TEMA 4

Estructuras de Datos Avanzadas

Contenido del Tema

4.1. Introducción

4.5. Arboles Binarios.

Arboles Binarios de Búsqueda

Introducción

Objetivos

- Especificación e Implementación de nuevas estructuras de datos ⇨ Técnica:

- Tipos de Estructuras de Datos:

1) Datos organizados por ⇨ Pilas , Colas

2) Datos organizados por ⇨ Arboles Binarios

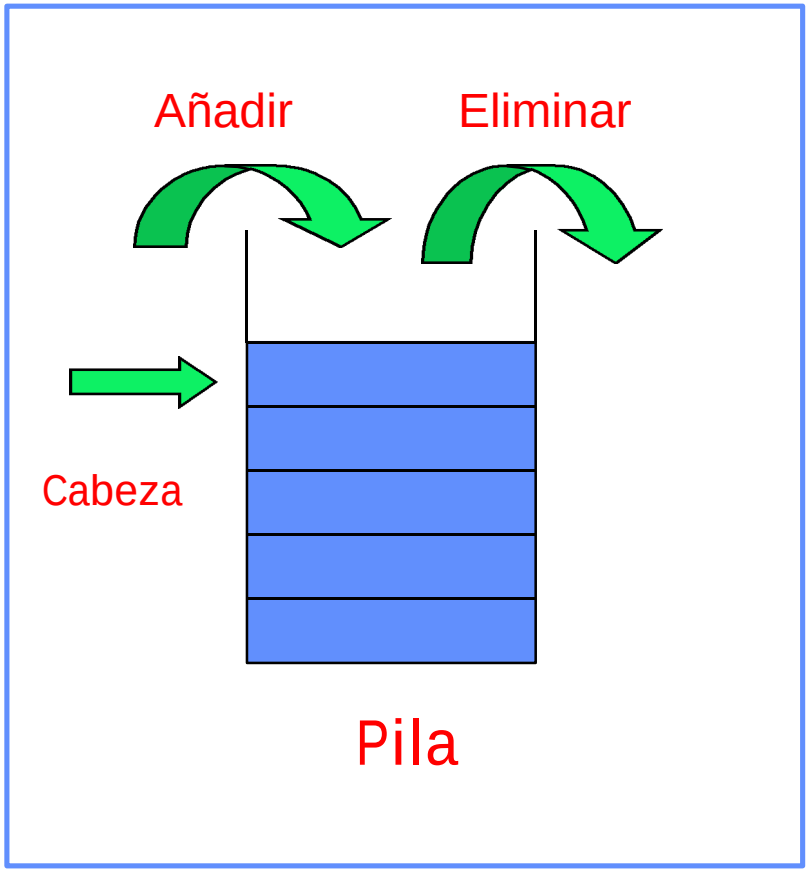
Introducción

- Estudio de las Estructuras de Datos:
  - de la Estructura de Datos e identificación de su
  - Desarrollo de diversas
  -

Pilas

Definición

- **Pila:** Grupo Ordenado, (de acuerdo al tiempo que llevan en la pila) de Elementos Homogéneos (todos del mismo tipo).
- **Acceso a la Pila:** añadir y eliminar elementos, SÓLO a través de la CABEZA de la Pila
- Estructura **LIFO** (**L**ast **I**nput **F**irst **O**utput)



Pilas. Operaciones

Conjunto de Operaciones

```
Modulo Pila
Definición
Tipos
      TipoElemento= (*cualquier tipo de datos*)
                  = (*por definir*)
Algoritmo   MeterPila(   VAR   pila:   TipoPila   ;
                          elem : TipoElemento)
(*Añade un elemento por la cabeza de la pila*)
Algoritmo SacarPila(VAR pila:TipoPila;VAR elem:
TipoElemento)
(* Saca un elemento por la cabeza de la Pila*)
```

Pilas. Operaciones

Conjunto de Operaciones

```
Algoritmo CrearPila():Tipopila
(* Crea una pila vacía*)
Algoritmo PilaVacia (pila :TipoPila):B
(*Operación lógica que nos dice si una pila está

Algoritmo PilaLlena(pila:TipoPila):B
(*Operación lógica que nos dice si una pila está
  llena   o   no.   Necesaria   en   determinadas
                    *)
Algoritmo DestruirPila(VAR pila:TipoPila)
(* Destruye una pila previamente creada*)
Fin
```

Pilas. Implementación

Implementación

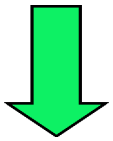
1) Con un Array

- Array estructura adecuada ⇨ Elementos Homogéneos
- Elementos almacenados de forma Secuencial

```
Constantes
    MaxPila=100
Tipos
TipoElemento=Z
TipoPila=ARRAY[1..MaxPila]DE TipoElemento
```

Pilas. Implementación

Sólo es posible acceder a la Cabeza de la Pila



¿ Cómo es posible conocer la posición de la cabeza?

1) Variable entera “cabeza” ⇨ Inconveniente: se ha de pasar como parámetro adicional a todas las operaciones sobre la

2) almacenaremos el índice del elemento que ocupa la cabeza actual

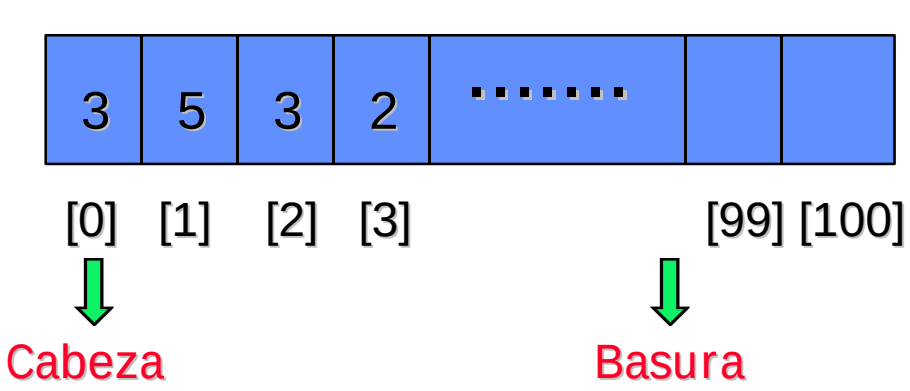
Pilas. Implementación

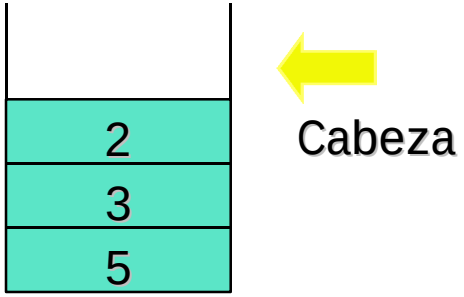
Constantes

```
Cabeza=0; MaxPila=100;
```

Tipos

```
TipoPila= ARRAY[Cabeza..MaxPila]DE TipoElemento
```

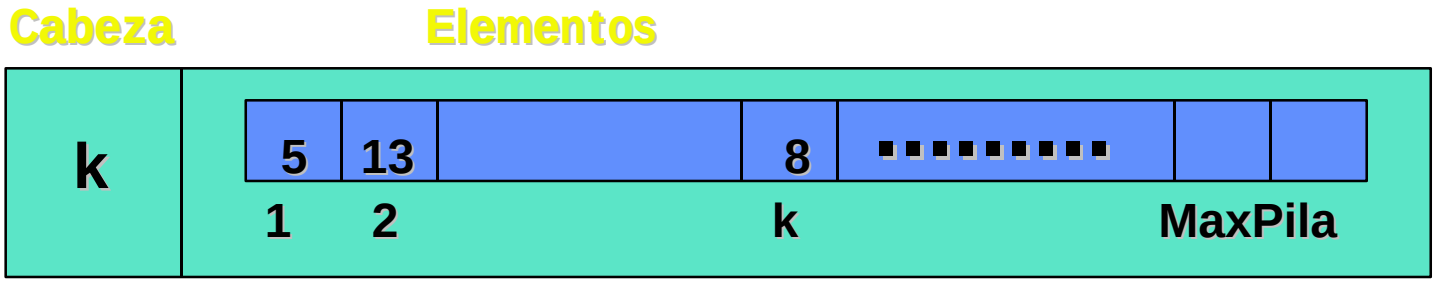




Pilas.Implementación

- Inconveniente: Solo es posible implementar una pila de enteros (no de cualquier otro tipo de datos)
- :

```
Constantes
  MaxPila=100
Tipos
  TipoElemento=(*cualquier tipo de datos*)
  TipoPila=REGISTRO
             Cabeza:Z
             Elementos: ARRAY[1..MaxPila] DE
                        TipoElemento
```



Pilas.Implementación

```
Algoritmo CrearPila(): TipoPila
VAR pila:TipoPila
Inicio
    Pila.Cabeza:= 0
    DEVOLVER (Pila)
Fin
Algoritmo      PilaVacia (pila : TipoPila ) : B
Inicio
DEVOLVER(pila.Cabeza=0)
Fin
```

```
Algoritmo PilaLlena(pila: TipoPila):B
Inicio
    DEVOLVER (pila.Cabeza = MaxPila)
Fin


Algoritmo DestruirPila(VAR pila:TipoPila)
Inicio

Fin
```

Pilas.Implementación

Implementación

```
Algoritmo SacarPila(
      VAR pila :TipoPila;
   VAR elem:TipoElemento)
Inicio
   elem := pila.Elementos
   [pila.Cabeza]
   pila.Cabeza :=
   pila.Cabeza - 1
Fin
```

(* precondición: la pila no ha de
*)

```
Algoritmo MeterPila(
      VAR pila :TipoPila;
      elem :TipoElemento)
Inicio
   pila.Cabeza :=
   pila.Cabeza + 1
   pila.Elementos
   [pila.Cabeza] := elem
Fin
```

(* precondición: la pila no ha de

Pilas.Implementación

```
Algoritmo MeterPila ( VAR
   pila:TipoPila;elem:TipoEle
   mento;VAR llena : B)
Inicio
   llena:= PilaLlena(pila)
    SI NOT llena ENTONCES
      pila.Cabeza:=
            pila.Cabeza + 1
      pila.Elementos
      [pila.Cabeza]:=elem
    FINSI
Fin
(* Sin Precondición.Introduce
   un elemento en la pila si
   no está llena*)
```

```
Algoritmo SacarPila(VAR pila
   :  TipoPila ; VAR elem :
   TipoElemento;VAR vacia:B)
Inicio
 vacia := PilaVacia(pila)
   SI NOT vacia ENTONCES
    elem := pila.Elementos
   [pila.Cabeza]
   pila.Cabeza :=
   pila.Cabeza - 1
   FINSI
Fin
(* Sin precondición. Saca un
   elemento de la pila si no
   está vacía*)
```

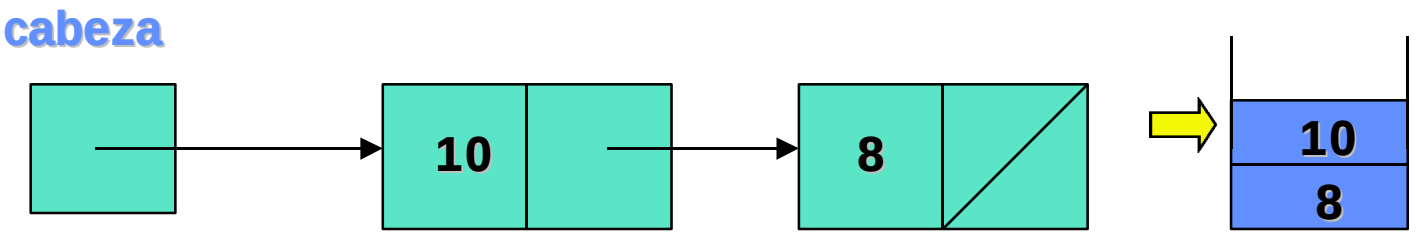
Pilas.Implementación

2) Con una Lista Enlazada de Punteros

- Comienzo de una lista enlazada ⇨ Cabeza de la Pila

Tipos

```
   TipoElemento=(*cualquier tipo de datos*)
   TipoPila=PUNTERO A ElementoPila
   ElemPila=REGISTRO
                  dato:TipoElemento
                  sig: TipoPila
                FINREGISTRO
```



Pilas.Implementación

```
Algoritmo MeterPila ( VAR
   pila:TipoPila;elem:
   TipoElemento)
Variables
   nuevonodo : TipoPila
Inicio
   Asignar(nuevonodo,
   TAMAÑO(ElemPila))
   nuevonodo^.dato := elem
   nuevonodo^.sig := pila
   pila := nuevonodo
Fin
```

```
Algoritmo SacarPila (VAR
   pila:TipoPila; VAR elem
   :TipoElemento)
Variables
      puntero: TipoPila
Inicio
   elem := pila^.dato
   puntero := pila
   pila := pila^.sig
   Liberar(puntero,
   TAMAÑO(ElemPila))
Fin
```

Pilas.Implementación

```
Algoritmo
CrearPila():TipoPila
Variables pila:TipoPila
Inicio
    pila := NIL
    DEVOLVER pila
Fin
Algoritmo       PilaVacia
(pila : TipoPila):B
Inicio
    DEVOLVER (pila= NIL)
Fin
```

```
Algoritmo  DestruirPila  (
   VAR pila:TipoPila)

Variables

      Ptr: TipoPila
Inicio
MIENTRAS(pila<>NIL)HACER
   Ptr:=pila
   pila:=pila^.sig
   Liberar(Ptr,
   TAMAÑO(ElemPila))
FINMIENTRAS
Fin
```

Pilas. Aplicaciones

Aplicaciones

- Ejemplo1: Leer una secuencia de caracteres desde teclado e imprimirla al revés
- : Verificar si una cadena de caracteres está balanceada en paréntesis o no

  abc((ijk mn)op)qr ⟹ SI

  (def)) (kl)m ⟹ NO
- : Reconocimiento del Lenguaje,

L={W$W´ / W es una cadena de caracteres y W´es su

(Suponemos que $ no está ni en W ni en W´)

Pilas. Ejemplo1

```
Algoritmo Inverso

Desde      Pila      Importa
   TipoPila,      CrearPila,
   MeterPila,     SacarPila,
   Pilavacia, DestruirPila
Tipos
   TipoElemento = $
Variables
   c : TipoElemento
   pila : TipoPila
Inicio
   pila:=CrearPila()
   Leer(c)
MIENTRAS c <> CHR(13)HACER
   MeterPila(pila,c)
   Leer(c)
FINMIENTRAS
MIENTRAS  NOT
   (PilaVacia(pila)) HACER
   SacarPila(pila,c)
   Escribir(c)
FINMIENTRAS
DestruirPila(pila)
Fin
```

Pilas. Ejemplo2

```
Algoritmo Balanceo
Desde    Pila    Importa    CrearPila,
    TipoPila,  MeterPila,  SacarPila,
    PilaVacia, DestruirPila
Tipos
    TipoElemento = $
Variables
    c : TipoElemento
    pila : TipoPila
    bien : B
Inicio
    pila:=CrearPila()
    bien := TRUE
    Leer(c)
    MIENTRAS(bien   AND   (c<>CHR(13)))
                                  HACER
    SI c= '(' ENTONCES
        MeterPila(pila,c)
    EN OTRO CASO
        SI c = ')' ENTONCES
          SI NOT PilaVacia(pila)
                        ENTONCES
                SacarPila(pila,c)
          EN OTRO CASO
                bien := FALSE
          FINSI
        FINSI
    FINSI
    Leer(c)
FINMIENTRAS
SI bien AND
        (PilaVacia(pila) ENTONCES
    Escribir("cadena balanceada ")
EN OTRO CASO
    Escribir("cadena no balanceada")
FINSI
DestruirPila(pila)
Fin
```

Pilas. Ejemplo3

```
Algoritmo Lenguaje_L
Desde    Pila    Importa    TipoPila,
    CrearPila,              MeterPila,
    SacarPila, DestruirPila
Tipos
    TipoElemento = $
Variables
    c1, c2 : TipoElemento
    pila : TipoPila
    bien : B
Inicio
    pila:=CrearPila()
    Leer(c1)
    MIENTRAS (c1 <> '$') HACER
    MeterPila(pila,c1);
    Leer(c1)
    FINMIENTRAS
    Leer(c1);
    bien := TRUE
```

```
MIENTRAS (bien AND (c1 <>
    CHR(13))) HACER
    SI PilaVacia(pila)ENTONCES
        bien := FALSE
    EN OTRO CASO
        SacarPila(pila,c2)
        SI (c1 <> c2) ENTONCES
          bien := FALSE
        EN OTRO CASO
          Leer(c1)
        FINSI
    FINSI
FINMIENTRAS
SI (bien AND
    PilaVacia(pila))ENTONCES
    Escribir (" Si pertenece")
EN OTRO CASO
    Escribir ("No pertenece")
FINSI
DestruirPila(pila)
Fin
```

Pilas. Aplicaciones

- Aplicaciones complejas que se pueden solucionar con **Expresiones Algebraicas**

  Operadores: +, -, *, /

  Operandos: Letras mayúsculas

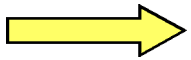
- **Notación Infija:**
- El operador binario está situado entre sus dos

- : Son necesarias reglas de precedencia y uso de paréntesis para evitar ambigüedades 

Pilas. Aplicaciones

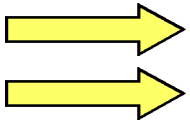
Notación Prefija

- El operador binario esta situado justo antes de sus dos



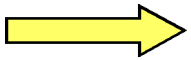
<expr_pref>::=<letra>|<operador>
<expr_pref><expr_pref>

<operador> ::= + | - | * | /



Notación Postfija

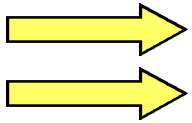
- El operador binario está situado justo después de sus dos



<exp_post>::=<letra>|<expr_post>
<exp_post><operador>

<letra> ::=A| B ....|Z

<operador> ::= + | - | * | /



Pilas. Aplicaciones

- Ventaja: Usando expresiones prefijas y reglas de precedencia, ni uso de paréntesis.

  Las gramáticas que las generan son muy simples, y los algoritmos que las reconocen y evaluan muy fáciles

- Ejemplo 4: Algoritmo que evalúa una expresión en notación

  1)Usaremos una pila

  se almacenará en un

  3)Los operadores serán: +, -, * y /

  serán letras mayúsculas (a cada una le podemos asignar un valor)

Pilas. Ejemplo4

Tipos

```
TipoArray = ARRAY
[1..20] DE $
TipoElemento=Z
```

```
Algoritmo Operando(c:$):Z
Inicio
   CASO  c SEA
      'A' : DEVOLVER(5)
      'B' : DEVOLVER(7)
      'C' : DEVOLVER(-1)
      'D' : DEVOLVER(11)
   EN OTRO CASO
      DEVOLVER 0
   FINCASO
Fin
```

Pilas. Ejemplo4

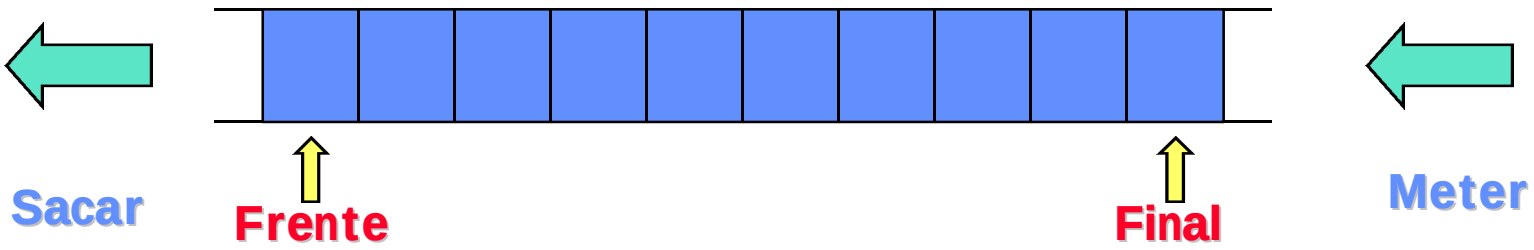
```
Algoritmo    Evaluar_postfija(exp: TipoArray; ultimo:Z):Z
Desde   Pila   Importa   TipoPila, CrearPila,          MeterPila, SacarPila, DestruirPila
Variables
pila : TipoPila
i, op1, op2, result : Z
c : $
conj: CONJUNTO DE $
Inicio
   pila:=CrearPila()
   conj:={`*´, `/´, `+´, `-´}
   PARA i := 1 HASTA ultimo HACER
     c := exp[i]
    SI c EN conj ENTONCES
      SacarPila(pila,op2)
      SacarPila(pila,op1)
      CASO  c SEA
      '+' : MeterPila(pila, op1+op2)
      '-' : MeterPila(pila, op1-op2)
      '*' : MeterPila(pila, op1*op2)
      '/' : MeterPila(pila, op1/op2)
      FINCASO
   EN OTRO CASO
      MeterPila(pila, Operando(c))
   FINSI
FINPARA
      SacarPila(pila,result)
      DestruirPila(pila)
      DEVOLVER result
Fin
```

Colas

Definición

- Cola: es un grupo ordenado (con respecto al tiempo que llevan en él) de elementos homogéneos (todos del mismo

- los elementos se añaden por un extremo ( ) y se sacan por el otro extremo ( )
- **FIFO** (**F**irst **I**nput **F**irst **O**utput)



Colas. Operaciones

Conjunto de Operaciones

```
Modulo Cola
Definición
Tipos
   TipoElemento=(*cualquier tipo de datos*)
   TipoCola=(*por definir*)
Algoritmo CrearCola(): TipoCola
(* Crea una cola vacia*)
Algoritmo ColaVacia(cola:TipoCola): B
(*  Operación  lógica  que  nos  dice  si  la  cola
   contiene algún elemento o no*)
```

Colas. Operaciones

```
Algoritmo ColaLlena(cola:TipoCola):B

(* Determina si una cola está llena (sólo
  necesaria en algunas implementaciones)*)

Algoritmo MeterCola(VAR cola: TipoCola;
                    elem: TipoElemento)

(* Introduce un elemento al final de la cola*)

Algoritmo SacarCola(VAR cola: TipoCola;
                                VAR elem: TipoElemento)
(*Saca un elemento del frente de la cola*)
Algoritmo DestruirCola(VAR cola:TipoCola)
(* Destruye una cola previamente creada*)
Fin
```

Colas. Implementación

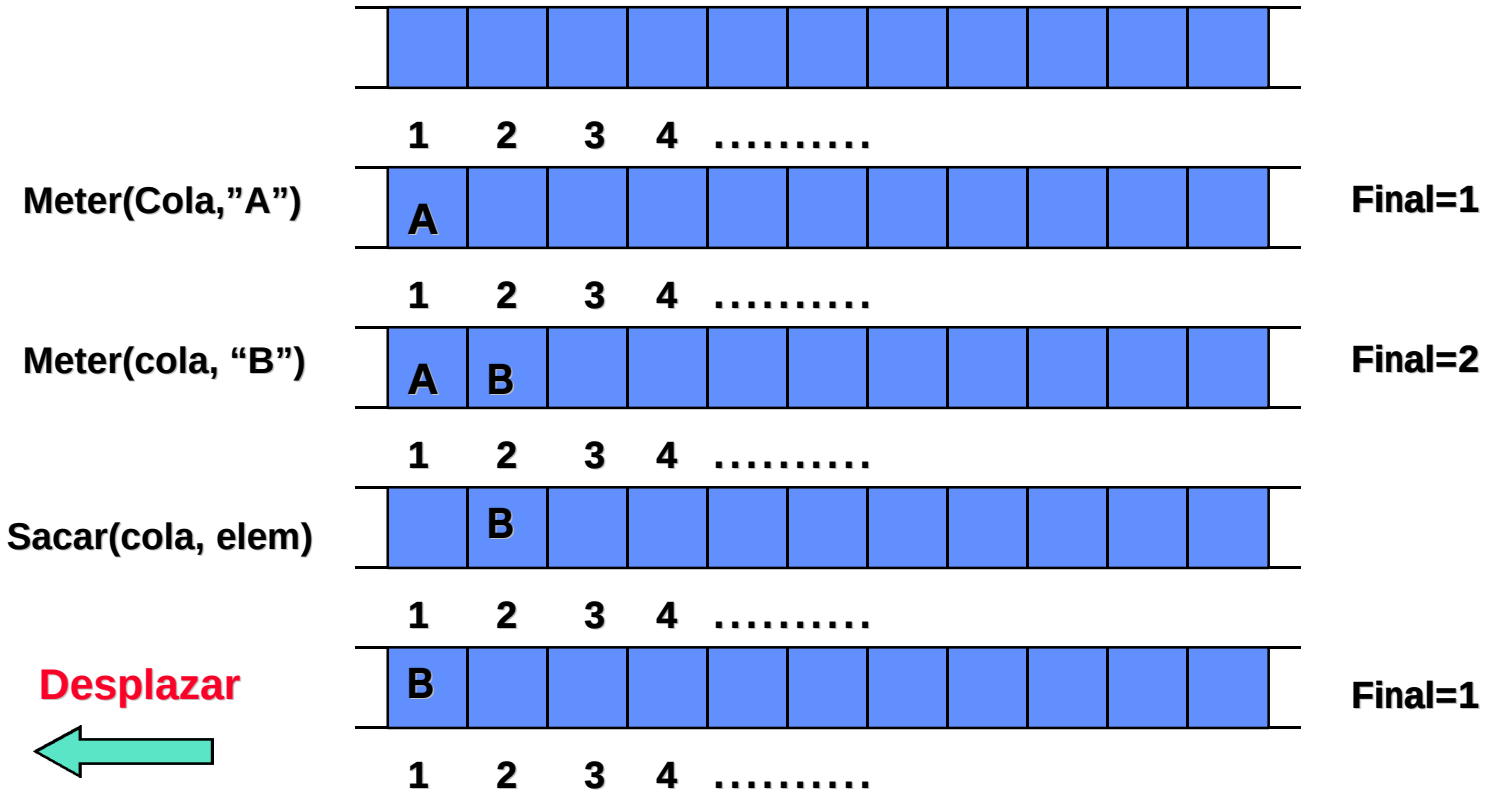
Implementación

1) Con un Array

- Se deja fijo el frente de la cola y se mueve el final a medida que se añaden nuevos elementos (Idem)
- , , y de una forma similar a sus análogas en Pilas
- : cada vez que saquemos un elemento de la cola se han de desplazar el resto una posición en el , para hacer coincidir el frente con la primera posición del
- ⇒
- ⇒ (colas con muchos elementos o elementos grandes)

Colas. Implementación

- Ejemplo:



Colas. Implementación

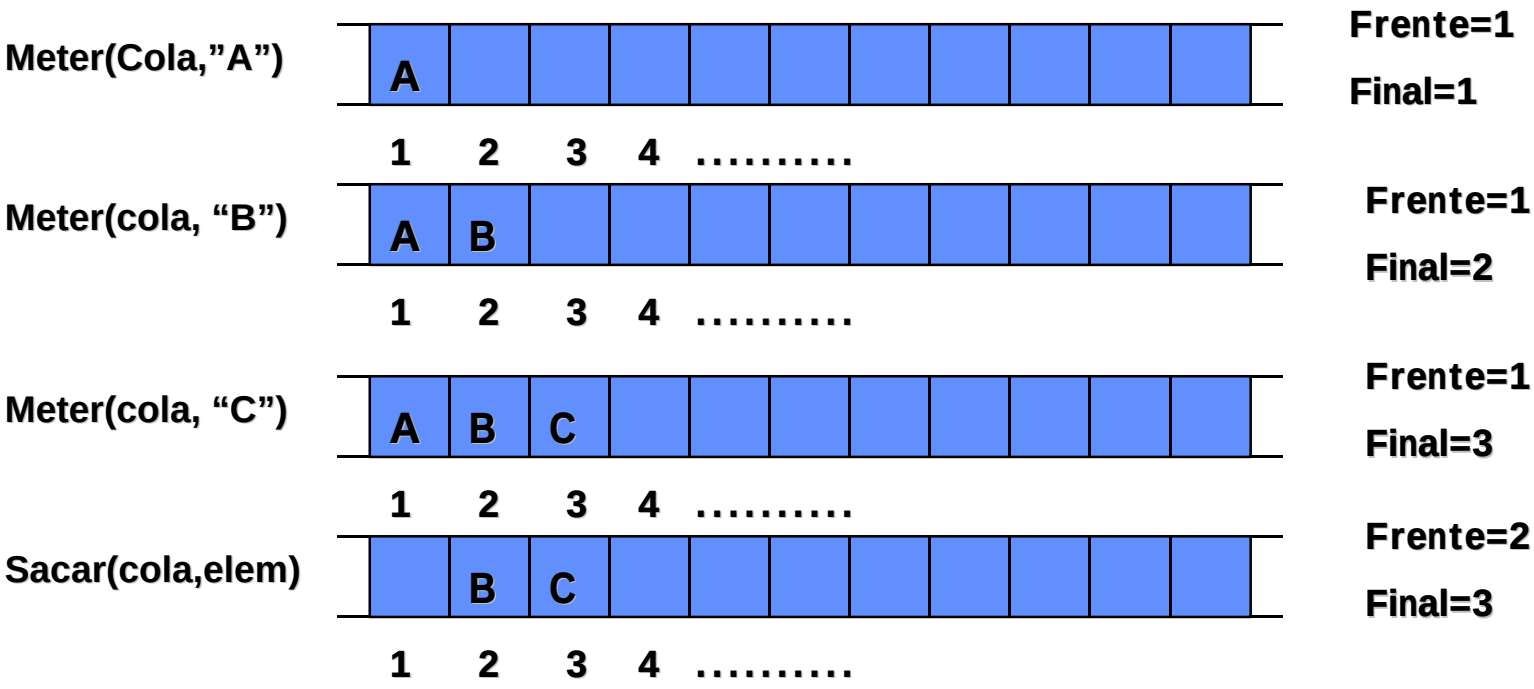
Solución:

- Utilizar un indice para el frente y otro para el final y permitir que ambos

-
- : Es posible que final sea igual a (última casilla del) y que la cola no esté llena

Colas. Implementación

- Ejemplo:



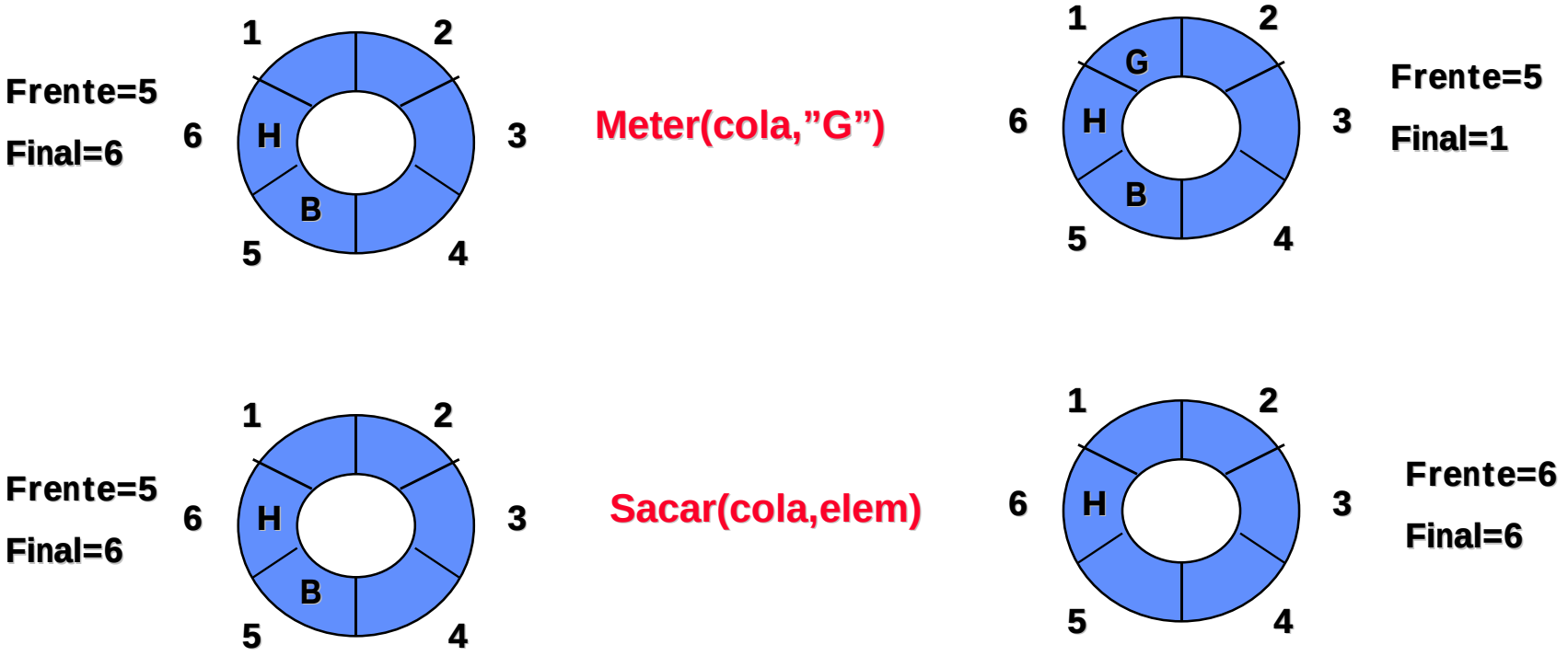
Colas. Implementación

- Solución:
- Tratar al array como una **Estructura Circular**, donde la última posición va seguida de la primera ⇨ Evitamos que el final de la cola alcance el final físico del

- ⇨ Añade elementos a las posiciones del e incrementa el índice final
- ⇨ Más sencilla. Sólo se incrementa el índice frente a la siguiente posición

Colas. Implementación

- **Ejemplo:**

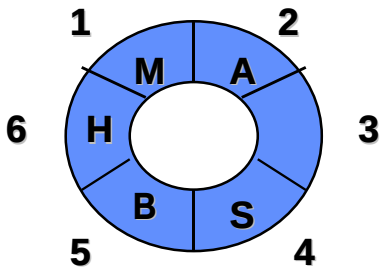


Colas. Implementación

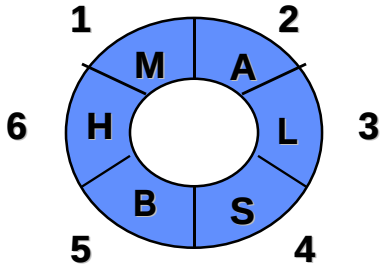
¿Como sabemos si la cola está vacía o llena?

Frente=4

Final=2



Meter(cola,"L")



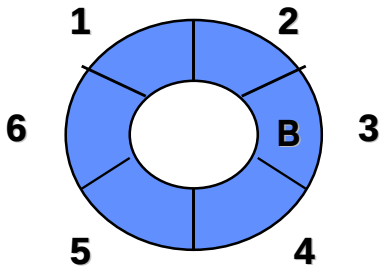
Frente=4

Final=3

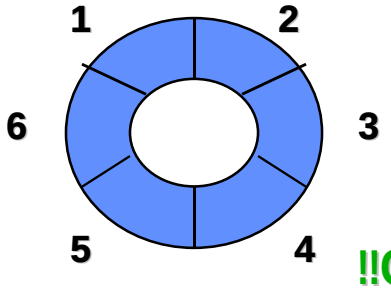
!!Cola llena!!

Frente=3

Final=3



Sacar(cola,elem)



Frente=4

Final=3

!!Cola Vacía!!

Colas. Implementación

- Solución:
- 1) Disponer de otra variable ⇨ Contabilizará los elementos almacenados en la cola

  Variable=0 ⇨ Cola vacía

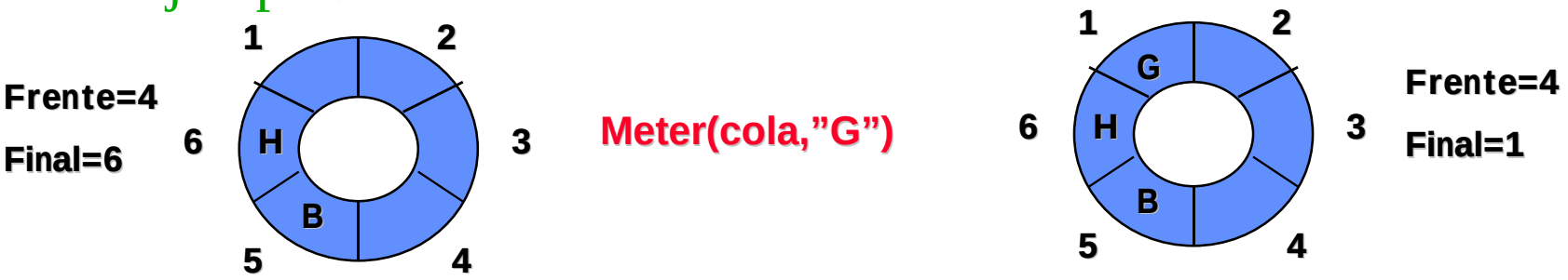
  ⇨ Cola llena

  : añade más procesamiento a las operaciones

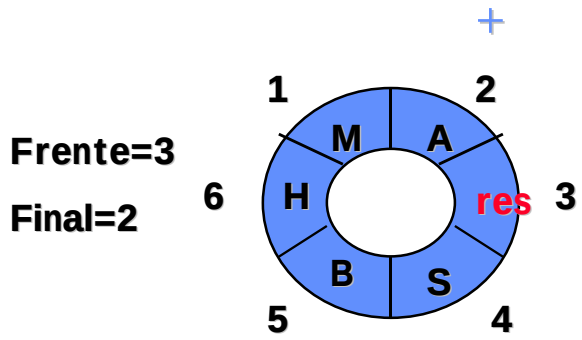
- 2) apunte a la casilla del elemento frente de la cola ⇨ Solución elegida

Colas. Implementación

- Ejemplo:



¿Como saber si la cola está llena? Es necesario que la posición a la que apunta frente en cada momento este



Colas. Implementación

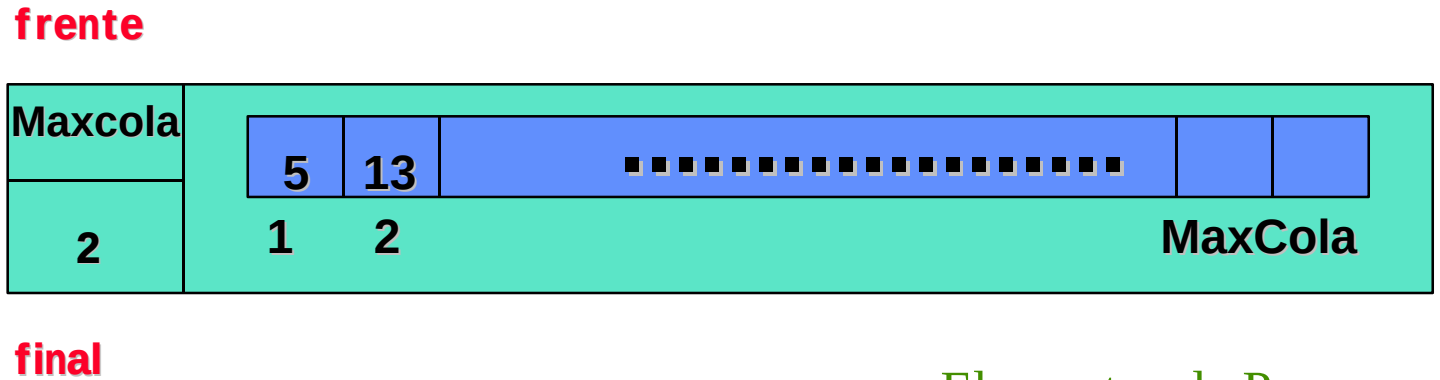
¿Como saber si la cola está vacía?

- que precede al elemento frente de la cola) y
  ⟹ Cola Vacía correcto (frente = final)

Colas. Implementación

- Agrupamos en un registro los índices frente y final, junto que contendrá los elementos de la cola

```
Constantes  MaxCola = 100
Tipos       TipoElemento = (* Cualquier tipo de datos *)
                           REGISTRO
                           elementos:ARRAY[1..MaxCola]DE
                                       TipoElemento
                           Frente : [1..MaxCola]
                           Final : [1..MaxCola]
                           FINREGISTRO
```



Colas. Implementación

```
Algoritmo      CrearCola(): TipoCola
Variables      cola: TipoCola
Inicio
   cola.Frente:= MaxCola
   cola.Final := MaxCola
   DEVOLVER cola
Fin
Algoritmo DestruirCola
   (VAR cola:TipoCola)
Inicio

Fin
```

```
Algoritmo ColaLlena
(cola:TipoCola):B
Inicio
   DEVOLVER    ((cola.Final
   MOD   MaxCola   +   1)   =
   cola.Frente)
Fin
Algoritmo ColaVacia
   (cola:TipoCola):B
Inicio
   DEVOLVER cola.Final =
   cola.Frente
Fin
```

Colas. Implementación

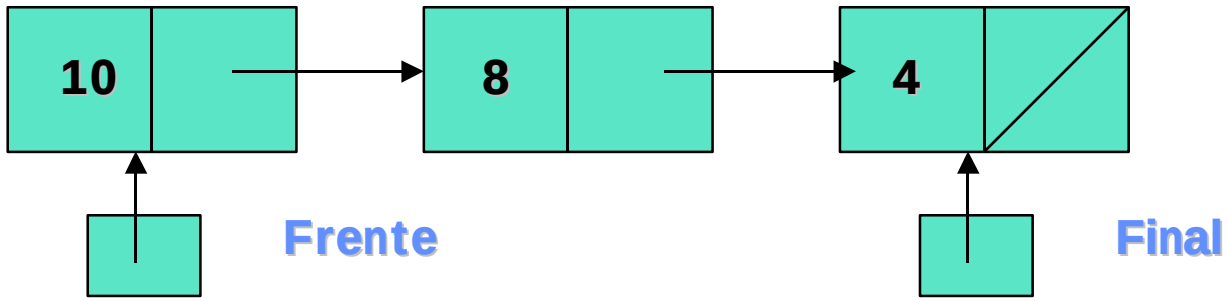
```
Algoritmo MeterCola (VAR
   cola:TipoCola; Elem :
   TipoElemento)
Inicio
   cola.Final:= cola.Final
   MOD MaxCola + 1
   cola.elementos
      [cola.Final] := elem
Fin
```

```
Algoritmo SacarCola (VAR
   cola:TipoCola; VAR elem
   : TipoElemento)
Inicio
   cola.Frente:=
   cola.Frente MOD MaxCola
   + 1
   elem:= cola.elementos
            [cola.Frente]
Fin
```

Colas. Implementación

2) Con listas enlazadas con Punteros

- dos variables de tipo puntero, frente y final, que apunten a los nodos que contienen los elementos frente y

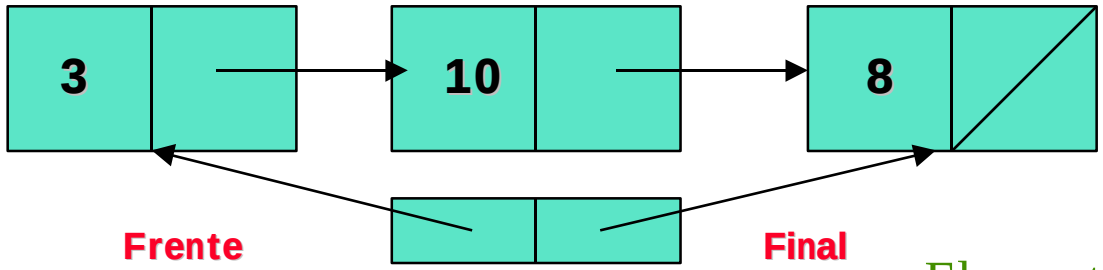


- ¿Que sucedería si intercambiáramos las posiciones de frente y final en la lista enlazada?

Colas. Implementación

- Agrupamos las variables frente y final en un registro

```
Tipos TipoElemento = (* cualquier tipo de datos *)
                     PUNTERO A Nodo
      Nodo = REGISTRO
                valor : TipoElemento
                sig : TipoPuntero
             FINREGISTRO
      TipoCola = REGISTRO
                   Frente : TipoPuntero
                   Final : TipoPuntero
                 FINREGISTRO
```



Colas. Implementación

```
Algoritmo    CrearCola():
             TipoCola

Variables cola:TipoCola
Inicio
      cola.Frente := NIL
      cola.Final := NIL
      DEVOLVER cola
Fin
Algoritmo   ColaVacia(cola   :
   TipoCola):B
Inicio
   DEVOLVER cola.Frente = NIL
Fin
Algoritmo   DestruirCola   (VAR
   cola:TipoCola)
```

```
Variables
nodo,siguiente: TipoPuntero
Inicio
SI(cola.Frente<>NIL) ENTONCES
   siguiente:=cola.Frente
   MIENTRAS     siguiente<>NIL
                     HACER
  nodo:=siguiente
  siguiente:=  siguiente^.sig
               TAMAÑO(Nodo))
  FINMIENTRAS
FINSI
cola.Frente:=NIL
cola.Final:=NIL
Fin
```

Colas. Implementación

```
Algoritmo MeterCola(VAR cola : TipoCola; elem: TipoElemento)
Variables
      nuevonodo : TipoPuntero
Inicio
   Asignar(nuevonodo, TAMAÑO(Nodo))
   nuevonodo^.valor := elem
   nuevonodo^sig := NIL
   SI   ColaVacia(cola)
                        ENTONCES
   cola.Frente:= nuevonodo
   EN OTRO CASO
   cola.Final^.sig:= nuevonodo
   FINSI
   cola.Final := nuevonodo
   Fin
```

```
Algoritmo SacarCola (VAR cola:TipoCola;VAR elem : TipoElemento)
Variables
   temp : TipoPuntero
Inicio
temp := cola.Frente
elem := cola.Frente^.valor
cola.Frente:=cola.Frente^.sig
SI  cola.Frente  =  NIL
                      ENTONCES
    cola.Final := NIL
FINSI
Liberar(temp, TAMAÑO(Nodo))
Fin
```

Colas. Aplicaciones

- **Ejemplo: Reconocimiento del lenguaje.**

L={W$W/W es una cadena que no

```
Algoritmo Lenguaje_L
Desde Cola Importa CrearCola,
   Colavacia,        SacarCola,
   MeterCola, DestruirCola
Tipos TipoElemento = $
Variables bien : B
   c1,c2 : TipoElemento
   cola : TipoCola
Inicio
 cola:=CrearCola(); Leer(c1)
 MIENTRAS (c1 <> '$') HACER
  MeterCola(cola,c1); Leer(c1)
 FINMIENTRAS
 Leer(c1); bien := TRUE
MIENTRAS (bien AND
   (c1<>CHR(13)) HACER
 SI colaVacia(cola) ENTONCES
       bien := FALSE
 EN OTRO CASO
       SacarCola(cola,c2)
       SI (c1 <> c2) ENTONCES
             bien := FALSE
       EN OTRO CASO
             Leer(c1)
       FINSI
 FINSI
FINMIENTRAS
SI (bien AND ColaVacia(cola))
                        ENTONCES
   Escribir("Pertenece")
EN OTRO CASO
   Escribir("No pertenece")
FINSI
DestruirCola(cola)
Fin
```