

ESTRUCTURAS DE DATOS

COLAS

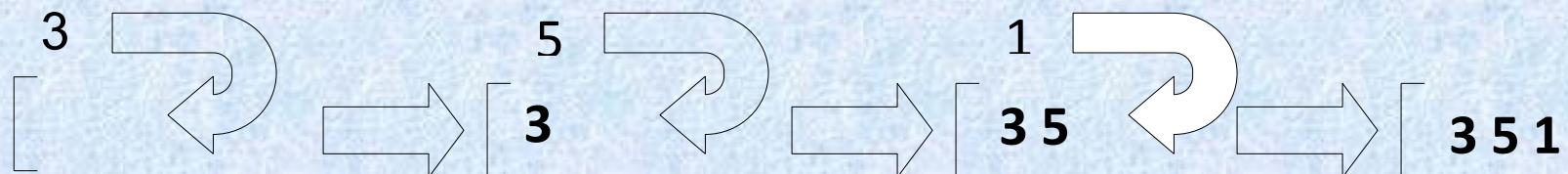
COLAS

Una cola C es una estructura lineal caracterizada porque las inserciones solo se permiten en uno de los extremos de la cola, llamado *final* o *último*, y las consultas o eliminaciones solo se permiten en el opuesto, llamado *principio* o *primero*.

- La cola puede no tener nada, situación que se denomina *cola vacía*.

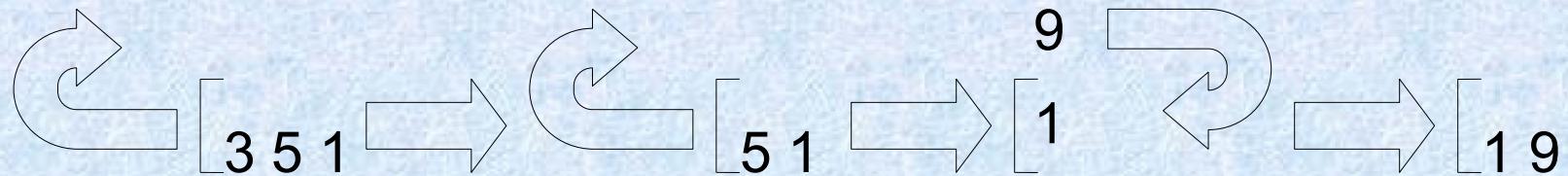
Las pilas se conocen también como estructuras FIFO (First In, First Out), por el modo en que se acceden los elementos.

Ejemplo: Poner los datos 3, 5 y 1 en una cola vacía.

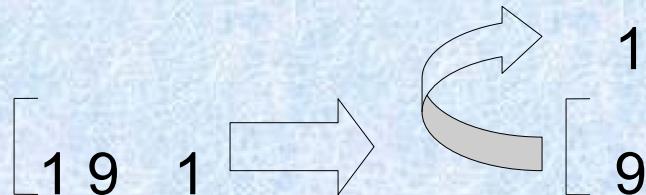


COLAS

Ejemplo: Quitar dos datos de la cola y poner un 9.



Ejemplo: Comprobar qué hay en la cola.



COLAS EN COMPUTACIÓN

Las colas se utilizan frecuentemente en simulación, dado que en prácticamente la totalidad de sistemas se encuentra algún tipo de cola.

- Los procesos que se ponen en una cola de espera según el orden en que se van lanzando.
- Puede haber distintas colas para diferentes prioridades.

ESPECIFICACIÓN: COLAS

{Como no sabemos qué tipo de elementos van a formar la cola, se pone una especificación genérica y usamos un parámetro formal}

espec COLA[ELEMENTO]

usa BOOLEANOS

parametro formal

generos elemento

fparametro

generos cola

ESPECIFICACIÓN: COLAS (2)

operaciones

{crear una cola vacía}

cvacia: → cola

{poner un elemento en la cola}

añadir: elemento cola → cola

Generadoras

{quitar un elemento de la cola }

parcial eliminar: cola → cola

Modificadoras

{ver el principio de la cola}

parcial primero: cola → elemento

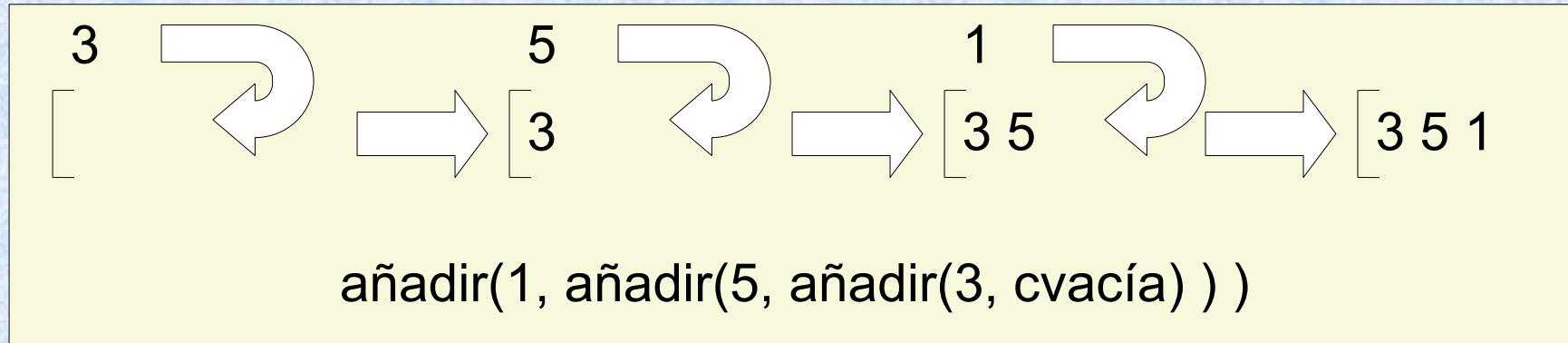
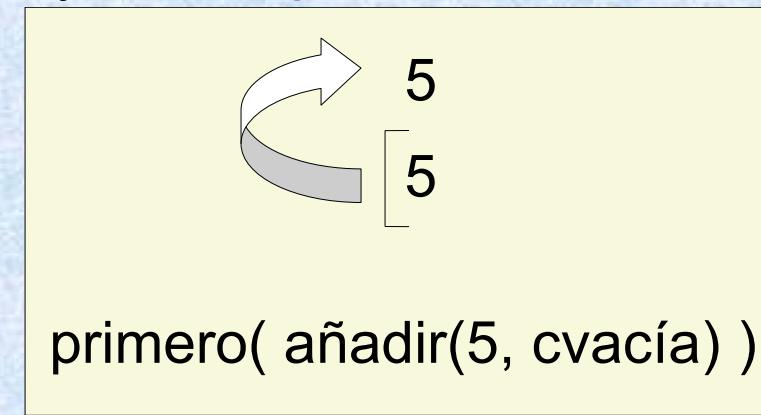
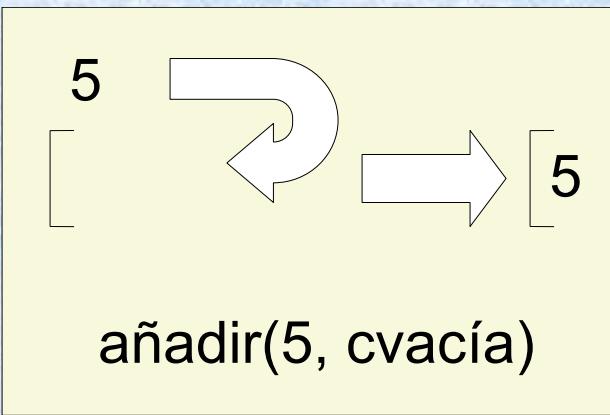
Observadoras

{ver si la cola está vacía}

vacía?: cola → bool

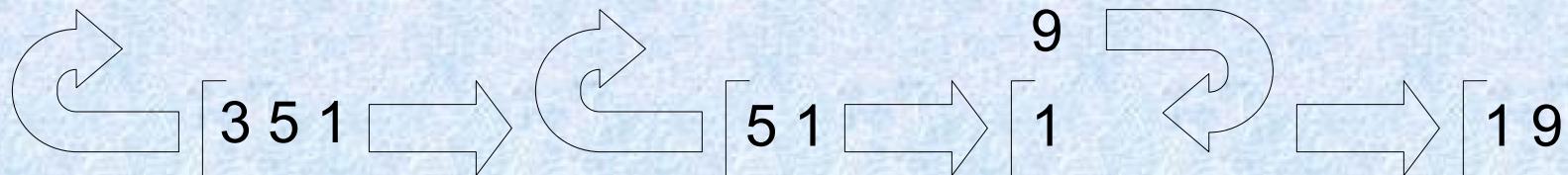
REPRESENTACIÓN DE LAS COLAS (1)

¿Qué podemos construir con números y estas operaciones?



REPRESENTACIÓN DE LAS COLAS (2)

¿Qué podemos construir con números y estas operaciones?



`añadir(9, eliminar(eliminar(añadir(1, añadir(5,añadir(3, vacía)))))))`



esta es la cola de la página anterior

ESPECIFICACIÓN: COLAS (3)

var c: cola; x: elemento

{Como el TAD tiene operaciones parciales hay que empezar por definir los datos sobre los que pueden usarse}

{Primera opción: indicando qué forma tienen que tener los datos} **ecuaciones de definitud**

Def (eliminar (añadir (x, c)))

Def (primero (añadir (x, c)))

{Segunda opción: con las propiedades que tienen que cumplir los datos para poder usar la operación} **ecuaciones de definitud**

vacía? (c) = F \Rightarrow Def (eliminar (c))

vacía? (c) = F \Rightarrow Def (primero (c))

ESPECIFICACIÓN: COLAS (4)

ecuaciones

eliminar(añadir(x,cvacía)) = cvacía

**vacía? (c)=F ⇒ eliminar(añadir(x,c)) =
añadir(x,eliminar(c))**

primero(añadir(x,cvacía)) = x

**vacía? (c)=F ⇒ primero(añadir(x,c)) =
primero(c)**

vacía? (cvacia) = T

vacía? (añadir(x,c)) = F

fespec

EJEMPLO 1

Ejemplo: Contar cuántos elementos tiene una cola.

- La operación (es observadora) es la siguiente:

contar: cola → natural

- Las ecuaciones pueden ser (basadas en generadoras)

contar(vacía) = 0

contar(añadir(x, c)) = suc(contar(c))

- Las ecuaciones pueden ser (basadas en propiedades)

vacía?(c) = T ⇒ contar(c) = 0

vacía?(c) = F ⇒

contar(c) = suc(contar(eliminar(c)))

EJEMPLO 1. PSEUDOCÓDIGO

Ejemplo: Contar cuántos elementos tiene una cola.

```
func contar (c:cola) dev n:natural           {recursiva}
      si vacia?(c) entonces devolver 0
      sino   desencolar(c)
              devolver 1+ contar(c)
      finsi
finfunc
```

EJEMPLO 1. PSEUDOCÓDIGO (2)

Ejemplo: Contar cuántos elementos tiene una cola.

```
func contar (c:cola) dev n:natural           {iterativo}  
var n:natural  
n←0  
mientras ¡vacía?(c) hacer  
    desencolar(c)  
    n←n+1  
finmientras  
finfunc
```

EJEMPLO 2

Ejemplo: Se conoce la operación `espar?`: `entero` → `bool`, que devuelve si un número entero es par o no; obtener la cantidad de números pares que hay en una cola de enteros.

- Es una operación observadora

`contar_pares`: `cola` → `natural`

- Las ecuaciones pueden quedar

`contar_pares (cvacia) = 0`

`espar? (x) = T` ⇒ `contar_pares (añadir (x, c)) =`

`suc (contar_pares (c))`

`espar? (x) = F` ⇒ `contar_pares (añadir (x, c)) =`

`contar_pares (c)`

EJEMPLO 3

Ejemplo: Especificar una operación para obtener la inversa de una cola, es decir, la cola resultante al cambiar el orden de los datos.

- Para invertir una cola, cogemos el primer dato que esté en la cola y lo ponemos al final de los otros datos, y se repite hasta que no queden cosas en la cola.

invertir: cola → cola

invertir(cvacía) = cvacía

vacía? (c) = F ⇒ invertir(c) =

añadir(primer(c), invertir(eliminar(c)))

- No es necesario usar un acumulador (como en las pilas) porque las colas tienen dos puntos de acceso distintos.

EJEMPLO 4

Ejemplo: Concatenar dos colas, poniendo la segunda después de la primera.

- Para concatenar una cola tras otra se pasan los datos uno a uno, de la segunda a la primera, hasta que no queden.

concatenar: *cola* *cola* → *cola*

- Usando propiedades:

vacía?(*c2*) = *T* ⇒ **concatenar**(*c1*, *c2*) = *c1*

vacía?(*c2*) = *F* ⇒ **concatenar**(*c1*, *c2*) =

concatenar(**añadir**(**primero**(*c2*), *c1*), **eliminar**(*c2*))

- Usando generadores:

concatenar(*c1*, **cvacía**) = *c1*

concatenar(*c1*, **añadir**(*x*, *c2*)) =

añadir(*x*, **concatenar**(*c1*, *c2*))

IMPLEMENTACIÓN DE COLAS

Las colas pueden representarse mediante **vectores**:

- La cola tiene el índice de la última celda ocupada.
- Al borrar un elemento, los restantes se desplazan.
- La cola tiene una capacidad fija (nuevas operaciones).

O mediante **vectores circulares**:

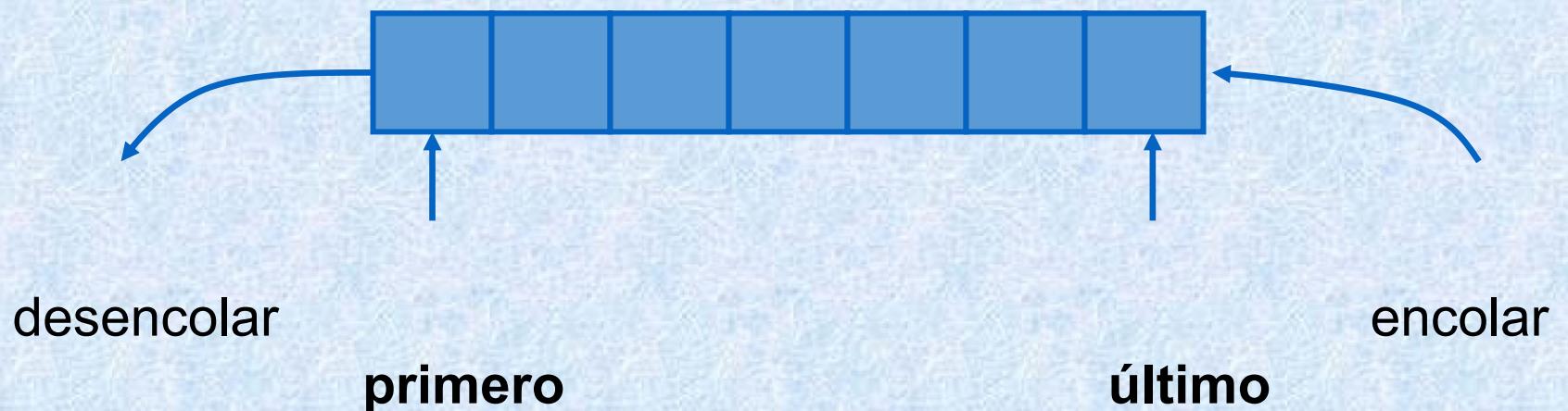
- Se evitan desplazamientos al borrar un elemento.
- Es necesario diferenciar cuando la cola está llena o vacía.

La implementación más habitual es la de **celdas enlazadas**:

- La cola tiene una referencia a la primera celda y a la última
 - Si la cola está vacía, ambos punteros son “NIL”.
 - Cada celda contiene un elemento y un puntero a la *siguiente* celda.

IMPLEMENTACIÓN DE COLAS

- IMPORTANTE: ¡Solo
- se accede a la cabeza de la cola!
- IMPORTANTE: ¡Solo se inserta en el final de la cola!



COLAS. TIPOS

tipos

posic

ion: 0 .. max_elementos;

cola=**reg**

 primero, ultimo: Posicion;

 elementos: **vector** [posicion] **de** elemento;

freg

ftipos

- Vector: inicio siempre primera posición de vector.
- Vector circular: inicio va cambiando al eliminar o *desencolar*.

COLAS. TIPOS

tipos

enlace-cola = **puntero a** nodo-cola

nodo-cola = **reg**

 valor: elemento

 sig: enlace-cola

freg

cola = **reg** *{ahora hay dos puntos de acceso}*

 primero: enlace-cola

 ultimo: enlace-cola

freg

ftipos

COLAS. CONSTRUCTORAS

{ Crear una cola vacía **cvacía** }

```
fun cola_vacía() dev c:cola
```

COLAS. CONSTRUCTORAS

{ Crear una cola vacía **cvacía** }

```
fun cola_vacía() dev c:cola  
    c.primero ← nil  
    c.ultimo ← nil  
ffun
```

COLAS. OBSERVADORAS

{ Ver si una cola está vacía **cvacía?** , obtener la cabeza **primero** }

fun es_colá_vacía(c:cola) **dev** b:bool

fun primero(c:cola) **dev** e:elemento

COLAS. OBSERVADORAS

{ Ver si una cola está vacía **cvacía?**, obtener la cabeza **primero** }

```
fun es_cola_vacia(c:cola) dev b:bool  
    b ← (c.primero = nil)
```

ffun

```
fun primero(c:cola) dev e:elemento  
    si es_cola_vacia(c) entonces  
        error(Cola vacía)  
    si no e ←  
        c.primero^.valor  
fsi ffun
```

COLAS. CONSTRUCTORAS (2)

{ Encolar un elemento añadir }

proc encolar(**E** e:elemento, **c**:cola)

COLAS. CONSTRUCTORAS (2)

{ *Encolar un elemento añadir* }

```
proc encolar(E e:elemento, c:cola)
var p: enlace-cola reservar(p)
p^.valor ← e
p^.sig ← nil
si es_cola_vacía(c) entonces
    c.primero ← p
si no
    c.ultimo^.sig ← p
fsi
c.ultimo ← p
fproc
```

COLAS. MODIFICADORAS

*{ Quitar el primer dato **eliminar** }*

proc desencolar (c:cola)

COLAS. MODIFICADORAS

{ Quitar la cabeza **desencolar** }

```
proc desencolar(c:cola)
var p: enlace-cola
si es_cola_vacía(c) entonces error(Cola vacía)
si no p ← c.primero
        c.primero ← c.primero^.sig
        si c.primero = nil entonces
                c.ultimo ← nil
fsi
        p^.sig ← nil      {por seguridad}
        liberar(p)
fsi
fproc
```

EJEMPLO 4. PSEUDOCÓDIGO DE CONCATENAR

Al trabajar con punteros no es necesario mover cada uno de los elementos, basta con redireccionar el final de la primera cola.

```
proc concatenar(c1,c2: cola)
si es_cola_vacía(c1) entonces
    c1.primero ← c2.primero
    c1.ultimo ← c2.ultimo
```

```
si no
```

```
    c1.ultimo^.sig ← c2.primero
    c1.ultimo ← c2.ultimo
```

```
fsi
```

fproc IMPORTANTE: ¡No se ha hecho una copia de los datos!
Si se cambian los valores que se encuentran en `c2`, también afecta a `c1` al ser enlaces directos a memoria, y viceversa.

EJERCICIOS

- Escribir en pseudocódigo las operaciones de los ejemplos 2, 3 y 4 (transparencias 14, 15 y 16).
- Escribir en pseudocódigo la implementación de las operaciones básicas de una cola utilizando un vector circular (transparencia 17).

EJEMPLO 2 . PSEUDOCÓDIGO

Ejemplo: Se conoce la operación **espar?**: **entero** → **bool**, que devuelve si un número entero es par o no; obtener la cantidad de números pares que hay en una cola de enteros.

func npares (c:cola) **dev** n:natural {iterativa}

n←0

mientras ¡ vacia?(c) **hacer**

si espar?(primero (C)) **entonces** n←n+1

desencolar(C)

finmientras

finfunc

EJEMPLO 2 . PSEUDOCÓDIGO (2)

```
func npares (c:cola) dev n:natural {recursiva}
    si vacia?(c) entonces devolver 0
    sino   si espar?(primero (c))
            entonces
                devolver 1+npares(desencolar(c))
            sino devolver npares(desencolar(c))
    finsi
finfunc
```

EJEMPLO 3 . PSEUDOCÓDIGO

Ejemplo: Especificar una operación para obtener la inversa de una cola, es decir, la cola resultante al cambiar el orden de los datos.

```
func inversa (c:cola):cola
    si vacia?(c) entonces devolver c
    sino
        e←primero(c)
        desencolar(c)
        devolver (encolar (e, inversa(c)))
    finsi
finfunc
```

EJEMPLO 3 . PSEUDOCÓDIGO (2)

```
func inversa (c:cola):cola {iterativa}
var ci:cola
ci←cvacia
paux←pvacia
mientras ¡vacia?(c) hacer
    apilar(primer(c), paux)
    desencolar(c)
finmientras
mientras ¡vacia?(paux) hacer
    encolar(cima(paux), ci)
    desapilar(paux)
finmientras
finfunc
```

EJEMPLO 4 . PSEUDOCÓDIGO

Ejemplo: Concatenar dos colas, poniendo la segunda después de la primera.

```
proc concatenar (c1, c2:cola) {iterativa}
{concatena los eltos de c2 al fnal de c1}
    mientras ¡ vacia?(c2) hacer
        encolar(primero(c2), c1)
        desencolar(c2)
    finmientras
finproc
```

EJEMPLO 4 . PSEUDOCÓDIGO (2)

```
proc concatenar (c1, c2:cola) {recursiva}
{concatena los eltos de c2 al fnal de c1}
    si vacia? (c2) entonces devolver c1
        sino
            e←primero (c2)
            desencolar(c2)
            concatenar(encolar(e, c1), c2)
    finsi
finproc
```

COLAS. TIPOS

{Cola implementada con **vector circular**}

tipos

posicion: 0..max_elementos;

cola=**reg**

 primero, ultimo:Posicion;

 elementos :**vector** [posicion] **de** elemento;

freg

ftipos

COLAS. CONSTRUCTORAS

{Crear una cola vacía **cvacía** }

fun cvacía() **dev** c:cola

{para diferenciar vector lleno y vector vacío dejaremos, al insertar/encolar datos, una posición vacía entre el ultimo y el primer elemento del vector}

c.primero = 0

c.ultimo = max_elementos

finfunc

COLAS. OPERACIÓN AUXILIAR

{posición en la que vamos a encolar si no está lleno}

fun pos_siguiente (p:posición) **dev** posicion

{Supondremos en las operaciones que el tipo posición es NATURAL, puede ser otro tipo enumerado, utilizaremos las operaciones +, MOD, =, < }

devolver 1+(p MOD max_elementos)

finfunc

COLAS. OBSERVADORAS

{Comprobar si una cola está vacía **cvacía?**}

```
fun cvacía?(c:cola) dev b:bool  
    devolver c.primero = pos_siguiente(c.ultimo)  
finfunc
```

COLAS. OBSERVADORAS

```
fun primero(c:cola) dev e:elemento  
si cvacia?(c) entonces error(Cola vacía)  
si no devolver c.elementos[primero]  
finsi  
finfunc
```

COLAS. CONSTRUCTORAS (2)

{Encolar/insertar un elemento, para diferenciar vector lleno y vector vacío dejaremos una posición vacía entre el ultimo y el primer elemento del vector}

```
proc encolar(E e:elemento, c:cola)
    si pos_siguiente(pos_siguiente (ultimo))=primero
        entonces error(Cola llena)
    si no
        c.ultimo ← pos_siguiente(c.ultimo)
        c.elementos[ultimo]←e
    finsi
finproc
```

COLAS. MODIFICADORAS

{ *Quitar el primer dato eliminar* }

```
proc desencolar(c:cola)
    si es_colá_vacía(c) entonces error(Cola vacía)
        si no c.primer ← pos_siguiente(c.primer)

    finsi

finproc
```