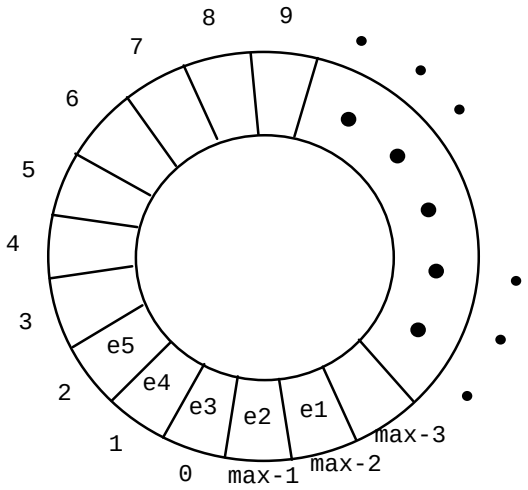
**Implementación estática (basada en un "vector circular") del TAD cola genérica**

En la figura se representa un ejemplo de cola (`e1,e2,e3,e4,e5`) almacenada en un vector (considerado como circular).

En este ejemplo, el índice `anterior` (en el sentido contrario a las agujas del reloj) al del primer elemento de la cola (`e1`) es `max-3`; el índice del último elemento de la cola (`e5`) es `2`.

En esta representación, `tamaño = 0` si y sólo si la secuencia es vacía. La capacidad máxima es de `max` elementos.




`anterior = max-3`

`último = 2`

`tamaño = 5`

```
módulo genérico colasGenéricasEstáticas
parámetro
  tipo elemento

exporta

  constante max = 100
  tipo cola  {Los valores del TAD cola representan secuencias de 0 o más elementos,
              con longitud máxima max; llamamos primer elemento de la cola al primero
              que fue añadido y último al último que fue añadido}

  procedimiento crear(sal c:cola)
  {Devuelve una cola vacía, sin elementos}
  procedimiento encolar(ent e:elemento; e/s c:cola; sal error:booleano)
  {Si c no está llena, añade e a c como último elemento; si está llena, devuelve error}
  función esVacía(c:cola) devuelve booleano
  {Devuelve verdad si y sólo si c no tiene elementos}
  procedimiento desencolar(e/s c:cola; sal error:booleano)
  {Si c es no vacía, devuelve la cola resultante de borrar su primer elemento;
   si es vacía, devuelve error}
  procedimiento primero(ent c:cola; sal e:elemento; sal error:booleano)
  {Si c es no vacía, devuelve en e su primer elemento; si es vacía, devuelve error}
  función longitud(c:cola) devuelve natural
  {Devuelve el número de elementos de c}

  procedimiento duplicar(sal cSal:cola; ent cEnt:cola)
  {Duplica la representación de la cola cEnt en la cola cSal}
  función iguales(c1,c2:cola) devuelve booleano
  {Devuelve verdad si y sólo si las colas c1 y c2 tienen la misma longitud y los
   mismos elementos en idénticas posiciones}

  {Añadimos las tres operaciones básicas de un iterador}
  procedimiento iniciarIterador(e/s c:cola)
  {Prepara el iterador para que el siguiente elemento a visitar sea el
   primero de la cola (situación de no haber visitado ningún elemento)}
  función haySiguiente(c:cola) devuelve booleano
  {Devuelve falso si ya se ha visitado el último elemento, cierto en caso contrario}
  procedimiento siguiente(e/s c:cola; sal e:elemento: sal error:booleano)
  {Si existe siguiente devuelve en e el siguiente elemento de c y avanza el cursor;
   si no, devuelve error}
```

**implementación**

```
  tipos vectorDatos = vector[0..max-1] de elemento;
        cola = registro
                 datos:vectorDatos;
                 anterior,último,actual:0..max-1;  {actual se usa para el iterador}
                 tamaño:0..max
               freg

  procedimiento crear(sal c:cola)
  principio
    c.anterior:=0;  {por ejemplo}
    c.último:=0;
    c.tamaño:=0
  fin

  procedimiento encolar(e/s c:cola; ent e:elemento; sal error:booleano)
  principio
    si c.tamaño<max entonces
      error:=falso;
      c.último:=(c.último+1) mod max;
      c.datos[c.último]:=e;
      c.tamaño:=c.tamaño+1
    sino
      error:=verdad
    fsi
  fin

  función esVacía(c:cola) devuelve booleano
  principio
    devuelve c.tamaño=0
  fin

  procedimiento desencolar(e/s c:cola; sal error:booleano)
  principio
    si esVacía(c) entonces
      error:=verdad
    sino
      error:=falso;
      c.anterior:=(c.anterior+1) mod max;
      c.tamaño:=c.tamaño-1
    fsi
  fin

  procedimiento primero(ent c:cola; sal e:elemento; sal error:booleano)
  principio
    si esVacía(c) entonces
      error:=verdad
    sino
      error:=falso;
      e:=c.datos[(c.anterior+1) mod max]
    fsi
  fin

  función longitud(c:cola) devuelve natural
  principio
    devuelve c.tamaño
  fin
```

```
  procedimiento duplicar(sal cSal:cola; ent cEnt:cola)
  variable i:natural
  principio
    si esVacía(cEnt) entonces
      crear(cSal)
    sino
      cSal.tamaño:=cEnt.tamaño;
      cSal.anterior:=cEnt.anterior;
      cSal.último:=cEnt.último;
      para i:=1 hasta cEnt.tamaño hacer
        cSal.datos[(cSal.anterior+i) mod max]:=cEnt.datos[(cEnt.anterior+i) mod max]
      fpara
    fsi
  fin

  función iguales(c1,c2:cola) devuelve booleano
  variable i:natural; igual:booleano
  principio
    si esVacía(c1) and esVacía(c2) entonces
      devuelve verdad
    sino_si longitud(c1)≠longitud(c2) entonces
      devuelve falso
    sino
      i:=1;
      igual:=verdad;
      mientrasQue igual and i≤longitud(c1) hacer
        igual:=c1.datos[(c1.anterior+i) mod max]=c2.datos[(c2.anterior+i) mod max];
        i:=i+1
      fmq;
      devuelve igual
    fsi
  fin

  procedimiento iniciarIterador(e/s c:cola)
  principio
    c.actual:=(c.anterior+1) mod max
  fin

  función haySiguiente(c:cola) devuelve booleano
  principio
    devuelve c.actual≠(c.último+1) mod max
  fin

  procedimiento siguiente(e/s c:cola; sal e:elemento: sal error:booleano)
  principio
    si haySiguiente(c) entonces
      error:=falso;
      e:=c.datos[c.actual];
      c.actual:=(c.actual+1) mod max
    sino
      error:=verdad
    fsi
  fin

fin
```