

Tema 4

TIPOS DE

DATOS ABSTRACTOS

DINÁMICOS LINEALES

Tutor: Enrique Carmona
Asignatura: Estructuras de Datos y Algoritmos

TEMA IV : TIPOS DE DATOS ABSTRACTOS DINÁMICOS LINEALES

4.1 Ejemplos de TDA dinámicos lineales

4.2 Pilas

4.2.1. Implementación mediante arreglos

4.2.2 Implementación mediante listas enlazadas

4.3 Colas

4.3.1. Implementación mediante arreglos

4.3.2 Implementación mediante listas enlazadas

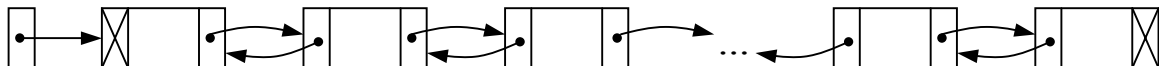
TDA's estáticas y dinámicas

Tipo TDA	Nº elementos	Reserva Memoria	Ejemplos
Estática	Fijo	En tiempo de compilación	(a) Arreglos (b) Registros
Dinámica	No predefinido	En tiempo de ejecución	(a) Lista enlazadas (b) Listas doblemente enlazadas

- A partir de la generalización del campo enlace del tipo nodo visto en las listas enlazadas, puede definirse cualquier TDA dinámico con diversos enlaces.

Listas doblemente enlazadas

- ***TDA lista doblemente enlazada.*** Colección de nodos ordenados según su posición, tal que cada uno de ellos es accedido mediante el puntero anterior del campo enlace del nodo siguiente y por el puntero siguiente del campo enlace del nodo anterior. En ella se pueden realizar operaciones de insertar, suprimir,...



➤ Ventajas/Inconvenientes respecto de las listas enlazadas

- Ventajas:
 - Permiten mayor flexibilidad en su manejo.
- Inconvenientes:
 - Requieren más memoria (un puntero más por nodo).
 - Coste de las funciones sobre ellas es mayor (doble actualizaciones de los enlaces).

Declaración de la estructura de la lista doblemente enlazada

```
TYPE
    tipo_datos= INTEGER;
    tipo_doble= POINTER TO nododoble;
    nododoble= RECORD
        datos: tipo_datos;
        sig,ant: tipo_doble
    END;
    tlista=tipo_doble;
VAR
    lista: tlista;
```

Procedimiento “inicia lista” e “inserción por el principio”

```
PROCEDURE inicia_lista(VAR l:tlista);
BEGIN
    l:=NIL;
END inicia_lista;
```

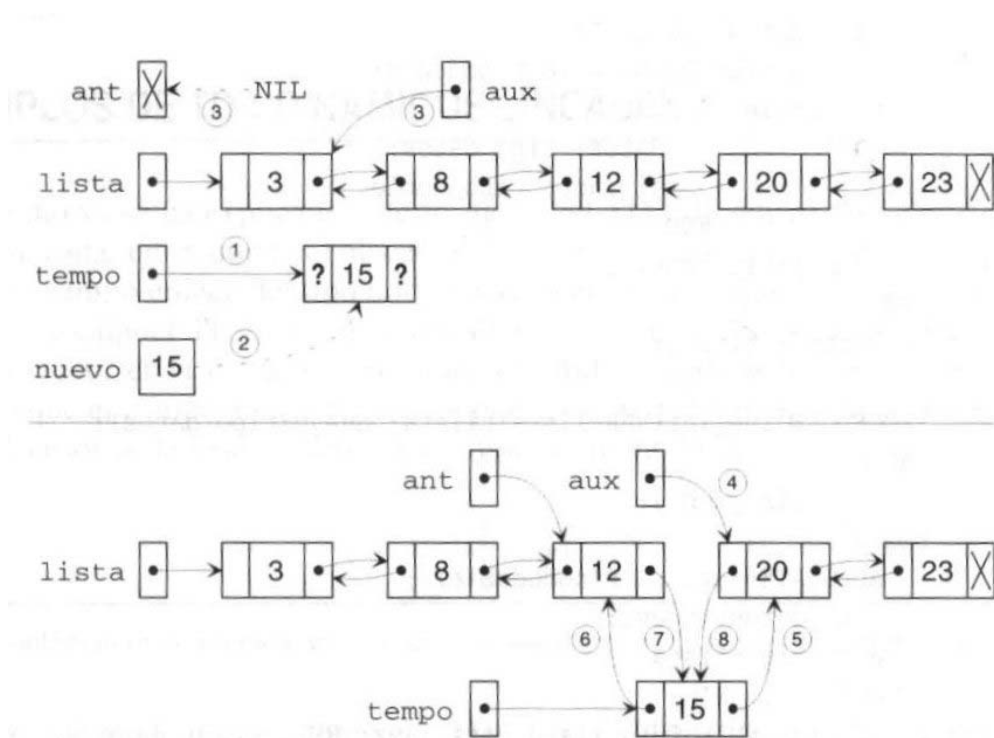
```
PROCEDURE ins_ini(VAR lista:tlista;nuevo:tipo_datos);
VAR
    aux: tipo_doble;
BEGIN
    ALLOCATE(aux, SIZE(nododoble));
    aux^.datos:=nuevo;
    aux^.sig:=lista;
    aux^.ant:=NIL;
    IF lista#NIL THEN lista^.ant:=aux; END;
    lista:=aux;
END ins_ini;
```

Procedimiento de insertar en orden

```

PROCEDURE ins_orden(VAR lista:tlista;nuevo: tipo_datos);
VAR
    aux,ant,tempo: tipo_doble;
BEGIN
    IF lista=NIL THEN
        ins_ini(lista,nuevo)
    ELSE
1      ALLOCATE(tempo,SIZE(nododoble));
2      tempo^.datos:=nuevo;
3      aux:=lista; ant:=NIL;
4      WHILE (aux#NIL) AND (nuevo>aux^.datos)DO
            ant:=aux;
            aux:=aux^.sig;
        END;
5      tempo^.sig:=aux;
6      tempo^.ant:=ant;
7      IF ant=NIL THEN lista:=tempo ELSE ant^.sig:=tempo END;
8      IF aux#NIL THEN aux^.ant:=tempo END;
    END;
END ins_orden;

```

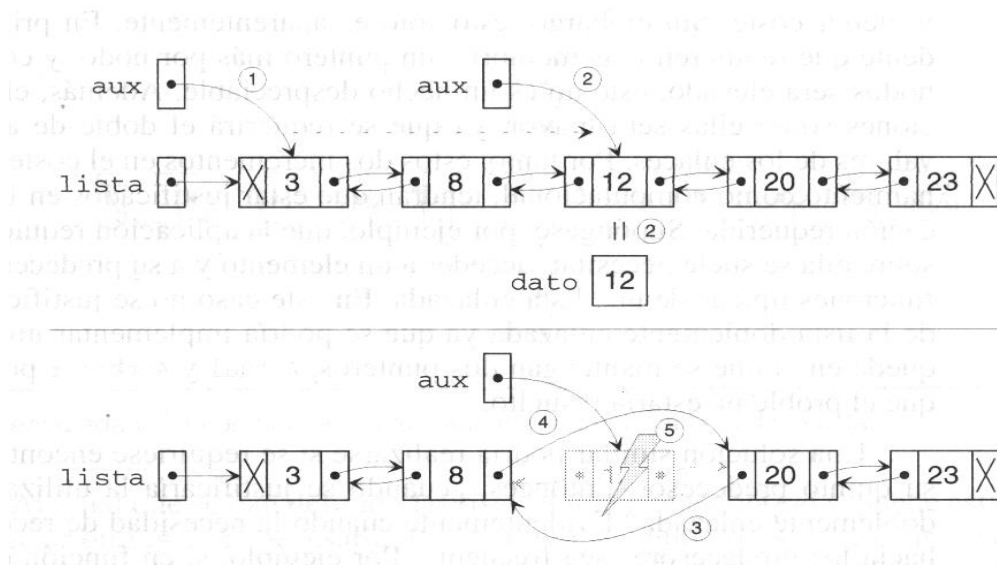


Procedimiento de eliminación de la primera aparición del elemento buscado

```

PROCEDURE sup_nodo(VAR lista:tlista;dato:tipo_datos);
VAR
  aux: tipo_doble;
BEGIN
  1 aux:=lista;
  2 WHILE (aux#NIL) & (aux^.datos#dato) DO aux:=aux^.sig; END;
  IF aux#NIL THEN (*Si existe el nodo*)
  3   IF aux^.sig#NIL THEN (*Si tiene siguiente*)
      aux^.sig^.ant:=aux^.ant;
    END;
  4   IF aux^.ant#NIL THEN (*Si tiene anterior*)
      aux^.ant^.sig:=aux^.sig
    ELSE
      lista:=aux^.sig
    END;
  5   DEALLOCATE(aux,SIZE(nododoble));
  END;
END sup_nodo;

```



Procedimiento imprimir lista enlazada en orden inverso

```
PROCEDURE Imprimir(lista: tlista);
(*Imprime lista en orden inverso*)
VAR
  aux: tipo_doble;
BEGIN
  aux:=lista;
  IF lista#NIL THEN
    WHILE aux^.sig#NIL DO
      aux:=aux^.sig
    END;
    WriteInt(aux^.datos,4); (*ultimo*)
    WHILE aux^.ant#NIL DO
      WriteInt(aux^.ant^.datos,4);
      aux:=aux^.ant
    END;
  END;
  WriteLn;
END Imprimir;
```

➤ Ejemplo en el que no es necesario recurrir a una lista doblemente enlazada:

“La aplicación requiere una lista y sobre ella se suele necesitar acceder a un elemento y a su predecesor”

Solución: Implementar una función de búsqueda en las que se mantengan dos punteros, por ejemplo, *Actual* y *Anterior*.

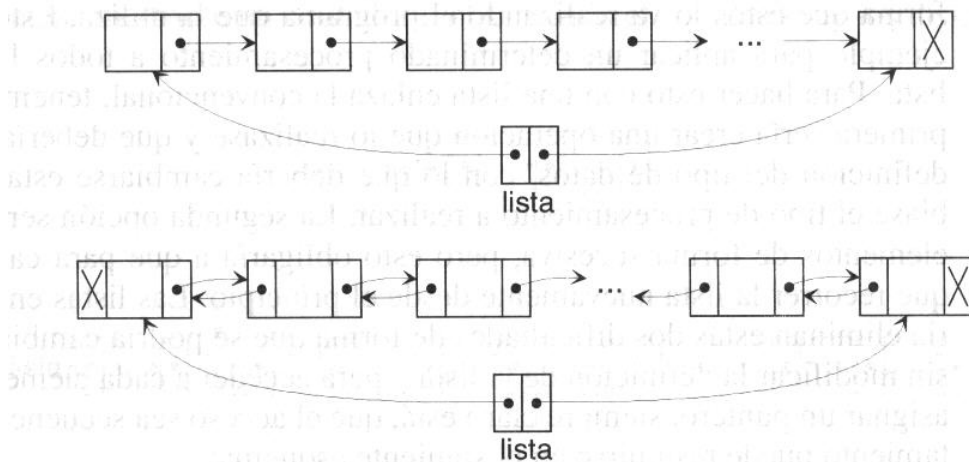
➤ **Ejemplos de uso de listas doblemente enlazadas:**

- Si en función de un valor en un nodo se necesita volver a examinar sus predecesores.

1 HOLA 1 BIEN 2 A NEUB 1 A DIOS 2 E TREUS

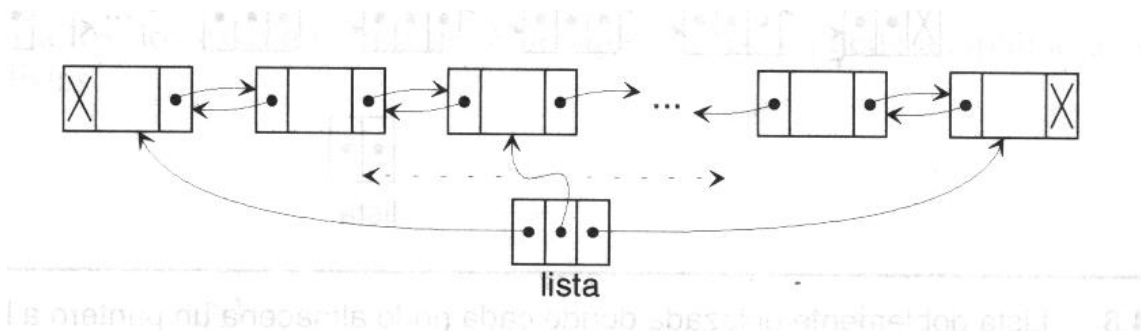
- Una lista formada por fichas de libros en la que, una vez localizado un libro, puede ser interesante analizar los libros adyacentes situados a su izquierda y a su derecha.

➤ **Listas con punteros al final de la lista (enlazadas/doblemente enlazadas):**



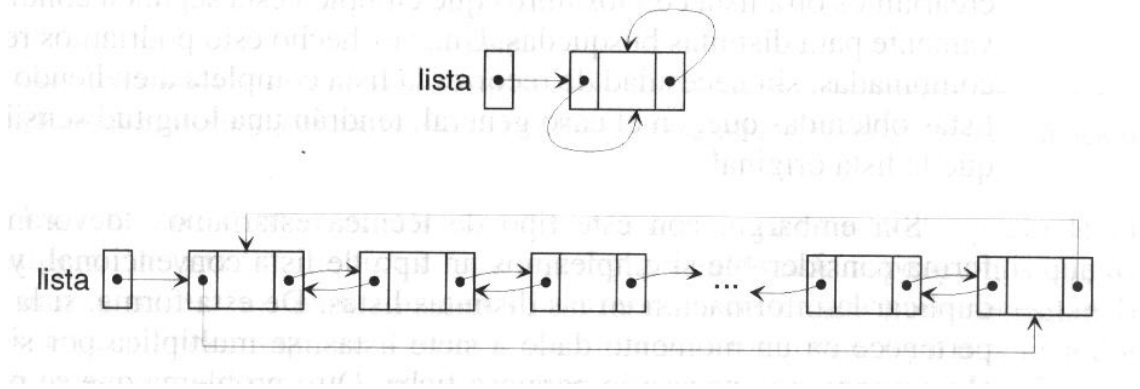
➤ **Listas con memoria (enlazadas/doblemente enlazadas):**

- Se añade un puntero adicional que apunta siempre al último elemento accedido
- Permite aplicar un procesamiento externo a todos los elementos de la lista



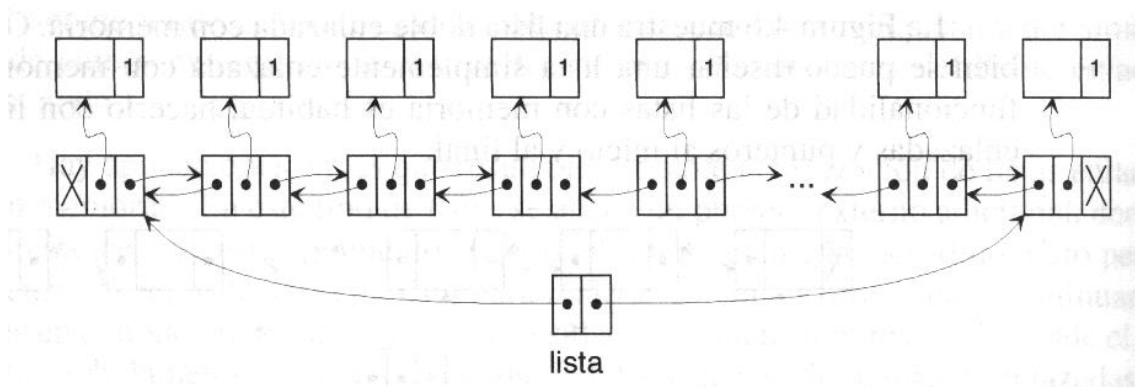
Listas circulares (enlazadas/doblemente enlazadas):

- Los recorridos que se realicen sobre la estructura deberá hacerse mediante un contador



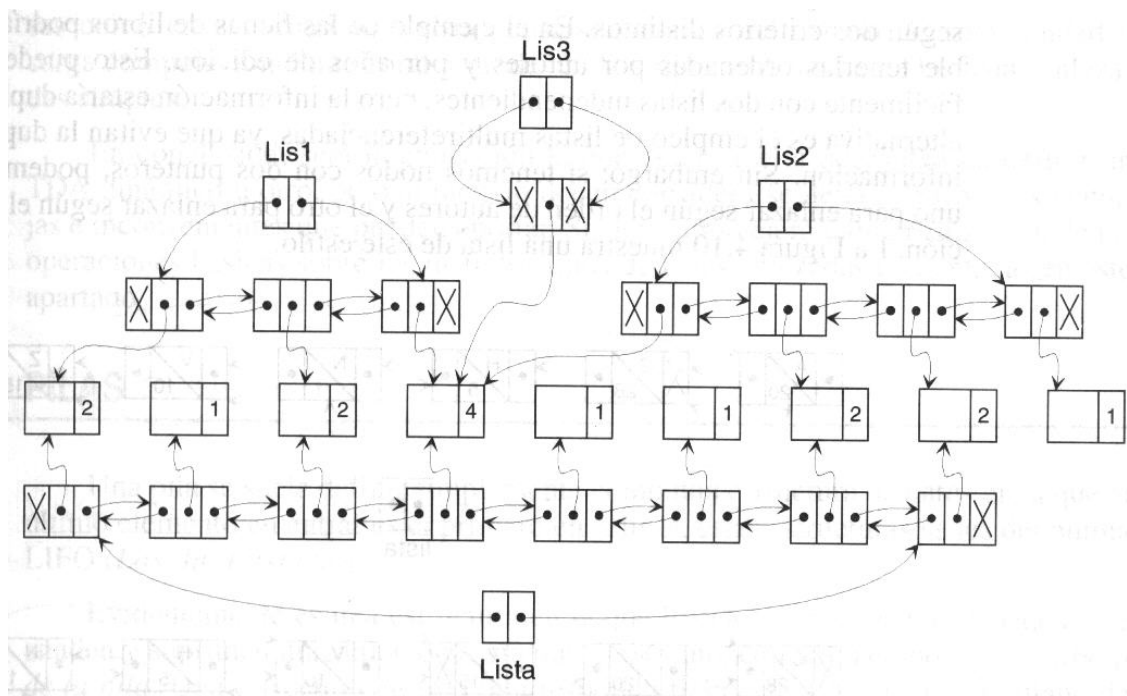
➤ Listas multireferenciadas (enlazadas/doblemente enlazadas):

- En estas listas, el nodo no contiene la información directamente sino un puntero a la misma
- **Aplicaciones:** Cuando una misma información puede pertenecer a más de una lista, por ejemplo, según distintas condiciones de búsqueda.



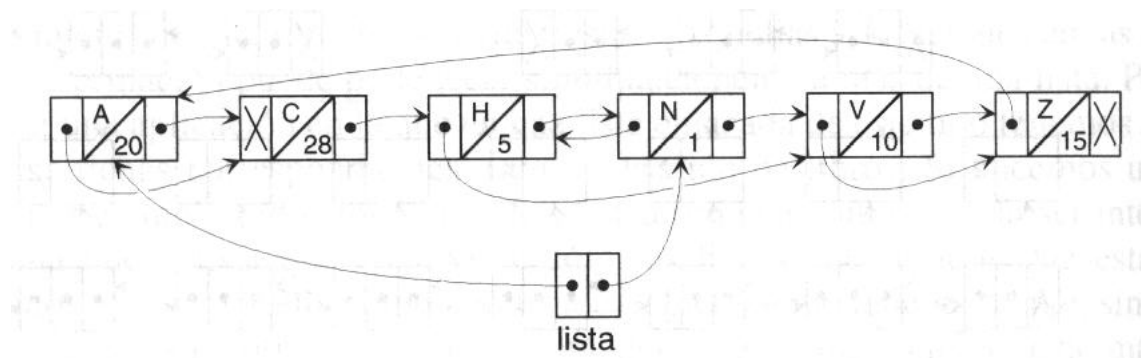
- **Ventajas:**

- (a) Obtiene listas más pequeñas que la original
 - (b) Ahorra memoria al no incluir el dato cada vez en la lista
 - (c) Evita la inconsistencia de la información
- Cada elemento de información tiene asociado un **contador** que indica el número de listas que lo referencian. Se utiliza como centinela para borrado.



➤ **Listas doblemente enlazadas no a nodos consecutivos:**

- Ejemplo: cuando una misma información, contenida en el nodo, la queremos mantener ordenada según distintos criterios



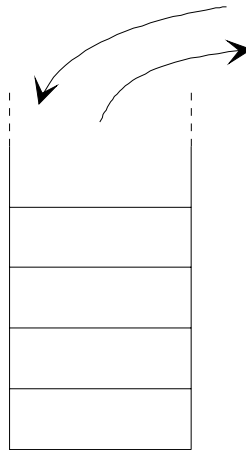
- Esta estructura puede generalizarse a cualquier número de punteros. Tantos como criterios de ordenación diferentes se quieran establecer.
- Inconveniente: Tratamiento de inserciones, búsquedas y eliminaciones bastante lento y complicado.

Conclusión: Será la aplicación la que determine la necesidad de definir un tipo de TDA dinámico u otro.

PILAS

- Una pila es un TDA dinámico homogéneo al que sólo tiene acceso por la cabeza o cima de la pila; dicho acceso es de tipo LIFO ("Last In-First Out": último elemento en entrar-primero en salir). Los operadores básicos asociados son *Introducir* y *Extraer* elementos de la pila, y los operadores auxiliares asociados son:
 - *Inicia_pila*: Crea una pila que no contiene elementos
 - *Pila_vacia*: consulta si pila está vacía
 - *Pila_Llena*: consulta si pila está llena
 - *Consultar_Pila*: consulta la cima de la pila sin extraer el elemento.

Inserción y eliminación de
elementos de una pila



- Es importante resaltar de la definición del TDA pila, que el carácter dinámico es conceptual. Este hecho es independiente de la implementación que se realice de la pila, en base a estructuras estáticas o dinámicas. En ambos casos, la pila es dinámica.

Implementación de pilas mediante arreglos

➤ Definición de la estructura Pila (Modula2):

```
CONST  MAXPILA=100;
```

```
TYPE  TipoIndice = [1..MAXPILA];
```

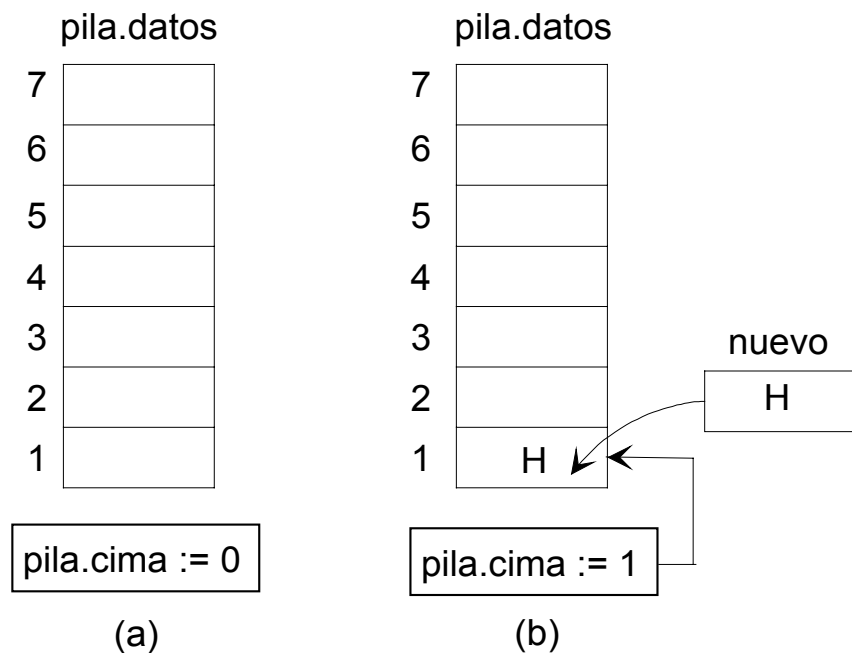
```
      TipoPila = RECORD
```

```
        datos: ARRAY[TipoIndice] OF TipoElemento;
```

```
        cima: Integer;
```

```
      END;
```

```
VAR   pila: TipoPila;
```



a) Pila vacía [Inicia_Pila() \Rightarrow pila.cima:=0]

b) Introducción de un nuevo elemento en la pila.

Operaciones con pilas

➤ *Operadores básicos:*

```
PROCEDURE Meter_pila(VAR pila:TipoPila; nuevo:TipoDatos);  
BEGIN  
    pila.cima := pila.cima+1; (*se incrementa antes de escribir*)  
    pila.datos[pila.cima] := nuevo;  
END;
```

```
PROCEDURE Sacar_pila(VAR pila:TipoPila; VAR dato:TipoDatos);  
BEGIN  
    dato:=pila.datos[pila.cima];  
    pila.cima := pila.cima-1; (*se decrementa despues de leer*)  
END;
```

➤ *Operadores auxiliares:*

```
PROCEDURE Inicia_pila(VAR pila: TipoPila)
BEGIN
    pila.cima:=0;
END;
```

```
PROCEDURE Pila_Vacia(pila: TipoPila): Boolean;
BEGIN
    RETURN pila.cima=0
END
```

```
PROCEDURE Pila_Llena(pila: TipoPila): Boolean;
BEGIN
    RETURN pila.cima=MAXPILA
END
```

```
PROCEDURE Consultar_pila(pila:TipoPila; VAR dato:TipoDato);
BEGIN
    dato:=pila.datos[pila.cima]; (*lee elemento de cima pero no elimina*)
END;
```

Implementación de pilas mediante listas enlazadas

➤ Declaración de la estructura (modula2)

```
TYPE Tipo_pila = POINTER TO Nodopila;  
    Nodopila = RECORD  
        enlace : Tipo_pila;  
        datos : Tipo_datos  
    END;
```

➤ Operadores básicos

```
PROCEDURE Meter_pila (VAR pila : Tipo_pila; nuevo_dato: Tipo_datos);  
(* Es equivalente a insertar por la cabeza en la lista enlazada *)
```

```
VAR  
    Nuevo_nodo : Tipo_pila;  
BEGIN  
    Allocate(Nuevo_nodo, SIZE(Nodopila));  
    Nuevo_nodo^.datos := nuevo_dato;  
    Nuevo_nodo^.enlace := pila;  
    pila := Nuevo_nodo  
END Meter_pila;
```

```
PROCEDURE Sacar_pila (VAR pila : Tipo_pila; VAR dato: Tipo_datos);  
(* Es equivalente a extraer por la cabeza en la lista enlazada *)
```

```
VAR  
    Aux : Tipo_pila;  
BEGIN  
    Aux := pila;  
    dato := pila^.datos;  
    pila := pila^.enlace;  
    Deallocate(Aux, SIZE(Nodopila))  
END;
```


➤ *Operadores auxiliares*

```
PROCEDURE Inicia_pila(VAR pila: Tipo_Pila)
BEGIN
    pila := NIL;
END Inicia_pila;
```

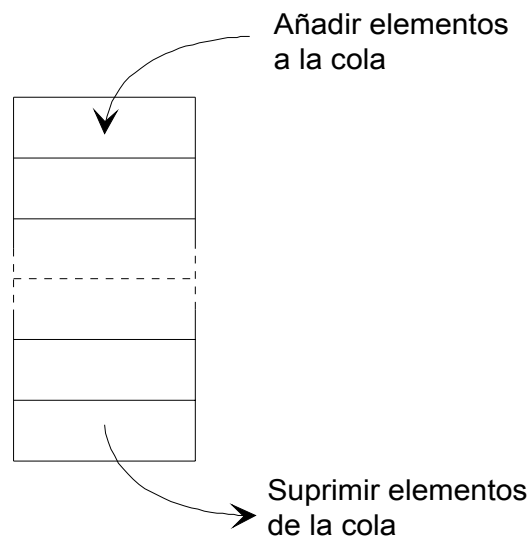
```
PROCEDURE Pila_Vacia(pila: Tipo_Pila): Boolean;
BEGIN
    RETURN pila = NIL
END Pila_Vacia;
```

```
PROCEDURE Pila_Llena(pila: Tipo_Pila): Boolean;
BEGIN
    RETURN ~Available(SIZE(nodopila));
END Pila_Vacia;
```

```
PROCEDURE Consultar_pila(pila: TipoPila; VAR dato: TipoDato);
BEGIN
    dato := pila^.datos; (*lee elemento de cima pero no elimina*)
END Consultar_pila;
```

Colas

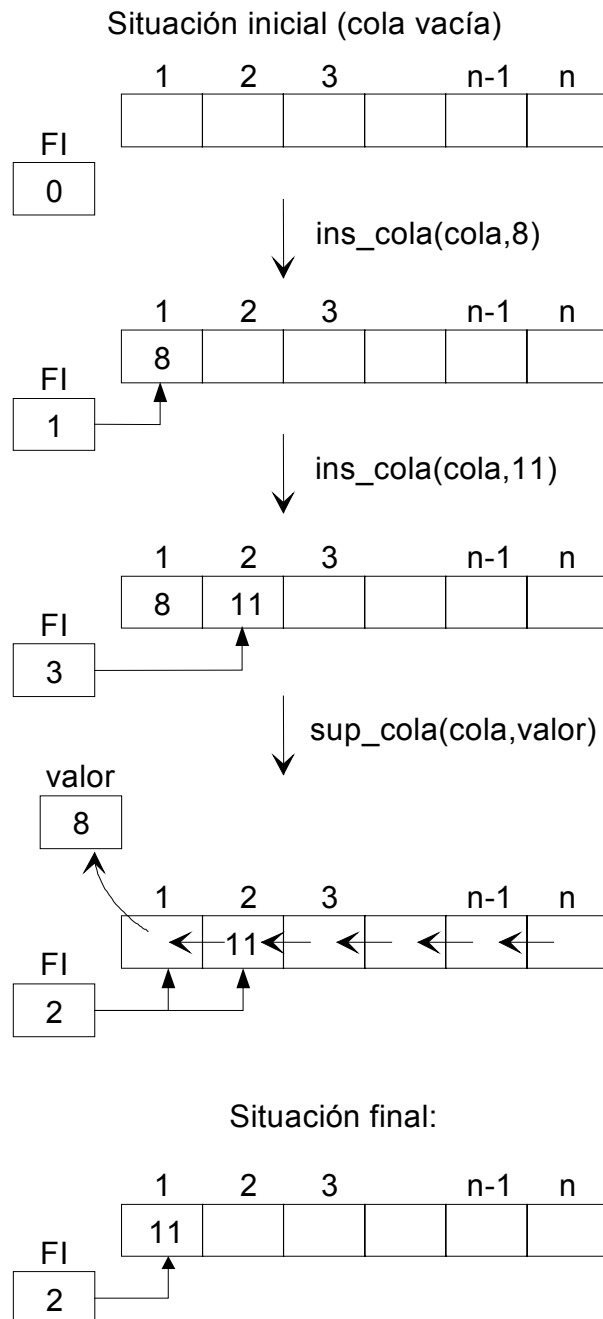
- Una cola es un TDA dinámico homogéneo al que sólo tiene acceso al principio de la cola para sacar elemento, y al final de la misma para meterlos; dicho acceso es de tipo FIFO ("First In-First Out": primer elemento en entrar-primero en salir). Los operadores básicos asociados son *Introducir* y *Extraer* elementos de la cola, y los operadores auxiliares asociados son:
- *Inicia_cola*: Crea una cola que no contiene elementos
 - *Cola_vacia*: consulta si cola está vacía
 - *Cola_Llena*: consulta si cola está llena
 - *Consultar_Cola*: consulta el primer elemento de la cola sin extraer el elemento.



- Como ocurriría con las pilas, el carácter dinámico de las colas es conceptual. Este hecho es independiente de la implementación que se realice de la cola, en base a estructuras estáticas o dinámicas. En ambos casos, la cola es dinámica.

Implementación de colas mediante arreglos

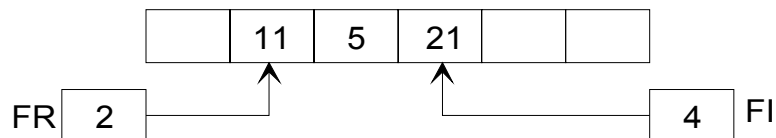
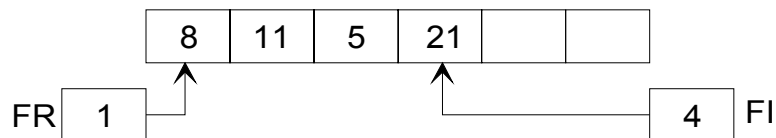
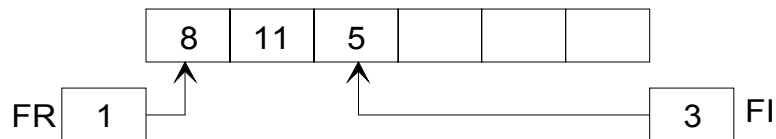
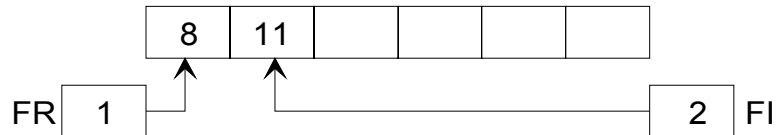
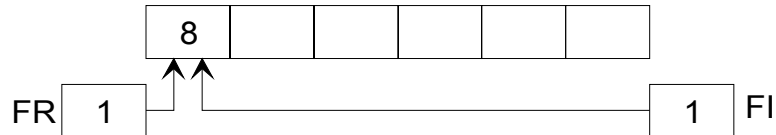
➤ Con un único índice: (FI = FInal de cola)



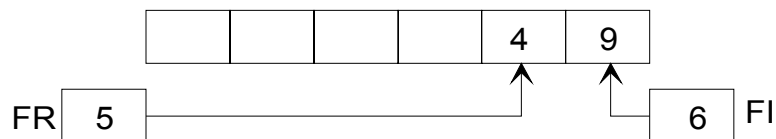
Inconveniente: al sacar un elemento de la cola (primera posición del arreglo) hay que desplazar el resto de los elementos.

➤ **Con dos índices y arreglo lineal:**

- FI (FInal de cola): apunta al último elemento introducido y aumenta al introducir un elemento.
- FR (FRente de cola): apunta al primer elemento a sacar y aumenta al eliminar un elemento.

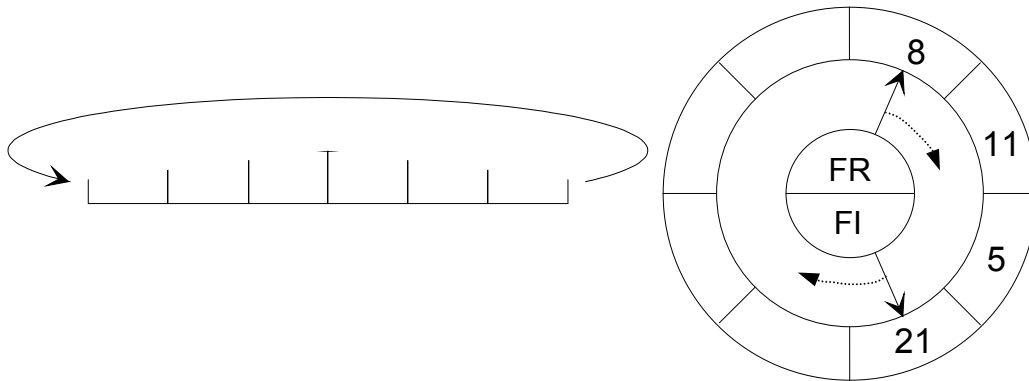


Tras varias operaciones:



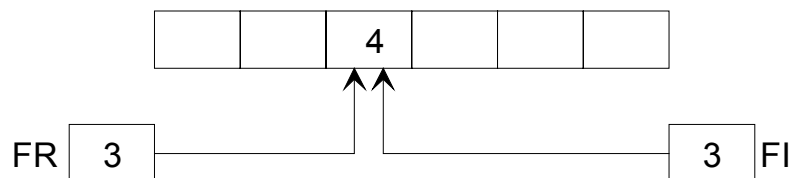
Inconveniente: al final hay muchos elementos libres pero, según el criterio de índices dado, no se puede seguir insertando.

➤ **Con dos índices y arreglo *circular*:**

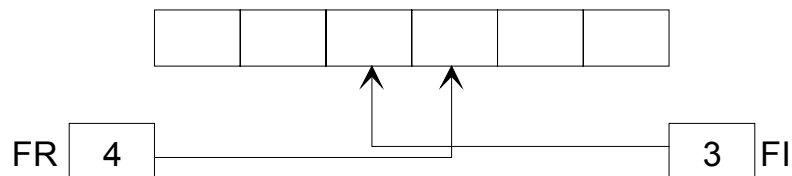


Problema: distinguir entre *cola llena* y *cola vacía*.

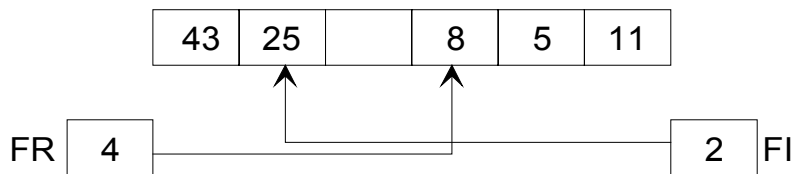
- La cola está vacía con $FR=4$ y $FI=3$:



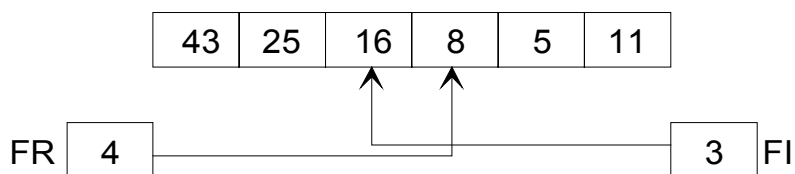
Al suprimir FR aumenta:



- La cola está llena con $FR=4$ y $FI=3$:



Al insertar FI aumenta:



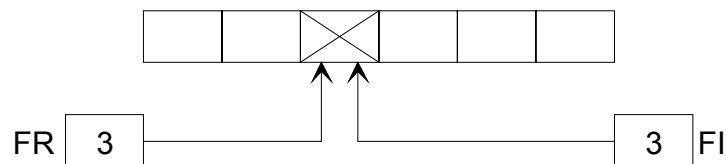
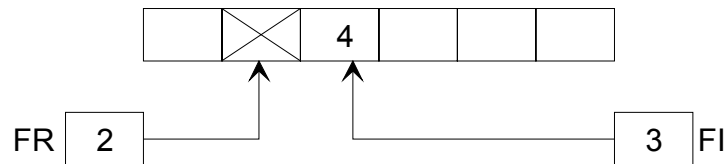
Solución: dejar una posición del arreglo libre

- FR: apunta a un elemento que nunca se rellena y que es el anterior al primero de la cola.
- FI: apunta al último elemento introducido.

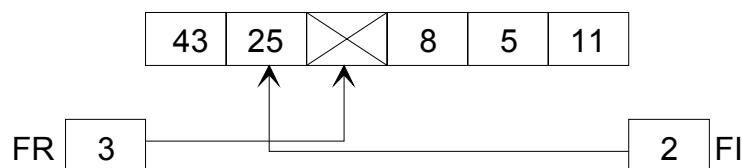
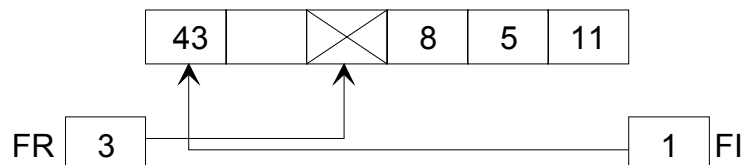
Condición de vacía: $FR=FI$

Condición de llena: $FR=FI+1$ o $(FI=MAX \text{ y } FR=1)$

- La cola está vacía con $FR=3$ y $FI=3$:



- La cola está llena con $FR=3$ y $FI=2$:



➤ Declaración de la estructura cola (Modula2):

```
CONST MAXCOLAN=100;
      MAXCOLA=MAXCOLAN+1;
TYPE  TipoIndice = [1..MAXCOLA];
      TipoCola = RECORD
          datos: ARRAY[TipoIndice] OF TipoElemento;
          FR, FI: TipoIndice;
          END;
VAR    cola: TipoCola;
```

➤ Operaciones básicas con colas:

```
PROCEDURE Meter_Cola(VAR cola:TipoCola; nuevo_dato:TipoDatos);
BEGIN
    IF cola.FI = MAXCOLA THEN  cola.FI := 1;
    ELSE cola.FI := cola.FI +1
    END (*primero incrementa y después escribe*)
    cola.datos[cola.FI] := nuevo_dato;
END
```

```
PROCEDURE Sacar_Cola(VAR cola:TipoCola; VAR dato:TipoElemento);
BEGIN
    IF cola.FR = MAXCOLA THEN  cola.FR := 1
    ELSE cola.FR := cola.FR +1;
    END (*primero incrementa y después lee*)
    dato := cola.datos[cola.FR]
END
```

➤ Operadores auxiliares:

```
PROCEDURE Inicia_Cola (VAR cola: TipoCola);  
  BEGIN  
    Cola.FR:= MAXCOLA;  
    Cola.FI:= MAXCOLA  
  END
```

```
PROCEDURE Cola_Vacia(cola: TipoCola): Boolean;  
  BEGIN  
    RETURN cola.FI = cola.FR  
  END
```

```
PROCEDURE Cola_Llena(cola: TipoCola): Boolean;  
  BEGIN  
    IF cola.FI = MAXCOLA RETURN cola.fr = 1  
    ELSE RETURN cola.fr = cola.fí + 1 END  
  END
```

```
PROCEDURE Consultar_final_cola(cola:Tipocola; VAR dato:TipoDato);  
  BEGIN  
    dato:=cola.datos[cola.FI]; (*lee elemento del final pero no elimina*)  
  END;
```

```
PROCEDURE Consultar_frente_cola(cola:Tipocola; VAR dato:TipoDato);  
  BEGIN  
    IF cola.FR = MAXCOLA THEN dato:=cola.datos[1];  
    ELSE dato:=cola.datos[cola.FR+1]; END;  
  END;
```

➤ **Nota:** Las condiciones de cola vacía y cola llena también se satisfacen con el siguiente criterio:

- **FR:** apunta al primer elemento de la cola.
- **FI:** apunta al elemento siguiente al último introducido.

➤ Implementación dinámica de colas

- En principio se resolvería mediante operaciones análogas a insertar por el final y suprimir por el principio (el frente de la cola sería el principio de la lista y el final de la cola sería el final de la lista). También cabe la situación inversa.

- **Problema:**

La inserción o la supresión por el final requiere un bucle hasta encontrarlo \Rightarrow N° de comparaciones de punteros es igual a la cantidad de elementos que tenga la cola. **Solución:** Utilizar un puntero externo al principio y otro al final. **Limitación:** la inserción en la cola ha de hacerse por el final de la lista y la extracción por el principio.

➤ Definición de la estructura dinámica (Modula2):

```
TYPE Ptr_nodoCola = POINTER TO NodoCola;  
  NodoCola = RECORD  
    enlace : Ptr_nodoCola;  
    datos : Tipo_datos  
  END;  
  TipoCola = RECORD  
    frente, final : Ptr_nodoCola;  
  END
```

➤ Operadores básicos:

PROCEDURE **Meter_cola** (VAR cola : Tipo_cola; nuevo_dato: Tipo_datos);
 (* escribe dato por el final de la lista *)

VAR

Nuevo_nodo : Ptr_Nodo_cola;

BEGIN

1 Allocate(Nuevo_nodo, SIZE(Nodo_cola));

2 Nuevo_nodo^.datos := nuevo_dato;

2 Nuevo_nodo^.enlace := NIL;

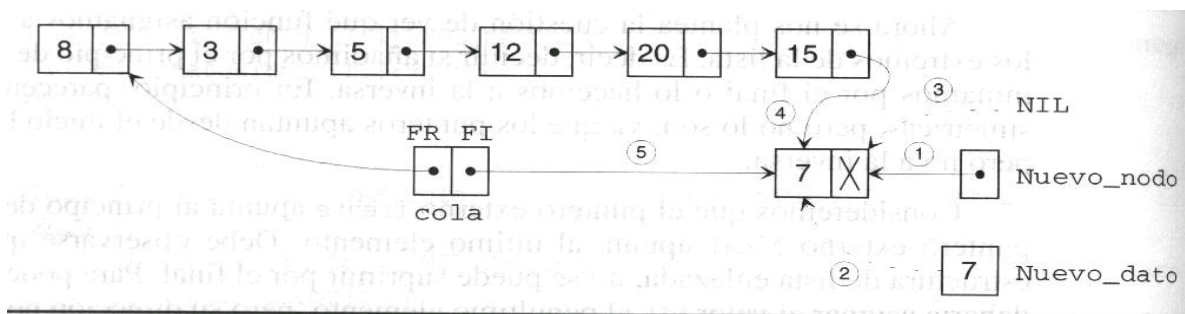
4 IF cola.final = NIL THEN

cola.frente := Nuevo_nodo

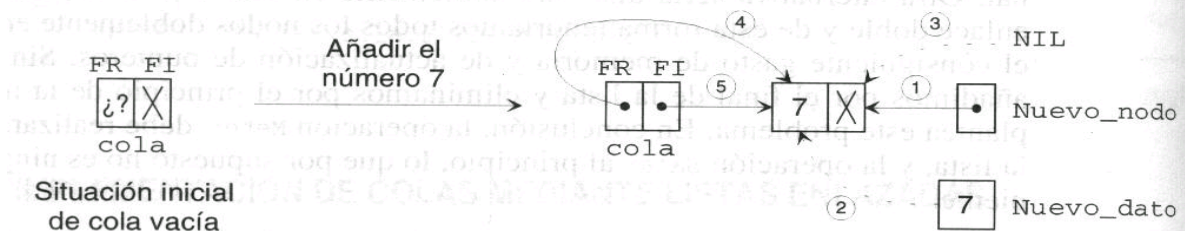
ELSE cola.final^.enlace := Nuevo_nodo END

5 cola.final := Nuevo_nodo

END Meter_cola;



Inserción en una cola no vacía implementada con asignación dinámica de memoria.

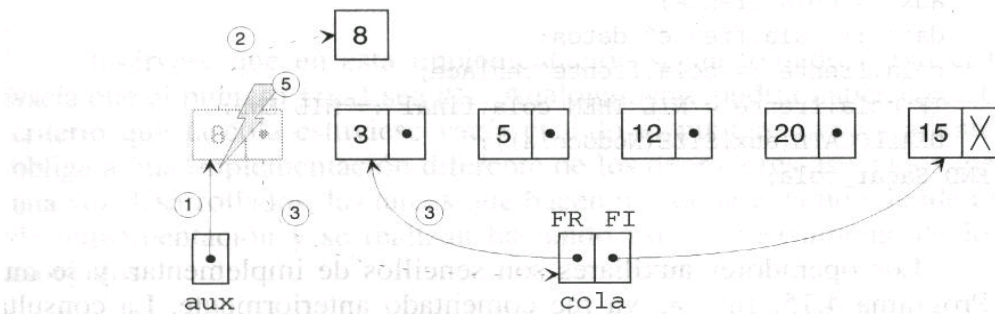


Inserción en una cola inicialmente vacía con asignación dinámica de memoria.

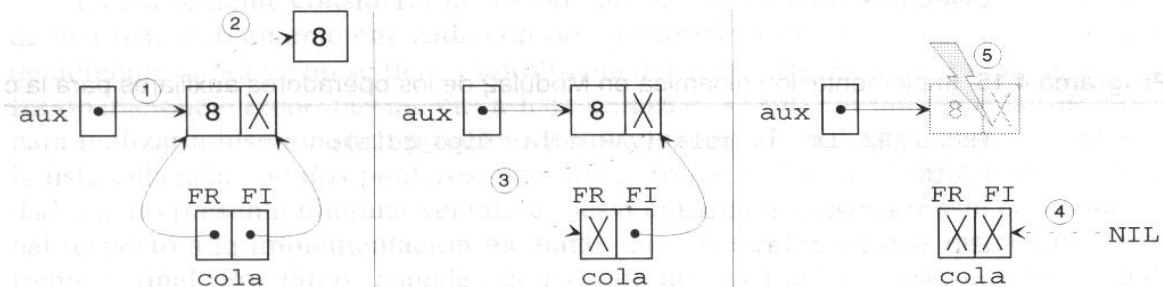
```

PROCEDURE SacarCola (VAR cola : TipoCola; VAR dato: TipoDatos);
(*lee dato por el principio de la lista*)
VAR
    Aux : Ptr_NodoCola;
BEGIN
    1  Aux := cola.frente;
    2  dato := cola.frente^.datos;
    3  cola.frente := cola.frente^.enlace;
    4  IF cola.frente = NIL THEN
        cola.final := NIL
    END
    5  Deallocate(Aux, SIZE(NodoCola));
END SacarCola;

```



Eliminación de un elemento no único en una cola implementada dinámicamente.



Eliminación del último elemento en una cola implementada dinámicamente. En este caso, el paso 3 no es necesario pero se mantiene por ser común con el caso general.

Operadores auxiliares:

```
PROCEDURE Inicia_cola (VAR cola: Tipo_cola)
BEGIN
    cola.final := NIL;
END;
```

```
PROCEDURE Cola_Vacia(cola: Tipo_cola): Boolean;
BEGIN
    RETURN cola.final = NIL
END Cola_vacia;
```

```
PROCEDURE Cola_Llena(cola: Tipo_Cola): Boolean;
BEGIN
    RETURN ~Available(SIZE(Nodo_Cola));
END Pila_Vacia;
```

```
PROCEDURE Consultar_frente_cola(cola:Tipo_cola; VAR dato:TipoDato);
BEGIN
    dato:=cola.frente^.datos; (*lee elemento de cima pero no elimina*)
END Consultar_frente_cola;
```

```
PROCEDURE Consultar_final_cola(cola:Tipo_cola; VAR dato:TipoDato);
BEGIN
    dato:=cola.final^.datos; (*lee elemento de cima pero no elimina*)
END Consultar_final_cola;
```

- **Nota:** Obsérvese que en esta implementación se ha tomado como criterio de cola vacía que el puntero final apunte a NIL. Otra posibilidad hubiese sido el criterio de cola vacía cuando el *puntero frente* apunta a NIL (esto implica una ligera modificación de los procedimientos).