sólo nodo convencional ocupa el equivalente a más de 100 nodos de una lista de este tipo. Si en lugar de considerar listas doblemente enlazadas, fuesen listas con un solo enlace, el ahorro sería aún mayor.

Las listas que incluyen más de un puntero en cada nodo, no siempre se enlazan simultáneamente con el anterior y el posterior según un criterio de ordenación único. Imaginemos que una misma información la queremos mantener ordenada según dos criterios distintos. En el ejemplo de las fichas de libros podría ser deseable tenerlas ordenadas por autores y por años de edición. Esto puede resolverse fácilmente con dos listas independientes, pero la información estaría duplicada. Otra alternativa es el empleo de listas multireferenciadas, ya que evitan la duplicación de información. Sin embargo, si tenemos nodos con dos punteros, podemos emplear uno para enlazar según el orden de autores y el otro para enlazar según el año de edición. La Figura 4.10 muestra una lista de este estilo.

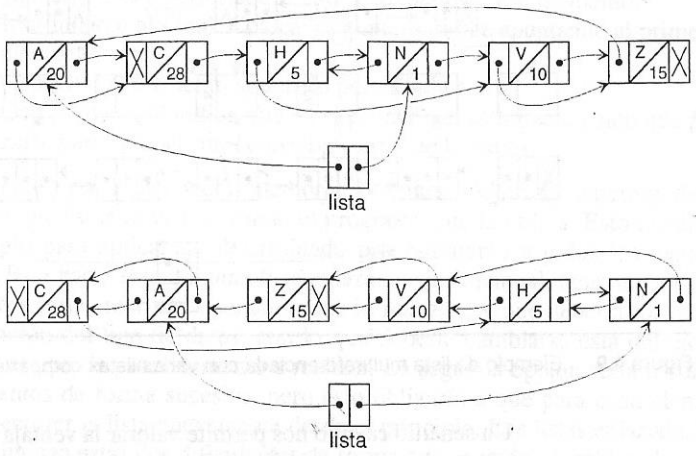

Figura 4.10 Lista con dos punteros que enlazan los elementos según dos criterios distintos. Las dos gráficas muestran la misma lista enlazada pero ordenando la representación gráfica de sus nodos según cada uno de los criterios.

Esta estructura puede generalizarse a cualquier número de punteros, tantos como criterios de ordenación distintos queramos mantener. En el caso extremo de que quisiéramos ordenar por todos los posibles campos de la información almacenada, habría un puntero por cada uno de estos campos.

Debe tenerse en cuenta que aunque sean listas doblemente enlazadas, son bastante distintas a las mostradas en los ejemplos anteriores, ya que ahora no se puede acceder desde un nodo a su predecesor, sino a dos sucesores. Cada nodo de esta lista indica quién es el nodo que le sigue según cada uno de los criterios pero no mantiene ninguna información sobre quién le precede, para ello sería necesario otro par de punteros adicional. Este tipo de listas deben considerarse como listas absolutamente independientes, y cada vez que se quiera hacer una manipulación con un nodo (inserción, búsqueda, eliminación, etc.), habría que tenerlo en cuenta en las dos listas de forma absolutamente independiente, lo que hace el tratamiento bastante lento y complicado.

Como puede observarse, hay aplicaciones en las que las listas doblemente enlazadas son sensiblemente mejores que las listas enlazadas, pero no son tan numerosas como en un principio podría pensarse, además del exceso de complejidad y carga computacional que conllevan; de ahí la amplia utilización de las listas enlazadas simples.

En conclusión, será la aplicación la que determine la necesidad de definir un TDA dinámico u otro, pero al hacerlo deben considerarse detenidamente las ventajas e inconvenientes que pueden aportar. Se deja como ejercicio la realización de las operaciones básicas sobre los distintos tipos de listas enlazadas comentadas en este apartado.

4.3 PILAS

Una pila se suele definir simplemente como una estructura de datos en la que el último elemento en entrar es el primero en salir. A estas estructuras se les denomina LIFO (*Last In, First Out*).

Evidentemente es una estructura en la que los elementos están ordenados y se añaden y suprimen únicamente por un único extremo, conocido como *tope o cabeza de la pila*. Una estructura de tipo pila responde fielmente al concepto cotidiano de pila al que estamos muy acostumbrados en la vida real. Un ejemplo típico es una pila de cajas. Sólo se puede coger la caja de arriba de la pila, y sólo ahí se puede dejar una caja. Obsérvese que esta disposición de las cajas es una pila porque se actúa así sobre ella. Efectivamente, podría pensarse en suprimir una caja que no es la primera o se podría ser lo suficientemente habilidoso como para introducir una entre dos ya colocadas. En este caso la disposición de las cajas no es una estructura pila, será otra cosa (una lista enlazada, por ejemplo).

Como puede observarse por otro lado, la estructura pila es dinámica ya que el número de elementos que la componen es variable. Por ello una pila puede encontrarse definida en la literatura, usualmente en la dedicada a principios básicos de programación estructurada, como una lista enlazada en la que el último elemento en entrar es el primero en salir.

Sin embargo, deben realizarse algunas consideraciones acerca de esta sencilla definición. La definición inicial de la pila sólo resalta la característica más sobresaliente del TDA pila, su acceso LIFO, y definirla como una lista enlazada sólo indica una manera de implementarla. Es decir, ninguna de ellas es una definición rigurosa de la pila como TDA. Esto no sólo tiene importancia formal o académica. Las definiciones de los TDA precisan los términos y las operaciones que pueden realizarse sobre los datos evitando ambigüedades. En el ejemplo de las pilas de cajas, ¿podría

mirarse el contenido de la última caja? Si es una lista enlazada por supuesto que sí, y la definición de acceso LIFO no se infringiría por 'mirar' la última caja. Pero si se tiene en cuenta que para mirar el contenido de la caja habría que sacarla entonces sí se violaría la condición LIFO. Para no hacerlo habría que pensar en cajas transparentes, por ejemplo. Podría hacerse así un conjunto de elucubraciones, todas ellas correctas o incorrectas según el punto de vista. Recuérdese que la abstracción y el encapsulamiento de datos tienen como objetivo establecer con rigor y sencillez las estructuras, y por ello, estas definiciones iniciales no pueden considerarse en absoluto correctas en el contexto de los TDA.

Atendiendo a las consideraciones anteriores el TDA pila se define de la siguiente manera:

Definición 4.2: Una pila es un TDA dinámico homogéneo al que sólo se tiene acceso por la cabeza o cima de la pila; dicho acceso es de tipo LIFO ('Last In - First Out': último en entrar - primero en salir), los operadores básicos asociados son `Meter` y `Sacar` elementos de la pila, y los operadores auxiliares asociados son:

- `Inicia_pila`: crea una pila o limpia una existente inicializándola a una pila vacía, que es aquella que no contiene elementos. Permite partir de un estado previamente establecido.
- `Pila_Vacia`: operación utilizada para consultar si la pila está vacía.
- `Pila_Llena`: operación utilizada para consultar si la pila está llena.
- `Consultar_Pila`: operación utilizada para consultar el contenido de la cima de la pila sin sacar el elemento de la misma.

La operación `Pila_Vacia` será necesaria cuando se vaya a sacar un elemento de la pila ya que no se pueden sacar elementos de una pila vacía, y `Pila_Llena` cuando se vaya a meter un elemento en la pila ya que no se pueden introducir elementos en una estructura llena.

La operación `Consultar_Pila` se utiliza para consultar el último elemento introducido en la pila sin sacarlo de la misma. Obsérvese que esta operación no infringe la característica de la pila que indica que *sólo* se tiene acceso por la cabeza o cima de la pila. La Figura 4.11 muestra la inserción y la eliminación de los elementos de una pila.

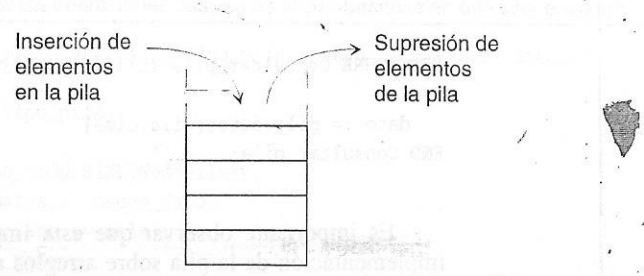

Figura 4.11 Inserción y eliminación de los elementos de una pila.

La Definición 4.2 establece el concepto de pila en sentido estricto, y así deberán considerarse todos los TDA. Con ella no existe ambigüedad alguna ya que todas las funciones a realizar deberán implementarse en función de sus operaciones básicas y auxiliares.

Es importante resaltar el hecho de que el carácter dinámico de la pila es conceptual, como en todos los TDA, y se debe a que el tamaño de la pila es variable, dependiendo de la cantidad de elementos almacenados en la misma. Este hecho es independiente de la implementación que se realice de la pila, en base a estructuras estáticas o dinámicas. Tanto con unas como con otras, la pila es dinámica. Seguidamente se presentan las implementaciones estática, mediante arreglos, y dinámica, mediante listas enlazadas, de la pila.

4.3.1 IMPLEMENTACIÓN DE PILAS MEDIANTE ARREGLOS

La implementación del TDA consta de dos partes. Por un lado, hay que definir la estructura sobre la que se almacenarán los datos, y por otro, la realización de los operadores básicos y asociados.

Dado que la pila es un tipo de datos homogéneo, es decir, sus componentes son del mismo tipo, es posible realizar su implementación mediante arreglos. Para ello, la pila vendrá representada por un registro con dos campos, uno el arreglo en el que se almacenarán los elementos de la pila y otro, denominado cima, que almacenará la posición del tope de la pila, es decir, la posición del último elemento introducido en la pila. Indexando el arreglo mediante enteros, el campo cima será de tipo entero. La declaración de una pila en Modula2 es la siguiente:

```
TYPE Tipo_Indice = [1..MAXPILA];
     Tipo_pila = RECORD
                 datos : ARRAY Tipo_indice OF Tipo_datos;
                 cima : INTEGER;
                 END;
VAR pila : Tipo_pila;
```

donde `MAXPILA` es el tamaño máximo de la pila.

La realización de los operadores básicos y auxiliares se describe a continuación.

La operación `Inicia_pila` deberá establecer que la pila no contiene elementos, en función de un valor dado por convenio. Evidentemente la forma más natural de inicializar la pila es considerar que el valor de la cima es cero, indicando que la pila contiene cero elementos. Así, se toma como *criterio* que *cima indica la posición del último elemento introducido*. Cuando cima es cero indica que no ha habido último elemento introducido, con lo que la pila está vacía. Esta operación se implementa en Modula2 mediante el procedimiento del Programa 4.5.

Programa 4.5 Procedimiento para la inicialización de una pila implementada mediante arreglos.

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

Para implementar los operadores básicos es necesario analizar su efecto sobre la pila. La Figura 4.12.a muestra una pila vacía. Obsérvese que para 'vaciar' la pila no es necesario realizar ninguna acción sobre los elementos del arreglo. Basta con considerar los posibles contenidos de `pila.datos` como 'basura'.

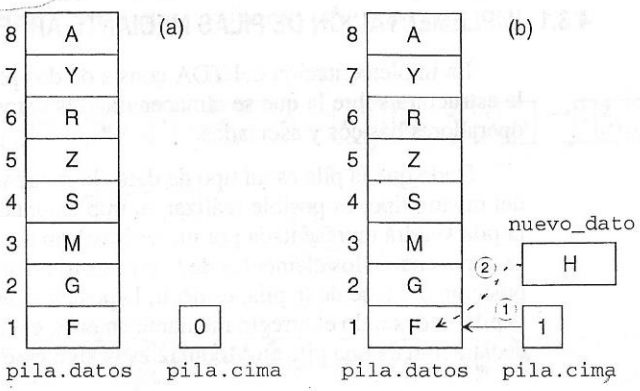

Figura 4.12 Pila vacía (a). Introducción de un nuevo elemento en la pila (b).

Dado que la cima indica por criterio la posición del último elemento introducido, para introducir el nuevo elemento habrá que incrementar primero la cima y seguidamente introducir el elemento, tal y como se ilustra en la Figura 4.12.b. Así el procedimiento en Modula2 que implementa la operación `Meter` viene dado por el Programa 4.6.

Programa 4.6 Implementación de la operación `Meter` de la pila implementada mediante arreglos.

```
PROCEDURE Meter_pila (VAR pila: Tipo_pila; nuevo_dato : Tipo_datos);
BEGIN
   pila.cima := pila.cima+1;
   pila.datos[pila.cima] := nuevo_dato;
END Meter_pila;
```


La operación `Sacar` actuará en orden inverso, de manera que primero se toma el elemento al que apunta `cima` y posteriormente se decrementa el valor de la cima. El Programa 4.7 muestra este procedimiento.

Programa 4.7 Implementación de la operación `Sacar` de la pila implementada mediante arreglos.

```
PROCEDURE Sacar_pila (VAR pila: Tipo_pila; VAR dato : Tipo_datos);
BEGIN
   dato := pila.datos[pila.cima];
   pila.cima := pila.cima-1;
END Sacar_pila;
```

Debe observarse el carácter dinámico de la pila. La cantidad de elementos que pertenecen a la pila es variable. Así, en la situación de la Figura 4.12.a la pila no tiene elementos mientras que en la de la Figura 4.12.b tiene un elemento. En esta situación los elementos del arreglo de índice 2, 3, etc., no forman parte de la pila. Ésta está formada únicamente por el elemento de índice 1. No debe confundirse la pila con el arreglo utilizado para implementarla.

Finalmente, las operaciones auxiliares de consulta vendrán dadas por las funciones mostradas en el Programa 4.8.

Programa 4.8 Operaciones auxiliares de consulta de la pila implementada mediante arreglos.

```
PROCEDURE Pila_vacia (pila:Tipo_pila) : BOOLEAN;
BEGIN
   RETURN pila.cima = 0;
END Pila_vacia;

PROCEDURE Pila_llena (pila:Tipo_pila) : BOOLEAN;
BEGIN
   RETURN pila.cima = MAXPILA
END Pila_llena;

PROCEDURE Consultar_pila (pila:Tipo_pila; VAR dato : Tipo_datos);
BEGIN
   dato := pila.datos[pila.cima]
END Consultar_pila;
```

Es importante observar que esta implementación no es la única válida. La implementación de la pila sobre arreglos anteriormente realizada ha tomado como criterio que la cima apunta a la posición del último elemento introducido. Sin embargo, se podría haber decidido que la *cima apunte a la posición del último ele-*

mento sacado; o lo que es lo mismo: apuntará a la posición donde deberá añadirse el próximo elemento. Con este criterio, la pila estará vacía cuando el valor del campo `cima` sea 1, para `Meter` un nuevo elemento habrá que introducir primero el elemento y seguidamente incrementar la cima y la operación `Sacar` actuará en orden inverso, de manera que primero se decrementa el valor de la cima y á continuación, se toma el elemento al que apunta cima. Las operaciones auxiliares de consulta cambiarán con el nuevo criterio de manera que la pila estará vacía si `cima` vale 1, y estará llena si vale `MAXPILA+1`. Finalmente la operación `Consultar_Pila` tomará el dato de la posición `cima-1`. Ambas implementaciones son igualmente válidas. Una vez realizada la implementación, el uso de la pila es independiente de la misma, es decir, para realizar cualquier función con la pila se utilizan los procedimientos y funciones de los operadores asociados y no se vuelve a atender a los detalles de implementación.

4.3.2 IMPLEMENTACIÓN DE PILAS MEDIANTE LISTAS ENLAZADAS

La implementación de pilas mediante estructuras dinámicas es inmediata partiendo de una lista enlazada. La declaración de la estructura en Modula2 será:

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

y se accede a la pila mediante el puntero externo que apunta a la lista enlazada. La cima de la pila es obviamente la cabeza de la lista enlazada.

La operación `Meter` se implementa de la misma forma que se hacía en la inserción por la cabeza de la lista enlazada, y análogamente la operación `Sacar` es equivalente a sacar por la cabeza de la lista enlazada. Los procedimientos que implementan estas operaciones básicas se muestran en el Programa 4.9.

Programa 4.9 Implementación de las operaciones básicas de la pila basadas en una lista enlazada.

```
PROCEDURE Meter_pila (VAR pila: Tipo_pila; nuevo_dato : Tipo_datos);
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);
VAR
   Aux : Tipo_pila;
BEGIN
   Aux := pila;
   dato := pila^.datos;
   pila := pila^.enlace;
   DEALLOCATE(Aux,SIZE(Nodopila));
END Sacar_pila;
```

Las operaciones auxiliares `Iniciar_Pila` y `Pila_Vacia` se implementan también directamente a partir de los conceptos elementales de listas enlazadas. Así, para inicializar la pila bastará que el puntero externo sea `NIL` y para comprobar si está vacía habrá que consultar si tiene este valor.

La operación `Pila_llena` no puede obviarse aunque la implementación sea dinámica. En el caso de la implementación estática el hecho de que la pila estuviese llena podía resolverse en memoria principal dotando al programa de procedimientos alternativos para manejar esta situación, utilizando otra pila por ejemplo. Sin embargo, debe observarse que con implementación dinámica el hecho de que la pila esté llena significa que no existe memoria libre suficiente para crear un nuevo nodo. Esta es una situación crítica en la que sólo se puede intentar liberar memoria principal, pasando datos a memoria secundaria, e informar que no queda memoria libre. El hecho de que no quede memoria libre se detecta en Modula2 mediante la función `Available` cuya declaración es:

```
PROCEDURE Available(size:CARDINAL): BOOLEAN;
```

En el Programa 4.10 se muestra la realización de estas funciones auxiliares de pilas implementadas como una lista enlazada.

Programa 4.10 Implementación de las operaciones auxiliares de la pila empleando una lista enlazada.

```
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_llena;

PROCEDURE Consultar_pila (VAR pila: Tipo_pila; VAR dato : Tipo_datos);
BEGIN
   dato := pila^.datos;
END Consultar_pila;
```

4.4 COLAS

Una cola se suele definir simplemente como una estructura de datos en la que el primer elemento en entrar es el primero en salir. A estas estructuras se les denomina FIFO (*First In, First Out*). Como ocurría con las pilas, es una estructura en la que los elementos están ordenados, se añaden por un extremo, denominado *final de cola*, pero en este caso se suprimen por el otro, conocido como *principio de cola*.

Un ejemplo típico es la cola de un cine. El primero que llega a la cola es el primero que compra la entrada y sale de la cola. Podría pensarse en introducirse por el medio, pero colarse no está permitido. Además, debe observarse que tampoco está permitido, por la definición de cola, abandonarla si no se está al principio. Por tanto, si se desiste de comprar la entrada y se sale en una posición intermedia la estructura ya no es una cola, puesto que no satisface su definición. Al igual que con las pilas, las colas responden fielmente al concepto cotidiano de cola y todo el mundo conoce sus reglas y las respeta. Si en una cola, no respetamos sus reglas, con toda seguridad seremos fuertemente increpados y obligados a someternos a las reglas establecidas. Nuevamente puede observarse que la estructura cola es dinámica, ya que el número de elementos que la componen es variable.

Aplicaciones típicas de colas, aunque hay muchas más, son todas aquellas en las que debamos sincronizar la velocidad de llegada de datos con la velocidad de procesamiento. Todos hemos empleado alguna vez el término cola de impresión. Es un ejemplo típico en el que el sistema operativo de una máquina forma una cola con los trabajos a imprimir y son atendidos por riguroso orden de llegada. En este caso, podemos suponer que la velocidad de impresión es uniforme pero en determinados momentos pueden llegar los trabajos con mayor frecuencia. Es exactamente la misma situación que en el cine, la velocidad de servir entradas puede considerarse constante, pero la afluencia de espectadores es variable y durante algún tiempo superior a la primera.

Como sucedía con las pilas, las colas pueden encontrarse definidas en la literatura orientada a la formación en principios básicos de programación estructurada, como una lista enlazada con acceso FIFO. Las consideraciones realizadas para las pilas en relación a este hecho son igualmente aplicables en este caso.

Al igual que en el caso de las pilas, esta definición inicial de la cola sólo resalta la característica más sobresaliente del TDA cola, su acceso FIFO, y definirla como una lista enlazada sólo hace referencia a una manera de implementarla.

En sentido estricto, que es el único correcto desde el punto de vista de la abstracción y el encapsulamiento, el TDA cola se define de la siguiente manera:

Definición 4.3: Una cola es un TDA dinámico homogéneo con acceso de tipo FIFO (primero en entrar - primero en salir), al que sólo se tiene acceso al principio de la cola para sacar elementos, y al final de la misma para meterlos. Los operadores básicos asociados son `Meter` y `Sacar` elementos de la cola, y los operadores auxiliares asociados son:

- `Inicia_cola`: crea una cola o limpia una existente inicializándola a una cola vacía, que es aquella que no contiene elementos. Permite partir de un estado previamente establecido.
- `Cola_Vacia`: operación utilizada para consultar si la cola está vacía.
- `Cola_Llena`: operación utilizada para consultar si la cola está llena.
- `Consultar_Cola`: operación utilizada para consultar el contenido del principio de la cola sin sacar el elemento de la misma.

La operación `Cola_Vacia` será necesaria cuando se vaya a sacar un elemento de la cola ya que no se pueden sacar elementos de una cola vacía, y `Cola_Llena` cuando se vaya a meter un elemento en la cola, ya que si está llena no se puede introducir más elementos en ella.

La operación `Consultar_Cola` se utiliza para consultar el último elemento introducido en la cola sin sacarlo de la misma. Esta operación, como ocurría en el TDA pila, no infringe la característica de la cola que indica que *sólo* se tiene acceso por el principio o final de ella. Análogamente podría definirse una operación que consultase el inicio de la cola.

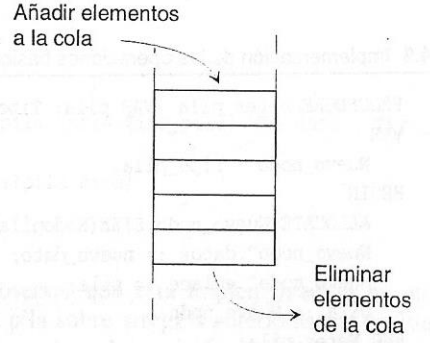

Figura 4.13 Inserción y la eliminación de los elementos de una cola.