden.

Una lista es una estructura particularmente flexible que es de mucha utilidad en la vida diaria como por ejemplo: lista de precios, lista de estudiantes, lista de candidatos, etc.

# ADT LISTA

Los elementos de una lista se distinguen por su posición en la misma, se habla del primer elemento, del segundo, del k-ésimo, del ultimo.

El ADT LISTA esta basado en el concepto de ***posición*** que es un valor entero. Se define en función de un tipo base ***item*** que representa el tipo de elementos que contendrá la lista. De esta manera se define el ADT independiente del tipo de elementos que contenga.

# Ejemplo: ADT LISTA

| Posicion | Contenido |
|---|---|
| 1 | UNIDAD I)  Algoritmos. |
| 2 | UNIDAD II) Tipos abstractos de datos básicos. |
| 3 | UNIDAD III) Tipos de datos no lineales. |
| 4 | UNIDAD IV) Ordenamiento. |
| 5 | UNIDAD V)  Búsqueda. |

# LISTA (ITEM)
## Especificación Algebraica

**OPERACIONES** **A) Sintaxis:**

LISTAVACIA : → LISTA

* GUARDAR : LISTA x ENTERO x ITEM → LISTA

INSERTAR : LISTA x ENTERO x ITEM → LISTA

OBTENER : LISTA x ENTERO → ITEM U {indefinido}

BORRAR : LISTA x ENTERO → LISTA

LONG : LISTA → ENTERO ≥ 0

OBSERVACION: la constructora GUARDAR es una ***constructora escondida*** del tipo, se usa solo para la especificación algebraica. El usuario tiene disponible solamente las constructoras LISTAVACIA e INSERTAR.

# LISTA (ITEM)
## Especificación Algebraica

**B) Semántica:** Para todo L $\in$ LISTA, $\forall$ i,j $\in$ ENTERO, $\forall$ a,b $\in$ ITEM

LONG(LISTAVACIA) $\equiv$ 0

LONG(GUARDAR(L,i,a)) $\equiv$ 1 + LONG(L)

OBTENER(LISTAVACIA, j) $\equiv$ indefinido

OBTENER(GUARDAR(L,i,a), j) $\equiv$ si i=j entonces a
sino OBTENER(L,j)

INSERTAR(LISTAVACIA,j,b) $\equiv$ GUARDAR(LISTAVACIA,j,b)

INSERTAR(GUARDAR(L,i,a),j,b) $\equiv$ si i $\geq$ j entonces
GUARDAR(INSERTAR(L,j,b),i+1,a)
sino GUARDAR(INSERTAR(L,j,b),i,a)

BORRAR(LISTAVACIA,j) $\equiv$ LISTAVACIA

BORRAR(GUARDAR(L,i,a),j) $\equiv$ si i=j entonces BORRAR(L,j)
sino si i>j entonces GUARDAR(BORRAR(L,j),i-1,a)
sino GUARDAR(BORRAR(L,j),i,a)

# LISTA
## Aplicación: igualdad de listas

**1) Dentro de la Especificación Algebraica**

Sintaxis:

IGUALI : LISTA x LISTA → BOOL

Semántica: $\forall$ L1,L2 $\in$ LISTA, $\forall$ i,j $\in$ ENTERO, $\forall$a,b $\in$ ITEM

IGUALI(LISTAVACIA, LISTAVACIA) $\equiv$ TRUE

IGUALI(LISTAVACIA, GUARDAR(L2,i,a)) $\equiv$ FALSE

IGUALI(GUARDAR(L1,i,a),LISTAVACIA) $\equiv$ FALSE

IGUALI(GUARDAR(L1,i,a), GUARDAR(L2,j,b)) $\equiv$

i=j AND a=b AND IGUALI(L1,L2)

*donde = es la operación que permite comparar 2 objetos del tipo item*

# LISTA
## Aplicación: igualdad de listas

**2) Como usuario del ADT LISTA,** función recursiva
Se usa una función auxiliar IGUAL:

Funcion IGUALISTA (L1,L2) **:** LISTA x LISTA → BOOL
    Si LONG(L1) ≠ LONG(L2) entonces
        Retorna FALSE
    Sino
        Retorna **IGUAL** (L1,L2,1)
Fin

# LISTA
## Aplicación: igualdad de listas

**2) Como usuario del ADT LISTA,** función recursiva

```
Funcion IGUAL(L1,L2,posicion): LISTAxLISTAxENTERO→ BOOL
    SI posicion > LONG(L1) ENTONCES
        RETORNA  TRUE
    SINO
        RETORNA
            OBTENER(L1, posicion) = OBTENER(L2, posicion)
            AND IGUAL (L1,L2, posicion + 1)
Fin
```

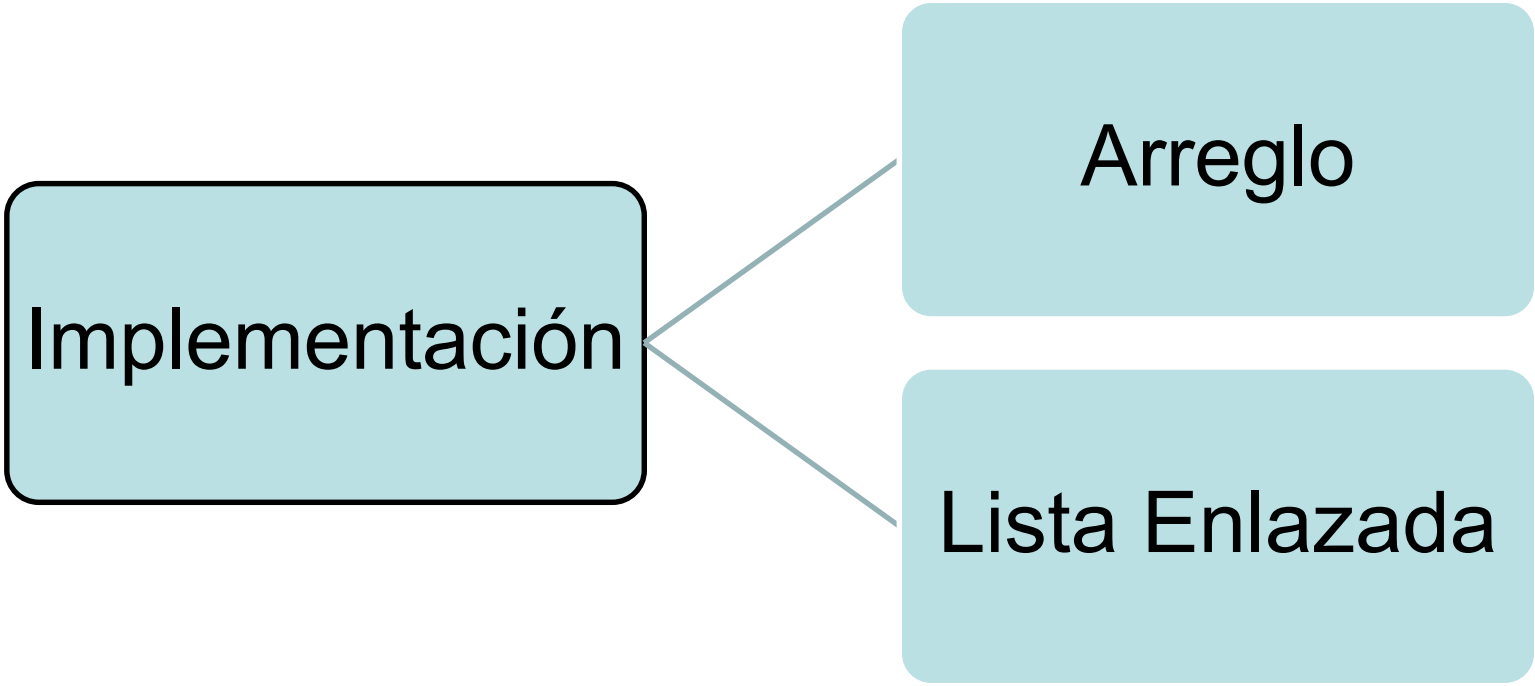
*donde = es la operación que permite comparar 2 objetos del tipo item*

# LISTA
## Aplicación: igualdad de listas

**3) Como usuario del ADT LISTA,** función iterativa

```
Funcion IGUALISTA (L1,L2) : LISTA x LISTA → BOOL
    Si LONG(L1) ≠ LONG(L2) entonces
        Retorna FALSE
    Sino
        L←LONG(L1)
        i←1
        Mientras i ≤ L AND  OBTENER(L1,i) = OBTENER(L2,i)
        Hacer
                    i←i+1
        Retorna i>L
Fin
```

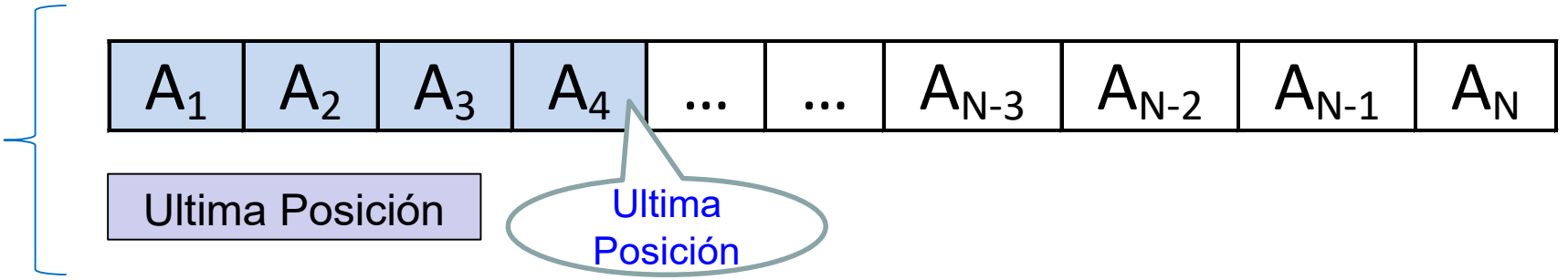
# LISTA




Ventajas y Desventajas

# Implementación Lista con arreglo

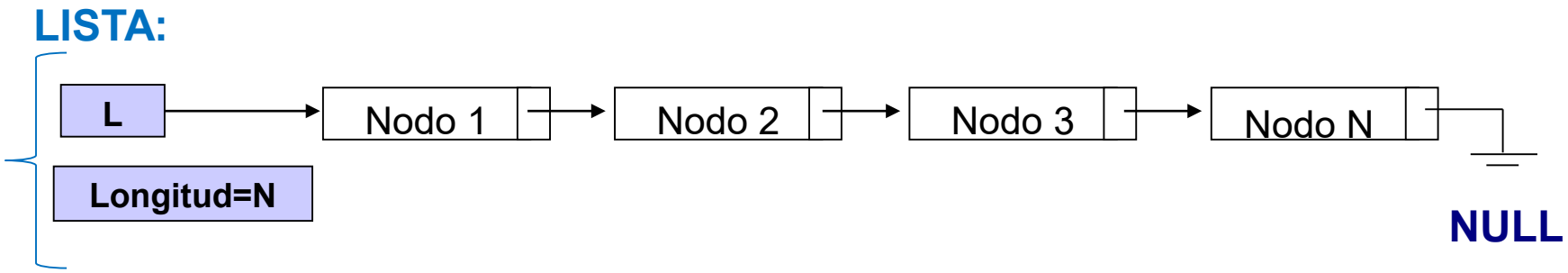
**LISTA:**




Costo de las Operaciones:

LISTAVACIA $\in$ O(1)
GUARDAR $\in$ O(1)
INSERTAR $\in$ O(N)
OBTENER $\in$ O(1)
BORRAR $\in$ O(N)
LONG $\in$ O(1)

# Implementación Lista con lista enlazada




Costo de las Operaciones:

| | |
|---|---|
| LISTAVACIA | $\in O(1)$ |
| GUARDAR | $\in O(N)$ |
| INSERTAR | $\in O(N)$ |
| OBTENER | $\in O(N)$ |
| BORRAR | $\in O(N)$ |
| LONG | $\in O(1)$ |