COLAS

2002

GRUPO # 22

Alumnos:

Aguilar Elba
Barrios Miguel
Camacho Yaquelin
Ponce Rodríguez Jhonny

ESTRUCTURA DE DATOS

TEMA 4

Estructura de datos Cola

ÍNDICE

4.1. Definición y ejemplos 3
4.2. El TAD Cola 4
4.3. Implementaciones del tipo Cola 4
4.3.1. Implementación mediante vectores 5
4.3.2. Implementación mediante punteros 15
4.4. Ejercicios 18

BIBLIOGRAFÍA

- (Joyanes y Zahonero, 1998), Cap. 8.
- (Joyanes y Zahonero, 1999), Cap. 7.
- (Dale y Lilly, 1989), Cap. 5.
- (Horowitz y Sahni, 1994), Cap. 3.

OBJETIVOS

- Conocer el concepto, funcionamiento y aplicaciones del tipo Cola.
- Conocer el tipo abstracto de datos Cola y sus operaciones asociadas.
- Saber implementar el TAD Cola mediante variables estáticas de tipo vector. Conocer y saber aplicar para ello el concepto de vector circular.
- Saber implementar el TAD Cola mediante el uso de variables dinámicas y punteros.
- Conocer las ventajas e inconvenientes de las distintas implementaciones del TAD.

4.1. Definición y ejemplos

El concepto de cola es ampliamente utilizado en la vida real. Cuando nos situamos ante la taquilla del cine para obtener nuestra entrada, o cuando esperamos en el autoservicio de un restaurante solemos hacerlo en una cola. Esto significa que formamos una fila en la que el primero que llega es el primero en obtener el servicio y salir de la misma. Esta política de funcionamiento se denomina FIFO (First In First Out), es decir, el primer elemento en entrar es el primer elemento en salir.

En la vida real puede perfectamente ocurrir que alguien pretenda saltarse su turno en una cola, o incluso que abandone la misma antes de que le toque el turno. Sin embargo, en ambos casos se está incumpliendo la política de funcionamiento de la cola y, estrictamente hablando, ésta deja de serlo.

En el ámbito de las estructuras de datos definiremos una cola del siguiente modo:

Definición

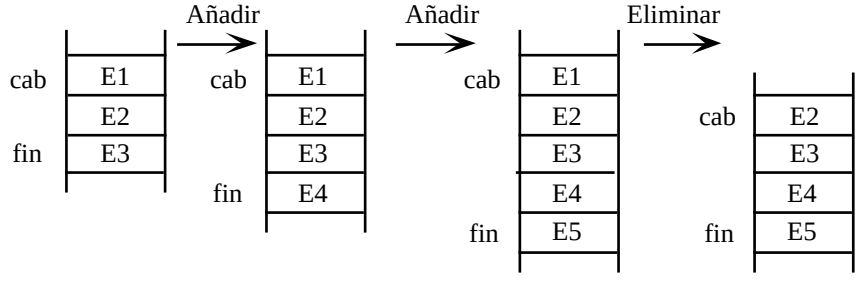
*Una cola es un conjunto ordenado de elementos homogéneos, en el cual los elementos se eliminan por uno de sus extremos, denominado **cabeza**, y se añaden por el otro extremo, denominado **final**. Las eliminaciones y añadidos se realizan siguiendo una política FIFO.*

Cuando hablamos de un conjunto ordenado, al igual que ocurre con las pilas, nos referimos a la disposición de sus elementos y no a su valor. Esto es, los elementos no tienen porque estar ordenados según su valor, sino que cada uno de ellos, salvo el primero y el último, tiene un anterior y un siguiente. Por otro lado, al decir que los elementos de la cola son homogéneos, queremos decir que son del mismo tipo base, aunque sin establecer ninguna limitación sobre este tipo.

Finalmente decir que en la literatura sobre estructuras de datos la cabeza de una cola suele denominarse también principio o frente de la misma, y el final suele denominarse fondo.

Al igual que las pilas, las colas se gestionan añadiendo y borrando elementos de las mismas. En este caso en particular las dos operaciones básicas de manipulación funcionan del siguiente modo:

- **Añadir:** Añade un elemento al final de la cola.
- **Eliminar:** Elimina un elemento de la cabeza de la cola.



La figura anterior muestra la evolución de una cola al aplicar sucesivamente las operaciones Añadir y Eliminar. Podemos ver que se trata de una estructura dinámica, ya que su tamaño cambia a medida que se añaden y eliminan elementos de la misma.

En el ámbito de la informática las colas son ampliamente utilizadas. Por ejemplo, los Sistemas Operativos suelen utilizar esta estructura de datos para gestionar los recursos que pueden ser compartidos por varios procesos (gestión de memoria, tiempo de procesador, etc.). En general, las colas se aplicarán cuando los objetos manejados sigan una política FIFO, tal y como la hemos descrito anteriormente.

4.2. El TAD Cola

En este apartado vamos a formalizar el concepto de cola mediante su especificación como tipo abstracto de datos.

TAD: Cola

Operaciones:

`CrearCola:` $\rightarrow$ `Cola`

Crea una cola vacía.

`Añadir: Cola x tipo_base` $\rightarrow$ `Cola`

Dada una cola c y un valor e, del tipo base, devuelve una nueva cola formada al añadir el elemento al final de c.

`Eliminar: Cola` $\rightarrow$ `Cola`

Dada una cola, devuelve la cola resultante de eliminar el elemento situado en su cabeza.

`Cabeza: Cola` $\rightarrow$ `tipo_base`

Dada una cola, devuelve el valor del elemento situado en su cabeza.

`ColaVacia: Cola` $\rightarrow$ `logico`

Dada una cola, devuelve el valor verdadero si la cola está vacía y falso en caso contrario.

Axiomas: $\forall$ `S` $\in$ `Cola,` $\forall$ `e` $\in$ `TipoBase,`

```
1)  ColaVacia(CrearCola) = cierto
2)  ColaVacia(Añadir(S,e)) = falso
3)  Eliminar(CrearCola) = error
4)  Eliminar(Añadir(S,e)) = Si ColaVacia(S) entonces CrearCola
                              sino Añadir(Eliminar(S),e)
5)  Cabeza(CrearCola) = error
6)  Cabeza(Añadir(S,e)) = Si ColaVacia(S) entonces e
                          sino Cabeza(S)
```

4.3. Implementaciones del tipo Cola

Al igual que ocurre con el tipo Pila, es posible realizar diversas implementaciones del TAD Cola. En primer lugar veremos como implementar de forma eficiente una cola utilizando vectores, y a continuación describiremos una implementación realizada mediante el uso de punteros y variables dinámicas. En ambos casos estudiaremos las ventajas e inconvenientes de utilizar cada implementación.

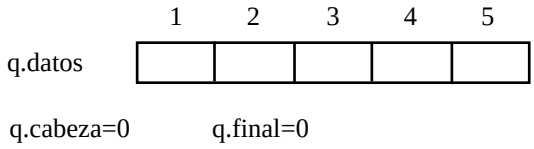
4.3.1. Implementación mediante vectores

Para realizar esta implementación, se toma como modelo la implementación del tipo Pila mediante vectores. Como en aquel caso, es más conveniente utilizar una sola entidad para contener la estructura de datos, y por tanto utilizaremos un registro e integraremos el vector como uno de sus campos. Además del vector, se necesitan unos índices que nos indiquen en cada momento qué elemento de dicho vector se está comportando como elemento cabeza y qué elemento se comporta como elemento final. Por lo tanto, el registro estará constituido por tres campos: un vector conteniendo los elementos de la cola, y dos valores, **cabeza** y **final**, que definen las posiciones del vector asociadas a ambos elementos de la cola. En Pascal este registro se definirá del siguiente modo:

```
CONST
  MAX= ...; {número máximo de elementos que puede contener la cola}
TYPE
  TipoCola = RECORD
             datos: ARRAY [1..MAX] OF TipoBase;
             cabeza, final: 0..MAX
           END;
```

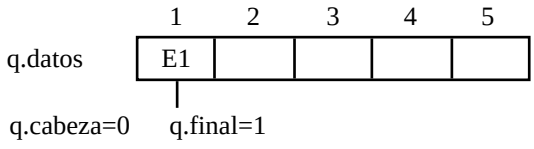
Antes de ver la implementación concreta de las operaciones, es conveniente realizar una pequeña discusión sobre su funcionamiento. Se parte, como en el caso de la Pila, de que al realizar la operación `CrearCola`, tanto cabeza como final indicarán la posición 0 :

`CrearCola(q):`



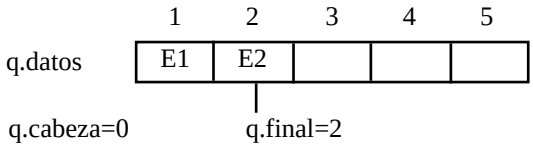
`Añadir(q, E1):`

Como final indica la última posición ocupada, una política sencilla es añadir en la siguiente (`final+1`); por lo tanto, la operación `Añadir` supondrá incrementar final y añadir en la posición indicada: `q.datos[q.final]`



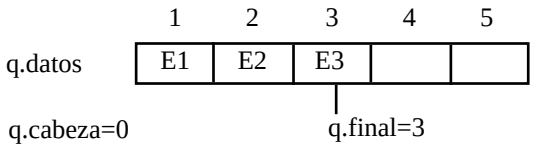
`Añadir(q, E2):`

Siguiendo la definición anterior de la operación:



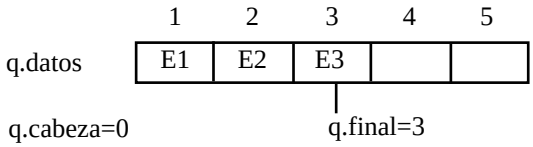
`Añadir(q, E3):`

Siguiendo la definición anterior de la operación:



`Cabeza(q):`

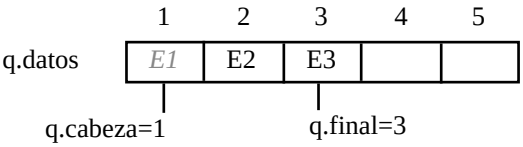
El elemento más antiguo en la cola es E1, que ocupa la posición 1. El valor de cabeza es 0, luego esta operación devolverá el elemento situado en la siguiente posición a la indicada por la cabeza: `q.datos[q.cabeza+1]`, que en este caso es E1.



Démonos cuenta que esta operación no modifica ninguno de los componentes de la cola; ni siquiera el valor de los campos cabeza y final.

Eliminar(q):

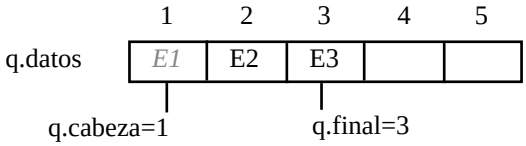
Dado que, como hemos dicho anteriormente, el campo cabeza indica la posición anterior al elemento más antiguo de la cola, la operación Eliminar supondrá eliminar de la cola el elemento situado en la posición `q.cabeza+1`. Para ello bastará con incrementar el valor del campo cabeza para que "apunte" al siguiente elemento de la cola. El elemento eliminado en este caso es el E1, y el más antiguo en la cola pasa a ser E2.



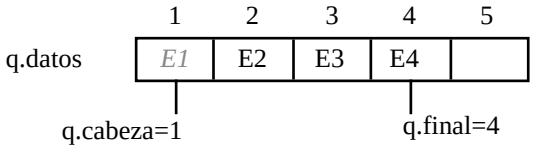
Debemos darnos cuenta de que no hemos sobreescrito el valor de E1 y que este sigue en la componente 1 del vector. Sin embargo, desde un punto de vista lógico, hemos eliminado este elemento de la cola. En este momento no existe ninguna operación valida del TAD Cola que nos permita acceder al mismo. En la figura representamos los elementos eliminados en cursiva. Estos elemento ya no forman parte de la cola.

Cabeza(q):

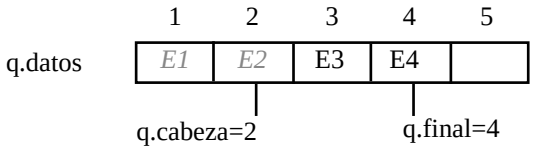
Al devolver q.datos[q.cabeza+1], el resultado es E2.



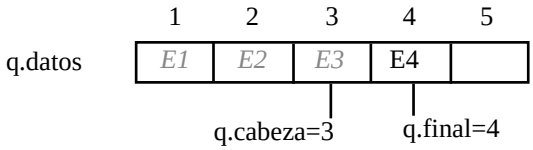
Añadir(q, E4):



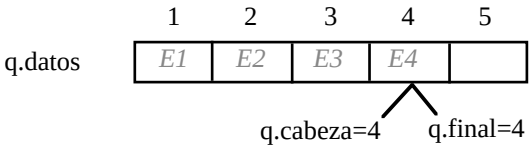
Eliminar(q):



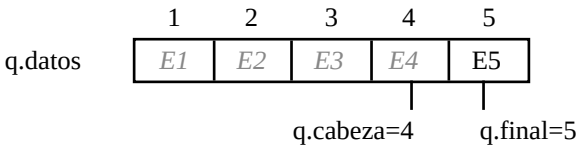
Eliminar(q):



Eliminar(q):



Añadir(q, E5):



...

En el ejemplo anterior hemos concretado varios aspectos relacionados con una posible implementación del tipo Cola mediante vectores:

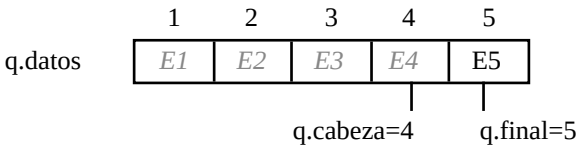
- En primer lugar utilizamos la siguiente definición de cabeza y final de la cola:
  - **cabeza**: contiene el índice de la componente del vector inmediatamente anterior a la que contiene el elemento más antiguo de la cola. Así pues, no apunta al elemento más antiguo, sino a la posición anterior al mismo.
  - **final**: contiene el índice de la última componente añadida a la cola. Así pues, apunta al elemento más reciente.
- La condición de ColaVacia es que (`q.cabeza=q.final`). Está condición se cumple inicialmente, cuando tanto cabeza como final valen 0 y también se cumple cuando se han eliminado todos los elementos añadidos.

La implementación de las operaciones del TAD Cola será la siguiente:

```
PROCEDURE CrearCola (VAR q: TipoCola);
BEGIN
  q.cabeza:=0;
  q.final:=0
END;

FUNCTION ColaVacia (q: TipoCola):BOOLEAN;
BEGIN
  ColaVacia := (q.cabeza=q.final)
END;

PROCEDURE Cabeza (q: TipoCola ; VAR e:Tipobase; VAR error: BOOLEAN);
BEGIN
  IF ColaVacia(q) THEN
    error := true
  ELSE
  BEGIN
    error := false;
    e := q.datos[q.cabeza+1]
  END
END;

PROCEDURE Eliminar (VAR q: TipoCola ; VAR error: BOOLEAN);
BEGIN
  IF ColaVacia(q) THEN
    error := true
  ELSE
  BEGIN
    error := false;
    q.cabeza := q.cabeza+1
  END
END;
```

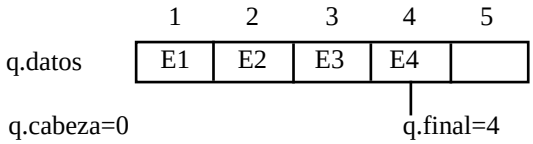
```
PROCEDURE Añadir (VAR q: TipoCola; e:Tipobase; VAR error: BOOLEAN);
BEGIN
  IF ColaLlena(q) THEN
    error := true
  ELSE
  BEGIN
    error := false;
    q.final := q.final+1;
    q.datos[q.final] := e
  END
END;
```

Pero, ¿cuál es la condición de `ColaLlena`? Podemos utilizar la condición (q.final=MAX) para afirmar que la cola está llena, tal y como ocurre en la siguiente figura:

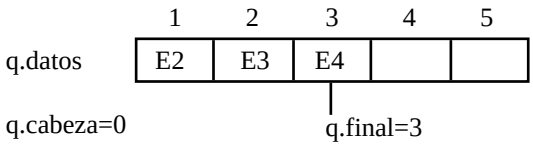


Sin embargo, si utilizamos la condición anterior, se da la paradoja de que hay sitio en el vector, pero no se puede añadir ningún elemento más. Una posible solución a este problema sería desplazar todos los elementos de la cola a la posición anterior del vector cada vez que eliminamos un elemento. De este modo, los elementos de la cola siempre estarían situados sobre las componentes iniciales del vector y no desaprovecharíamos espacio del mismo. Veámoslo con un ejemplo:

Si partimos de una cola como la siguiente:

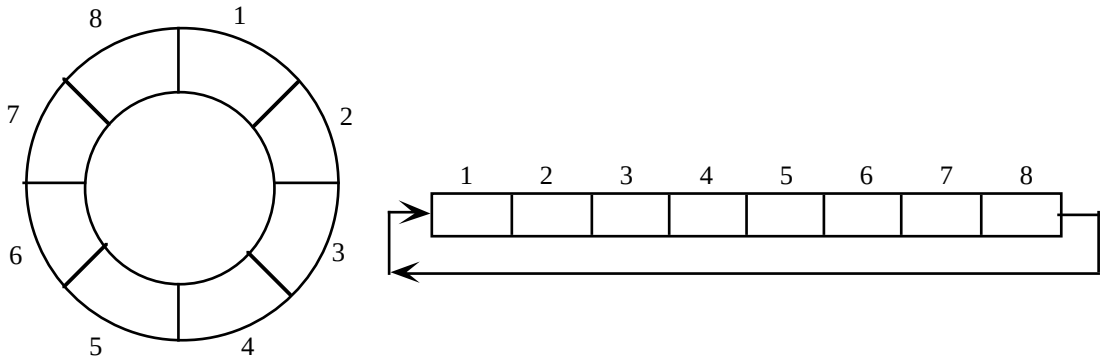


la operación `Eliminar(q)` dará como resultado:



Esta implementación alternativa nos permite mantener la cabeza de la cola siempre fija, pero supone mover múltiples elementos cada vez que queramos eliminar uno de ellos. Esto puede suponer un incremento sustancial del coste, por lo que optamos por otra solución al problema de llenado de la cola.

La solución que vamos a describir implica reutilizar las componentes del vector que contenían elementos ya eliminados. Esto es, cuando durante el proceso de añadido lleguemos al final del vector, comenzaremos a llenar de nuevo las componentes iniciales del mismo si se encuentran vacías. Para logra esto, manejaremos el vector como si fuese un "vector circular". Esto significa que no consideraremos la componente `MAX` del vector como la última del mismo, sino que consideraremos que la siguiente componente a está es otra vez la primera del vector. Gráficamente:



La implementación de la estructura de datos es la misma que hemos utilizado anteriormente. Así, se utiliza un registro con tres campos: uno para guardar los elementos de la cola, y dos para guardar la posición de la cabeza y el final de la misma. La diferencia estriba en cómo se manejan estos campos en las operaciones de manipulación del TAD.

Al utilizar un vector circular para la implementación, la siguiente componente a la posición `i` en el vector no siempre será `i+1`, sino que vendrá dada por la función:

```
FUNCTION Siguiente ( i:INTEGER ):INTEGER;
BEGIN
  IF (i<>MAX) THEN
    Siguiente := i+1
  ELSE
    Siguiente := 1
END;
```

Existe otra forma más sencilla de implementar la función `Siguiente` que se basa en el uso del operador `MOD`. Recordemos que este operador nos devuelve el resto de la división entera entre dos números.

```
FUNCTION Siguiente ( i:INTEGER ):INTEGER;
BEGIN
  Siguiente := i MOD MAX + 1
END;
```

Haciendo un uso adecuado de la función `Siguiente` podemos implementar las distintas operaciones del TAD Cola mediante un vector circular del siguiente modo:

```
PROCEDURE CrearCola ( VAR q: TipoCola );
BEGIN
  q.cabeza := MAX;
  q.final := MAX
END;
```

Siguiendo el mismo criterio que con el vector lineal, `q.cabeza` apuntará a la componente "anterior" a donde se sitúa el elemento más antiguo. El primer elemento se insertará en la componente 1 del vector, por lo que inicialmente `q.cabeza` debe apuntar a la posición anterior a esta. En un vector circular, la posición anterior a la primera en el vector es la última, por lo que `q.cabeza` se inicializa a `MAX`. Por otro lado, consideraremos que la cola se encuentra vacía cuando coincidan `q.cabeza` y `q.final`. Así pues, también debemos inicializar `q.final` a MAX.

```
FUNCTION ColaVacia ( q: TipoCola ): BOOLEAN;
BEGIN
  ColaVacia := (q.cabeza=q.final)
END;
```

La cola se considera vacía cuando la cabeza y el final de la misma apuntan a la misma posición del vector circular. Esta condición también se da con una cola recién creada.

```
PROCEDURE Cabeza (q: TipoCola ; VAR e:Tipobase; VAR error: BOOLEAN);
BEGIN
  IF ColaVacia(q) THEN
    error := true
  ELSE
  BEGIN
    error := false;
    e := q.datos[Siguiente(q.cabeza)]
  END
END;
```

El elemento cabeza de la cola es el que ocupa la siguiente posición a la apuntada por el campo `q.cabeza`. Además esta operación devuelve un error cuando la cola está vacía.

```
PROCEDURE Eliminar ( VAR q: TipoCola ; VAR error: BOOLEAN);
BEGIN
  IF ColaVacia(q) THEN
    error := true
  ELSE
  BEGIN
    error := false;
    q.cabeza := Siguiente(q.cabeza)
  END
END;
```

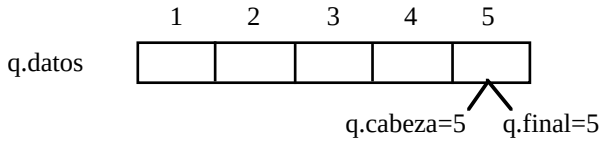
Para eliminar un elemento de la cola basta con hacer que el campo `q.cabeza` apunte al siguiente elemento del vector. De nuevo, esta operación devuelve un error si la cola está vacía.

```
PROCEDURE Añadir (VAR q: TipoCola; e:Tipobase; VAR error: BOOLEAN);
BEGIN
  IF ColaLlena(q) THEN
    error := true
  ELSE
  BEGIN
    error := false;
    q.final := Siguiente(q.final);
    q.datos[q.final] := e
  END
END;
```

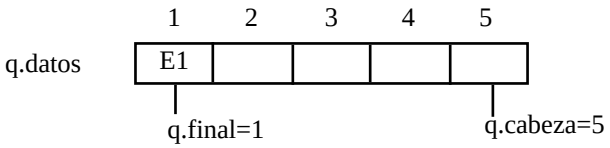
Para añadir un elemento, se hace que el campo `q.final` apunte al siguiente de la cola y se coloca en esa posición. Obviamente no podremos añadir un elemento a la cola si está llena.

Antes de discutir cuál es el enunciado de la operación `ColaLlena`, conviene comprobar cuál es el comportamiento de la Cola cuando utilizamos un vector circular y las operaciones anteriores; volviendo al ejemplo anterior, se tiene:

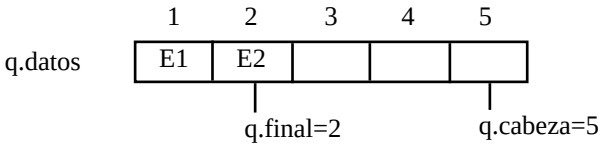
`CrearCola(q):`



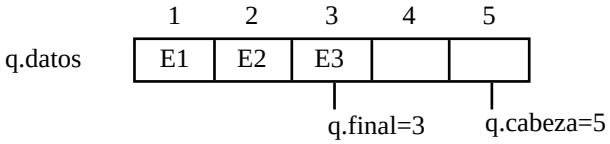
`Añadir(q, E1):` `Siguiente(q.final) = 5 MOD 5 + 1 = 1`



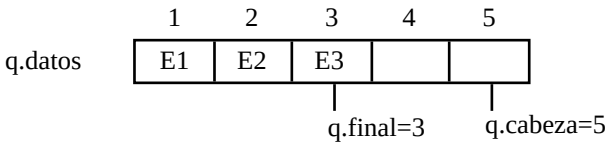
`Añadir(q, E2):` `Siguiente(q.final) = 1 MOD 5 + 1 = 2`



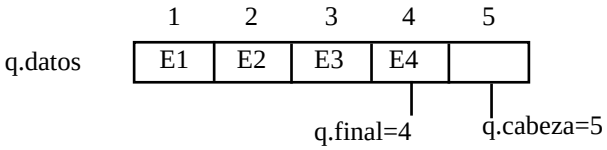
Añadir(q, E3): Siguiente(q.final) = 2 MOD 5 + 1 = 3



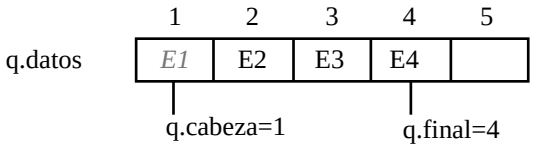
Cabeza(q): Siguiente(q.cabeza) = 5 MOD 5 + 1 = 1; Cabeza vale E1



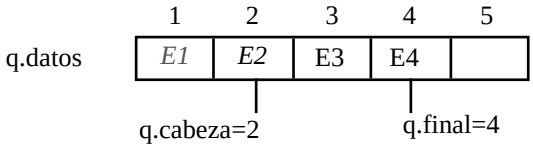
Añadir(q, E4): Siguiente(q.final) = 3 MOD 5 + 1 = 4



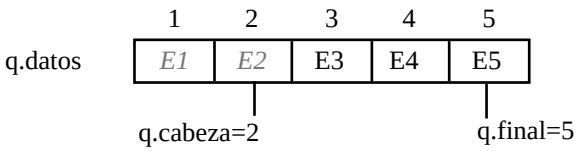
Eliminar(q): Siguiente(q.cabeza) = 5 MOD 5 + 1 = 1



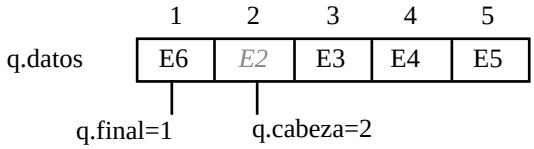
Eliminar(q): Siguiente(q.cabeza) = 1 MOD 5 + 1 = 2



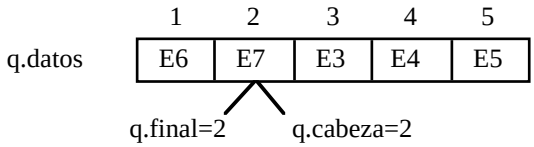
Añadir(q, E5): Siguiente(q.final) = 4 MOD 5 + 1 = 5



Añadir(q, E6): `Siguiente(q.final) = 5 MOD 5 + 1 = 1`



Añadir(q, E7): `Siguiente(q.final) = 1 MOD 5 + 1 = 2`



En este punto, ¿la cola está Llena o Vacía?. Sabemos que se cumple la condición de cola vacía porque `q.final` y `q.cabeza` son iguales. Sin embargo, tal y como se ha desarrollado el ejemplo, sabemos que la cola está llena. Hemos de establecer algún mecanismo para diferenciar ambas situaciones.

La primera solución a este problema consiste en utilizar un campo adicional en el registro que representa la Cola. Este campo, que podemos denominar longitud, nos dará el número de elementos que contiene. Cuando valga 0 sabremos que la cola está vacía y cuando valga `MAX` (la longitud del vector) sabremos que la cola está llena. Sin embargo, si utilizamos esta solución hemos de modificar todas las operaciones del TAD. Además, nos vemos obligados a manejar un campo adicional, con el consiguiente aumento del coste y de la dificultad de los procedimientos.

Otra solución más extendida y menos costosa consiste en dejar siempre una posición libre en la Cola. Esta posición se encontrará en la componente apuntada por la cabeza. Sabemos que esta componente es justo la anterior en el vector circular a la ocupada por el elemento más antiguo de la cola. En esta implementación, esta posición será especial, dado que no podrá ser ocupada por ningún elemento. Con esta restricción, la condición de cola llena se producirá cuando el final de la cola se encuentre justo antes de esta posición (`q.cabeza`), es decir, cuando el siguiente elemento debería añadirse en la misma. Por lo tanto, la definición de Cola Llena queda de la siguiente forma:

```
FUNCTION ColaLlena ( q: TipoCola ): BOOLEAN;
BEGIN
  ColaLlena := (q.cabeza = Siguiente(q.final))
END;
```

Con esta definición de cola llena, la última operación del ejemplo anterior habría devuelto un error. La condición de cola llena se habría cumplido y no se habría podido añadir el elemento E7.

Obviamente, la solución que hemos utilizado con un vector circular desaprovecha una de las posiciones del vector, pero es mejor que la que usa un vector lineal que acaba llenándose con mucha mayor facilidad.

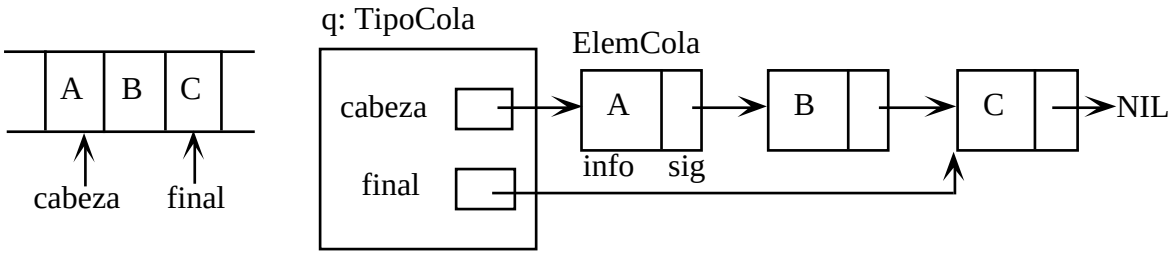
En todo caso, todas las implementaciones que hacen uso de un vector adolecen del mismo problema: se usa una estructura estática, el vector, para manejar una estructura dinámica, la cola. Esto da lugar a que la cola pueda llenarse, hecho que queda fuera de la definición del TAD. Además, si queremos evitar que esto se produzca, habremos de reservar espacio para un vector de gran dimensión, con el consiguiente peligro de desaprovechar espacio en memoria. Estos problemas nos llevan a una implementación de la cola mediante el uso de variables dinámicas y punteros.

4.3.2. Implementación mediante punteros

En este apartado estudiaremos la implementación del TAD Cola mediante el uso de variables dinámicas y punteros. Al igual que en la definición de la Pila, al utilizar punteros para definir una Cola, se llega a una definición recursiva:

```
TYPE
  TipoPuntero = ^ElemCola;
  ElemCola = RECORD
               info: <TipoBase>;
               sig: TipoPuntero
             END;
  TipoCola = RECORD
             cabeza, final : TipoPuntero
           END;
```

Gráficamente:

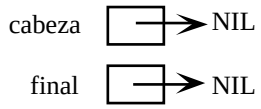


TAD Cola Implementación mediante punteros

Con esta definición se obtiene la siguiente implementación para las operaciones:

```
PROCEDURE CrearCola ( VAR q: TipoCola );
BEGIN
  q.cabeza := NIL;
  q.final := NIL
END;
```

Al crear la cola tanto la cabeza como el final de la misma están indefinidos. Así pues, se asigna el valor NIL a los punteros asociados.




```
FUNCTION ColaVacia ( q: TipoCola ): BOOLEAN;
BEGIN
  ColaVacia := (q.cabeza=NIL) AND (q.final=NIL)
END;
```

Consideraremos que la cola está vacía cuando tanto la cabeza como el final apunten a NIL. Este caso se dará nada más ser creada y cuando se acabe de eliminar su último elemento.

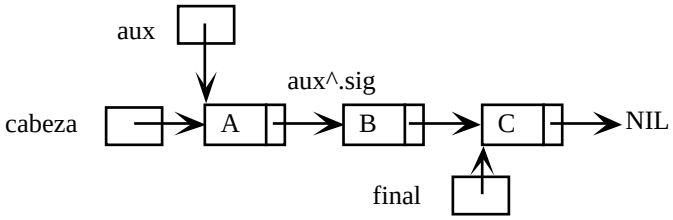
```
PROCEDURE Cabeza (q:TipoCola; VAR e:<Tipobase>; VAR error: BOOLEAN);
BEGIN
  IF ColaVacia(q) THEN
    error := true
  ELSE
  BEGIN
    error := false;
    e := q.cabeza^.info
  END
END;
```

Para conocer la cabeza de la cola basta con acceder al nodo apuntado por `q.cabeza`. El campo `info` de este nodo contendrá el elemento de la cola deseado. Esta operación debe devolver un error cuando la cola se encuentre vacía.

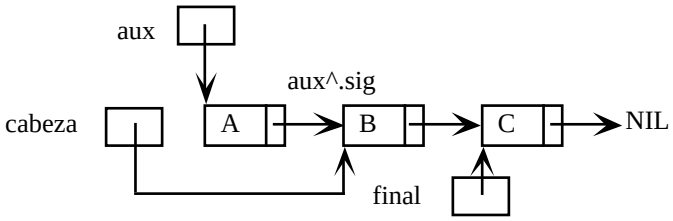
```
PROCEDURE Eliminar ( VAR q: TipoCola; VAR error: BOOLEAN);
VAR
  aux: TipoPuntero;
BEGIN
  IF ColaVacia(q) THEN
    error := true
  ELSE
  BEGIN
    error := false;
    aux := q.cabeza;
    q.cabeza := aux^.sig;
    IF ( q.cabeza = NIL ) THEN { hemos vaciado la cola }
      q.final := NIL;
    dispose(aux)
  END
END;
```

Para eliminar un elemento hay que hacer que la cabeza de la cola apunte al siguiente elemento de la misma. A continuación se borra el elemento que ocupa la cabeza en la actualidad. Veamos los pasos seguidos:

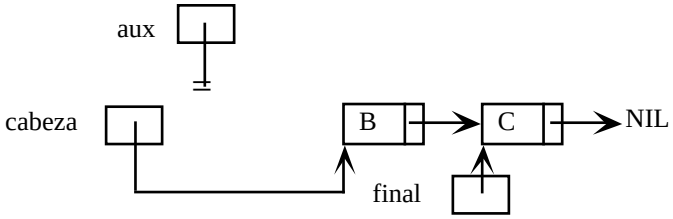
- En primer lugar se usa una variable auxiliar de tipo TipoPuntero para acceder al elemento de la cola situado en la cabeza:



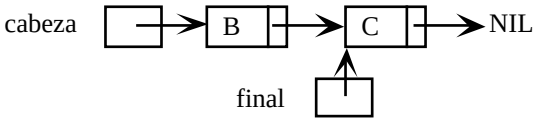
- A continuación se hace que la cabeza pase a apuntar al siguiente elemento, dado por `aux^.sig`.



- Finalmente se libera la memoria correspondiente al nodo que ocupa la cabeza.



- Al salir del procedimiento se libera la memoria de la variable local `aux`.

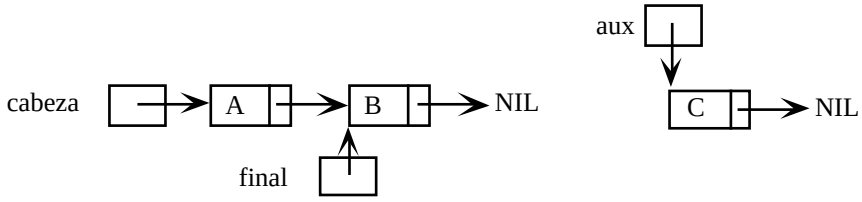


Al implementar la rutina de eliminado hemos de tener en cuenta dos casos particulares.

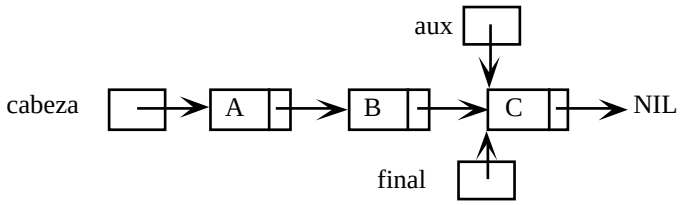
- Si la cola estaba vacía, el procedimiento devolverá un error
- Si la cola contenía un solo elemento, tras eliminarlo quedará vacía. Cuando nos encontremos en este caso, el procesamiento normal nos llevará a que `q.cabeza` valga NIL. Cuando se produzca este hecho debemos hacer que también `q.final` valga NIL.

```
PROCEDURE Añadir ( VAR q: TipoCola ; e: <TipoBase>);
VAR
  aux: TipoPuntero;
BEGIN
  new(aux);
  aux^.info := e;
  aux^.sig := NIL;
  IF ColaVacia(q) THEN
    q.cabeza := aux
  ELSE
    q.final^.sig := aux;
  q.final := aux
END;
```

- Para añadir un nuevo elemento, las tres primeras instrucciones del procedimiento crean y rellenan un nuevo nodo apuntado por `aux`:



- A continuación, si la cola estaba vacía, se hace que la cabeza apunte al nuevo nodo. Si la cola no estaba vacía, el último nodo de la cola debe apuntar al que se va a añadir. En ambos casos, se hace apuntar el final de la cola al nuevo nodo.



Con esta implementación mediante variables dinámicas, no es necesario tener en cuenta el caso de la cola llena, ya que el número de elementos a almacenar no está limitado en principio.

4.4. Ejercicios

1. Un concesionario de coches tiene un número limitado *m* de modelos, todos en un número limitado *c* de colores distintos. Cuando un cliente quiere comprar un coche, pide un coche de un modelo y color determinados. Si el coche de ese modelo y color no está disponible en el concesionario, se toman los datos del cliente (nombre y dirección), que verá atendida su petición cuando el coche esté disponible. Si hay más de una petición de un coche de las mismas características, se atienden las peticiones por orden cronológico. Se pide:

   a) Definir la estructura de datos más adecuada capaz de contener las peticiones de un modelo y color de coche.

b) Definir la estructura de datos global más adecuada, capaz de contener las peticiones para todos los modelos y colores de coches del concesionario.

c) Definir una operación que, dado un cliente (nombre y dirección) que desea comprar un coche de un modelo y color determinado, coloque sus datos como **última petición** de ese modelo y color.

d) Definir una operación que, dado un modelo del que se han recibido k coches de determinado color, elimine los **k primeros** clientes de la lista de peticiones de ese coche y los devuelva en un vector, sabiendo que $k \leq 20$.

2. Se utiliza, para implementar una estructura de datos Cola, vectores que a lo sumo pueden contener MAXIMO elementos. (Siendo MAXIMO una constante que se supone definida).

a) Definir la estructura de datos necesaria para poder utilizar colas con una capacidad máxima de 2*MAXIMO elementos.

b) Desarrollar los algoritmos que implementan las operaciones básicas sobre datos del tipo cola, según la estructura elegida en el apartado a).

3. Se desea tener una estructura de datos cola que tenga disponible en todo momento qué cantidad de elementos contiene. Se pide:

a) Definición de tipos necesarios, sabiendo que se trata de una estructura dinámica.

b) Definición de las operaciones de cola, y además de una operación longitud, que devuelva el número de elementos de la cola.

4. Una librería registra las peticiones de cualquier libro que no tiene en ese momento. La información de cada libro consiste en el título del libro, el precio (en pesetas), el número de libros en *stock*, y las peticiones del libro en estricto orden de llegada. Cada petición consiste en el nombre de una persona y su dirección.

a) Define los tipos de datos capaces de describir toda esta información para un total de 2000 libros. Justifica la elección de estructuras para las peticiones y para la estructura general que incluye los datos de todos los libros.

b) Implementa una operación que dado un cliente que pide un libro, vea si hay en stock, y si quedan, actualice el stock con la venta de ese libro, y si no, guarde los datos del cliente como **última petición** de ese libro.

c) Implementa otra operación que dado un libro del que se reciben ***k*** ejemplares vea si hay peticiones y si las hay escriba en pantalla **los *k* primeros** clientes y elimine sus peticiones. Si hay más de *k* peticiones el stock seguirá a 0. Si hay menos de *k* peticiones, habrá que actualizar el stock con los libros que queden.

5 Una agencia de viajes ofrece **n** destinos; para cada destino se puede optar por 5 clases de viaje, **super**, **luxe**, **normal**, **turista** y **estudiante**, y además se ofrecen tres tipos de alojamiento: **AD** (alojamiento y desayuno), **MP** (media pensión) y **PC** (pensión completa). Cada programa de viaje se caracteriza por la información (destino, clase, alojamiento). Por cada programa de viaje se quiere saber el número de plazas disponibles, de manera que cuando un cliente contrata un determinado programa, el número de plazas disponibles se decrementa. Cuando un programa no dispone de plazas, entonces la información del cliente (nombre, dirección y NIF) se almacena en orden cronológico. Así, cuando se disponga de nuevas plazas en ese programa se atenderán las peticiones en orden.

Se desea informatizar la gestión de esta agencia. Entre otras cosas, es preciso:

a) La definición de la estructura de datos que soporte la información descrita (es decir, la estructura que permita almacenar todos los programas de viaje de la agencia).

b) Escribir un algoritmo que, **dado un cliente** y **un determinado programa** de viajes, compruebe si es posible o no que el cliente lo contrate. En cualquier caso, habrá que realizar las acciones oportunas para que la estructura se actualice de forma conveniente.

c) Escribir un algoritmo que indique todos los **destinos** con **alguna** plaza disponible. La solución debe incluir la definición de la **estructura de datos más idónea** para devolver la información, sabiendo que n < 100.

6. Un restaurante dispone de **m** mesas. Por cada mesa se sabe su código de identificación y su capacidad en comensales.

   Hay una cola de espera para ir ocupando las mesas, de forma que, para cada elemento, se sabe el nombre de la persona que ha hecho la reserva y el número de comensales. Se pide:

   a) Definir las estructuras de datos **restaurante** y **espera**.

   b) Definir la operación **Maître** que, dado un identificador de mesa, devuelve el nombre de una persona que haya hecho una reserva y que ya puede pasar al comedor junto con sus acompañantes. La estructura **espera** debe quedar convenientemente actualizada. Se deben hacer dos versiones de la operación:

      b.1) La elección se hace buscando la primera reserva que cabe en la mesa.

      b.2) La elección se hace optimizando la ocupación de las mesas.

7. Se desea realizar un enriquecimiento del TAD cola que llamaremos cola con prioridad. En la cola con prioridad habrá dos tipos de elementos, los que tienen prioridad y los que no la tienen. Además tendremos dos operaciones para insertar elementos, la operación **Añadir**, que inserta elementos sin prioridad (hace lo mismo que en la cola normal), y la operación **Añadir_con_prioridad**, la cual inserta un elemento de modo que queda colocado antes de todos aquellos que no tiene prioridad y después de aquellos con prioridad que ya estuvieran en la cola. Se pide:

   a) Definir la estructura de datos cola con prioridad utilizando variables dinámicas.

   b) Modificar las operaciones del TAD Cola que sean necesarias para que se adecúen al enriquecimiento.

   c) Implementar la operación **Añadir_con_prioridad** mediante el procedimiento que debe tener el siguiente perfil:

   ```
   procedure Añadir_con_prioridad(var q:Cola_prioridad; e:TB);
   ```

8. En un supermercado hay 20 cajas registradoras, en cada una de las cuales se colocan los clientes con sus carros de la compra en orden de llegada. Por cada caja registradora queremos guardar el nombre de la cajera, la recaudación acumulada y los carros en espera.

   Por otro lado, en cada carro se amontonan los distintos productos, de modo que tan sólo puede añadirse o extraerse el situado en la parte superior. Por cada producto guardamos su nombre y precio.

   a) Estructuras de datos. Utilizar en cada subapartado las estructuras de los anteriores.

a.1) Definir la estructura más adecuada para guardar un **carro** de la compra.

a.2) Definir la estructura más adecuada para guardar una **caja** registradora.

a.3) Definir la estructura más adecuada para guardar el **supermercado**.

b) Implementar un algoritmo denominado **atender_cliente** que dado un carro de la compra, pase los productos que contiene por caja y calcule el precio total.

c) Implementar un algoritmo que calcule la recaudación de las cajas después de pasar los primeros *x* carros por cada una de ellas. *x* será un argumento de entrada que puede ser mayor que el número de carros en espera en algunas de las cajas.

Utilizar la estructura de datos más adecuada para devolver el resultado pedido.

Nota. Utilizar de modo adecuado el procedimiento implementado en el apartado b)

9. Una Bicola es una estructura de datos similar a una Cola, pero en la que pueden insertarse y eliminarse elementos en los dos extremos (y en ninguna otra posición).

Supongamos la siguiente especificación en lenguaje natural de las operaciones de manejo de una bicola.

`Crearbicola(bicola):`
Crea una bicola vacía.

`Bicolavacia(bicola)`:
Devuelve verdadero si la bicola está vacía y falso en caso contrario.

`Elimizq(bicola)`:
Elimina el elemento situado en el extremo izquierdo de la bicola.

`Elimder(bicola)`:
Elimina el elemento situado en el extremo derecho de la bicola.

`Añadizq(bicola, x)`:
Añade el elemento x en el extremo izquierdo de la bicola.

`Añadder(bicola,x)`:
Añade el elemento x en el extremo derecho de la bicola.

`Cabizq(bicola, x)`:
Devuelve en x el elemento situado en el extremo izquierdo de la bicola sin eliminarlo.

`Cabder(bicola, x)`:
Devuelve en x el elemento situado en el extremo derecho de la bicola sin eliminarlo.

a) Definir la estructura de datos dinámica más adecuada para almacenar una bicola. Justificar la respuesta.

b) Implementar la operación Elimder(bicola);

c) Dada una bicola de caracteres, implementar un algoritmo que nos diga si la cadena de caractéres que contiene es un palindromo (capicúa).

Ejemplos de cadenas capicúa son: a, aa, aba, abba.