TEMA 5

# Estructuras de Datos Avanzadas

## Contenido del Tema





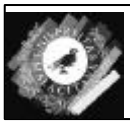
## Introducción

### Objetivos

- Especificación e Implementación de nuevas estructuras de datos ⇨ Técnica: Abstracción de Datos
- Tipos de Estructuras de Datos:

1) Datos organizados por Posición ⇨ Pilas , Colas y Listas

2) Datos organizados por Valor ⇨ Arboles Binarios



# Introducción

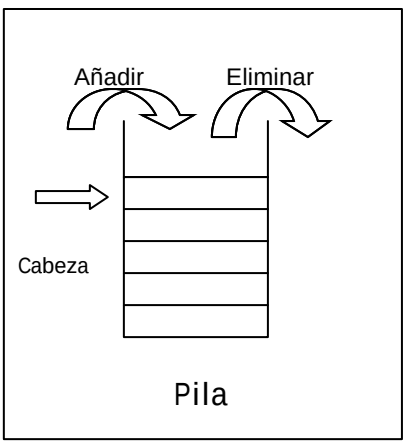
- Estudio de las Estructuras de Datos:
  - Definición de la Estructura de Datos e identificación de su Conjunto de Operaciones
  - Presentación de Aplicaciones
  - Desarrollo de diversas Implementaciones

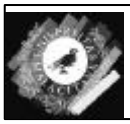


# Pilas

**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)





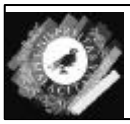
# Pilas. Operaciones

```
INTERFAZ CLASE CPila
   TIPOS
      TipoElemento ... // cualquier tipo de datos
   METODOS
      // Añade un elemento por la cabeza de la pila
      Apilar( E TipoElemento elem)
      // Saca un elemento por la cabeza de la Pila
      Desapilar()
      // Devuelve el elemento de la cabeza de la Pila
      TipoElemento Cima()
         ...
```



# Pilas. Operaciones 2

```
      ...
      // Crea una pila vacía
      Crear()
      //Operación lógica que nos dice si una pila está vacía o no
      B EstáVacía ()
      //Operación lógica que nos dice si una pila está llena o no.
      //Necesaria en determinadas implementaciones
      B EstáLlena()
      // Destruye una pila previamente creada
      Destruir()
FIN CPila
```



# Pilas. Aplicaciones

**Aplicaciones**

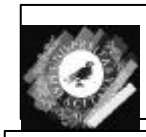
- Ejemplo1: Leer una secuencia de caracteres desde teclado e imprimirla al revés
- Ejemplo2: Verificar si una cadena de caracteres está balanceada en paréntesis o no

  abc(defg(ijk))(l(mn)op)qr ⟹ SI

  abc(def))ghij(kl)m ⟹ NO
- Ejemplo3: Reconocimiento del Lenguaje,

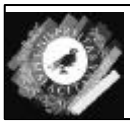
  L={W$W´ / W es una cadena de caracteres y W´es su inversa} (Suponemos que $ no está ni en W ni en W´)





# Pilas. Ejemplo1

```
Algoritmo Inverso
Tipos
      TipoElemento = C
Variables
       TipoElemento c
       CPila pila // Se llama automáticamente al constructor
Inicio
      Leer(c)
      MIENTRAS c != CHR(13)HACER
            pila.Apilar(c)
            Leer(c)
      FINMIENTRAS
      MIENTRAS NO (pila.EstáVacía()) HACER
            c = pila.Cima()
            pila.Desapilar()
            Escribir(c)
      FINMIENTRAS
      pila.Destruir()
Fin
```



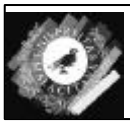
# Pilas. Ejemplo2

```
Algoritmo Balanceo
Tipos
   TipoElemento = C
Variables
    TipoElemento c
    CPila pila
    B bien
Inicio
   bien = VERDADERO
   Leer(c)
   MIENTRAS  (bien  Y  (c!=CHR(13)))
   HACER
     SI c== '(' ENTONCES
        pila.Apilar(c)
     SINO
     SI c == ')' ENTONCES
       SI (!pila.EstáVacía()) ENTONCES
             pila.Desapilar()
       SINO
          bien = FALSO
       FINSI
     FINSI
   FINSI
   Leer(c)
FINMIENTRAS
SI bien Y pila.EstáVacía() ENTONCES
   Escribir("cadena balanceada ")
SINO
   Escribir("cadena no balanceada")
FINSI
pila.Destruir()
Fin
```



# Pilas. Ejemplo3

```
Algoritmo Lenguaje_L
Tipos
   TipoElemento = $
Variables
    TipoElemento c1, c2
    CPila pila
    B bien
Inicio
  Leer(c1)
  MIENTRAS (c1 != '$') HACER
        pila.Apilar(c1)
        Leer(c1)
  FINMIENTRAS
  Leer(c1)
  bien = VERDADERO
  MIENTRAS (bien AND
           (c1 <> CHR(13))) HACER
   SI pila.EstáVacía()ENTONCES
       bien= FALSO
   SINO
       c2 = pila.Cima()
       pila.Desapilar()
       SI (c1 != c2) ENTONCES
         bien = FALSE
       SINO
         Leer(c1)
       FINSI
   FINSI
FINMIENTRAS
SI (bien AND pila.EstáVacía())ENTONCES
   Escribir (" Si pertenece")
SINO
   Escribir ("No pertenece")
FINSI
pila Destruir()
Fin
```



# Pilas. Aplicaciones

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

  Operadores: +, -, *, /

  Operandos: Letras mayúsculas
- **Notación Infija:**
- El operador binario está situado entre sus dos operandos ⇨ A+ B
- Inconveniente: Son necesarias reglas de precedencia y uso de paréntesis para evitar ambigüedades ⇨ A+B*C



# Pilas. Aplicaciones

**Notación Prefija**

- El operador binario esta situado justo antes de sus dos operandos ⇨ +AB
- Gramática:

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

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

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

- Ejemplos:

  A+(B*C) ⇨ +A*BC

  (A+B)*C ⇨ *+ABC

**Notación Postfija**

- El operador binario está situado justo después de sus dos operandos ⇨ AB+
- Gramática:

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

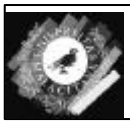
<letra> ::=A| B ....|Z

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

- Ejemplos:

  A+(B*C) ⇨ ABC*+

  (A+B)*C ⇨ AB+C*



# Pilas. Aplicaciones

- Ventaja: Usando expresiones prefijas y postfijas no son necesarias reglas de precedencia, ni uso de paréntesis.

  Las gramáticas que las generan son muy simples, y los algoritmos que las reconocen y evalúan muy fáciles
- Ejemplo 4: Algoritmo que evalúa una expresión en notación Postfija

  1)Usaremos una pila

  2)La expresión postfija se almacenará en un array y será correcta

  3)Los operandores serán: +, -, * y /

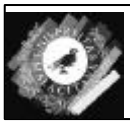
  4)Los operandos serán letras mayúsculas (a cada una le podemos asignar un valor)



# Pilas. Ejemplo4

**Tipos**
C TipoArray[1..20]
**Z** TipoElemento

```
Algoritmo B Operando(C c)
Inicio
   DEVOLVER (c=='+' O c=='*' O c=='/' O c=='-');
Fin Operando
Algoritmo Operador(c:$):Z
Inicio
   CASO  c SEA
      'A' : DEVOLVER(5)
      'B' : DEVOLVER(7)
      'C' : DEVOLVER(-1)
      'D' : DEVOLVER(11)
      SINO
            DEVOLVER 0
      FINCASO
Fin Operando
```



# Pilas. Ejemplo4

```
Algoritmo Z Postfija(E TipoArray exp,
                     E Z ultimo)
Variables
   CPila pila
   Z i, op1, op2, result
   C c
Inicio
   PARA i = 1 HASTA ultimo HACER
      c = exp[i]
      SI Operador(c) ENTONCES
                  op2 = pila.Cima()
            pila.Desapilar()
            op1 = pila.Cima()
            pila.Desapilar()
      CASO  c SEA
      '+' : pila.Apilar(op1+op2)
      '-' : pila.Apilar(op1-op2)
      '*' : pila.Apilar(op1*op2)
      '/' : pila.Apilar(op1/op2)
      FINCASO
    SINO
      pila.Apilar(Operando(c))
    FINSI
  FINPARA
  result = pila.Cima()
  pila.Destruir()
  DEVOLVER result
Fin
```



# Pilas. Implementación

**Implementación**

**1) Con un Array**

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

**Constantes**

```
MaxPila=100
```

**Tipos**

```
Z TipoElemento
TipoElemento TDatos[1..MaxPila]
```

# Pilas. Implementación

Sólo es posible acceder a la Cabeza de la Pila

⇩

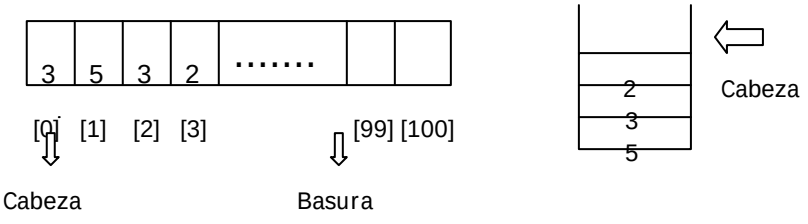
¿ Cómo es posible conocer la posición de la cabeza?

1) Variable entera "cabeza" ⇨Inconveniente: se ha de pasar como parámetro adicional a todas las operaciones sobre la pila
2) Extender el array, en pila[0] almacenaremos el índice del elemento que ocupa la cabeza actual



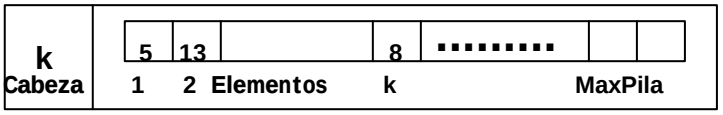
# Pilas. Implementación

```
CONSTANTES
  Cabeza=0; MaxPila=100;
TIPOS
  TipoElemento TDatos[Cabeza..MaxPila]
```



# Pilas. Implementación

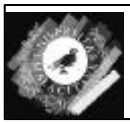
- Inconveniente: Solo es posible implementar una pila de ordinales (no de cualquier otro tipo de datos)
- 3) Solución: Registro = cabeza + array de datos






# Pilas.Implementación

```
IMPLEMENTACION CLASE CPila
   CONSTANTES
      MAXPILA = // Valor adecuado
   ATRIBUTOS
      Z cabeza
      TipoElemento datos[1..MAXPILA]
MÉTODOS
      Crear ()
      INICIO
        cabeza = 0
      FIN Crear
      Destruir()
      INICIO
          cabeza=0
      FIN Destruir
```



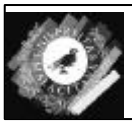
# Pilas.Implementación

```
B EstáVacia ()
INICIO
  DEVOLVER (cabeza == 0)
FIN   EstáVacia

B EstáLlena()
INICIO
  DEVOLVER (cabeza == MAXPILA)
FIN EstáLlena
```



# Pilas.Implementación

```
Apilar(E TipoElemento elem)
INICIO // precondición: la pila no ha de estar
llena
  cabeza = cabeza + 1
  datos[cabeza] = elem
FIN Apilar

Desapilar()
INICIO // precondición: la pila no ha de estar
vacía
  cabeza = cabeza - 1
FIN Desapilar

TipoElemento Cima()
INICIO // precondición: la pila no ha de estar
vacía
  DEVOLVER datos[cabeza]
FIN Cima
FIN CPila
```



# Pilas.Implementación

```
// Sin precondición. Mete un elemento de la pila si no
está llena
Apilar(E TipoElemento elem;S B llena)
INICIO
   llena = EstáLlena()
   SI NOT llena ENTONCES
      cabeza = cabeza + 1
      datos[cabeza] =  elem
   FINSI
FIN Apilar
```



# Pilas.Implementación

```
// Sin precondición. Saca un elemento de la pila si no
está vacía
Desapilar(S B vacia)
INICIO
   vacia = EstáVacia()
   SI NOT vacia ENTONCES
      cabeza = cabeza – 1
   FINSI
FIN Desapilar
```

# Pilas. Implementación

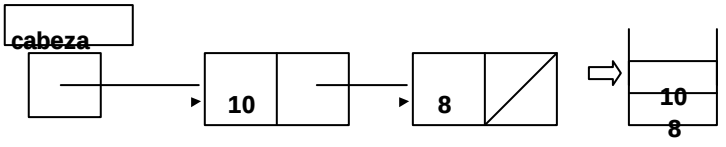
```
// Sin precondición. Mira el 1er elemento de la pila si no
está vacía
TipoElemento Cima(S B vacia)
VAR
   TipoElemento elem
INICIO
   vacia = PilaVacia()
   SI NOT vacia ENTONCES
     elem=datos[cabeza]
   FINSI
// Sin precondición. Devuelve el elemento de la cima de la
// pila si no está vacía. Si está vacía devuelve un valor
// basura
    devolver elem;
FIN Cima
```

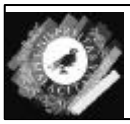


# Pilas.Implementación

**2) Con una Lista Enlazada de Punteros**

- Comienzo de una lista enlazada ⇨ Cabeza de la Pila
- Se inserta y se extrae por el inicio de la lista





# Pilas. Implementación

```
IMPLEMENTACION CLASE CPila
  TIPOS
    REGISTRO NodoPila
        TipoElemento dato
        NodoPila *sig
    FINREGISTRO
  ATRIBUTOS
       NodoPila *cabeza
  METODOS
    Crear ()
    INICIO
      cabeza = NULO
    FIN Crear
    B EstáVacia ()
    INICIO
      DEVOLVER (cabeza == NULO)
    Fin EstáVacía
```



# Pilas.Implementación

```
Apilar (E TipoElemento elem)
VAR
  NodoPila *nuevonodo
INICIO
  Asignar(nuevonodo)
  nuevonodo->dato = elem
  nuevonodo->sig = cabeza
  cabeza = nuevonodo
FIN Apilar
```

# Pilas.Implementación

Suponiendo que la pila tiene al menos un elemento

```
Desapilar ()
VAR
   NodoPila *ptr
INICIO
   ptr = cabeza
   cabeza = cabeza->sig
   Liberar(ptr)
FIN Desapilar

TipoElemento Cima ()
INICIO
   DEVOLVER cabeza->dato
FIN Cima
```



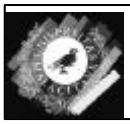
# Pilas. Implementación

Controlando la posibilidad de que la pila este vacía

Los métodos de la misma clase se pueden llamar como:
Método() o este.Metodo()

```
Desapilar (S B vacia)
VAR
    NodoPila *ptr
INICIO
   vacia = EstáVacia()
   SI NO vacia ENTONCES
     ptr = cabeza
     cabeza = cabeza->sig
     Liberar(ptr)
   FINSI
FIN Desapilar
```

```
TipoElemento Cima (S B vacia)
VAR
   TipoElemento elem
INICIO
   vacia = EstáVacía()
   SI NO vacia ENTONCES
     elem = cabeza->dato
   FINSI
   DEVOLVER elem
FIN Cima
```



# Pilas.Implementación

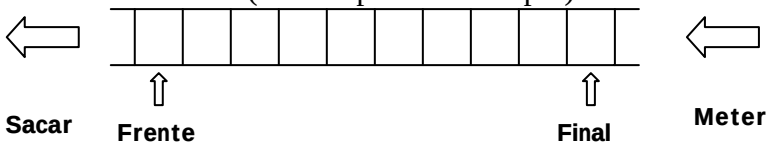
```
Destruir ()
VAR
  NodoPila *ptr
INICIO
  MIENTRAS(cabeza!=NULO) HACER
    ptr=cabeza
    pila=cabeza->sig
    Liberar(ptr)
  FINMIENTRAS
FIN Destruir
```

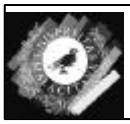


# Colas

## Definición

- Cola: es un grupo ordenado (con respecto al tiempo que llevan en él) de elementos homogéneos (todos del mismo Tipo)
- Acceso: los elementos se añaden por un extremo (final) y se sacan por el otro extremo (frente)
- Estructura **FIFO** (**F**irst **I**nput **F**irst **O**utput)





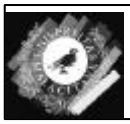
# Colas. Interfaz

```
INTERFAZ CLASE CCola
  TIPOS
    TipoElemento=//cualquier tipo de datos
  MÉTODOS
    Crear()
      // Crea una cola vacía
    B EstáVacía()
      //Devuelve VERDADERO si la cola no contiene
      //elementos. FALSO en caso contrario.
```



# Colas. Operaciones

```
    B EstáLlena()
    // Devuelve VERDADERO si la cola no permite insertar
    // más elementos. FALSO en caso contrario.
    Encolar(E TipoElemento elem)
    // Introduce un elemento al final de la cola
    Desencolar()
    //Saca el elemento del frente de la cola
    TipoElemento Frente()
    // Devuelve el elemento del frente de la cola
    Destruir()
    // Destruye una cola previamente creada
  FIN CCOLA
```



# Colas. Aplicaciones

**Ejemplo: Reconocimiento del lenguaje.**
**L={W$W/W no contiene a $}**

```
B ALGORITMO Lenguaje_L(E C cad[]
IMPORTA CCola

TIPOS
  TipoElemento = C // caracteres
VARIABLES
  B bien = VERDADERO
  TipoElemento el
  CCola cola
  N ind
INICIO
  ind = 1
  MIENTRAS (cad[ind] <> '$') Y (cad[ind] <> '\n') HACER
    cola.Encolar(cad[ind])
    INC(ind)
  FINMIENTRAS
```



# Colas. Aplicaciones

```
  SI cad[ind] <> '\n' ENTONCES
    INC(ind)
    MIENTRAS (bien Y (c1<>'\n') HACER
      SI cola.EstaVacia() ENTONCES
        bien = FALSO
      SI NO
        SI (cad[ind] <> cola.Frente()) ENTONCES
          bien = FALSO
        SI NO
          cola.Desencolar()
          INC(ind)
        FINSI
      FINSI
    FINMIENTRAS
  SI NO
    bien = FALSO
  FINSI
FIN Lenguaje_L
```

# Colas. Implementación

## Implementación

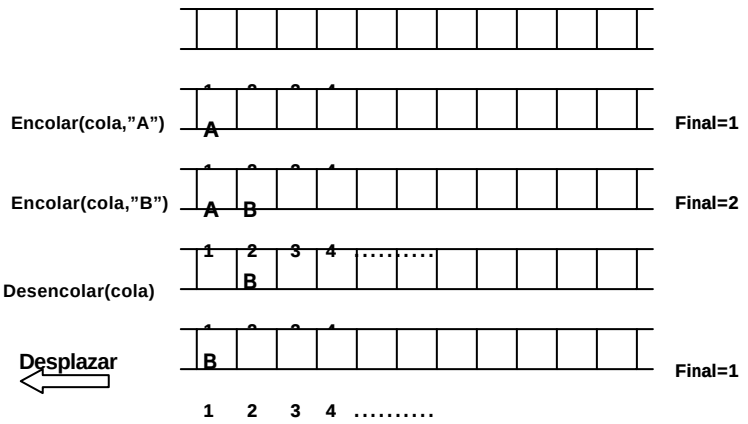
1) Con un Array

- Se deja fijo el frente de la cola y se mueve el final a medida que se añaden nuevos elementos (Idem Pila)
- Las operaciones Encolar, Frente, EstáVacía y EstáLlena se implementan de una forma similar a sus análogas en Pilas
- La operación de Desencolar es más complicada: cada vez que saquemos un elemento de la cola se han de desplazar el resto una posición en el array, para hacer coincidir el frente con la primera posición del array
- Ventaja ⇒ Simplicidad
- Inconveniente ⇒ Ineficiente (colas con muchos elementos o elementos grandes)



# Colas. Implementación

- Ejemplo:



# Colas. Implementación

Solución:

- Utilizar un índice para el frente y otro para el final y permitir que ambos fluctúen por el array

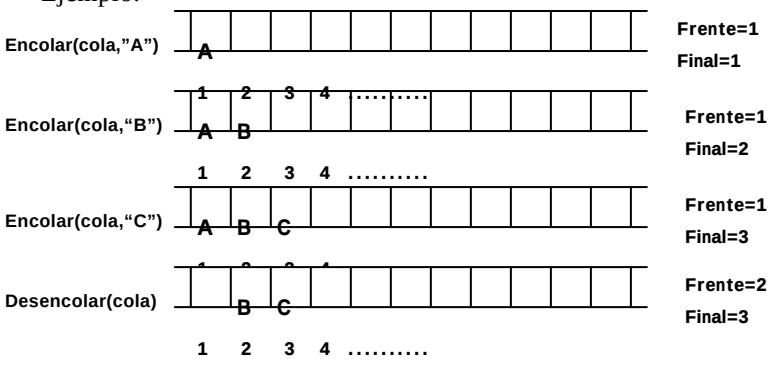
⇩

- Ventaja: operación Desencolar más sencilla
- Inconveniente: Es posible que final sea igual a MAXCOLA (última casilla del array) y que la cola no esté llena



# Colas. Implementación

- Ejemplo:



# Colas. Implementación

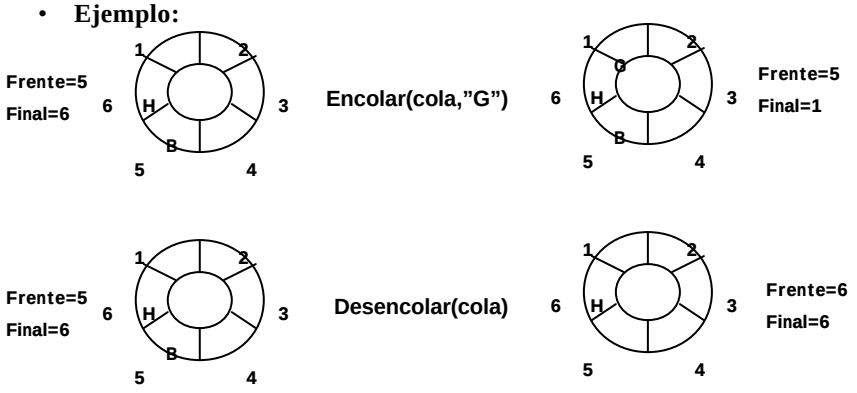
Solución:

- 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 array y no esté llena
- Operación Encolar ⇨Añade elementos a las posiciones del array e incrementa el índice final
- Operación Desencolar ⇨Más sencilla. Sólo se incrementa el índice frente a la siguiente posición



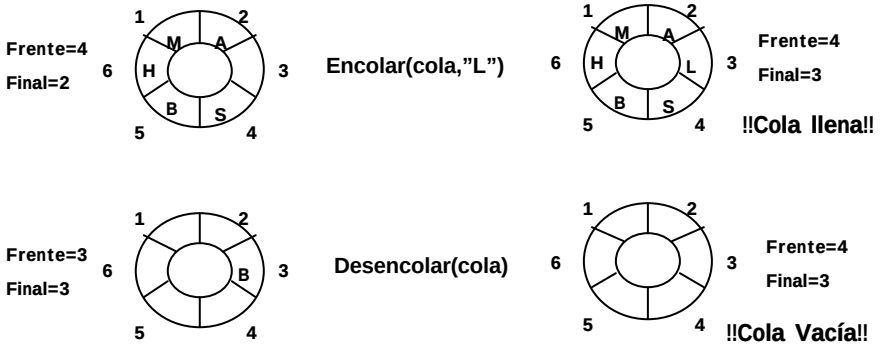
# Colas. Implementación

- **Ejemplo:**



# Colas. Implementación

¿Cómo sabemos si la cola está vacía o llena?






# Colas. Implementación

Solución:

1) Disponer de otra variable. Contabilizará los elementos almacenados en la cola.

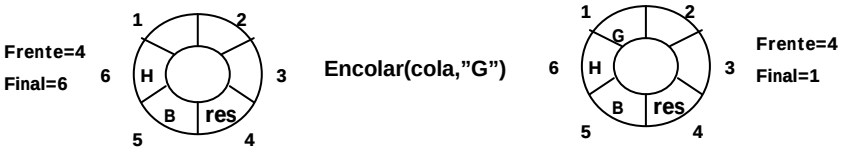
   Variable=0 ⇨ Cola vacía

   Variable=MaxCola ⇨ Cola llena

2) Frente apunte a la casilla del array que precede a la del elemento frente de la cola

# Colas. Implementación

Con la segunda solución es necesario que la posición a la que apunta `frente` esté Reservada

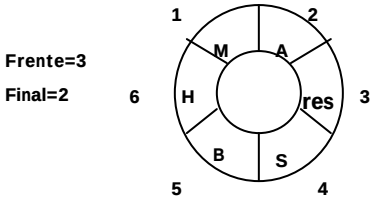


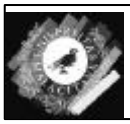



# Colas. Implementación

¿Cómo saber si la cola está llena?

Cola Llena: Frente == Final + 1

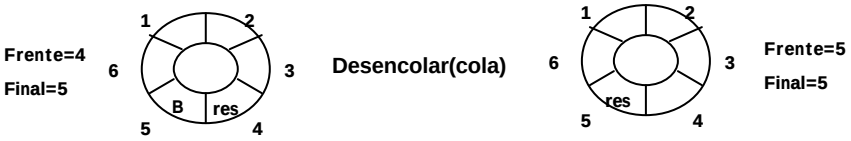




# Colas. Implementación

¿Cómo saber si la cola está vacía?

Cola Vacía: Frente == Final

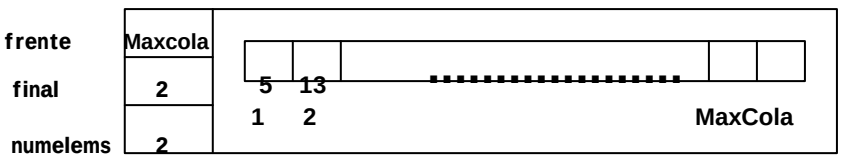


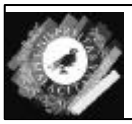

- Crear la cola (inicializarla vacía): frente =Maxcola (índice del array que precede al elemento frente de la cola) y final= Maxcola $\Longrightarrow$ Cola Vacía correcto (frente == final)



# Colas. Implementación

- Definiremos como atributos los índices `frente` y `final` y el número de elementos, junto con el array que contendrá los elementos de la cola





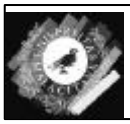
# Colas.Implementación

```
IMPLEMENTACIÓN CLASE CCola
  CONSTANTES
    MAXCOLA = 100
  ATRIBUTOS
    TipoElemento elementos[1..MAXCOLA]
    N NumElems
    N Principio
    N Final
```



# Colas. Implementación

```
MÉTODOS
  Crear ()
  INICIO
    NumElems = 0
    Principio = 1
    Final = 0
  FIN Crear

  Destruir()
  INICIO

  FIN Destruir
```



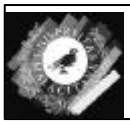
# Colas. Implementación

```
B EstáVacia ()
INICIO
  DEVOLVER NumElems == 0
FIN EstáVacia


B EstáLlena ()
INICIO
  DEVOLVER NumElems == MAXCOLA
FIN EstáLlena
```



# Colas. Implementación

```
Encolar (E TipoElemento Elem)
INICIO
  SI NO EstáLLena() ENTONCES
    INC(NumElems)
    Final = (Final MOD MAXCOLA) + 1
    elementos[Final] = elem
  FINSI
Fin Encolar

Desencolar ()
INICIO
  SI NOT EstáVacía() ENTONCES
    DEC(NumElems)
    Principio = (Principio MOD MAXCOLA) + 1
  FINSI
FIN Desencolar
```



# Colas. Implementación

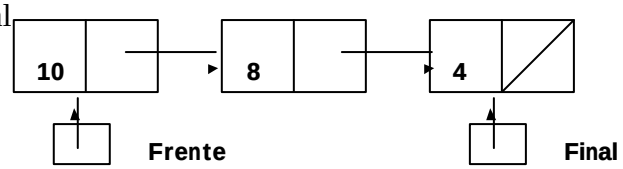
```
    TipoElemento Frente ()
    VARIABLES
      TipoElemento elem
    INICIO
      SI NO EstaVacia() ENTONCES
        elem = elementos[Principio]
      FINSI
      DEVOLVER elem
    FIN Frente
FIN CCOLA
```



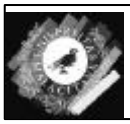
# Colas. Implementación

2) Con listas enlazadas con Punteros

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




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



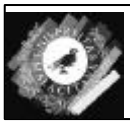
# Colas. Implementación

```
IMPLEMENTACIÓN CLASE CCola
  TIPOS
    NodoCola *TipoCola
    REGISTRO NodoCola
      TipoElemento valor
      TipoCola sig
    FINREGISTRO

  ATRIBUTOS
    TipoCola Principio
    TipoCola Final
```



# Colas. Implementación

```
MÉTODOS
  Crear ()
  INICIO
    Principio = NULO
    Final  = NULO
  FIN Crear

  B EstáVacía()
  INICIO
    DEVOLVER Principio == NULO
  FIN EstáVacía
```



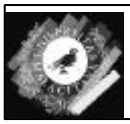
# Colas.Implementación

```
Destruir ()
VARIABLES
  TipoCola nodo, siguiente
INICIO
  siguiente = Principio
  MIENTRAS siguiente <> NULO HACER
    nodo = siguiente
    siguiente = siguiente->sig
    LIBERAR(nodo)
  FINMIENTRAS
  Principio = NULO
  Final = NULO
FIN Destruir
```



# Colas. Implementación

```
Encolar(E TipoElemento elem)
VARIABLES
     TipoPuntero nuevonodo
INICIO
  ASIGNAR(nuevonodo)
  nuevonodo->valor = elem
  nuevonodo->sig = NULO
  SI EstáVacia() ENTONCES
    Principio = nuevonodo
  EN OTRO CASO
    Final->sig = nuevonodo
  FINSI
  Final = nuevonodo
FIN Encolar
```



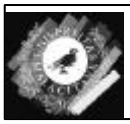
# Colas.Implementación

```
Desencolar ()
VARIABLES
  TipoPuntero temp
INICIO
  SI NO EstaVacía() ENTONCES
    temp = Principio
    Principio = Principio ->sig
    SI Principio == NULO ENTONCES
      Final = NULO
    FINSI
    LIBERAR(temp)
  FINSI
FIN Desencolar
```



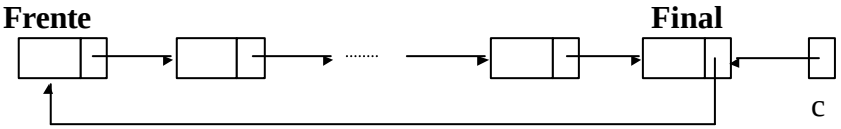
# Colas.Implementación

```
TipoElemento Frente ()
VARIABLES
  TipoElemento elem
INICIO
  SI NO EstáVacía() ENTONCES
    elem = Principio->valor
  FINSI
  DEVOLVER elem
FIN Frente
FIN Ccola
```



# Colas.Implementación

- Con una lista enlazada circular es muy fácil implementar una Cola, sin tener que disponer de un registro con dos campos para el frente y para el final.




**Implementación**