```
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.

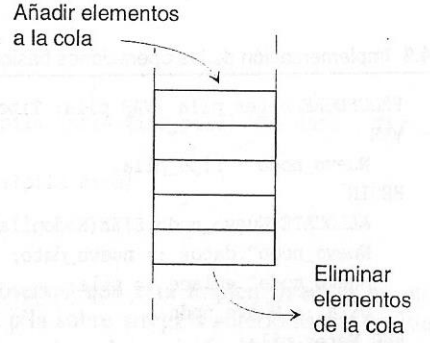

Figura 4.13 Inserción y la eliminación de los elementos de una cola.

La Figura 4.13 muestra la inserción y la eliminación de los elementos de una cola.

Con esta definición del TDA cola, no existe ambigüedad alguna ya que todas las funciones a realizar deberán implementarse en función de sus operaciones básicas y auxiliares.

Como ocurría con las pilas, el carácter dinámico de las colas es conceptual y se debe a que el tamaño de la cola depende de la cantidad de elementos almacenados en la misma, y por tanto, es variable independientemente de su implementación. Seguidamente se presentan las implementaciones mediante arreglos y mediante listas enlazadas.

4.4.1 IMPLEMENTACIÓN DE COLAS MEDIANTE ARREGLOS

La cola es un TDA homogéneo, por lo que es factible su implementación mediante arreglos. Seguidamente se exploran diversas implementaciones con el fin de analizar la más adecuada.

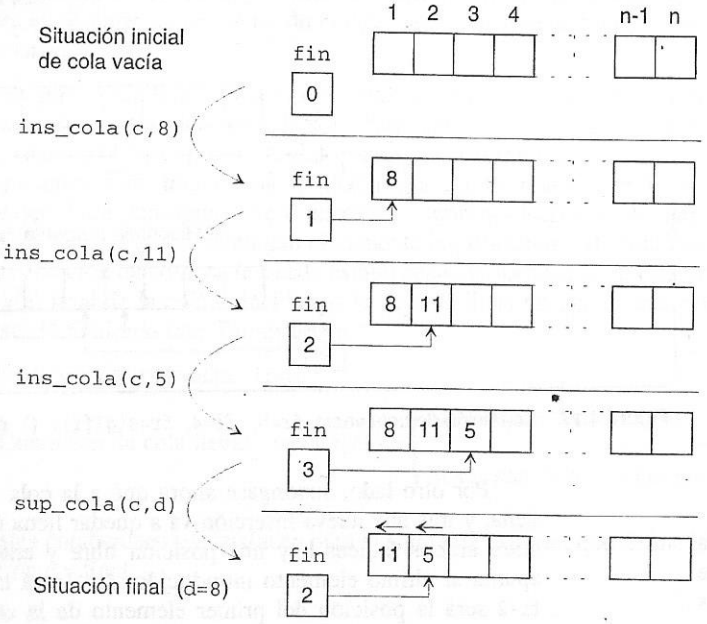

Figura 4.14 Cola con un único índice (`fin` = índice de final de cola).

En una primera implementación con arreglos se podría considerar la utilización de un único índice para almacenar el final, presuponiendo que el primer elemento de la cola está siempre en la primera posición del arreglo. Sin embargo, esta estructura es inadecuada por ineficaz ya que al sacar un elemento por el principio de la cola

(posición 1 del arreglo) deberemos seguidamente mover todos los elementos del arreglo una posición hacia el principio, tal y como se ilustra en la Figura 4.14. Este elevado coste computacional hace que la estructura no sea admisible, aunque 'funcione'. La situación en la cola del cine es básicamente la misma. Sin embargo, podemos pensar que se trata de un sistema en el que cada elemento dispone de su propio procesador para realizar el desplazamiento. Supóngase que alguien tuviese que encargarse de hacer avanzar un paso a los espectadores de la cola, con toda seguridad requeriría mucho más esfuerzo. Si, como sucede con los elementos de la cola, los espectadores no dispusieran de movilidad propia, sería mucho más eficiente que fuese la persona que sirve las entradas la que avanza sobre la cola de forma que bastaría mover un único elemento.

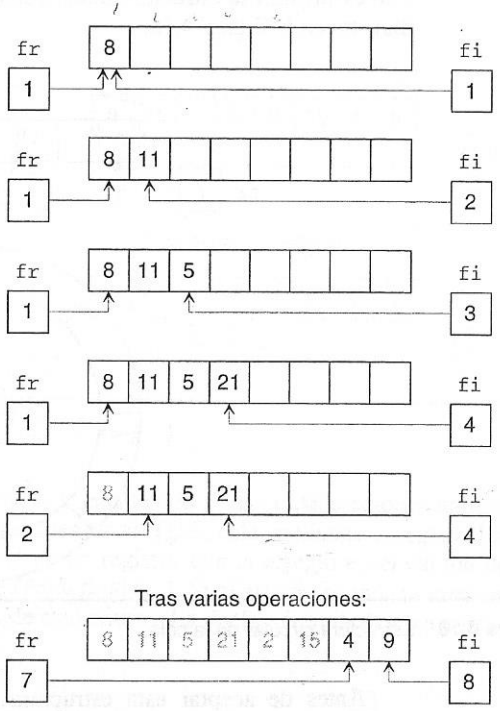

Figura 4.15 Implementación de la cola con dos índices. Obsérvese la situación final: hay muchos elementos libres pero, según el criterio de índices dado, no se puede seguir insertando.

Por tanto, es conveniente utilizar dos índices, uno que indique el final (`fi`) y otro el frente (`fr`) de la cola. Puede pensarse en adoptar como *criterio* que el final, `fi`, indique el último elemento introducido, aumentando cada vez que se introduce un nuevo elemento en la cola, y que frente, `fr`, apunte al primer elemento a sacar, aumentando al eliminar un elemento. Con esta implementación, las posiciones del arreglo en las que se han introducido elementos y posteriormente se han sacado que-

dan inutilizadas, debido al hecho de que los índices no se decrementan nunca. Este comportamiento que se ilustra en la Figura 4.15 hace que se alcance el final del arreglo y no se puedan seguir añadiendo elementos aunque haya numerosas posiciones libres al comienzo del mismo. Si se piensa en solucionar este problema decrementando los índices para reutilizar esas posiciones, esto implica mover todos los elementos de la cola, como sucedía en el caso de la estructura con un único índice y, por los mismos motivos que antes, esta opción debe desestimarse.

Para aprovechar las posiciones del arreglo que han sido utilizadas y están disponibles, se vuelve a comenzar por el principio una vez que hayamos llegado al final. De esta manera estamos considerando el arreglo como si fuera circular, de tal forma que el elemento que sigue al último es el primero. Lo que se consigue con esto es sustituir la estructura lineal del arreglo por una de tipo circular, tal y como se muestra en la Figura 4.16.

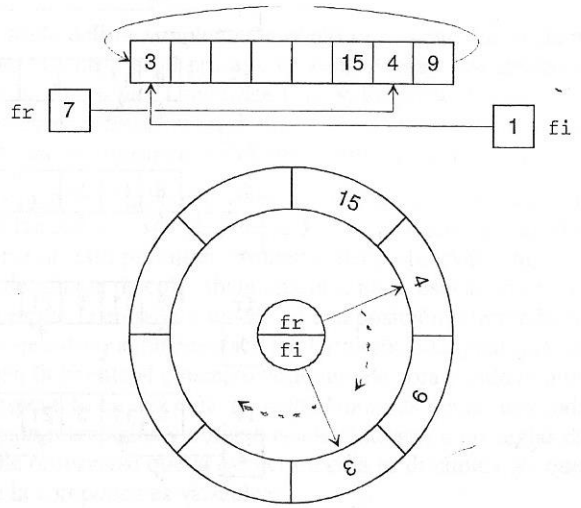

Figura 4.16 Estructura circular de la cola.

Antes de aceptar esta estructura debe analizarse el comportamiento de la misma. Las operaciones `Meter` y `Sacar` elementos no presentan ningún inconveniente ni en cuanto a economía de almacenamiento (como ocurría en la opción con dos índices y arreglo lineal) ni en relación con el coste computacional (como ocurría en la opción con un único índice).

Sin embargo, debe tenerse en cuenta que con la estructura circular la cola puede quedar vacía o llena en posiciones intermedias del arreglo. Analicemos detenidamente cómo detectar si la cola está vacía o llena.

Recuérdese que el *criterio* adoptado para los apuntadores es:

- Final (`fi`) apunta al último elemento introducido, y aumenta cada vez que se introduce un nuevo elemento en la cola.
- Frente (`fr`) apunta al primer elemento a sacar, y aumenta al eliminar un elemento de la cola.

Supóngase que la cola tiene un único elemento y va a quedar vacía. En esta situación ambos índices tienen el mismo valor (`fr=fi`) ya que el último elemento introducido y el primero a sacar son el mismo. Cuando el elemento es eliminado de la cola, el índice frente (`fr`) se incrementa. Esta situación se ilustra en la Figura 4.17. Por tanto, la condición de cola vacía será tal que el frente esté apuntando al siguiente elemento al que apunta final. Este siguiente elemento ofrece dos alternativas según sea el último del arreglo o sea otro cualquiera. Si es un elemento cualquiera distinto del último, el siguiente se obtiene incrementando en 1, si por el contrario es el último elemento su siguiente será el primero:

Condición de cola vacía: `fr=sig(fi)` =
- Si `fi=MAXCOLA`: `fr=1`
- Si `fi≠MAXCOLA`: `fr=fi+1`

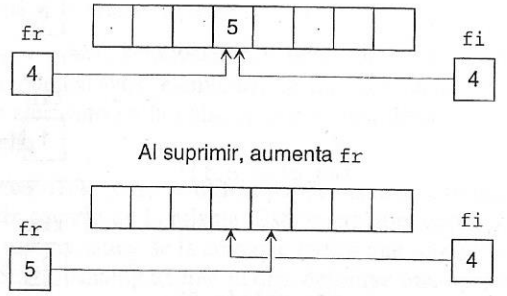

Figura 4.17 Condición de cola vacía: `fr=5, fi=4, fr=sig(fi): (1 o fi+1)`.

Por otro lado, supóngase ahora que a la cola le falta un elemento para estar llena, y tras una nueva inserción va a quedar llena (Figura 4.18). En esta situación entre ambos índices hay una posición libre y entonces `fr=fi+2`, ya que el final apunta al último elemento introducido, `fi+1` será la posición que aún está libre y `fi+2` será la posición del primer elemento de la cola, y por tanto el frente de la misma. Cuando un nuevo elemento es introducido en la cola, el índice final (`fi`) se incrementa, quedando la cola llena. Por tanto, la condición de cola llena será:

Condición de cola llena: `fr=sig(fi)` =
- Si `fi=MAXCOLA`: `fr=1`
- Si `fi≠MAXCOLA`: `fr=fi+1`

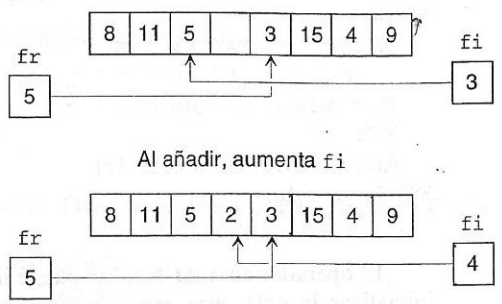

Figura 4.18 Condición de cola llena, `fi=4` y `fr=5`, `fr=sig(fi)`: (`1` o `fi+1`).

En conclusión, con el criterio adoptado para los apuntadores, tal que final (`fi`) apunta al último elemento introducido, y aumenta cada vez que se introduce un nuevo elemento en la cola, y frente (`fr`) apunta al primer elemento a sacar, y aumenta al eliminar un elemento de la cola, *no es posible distinguir si la cola está llena o vacía.*

Una primera solución que puede considerarse es mantener un contador que indique los elementos que hay en la cola. Esta solución lleva a la necesidad de incrementar este contador con cada nueva inserción y decrementarlo con cada eliminación. Siempre será mejor una solución que evite realizar estas operaciones adicionales. Para conseguirlo será necesario cambiar el criterio de asignación de índices, de manera que se distingan claramente las situaciones de cola llena y vacía. Así, la condición de cola vacía puede establecerse de forma que en esta situación el frente y el final de la cola coincidan, y la de cola llena tal que en esta situación el frente sea el final más uno. En resumen:

Condición de cola vacía: `fr=fi`

$$\text{Condición de cola llena: } \texttt{fr=sig(fi)} = \begin{cases} \text{Si } \texttt{fi=MAXCOLA:} & \texttt{fr=1} \\ \text{Si } \texttt{fi}\neq\texttt{MAXCOLA:} & \texttt{fr=fi+1} \end{cases}$$

Estas condiciones se satisfacen para el siguiente *criterio*, que define los apuntadores frente y final.

Criterio 1

- El frente, `fr`, apunta al elemento anterior al primero de la cola.
- El final, `fi`, apunta al último elemento introducido en la cola.

Obsérvese que frente apunta a un elemento que nunca se rellena, con lo que cuando la cola está llena se está dejando una posición libre en el arreglo.

La Figura 4.19 muestra la situación en la que la cola queda vacía y la Figura 4.20 muestra la cola llena.

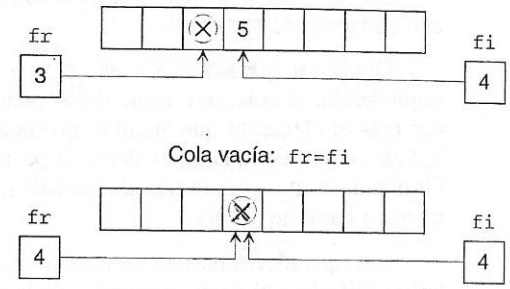

Figura 4.19 Cola vacía con `fr=4` y `fi=4` (`fr=fi`).

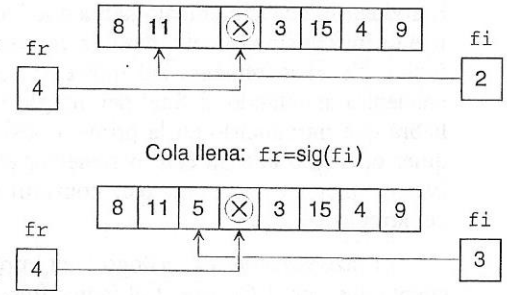

Figura 4.20 Cola llena con `fr=4` y `fi=3` `fr=sig(fi)`.

Con el *Criterio* 1 y atendiendo a las consideraciones anteriores ya se está en disposición de implementar el TDA cola mediante arreglos. En primer lugar, la estructura consistirá en un registro con el arreglo en el campo de datos, y los dos apuntadores frente (`fr`) y final (`fi`). Si el arreglo se indexa mediante enteros los dos apuntadores serán de tipo entero. La declaración de la estructura en Modula2 es la siguiente:

```
CONST MAXCOLA=MAXCOLAN+1;
TYPE
   TipoIndice = [1..MAXCOLA];
   Tipo_cola = RECORD
                  datos : ARRAY TipoIndice OF Tipo_datos;
                  fr,fi : TipoIndice;
               END;
VAR cola: Tipo_cola;
```

En estas declaraciones MAXCOLAN es el máximo número de elementos que pueden almacenarse en la cola.

Obsérvese que MAXCOLA = MAXCOLAN + 1 es el tamaño necesario del arreglo para implementar la cola, que viene determinado por el número de elementos a almacenar más el elemento que siempre quedará libre. El apuntador fi es el índice que indica el final de la cola, es decir, la posición del arreglo en la que se introdujo el último elemento, y fr indica el frente de la cola, es decir, la posición anterior al elemento a sacar de la cola.

Los operadores básicos se implementan de forma inmediata atendiendo al criterio utilizado y al carácter circular de la cola. Así, habrá que considerar dos situaciones, una cuando las operaciones se realizan en los extremos del arreglo sobre el que se implementa la cola, y otra cuando se realizan en posiciones intermedias del mismo. De este modo, para implementar el operador Meter_cola, se manipulará el índice de final (fi). Como fi apunta al último elemento introducido en la cola, para introducir el nuevo elemento habrá que incrementar primero el final, fi, y seguidamente introducir el nuevo dato en la posición que esté siendo apuntada por este índice. En el incremento del índice fi se deben considerar dos casos: si éste se encuentra apuntando al final del arreglo (cola.fi = MAXCOLA), el nuevo elemento habrá que introducirlo en la primera posición del mismo (cola.fi:=1), y en cualquier otro caso bastará con incrementar el índice fi en 1 (cola.fi := cola.fi+1). Esta distinción es necesaria para convertir en circular la estructura puramente lineal del arreglo.

El razonamiento es análogo para implementar el operador Sacar. Teniendo en cuenta que por el *Criterio 1* el índice frente (fr) apunta al elemento anterior al primero de la cola, para sacar un elemento habrá que incrementar primero el índice fr y posteriormente asignarlo. De nuevo, si el índice se encuentra en el final del arreglo (cola.fr = MAXCOLA) el primer elemento de la cola estará en la primera posición del arreglo (cola.fr:=1), y si está en las posiciones intermedias del arreglo basta con incrementar el índice fr en 1 (cola.fr := cola.fr+1).

El Programa 4.11 muestra la implementación en Modula2 de los operadores básicos de la cola implementada estáticamente.

Programa 4.11 Implementación de los operadores básicos de la cola implementada estáticamente.

```
PROCEDURE Meter_cola (VAR cola : Tipo_cola; nuevo_dato : Tipo_datos);
BEGIN
   IF cola.fi = MAXCOLA THEN
      cola.fi := 1
   ELSE cola.fi := cola.fi+1
   END;
   cola.datos[cola.fi] := nuevo_dato
END Meter_cola;
```



```
PROCEDURE Sacar_cola (VAR cola: Tipo_cola; VAR dato : Tipo_datos);
BEGIN
   IF cola.fr = MAXCOLA THEN
      cola.fr := 1
   ELSE cola.fr := cola.fr + 1
   END;
   dato := cola.datos[cola.fr]
END Sacar_cola;
```

El operador auxiliar Iniciar puede implementarse de muy diversas formas. Al inicializar la cola, ésta estará vacía y por tanto, debe satisfacerse que cola.fr= cola.fi. Dado que la estructura del arreglo es circular, cualquier elemento es igualmente válido para imponer esta condición, de ahí las múltiples implementaciones posibles de este operador.

Si se adopta el criterio de que partiendo de una cola vacía, el primer elemento se introducirá en la primera posición del arreglo, entonces con la cola vacía, fi deberá apuntar a MAXCOLA para que al incrementarse para introducir el nuevo elemento, apunte a la primera posición del arreglo. Y como la condición de cola vacía exige fr = fi, entonces la cola se inicializará con:

cola.fr=cola.fi=MAXCOLA.

El operador de consulta Cola_Vacia se implementa sencillamente comprobando si fr=fi, y Cola_Llena comprobará la condición cola.fr=cola.fi+1 en las posiciones intermedias del arreglo, y en el extremo del arreglo, cuando cola.fi=MAXCOLA, la cola estará llena si cola.fr=1.

Los operadores de consulta al principio y al final de la cola se implementan de forma análoga. Las implementaciones estáticas en Modula2 de los operadores auxiliares de la cola se muestran en el Programa 4.12.

Programa 4.12 Implementación de los operadores auxiliares de la cola implementada estáticamente.

```
PROCEDURE Inicia_cola (VAR cola: Tipo_cola);
BEGIN
   cola.fr := MAXCOLA;
   cola.fi := MAXCOLA
END Inicia_cola;


PROCEDURE Cola_vacia (cola:Tipo_cola) : BOOLEAN;
BEGIN
   RETURN cola.fi = cola.fr;
END Cola_vacia;
```

```
PROCEDURE Cola_llena (cola:Tipo_cola) : BOOLEAN;
BEGIN
   IF cola.fi = MAXCOLA THEN RETURN cola.fr =1
   ELSE RETURN cola.fr = cola.fi + 1;
   END
END Cola_llena;


PROCEDURE Consultar_final_cola (cola : Tipo_cola; VAR dato : Tipo_datos);
BEGIN
   dato := cola.datos[cola.fi];
END Consultar_final_cola ;

PROCEDURE Consultar_frente_cola (cola: Tipo_cola; VAR dato : Tipo_datos);
BEGIN
   IF cola.fr = MAXCOLA THEN
      dato := cola.datos[1]
      ELSE dato  := cola.datos[cola.fr + 1]
   END;
END Consultar_frente_cola;
```

La implementación desarrollada, que se basa en el *Criterio 1* no es la única válida. En efecto, las condiciones de distinguibilidad de cola vacía y cola llena también se satisfacen con el siguiente criterio de definición de los apuntadores frente y final:

Criterio 2

- El frente, `fr`, apunta al primer elemento de la cola.
- El final, `fi`, apunta al elemento siguiente al último introducido en la cola.

La Figuras 4.21 y 4.22 ilustran este criterio.

Como puede observarse los índices del *Criterio 2* son los del *Criterio 1* incrementados una posición. A continuación recordamos el primer criterio con el fin de observar este hecho:

Criterio 1

- El frente, `fr`, apunta al elemento anterior al primero de la cola.
- El final, `fi`, apunta al último elemento introducido en la cola.

La implementación de los operadores básicos y auxiliares es diferente según el criterio que se decida emplear, aunque ambos satisfagan la misma condición de distinguibilidad de cola llena y cola vacía. Sin embargo, una vez desarrollada la implementación de los operadores, las tareas a desarrollar con la cola se realizan utilizándolos sin atender a sus detalles de implementación.

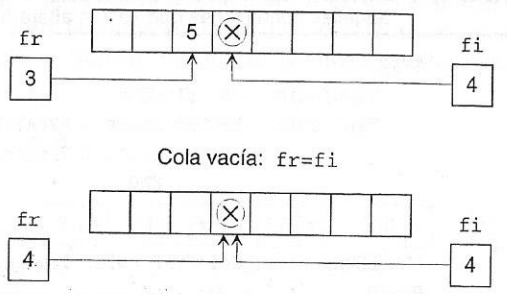

Figura 4.21 Cola vacía con `fr=4` y `fi=4` (`fr=fi`).

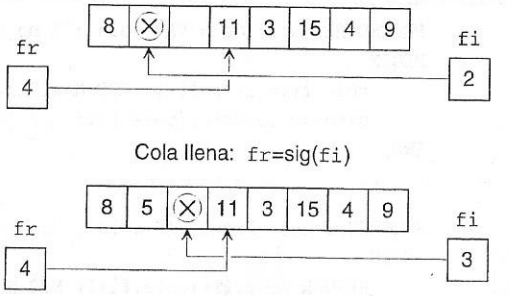

Figura 4.22 Cola llena con `fr=4` y `fi=3` (`fr=fi+1`).

Antes de pasar a comentar la implementación de las colas con asignación de memoria dinámica basadas en listas enlazadas, veamos una forma alternativa y muy habitual de recircular los índices de frente y final de una cola. Para ello se hace uso del operador resto de una división entera y se define el arreglo que contiene los datos entre `0` y `MAXCOLA-1`. En este caso el siguiente elemento se calcula con `sig:=(actual+1) MOD MAXCOLA`. Esto es válido para todos los elementos incluído el último y no es necesario considerar este último elemento de distinta forma al resto. Supongamos como ejemplo que `MAXCOLA=10` y que el índice de final vale 7. Al aplicar la expresión anterior obtendríamos (7+1) MOD 10=8, que es equivalente a 7+1. Supongamos ahora que está en la última posición del arreglo (`fi=9`). Si aplicamos nuevamente la misma expresión tendremos: (9+1) MOD 10=0, con lo que el índice de final pasa a apuntar a la primera posición tal y como se requiere para recircular los índices a lo largo del arreglo.

El Programa 4.13 muestra las declaraciones necesarias y la implementación de las operaciones fundamentales que deben ser reescritas para emplear esta alternativa de recirculación de índices.

Programa 4.13 Implementación de una cola circular empleando el operador módulo. Sólo se muestran aquellas operaciones que se ven alteradas por esta nueva estrategia de recirculación.

```
TYPE
    TipoIndice = 0..MAXCOLA;
    Tipo_cola = RECORD datos : ARRAY[0..MAXCOLA-1] OF Tipo_datos;
                       fr,fi : TipoIndice;
                       END;

PROCEDURE Meter_Cola(VAR cola: Tipo_cola; nuevo : Tipo_datos);
BEGIN
     cola.fi:=(cola.fi+1) MOD MAXCOLA;
     cola.datos[cola.fi]:=nuevo;
END;

PROCEDURE Sacar_Cola(VAR cola : Tipo_cola; VAR dato : Tipo_datos);
BEGIN
     cola.fr:=(cola.fr+1) MOD MAXCOLA;
     dato:=cola.datos[cola.fr];
END;

PROCEDURE Cola_Llena(VAR cola : Tipo_cola) : BOOLEAN;
BEGIN
     RETURN cola.fr=(cola.fi+1) MOD MAXCOLA;
END;
```

4.4.2 IMPLEMENTACIÓN DE COLAS MEDIANTE LISTAS ENLAZADAS

En principio, puede pensarse en utilizar una lista enlazada para implementar una cola, empleando operaciones análogas a insertar por la cabeza y suprimir por el final (el frente de la cola sería el final de la lista y el final de la cola sería el principio de la lista), o bien insertar por el final y suprimir por la cabeza (el frente de la cola sería el principio de la lista y el final de la cola sería el final de la lista). En cualquier caso, con una lista enlazada, la inserción y la supresión por el final requiere un bucle hasta alcanzar este extremo. Esto significa que siempre una operación básica (`Sacar` o `Meter`) tiene un número de comparaciones y asignaciones de punteros igual a la cantidad de elementos que tenga la cola.

La solución es sencilla, y consiste en modificar la lista enlazada dotándola de un puntero externo más, que apunte al final de la misma. Esta estructura ya fue presentada en la Figura 4.5.

La declaración de la cola dinámica en Modula2 será, por tanto:


```
TYPE Ptr_nodo = POINTER TO Nodocola;
   Nodocola= RECORD
         datos : Tipo_datos;
         enlace : Ptr_nodo;
      END;
   Tipo_cola = RECORD
         frente,final : Ptr_nodo;
      END;
```

Ahora se nos plantea la cuestión de ver qué función asignamos a cada uno de los extremos de la lista. Es decir, decidir si añadimos por el principio de la lista y eliminamos por el final o lo hacemos a la inversa. En principio, parecen situaciones simétricas, pero no lo son, ya que los punteros apuntan desde el inicio hacia el final pero no a la inversa.

Consideremos que el puntero externo `frente` apunta al principo de la lista y el puntero externo `final` apunta al último elemento. Debe observarse que, con esta estructura de lista enlazada, no se puede suprimir por el final. Para poder hacerlo, se debería asignar el valor `NIL` al penúltimo elemento, pero su dirección no es conocida a partir del último nodo o del puntero externo `final`. Para conseguirlo, habría que emplear un bucle que recorriese toda la lista desde el principio, estando en la misma situación que una lista con un único puntero, con su considerable costo computacional. Otra alternativa sería una lista doblemente enlazada, pero sólo se necesita un enlace doble y de esta forma tendríamos todos los nodos doblemente enlazados con el consiguiente gasto de memoria y de actualización de punteros. Sin embargo, si añadimos por el final de la lista y eliminamos por el principio de la misma, no se plantea este problema. En conclusión, la operación `Meter` debe realizarse al final de la lista, y la operación `Sacar` al principio, lo que por supuesto no es ningún inconveniente.

Como es habitual, es necesario establecer un criterio para determinar de forma única la situación en que la cola está vacía. Estableciendo como criterio que la cola está vacía cuando el puntero `final` sea `NIL`, la inicialización de la cola se realiza simplemente asignando `NIL` a `cola.final`.

La operación de `Meter` se realizará considerando el puntero `final`. En primer lugar se crea un nuevo nodo, a continuación se introduce el nuevo dato en su campo `datos`, y se establece `NIL` en su enlace, indicando que será el último elemento. Estos tres primeros pasos se muestran en las Figuras 4.23 y 4.24. Seguidamente se distinguen dos situaciones: el caso general en que la cola ya contiene elementos y el caso especial en el que esté inicialmente vacía.

Si la cola ya contiene elementos (Figura 4.23) se enlaza el último nodo de la lista con el nuevo nodo (`cola.final^.enlace:=Nuevo_nodo`). Sin embargo, si la cola está vacía (Figura 4.24) no puede hacerse esto. En este caso es el puntero externo `frente` el que debe apuntar a este nuevo nodo.

Obsérvese que, en principio, el puntero `frente` sólo se modifica al eliminar y el puntero `final` sólo al añadir. Sin embargo, como `frente` apunta al primer elemento que saldrá de la cola, si el que se inserta es el primero (y por tanto único), habrá que hacer que `frente` apunte a él.

Por último, se asigna la dirección del nuevo nodo al puntero externo `final` independientemente del estado inicial de la cola. La Figura 4.23 ilustra la inserción en una cola de este tipo que ya contiene elementos y en la Figura 4.24 se muestra la misma acción pero sobre una cola inicialmente vacía. En estas figuras, `FR` y `FI` representan los campos `frente` y `final` del registro que forma la cola.

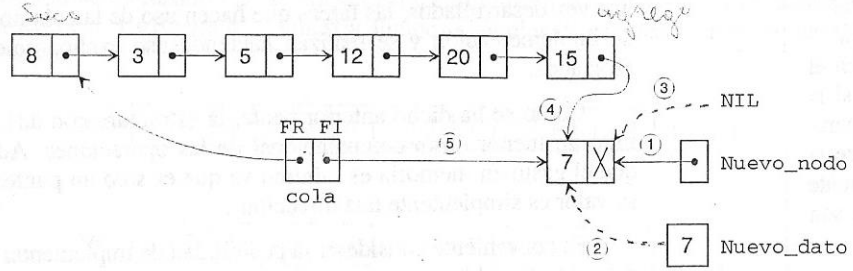

Figura 4.23 Inserción en una cola no vacía implementada con asignación dinámica de memoria.

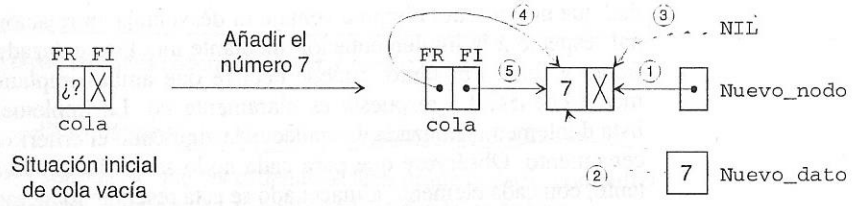

Figura 4.24 Inserción en una cola inicialmente vacía con asignación dinámica de memoria.

La operación `Sacar` toma los datos de la variable referenciada a la que apunta `cola.frente` y asigna el puntero externo `frente` al siguiente nodo (`cola.frente:= cola.frente^.enlace;`). Nuevamente hay que distinguir la situación de elemento único (al sacarlo la cola quedará vacía) de la situación en que la lista tiene más de un elemento. Por tanto, es necesario comprobar si tras la eliminación la cola ha quedado vacía, en cuyo caso debe asignarse `NIL` al puntero externo `final`, indicativo de este hecho para posteriores operaciones. Como puede verse la situación es prácticamente simétrica a la que se tenía al insertar: sólo se actúa sobre el puntero `final` si el elemento eliminado es el último, en caso contrario basta con manipular adecuadamente el puntero `frente`. Finalmente, se libera el nodo con la ayuda de un puntero auxiliar (`aux`) que se habrá asignado al último elemento como primer paso de la eliminación. La Figura 4.25 muestra la eliminación en una cola con más de un elemento y la Figura 4.26 la eliminación en una cola con un único elemento. El Programa 4.14 muestra la implementación en Modula2 de estos operadores básicos.

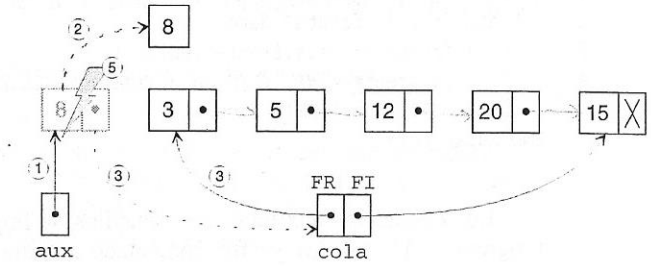

Figura 4.25 Eliminación de un elemento no único en una cola implementada dinámicamente.

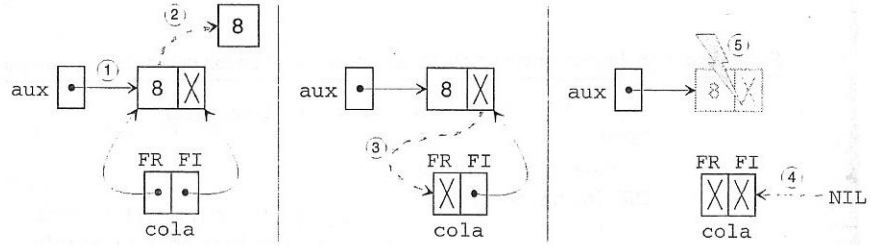

Figura 4.26 Eliminación del último elemento en una cola implementada dinámicamente. En este caso, el paso 3 no es necesario pero se mantiene por ser común con el caso general.

Programa 4.14 Implementación en Modula2 de los operadores básicos `Meter` y `Sacar` para la cola con implementación dinámica.

```
   PROCEDURE Meter_cola (VAR cola : Tipo_cola; nuevo_dato : Tipo_datos);
   VAR
      Nuevo_nodo : Ptr_nodo;
   BEGIN
1     ALLOCATE(Nuevo_nodo,SIZE(Nodocola));
2     Nuevo_nodo^.datos := nuevo_dato;
3     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;
```

```
   PROCEDURE Sacar_cola (VAR cola: Tipo_cola; VAR dato : Tipo_datos);
   VAR
      aux : Ptr_nodo;
   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 Sacar_cola;
```

Los operadores auxiliares son sencillos de implementar, y se muestran en el Programa 4.15. `Iniciar` ya fue comentado anteriormente. La consulta sobre si la cola está vacía se lleva a cabo siguiendo el criterio adoptado mediante la comparación del puntero externo `final` con `NIL`, y las consideraciones realizadas sobre la operación `Pila_llena` en la implementación dinámica de las pilas son igualmente aplicables en este caso. Las operaciones de consulta en los extremos de la cola son evidentes.

Programa 4.15 Implementación dinámica en Modula2 de los operadores auxiliares para la cola.

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


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));
END Cola_llena;

PROCEDURE Consultar_frente_cola (cola: Tipo_cola; VAR dato : Tipo_datos);
BEGIN
   dato := cola.frente^.datos;
END Consultar_frente_cola;
```



```
PROCEDURE Consultar_final_cola (cola: Tipo_cola; VAR dato : Tipo_datos);
BEGIN
   dato := cola.final^.datos;
END Consultar_frente_cola;
```

Obsérvese que en esta implementación se ha tomado como criterio de cola vacía que el puntero `final` sea `NIL`. Análogamente podría haberse establecido como criterio que la cola estuviese vacía cuando el puntero `frente` apunte a `NIL`. Esto obliga a una implementación diferente de los operadores asociados, pero de nuevo, una vez desarrollados, las tareas que hacen uso de la cola no atienden a los detalles de implementación y se realizan haciendo uso exclusivamente de los operadores asociados.

Como se ha dicho anteriormente, la estructura con dos punteros externos permite un menor coste computacional de las operaciones. Además debe observarse que el gasto en memoria es mínimo ya que es sólo un puntero más (y recordar que su valor es simplemente una dirección).

Es conveniente considerar la posibilidad de implementar una cola haciendo uso de una lista doblemente enlazada con dos punteros externos (`frente` y `final`). Evidentemente ahora ya no se tiene el problema del costo del bucle para realizar una de las operaciones, como pasaba en la lista enlazada simple, ni tampoco la dificultad para realizarla insertando al principio y suprimiendo al final, tal y como sucedía en la lista enlazada con dos punteros externos al frente y al final. Pero también es verdad que no presenta ninguna ventaja ni desventaja en relación al coste computacional respecto a la implementación mediante una lista enlazada con los dos punteros frente y final. Por tanto, ¿puede decirse que ambas implementaciones son igualmente buenas? La respuesta es claramente no. La implementación mediante una lista doblemente enlazada es inadecuada siguiendo el criterio de economía de almacenamiento. Obsérvese que para cada nodo se ha declarado un puntero más y por tanto, con cada elemento almacenado se está reservando memoria que no se necesita para implementar la cola. Además, al tener dos enlaces por nodo, la actualización de punteros requerirá el doble de operaciones. Por tanto, sería una solución incorrecta, aunque 'funcione'.

En conclusión, la implementación desarrollada mediante una lista enlazada con dos punteros externos, `frente` y `final`, es la más adecuada.

4.5 EJEMPLOS DE APLICACIÓN

4.5.1 EVALUACIÓN DE LA PARENTIZACIÓN DE EXPRESIONES

Las expresiones aritméticas, como es bien sabido de matemáticas básicas, se representan habitualmente de manera que los operadores se encuentran entre los operandos. Así, la suma de los operandos A y B se expresa como $A+B$. Además, se