Estructura de datos y algoritmos

Tema IV:

TIPOS DE DATOS

ABSTRACTOS DINÁMICOS LINEALES

TIPOS DE DATOS ABSTRACTOS DINÁMICOS LINEALES

- 4.1 Introducción
- 4.2 Ejemplos de TDA dinámicos lineales
- 4.3 Pilas
  - 4.3.1 Implementación mediante arreglos
  - 4.3.2 Implementación mediante listas enlazadas
- 4.4 Colas
  - 4.4.1 Implementación mediante arreglos
  - 4.4.2 Implementación mediante listas enlazadas
- 4.5 Ejemplos de aplicación
- 4.6 Consideraciones generales
- 4.7 Conclusiones

Definiciones

- Tipo de Datos
  - Conjunto de elementos.
- Tipo de Datos Abstracto (TDA)
  - Es un conjunto de elementos definidos de forma única mediante un tipo y un conjunto de operaciones sobre dichos elementos.
- Estructura de Datos
  - Implementación física de un TDA.

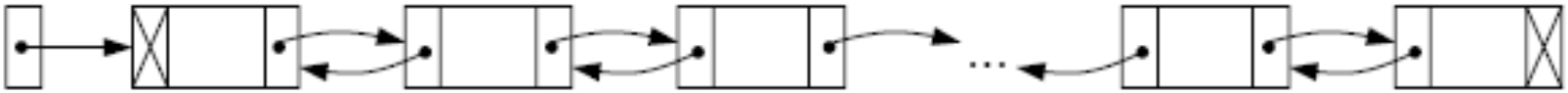
4.1 Introducción

- En este capítulo se presentarán los TDA dinámicos **lineales**, es decir, aquellos que mantienen una organización secuencial con **un predecesor y un sucesor**.
- En primer lugar se introducen algunos TDA similares a las listas enlazadas, como ejemplo de construcción de TDA dinámicos.
- Seguidamente se presentan los dos TDA dinámicos lineales más característicos: las pilas y las colas.

Tipo TDA	N° 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

4.2 Ejemplos de TDA dinámicos lineales

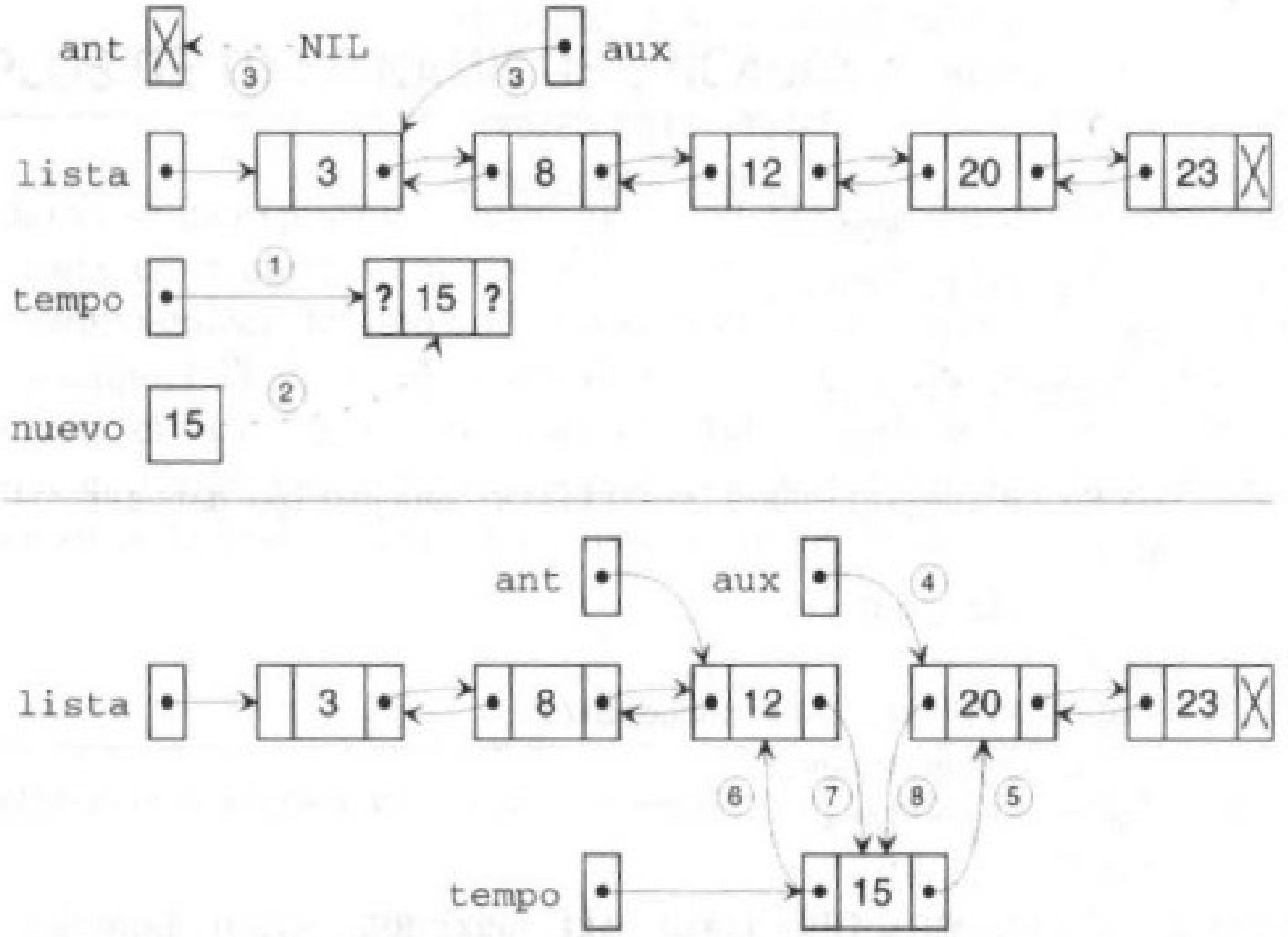
- A partir de la generalización del campo enlace del tipo nodo visto en las listas enlazadas, puede definirse cualquier TDA dinámico con diversos enlaces.
- TDA Listas doblemente enlazadas
  - Colección de nodos ordenados según su posición, tal que cada uno de ellos es accedido mediante el puntero anterior 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,...



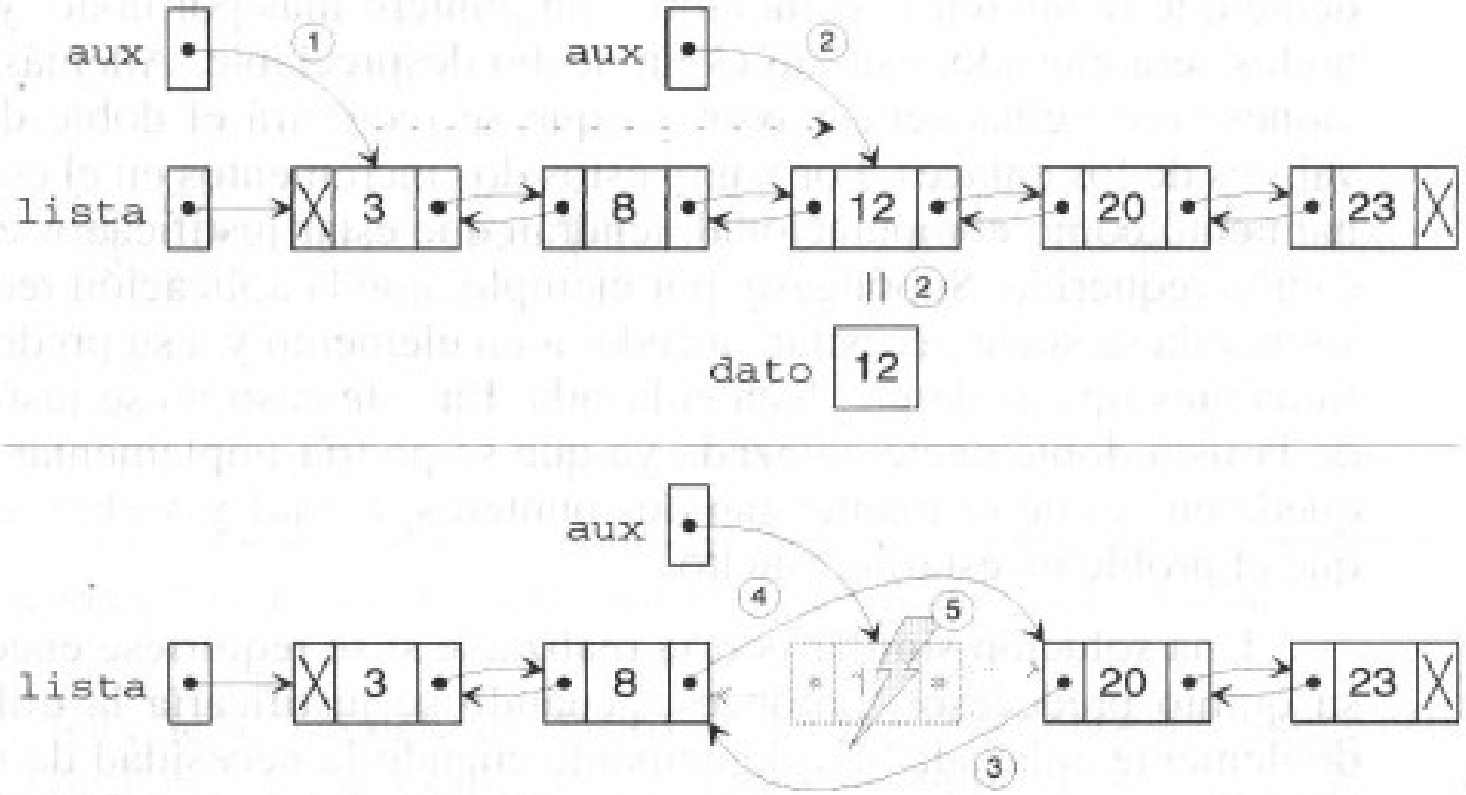
Ventajas/Inconvenientes respecto de las listas enlazadas

- 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).

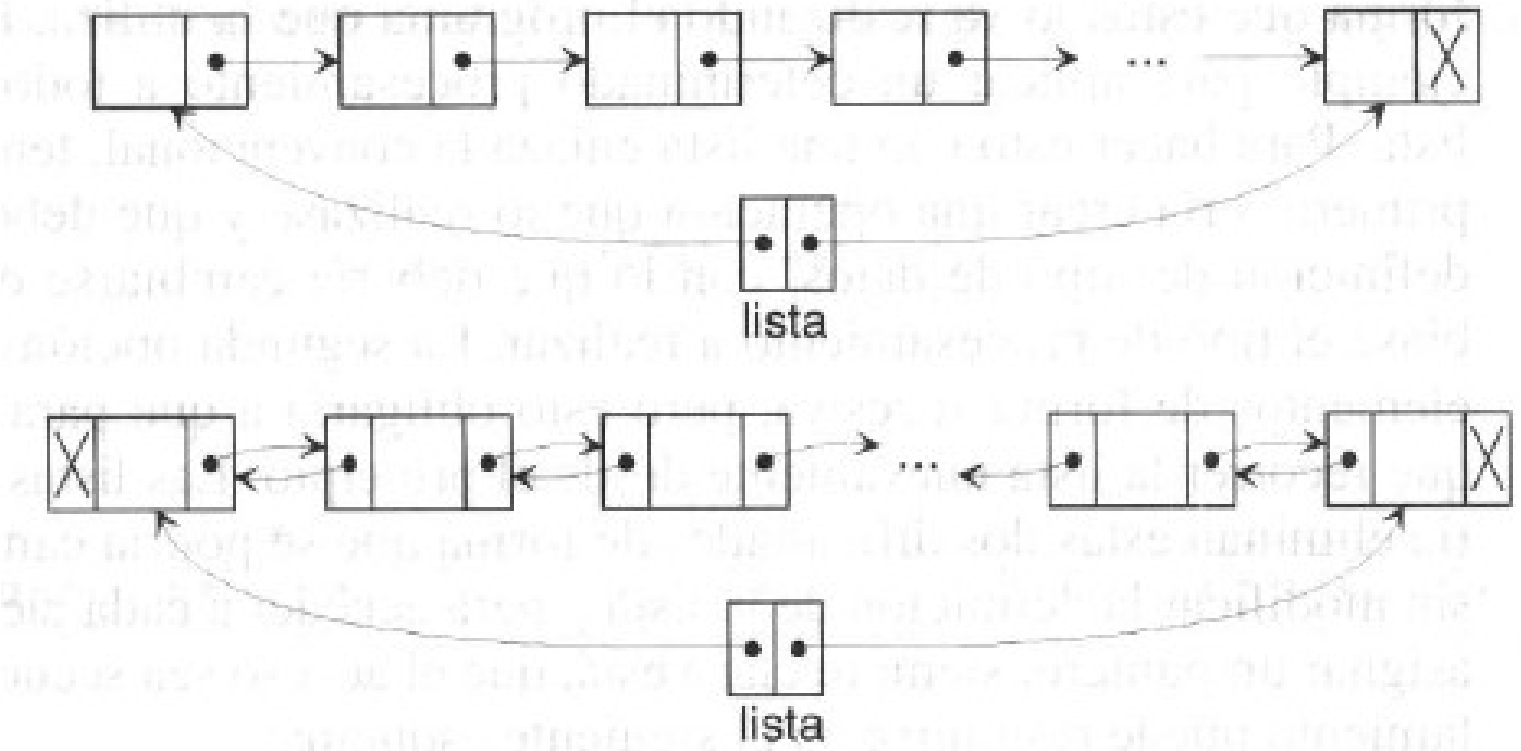
Insertar en orden



Eliminación de la primera aparición del elemento buscado

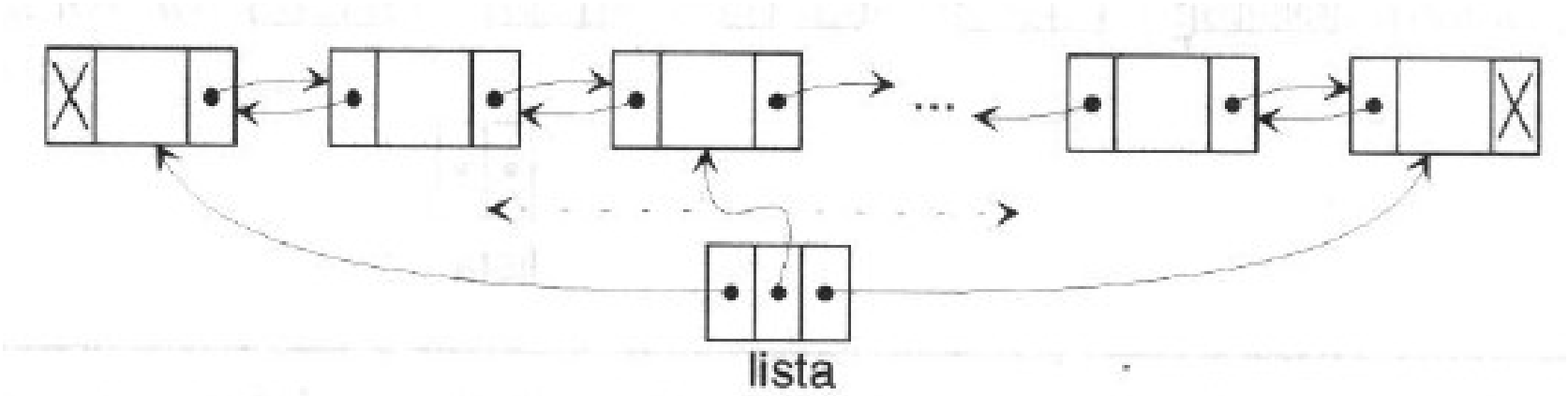


Listas con punteros al final de la lista (enlazadas/doblemente enlazadas):



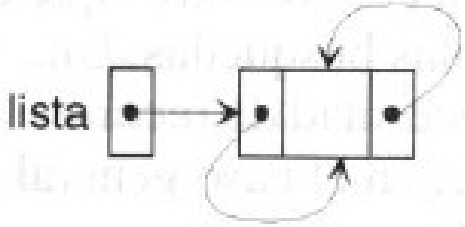
Listas con memoria (enlazadas/doblemente enlazadas)

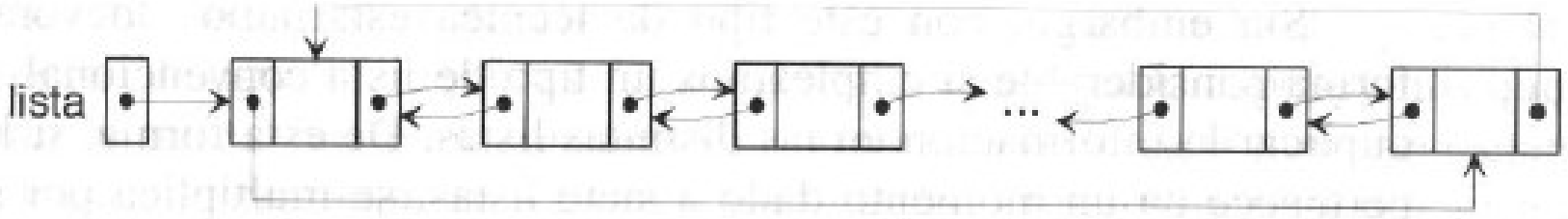
- Se añade un puntero adicional que apunta siempre al último elemento accedido
- Permite aplicar un procesamiento externo a todos los elementos de la lista



Listas circulares (enlazadas/doblemente enlazadas)

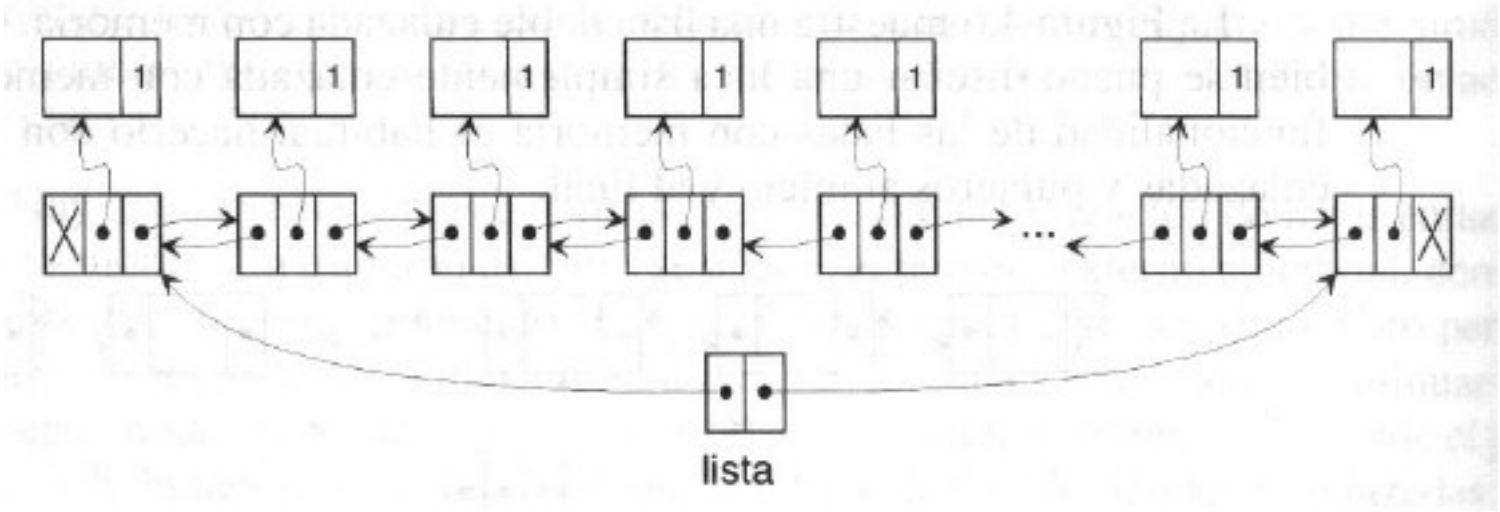
- El último tiene como sucesor al primero y el primero al último





Listas multirreferenciadas (enlazadas/doblemente enlazadas)

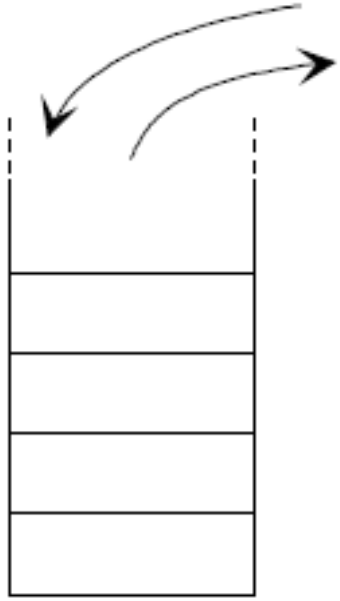
- En estas listas, el nodo no contiene la información directamente sino un puntero a la misma
- **Aplicaciones**:
  - Cuando una misma información puede pertenecer a más de una lista, por ejemplo, según distintas condiciones de búsqueda.



4.3 Pilas

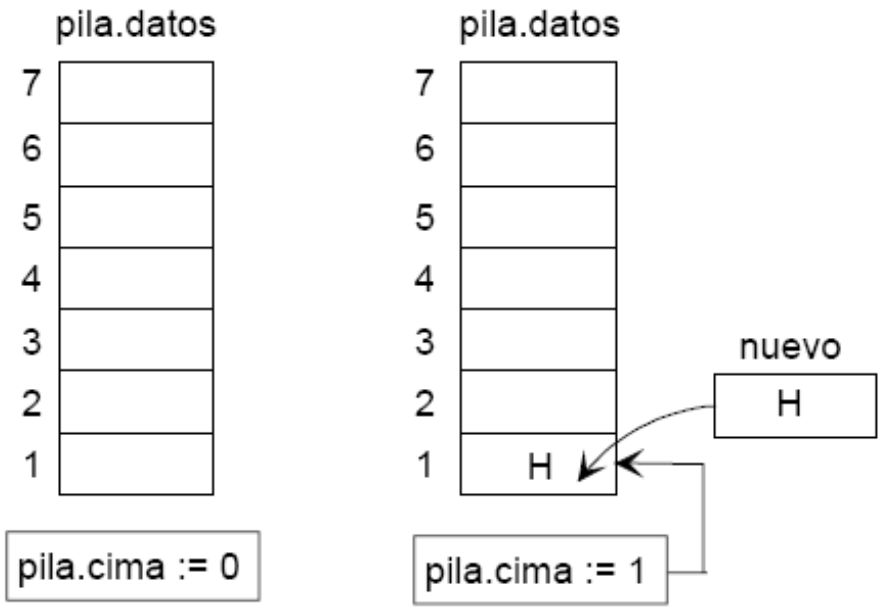
- Una pila es un TDA **dinámico** homogéneo al que sólo tiene acceso por la cabeza o cima de la pila; dicho acceso es de tipo LIFO ("Last In-First Out": último elemento en entrar-primero en salir).
- Los operadores básicos asociados son:
  - **Introducir y Extraer** elementos de la pila, y
- Los operadores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



4.3.1. Implementación mediante arreglos

```
CONST    MAXPILA=100;

TYPE TipoIndice = [1..MAXPILA];
       TipoPila = RECORD
                    datos: ARRAY[TipoIndice] OF TipoElemento;
                    cima: Integer;
                  END;
VAR  pila: TipoPila;
```

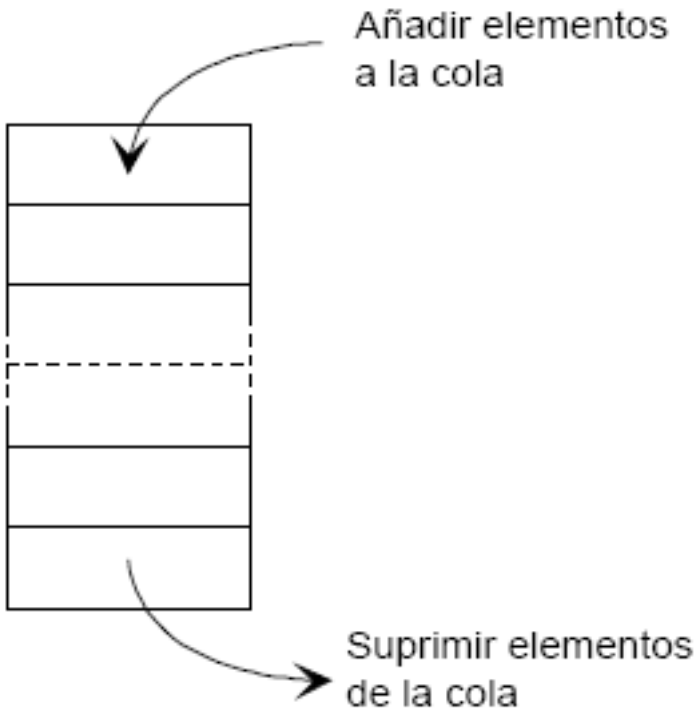


4.4.2 Implementación mediante listas enlazadas

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

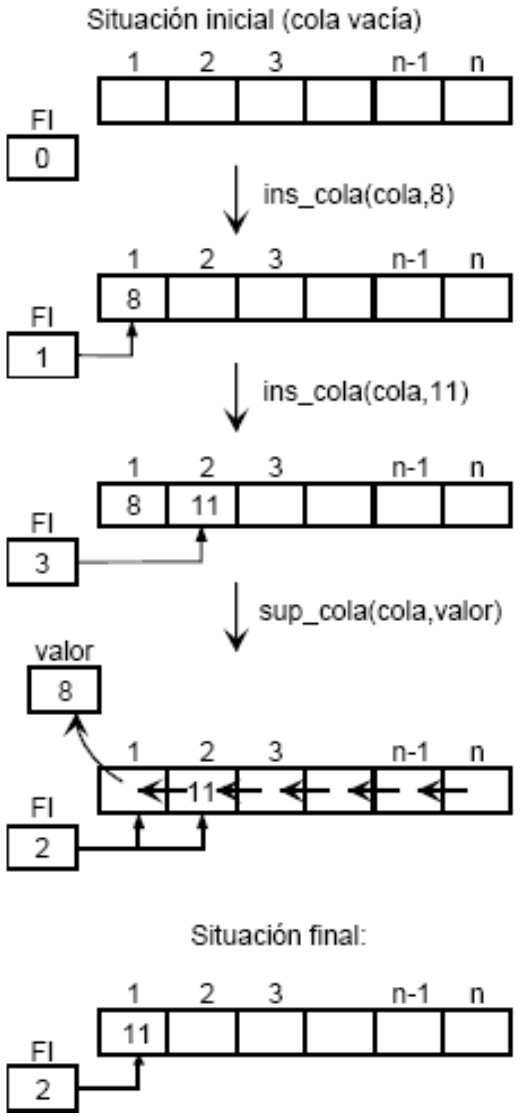
4.4 Colas

- 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
- El 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
- Los operadores auxiliares asociados son:
  - Inicia_cola:
    - Crea una cola que no contiene elementos
  - Cola_vacía:
    - 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.



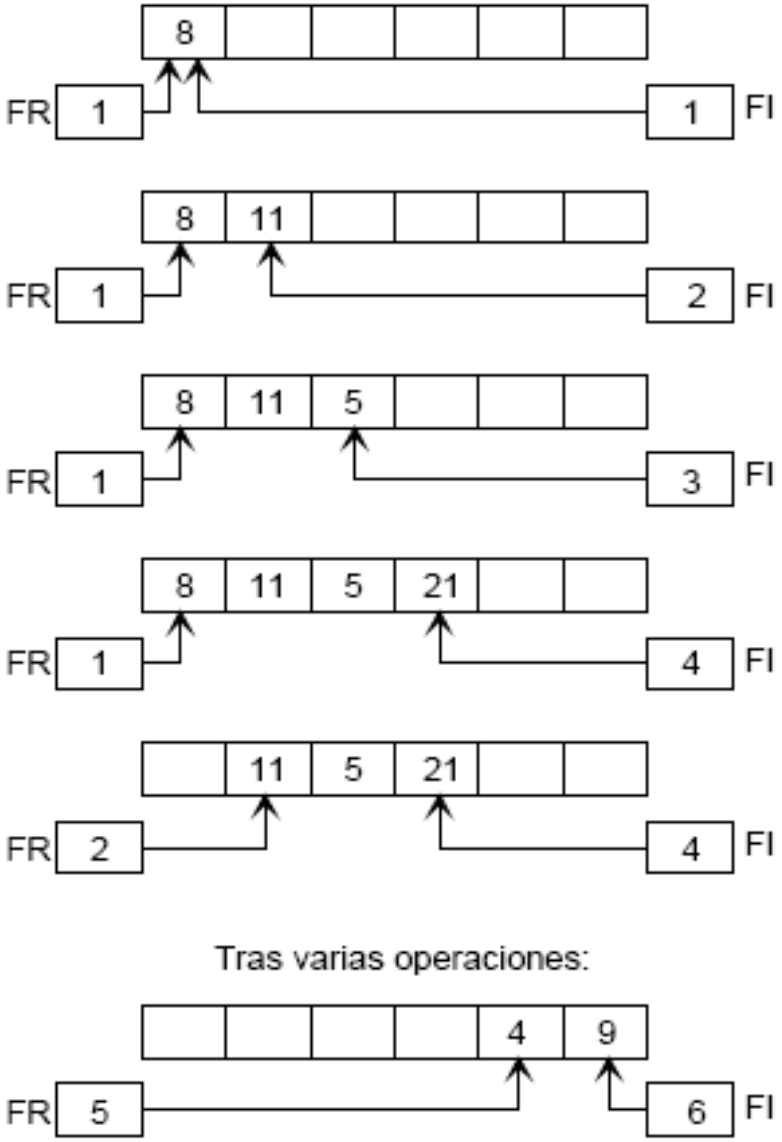
4.4.1 Implementación de colas mediante arreglos

- 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.



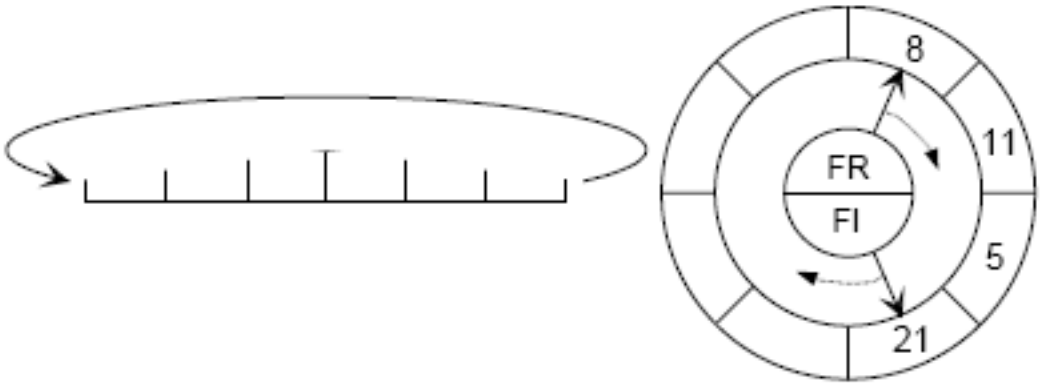
Con dos índices y arreglo lineal

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

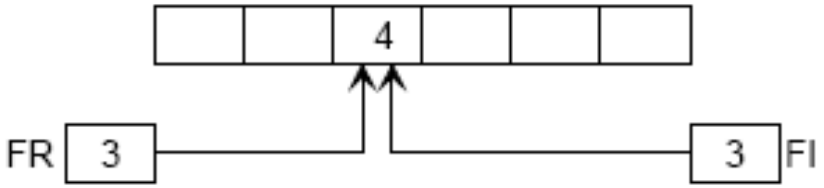


Con dos índices y arreglo circular

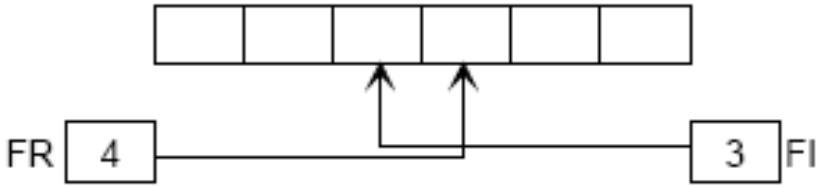
- Se aprovechan las posiciones vacías y no es necesario desplazar los elementos
- Problema:
  - Distinguir entre cola llena y cola vacía.
  - No hay diferencia ente la situación de cola vacía y cola llena



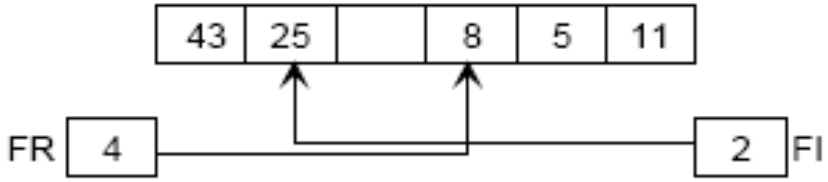
- La cola está vacía con FR=4 y FI=3:



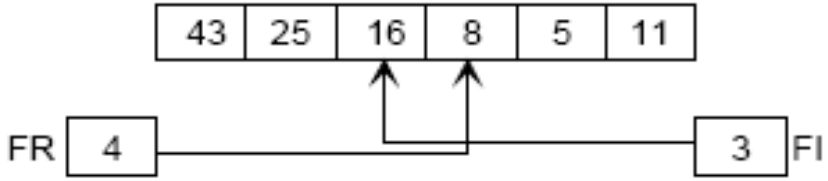
Al suprimir FR aumenta:



- La cola está llena con FR=4 y FI=3:



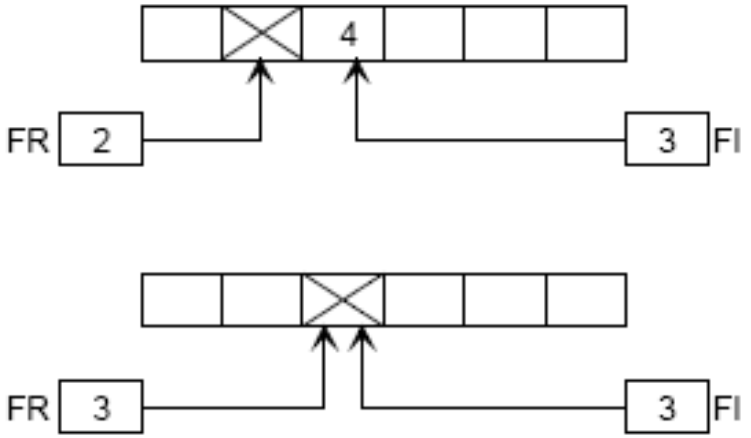
Al insertar FI aumenta:



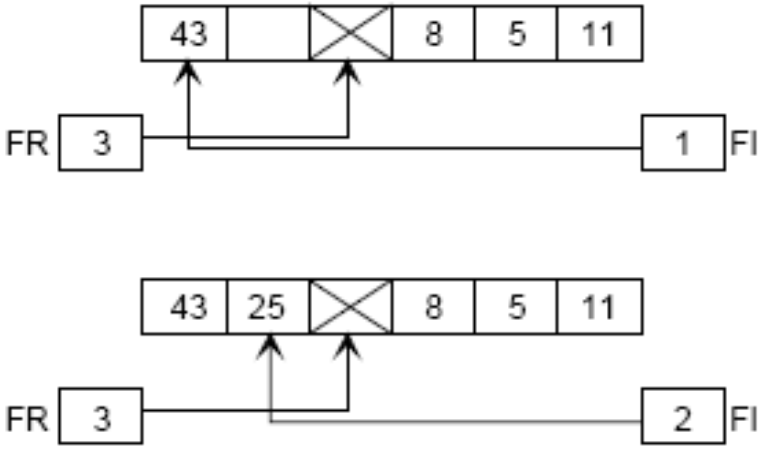
Solución

- 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 y FR=1)

- La cola está vacía con FR=3 y FI=3:



- La cola está llena con FR=3 y FI=2:



Criterio 2

- 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.

➢ Declaración de la estructura cola (Modula2):

```
CONSTMAXCOLAN=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
```

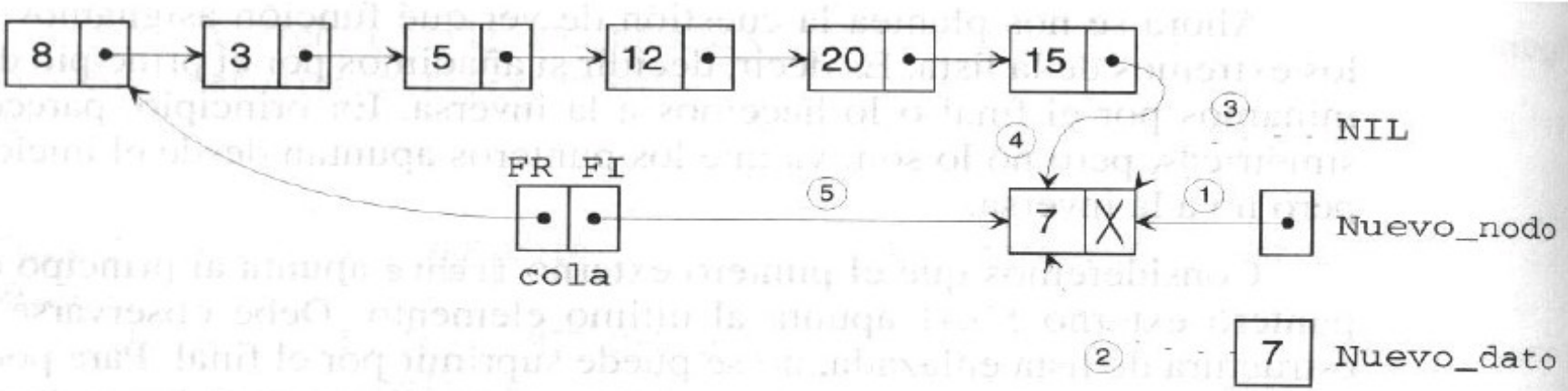
```
PROCEDURE Sacar_Cola(VAR cola:TipoCola; VAR dato:TipoElemento);
 BEGIN
     IF cola.FR = MAXCOLA THEN   cola.FR := 1
     ELSE  cola.FR := cola.FR +1;
     END (*primero incrementa y después lee*)
     dato := cola.datos[cola.FR]
 END
```

- Operadores auxiliares:

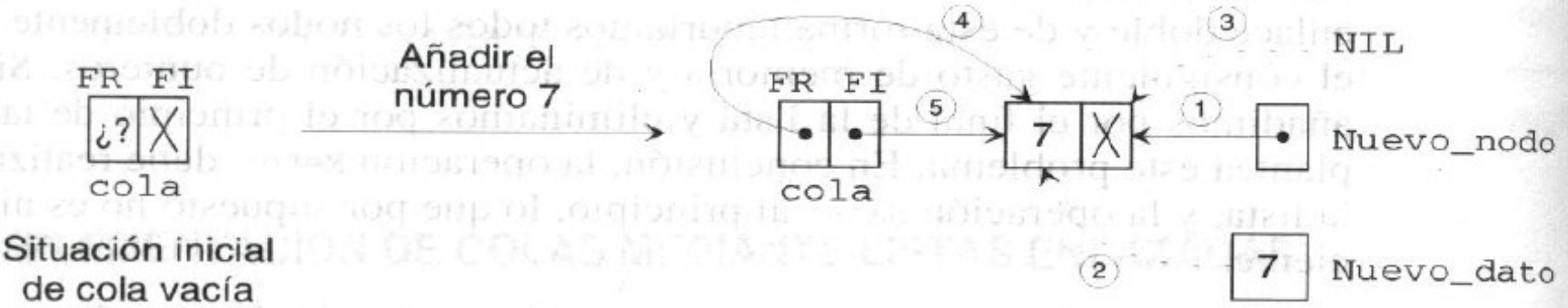
```
PROCEDURE Inicia_Cola (VAR cola: TipoCola);
      BEGIN
        Cola.FR:= MAXCOLA:
        Cola.FI:= MAXCOLA
      END

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.fi + 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;
```

4.4.2 Implementación de colas mediante listas enlazadas

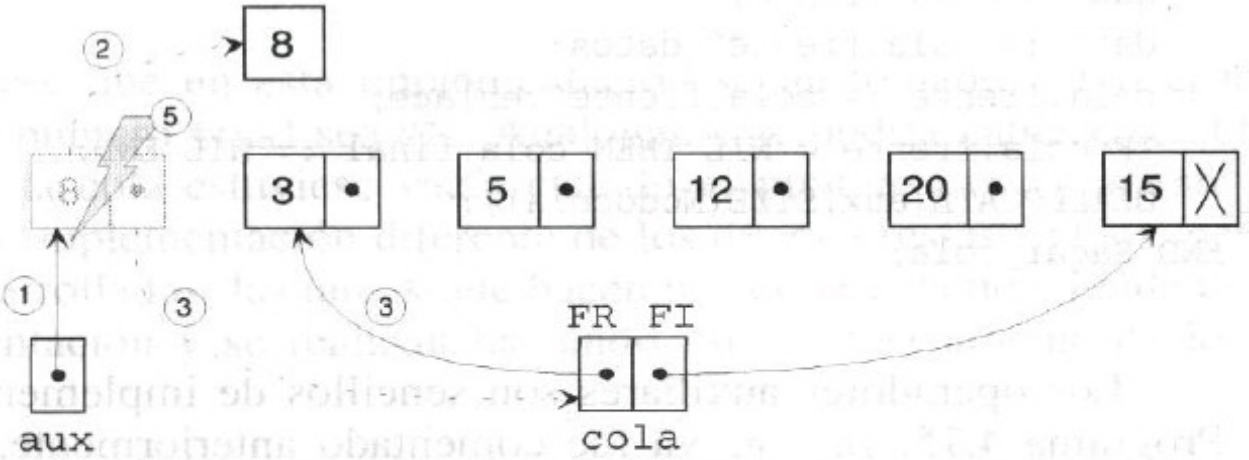
- Operaciones
  - Meter: por el final y
  - Eliminar: por el principio
- Problema
  - La inserción o la supresión por el final requiere un bucle hasta encontrarlo
    - El número de comparaciones de punteros es igual a la cantidad de elementos que tenga la cola.
- Solución:
  - Utilizar un puntero externo 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.



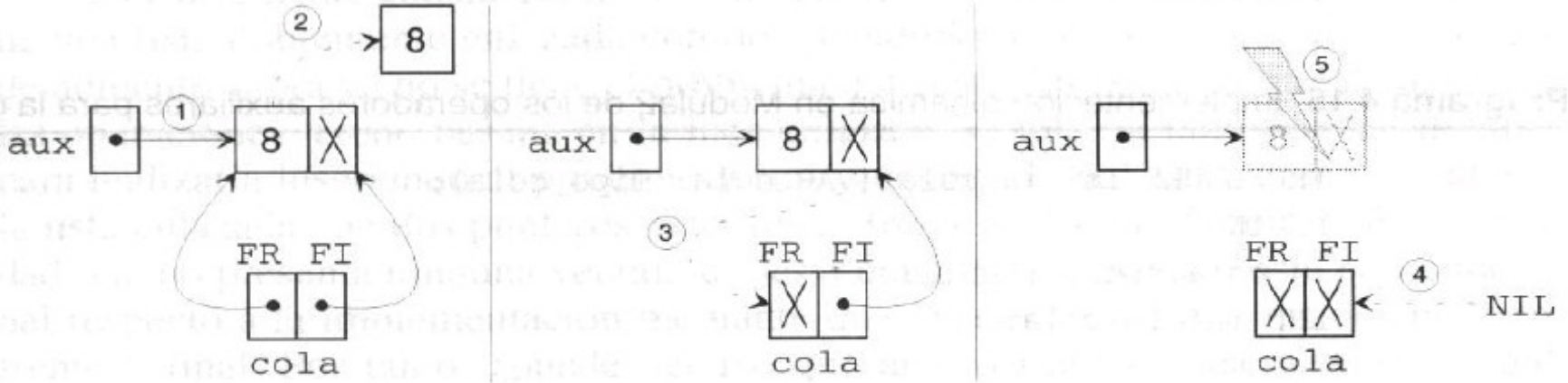
Inserción en una cola no vacía implementada con asignación dinámica de memoria.



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



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.

4.5 Ejemplos de aplicación

- 4.5.1 Evaluación de parentización de expresiones
- 4.5.2 Notación de expresiones prefija y postfija
- 4.5.3 Búsqueda de infinitivo en un texto