Programación II

Tema 3. Colas

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Contenidos

- El TAD Cola
- Estructura de datos y primitivas de Cola
- Estructura de datos de Cola como array circular
- Implementación en C de Cola
  - Implementación con front y rear de tipo entero

- Anexo
  - Implementación con front y rear de tipo puntero

Contenidos

- **El TAD Cola**
- Estructura de datos y primitivas de Cola
- Estructura de datos de Cola como array circular
- Implementación en C de Cola
  - Implementación con front y rear de tipo entero

- Anexo
  - Implementación con front y rear de tipo puntero

El TAD Cola

- **Cola** (***queue*** en inglés)
  - Colección de elementos FIFO - *First In, First Out*: “el primero que entra, el primero que sale”

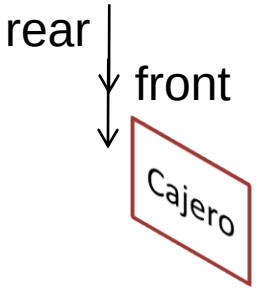
El TAD Cola

- **Definición de Cola**
  - Contenedor de elementos que son insertados y extraídos siguiendo el principio de que el primero que fue insertado será el primero en ser extraído (*FIFO – First In, First Out*)
    - Los elementos se insertan de uno en uno: **insertar**
    - Los elementos se extraen de uno en uno: **extraer**
    - La posición de la cola donde se encuentra el primer elemento, es decir, el siguiente elemento a ser extraído, se denomina **front** (o *head*, inicio)
    - La posición de la cola donde se colocará el siguiente elemento que se inserte se denomina **rear** (o *tail*, fin)

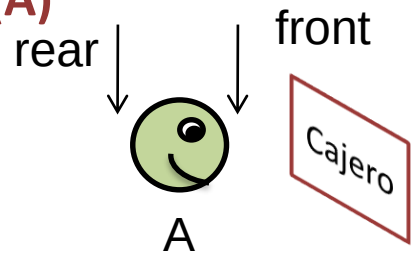
El TAD Cola

- **Cola**: contenedor de elementos en el que…
  - la inserción se realiza por un único punto: **rear** / tail / fin
  - la extracción se realiza por un único punto: **front** / head / inicio

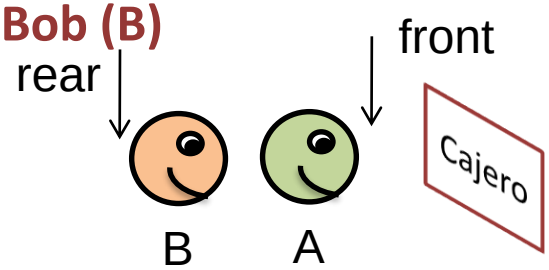
1. Cola Vacía



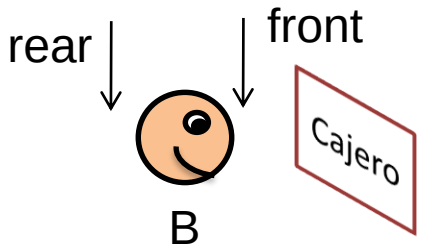
2. Entra Alice (A)



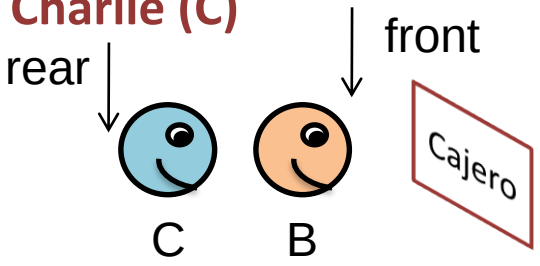
3. Entra Bob (B)



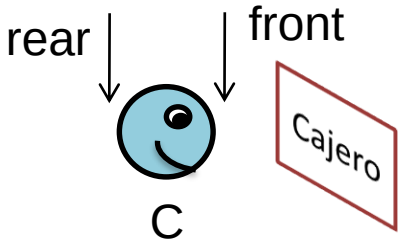
4. Sale Alice



5. Entra Charlie (C)

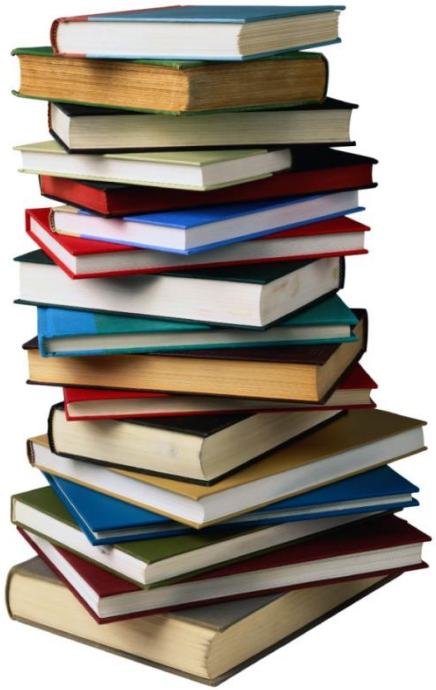


6. Sale Bob



El TAD Cola

- **Diferencias entre los TAD Pila y Cola**
  - Pila tiene un único punto de entrada y salida; Cola tiene dos
  - Pila es LIFO (*Last In, First Out*); Cola es FIFO (*First In, First Out*)



El TAD Cola

- Colas en el mundo real: para pagar en comercios, comprar tickets para un espectáculo, sacar dinero de un cajero, …
  - **Una cola gestiona un acceso concurrente a un único recurso**
- En Informática existen muchos ejemplos de uso de colas
  - Trabajos enviados a impresoras
    - El primer trabajo en llegar es el primero que se imprime: *First Come, First Served* (FCFS)
  - Peticiones a servidores
  - Uso del procesador
    - El sistema operativo a veces planifica la ejecución de los procesos en el orden de llegada.
- ¡OJO! A veces los elementos no tienen la misma prioridad y hay que permitir modificar el orden FIFO

  → **colas de prioridad** (tema 6)

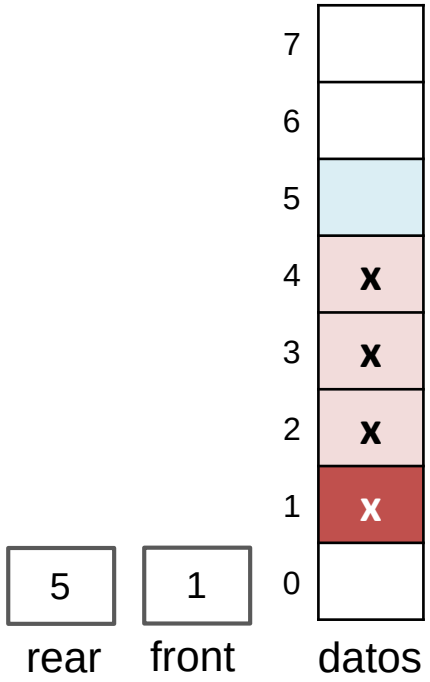
Contenidos

- El TAD Cola
- **Estructura de datos y primitivas de Cola**
- Estructura de datos de Cola como array circular
- Implementación en C de Cola
  - Implementación con front y rear de tipo entero

- Anexo
  - Implementación con front y rear de tipo puntero

Estructura de datos y primitivas de Cola

- **Una cola está formada por:**
  - **datos:** conjunto de elementos, en general del mismo tipo, almacenados de forma secuencial y accesibles desde dos puntos: *front* y *rear*
  - **front**: indicador de la posición del próximo elemento a extraer
  - **rear**: indicador de la posición donde colocar el próximo elemento que se inserte



(en este dibujo se asume que la cola tiene tamaño máximo de 8, pero no tiene por qué ser así)

Estructura de datos y primitivas de Cola

- **Primitivas**

`Cola` **`cola_crear`**`()`: crea, inicializa y devuelve una cola

`cola_liberar``(Cola s)`: libera (la memoria ocupada por) la cola

`boolean` **`cola_vacia`**`(Cola s)`: devuelve *true* si la cola está vacía y *false* si no

`boolean` **`cola_llena`**`(Cola s)`: devuelve *true* si la cola está llena y *false* si no

`status` **`cola_insertar`**`(Cola s, Elemento e)`: inserta un dato en una cola

`Elemento` **`cola_extraer`**`(Cola s)`: extrae el dato que ocupa el front de la cola

`Elemento` **`cola_front`**`(Cola s)`: accede al dato que ocupa el front de la cola sin extraerlo

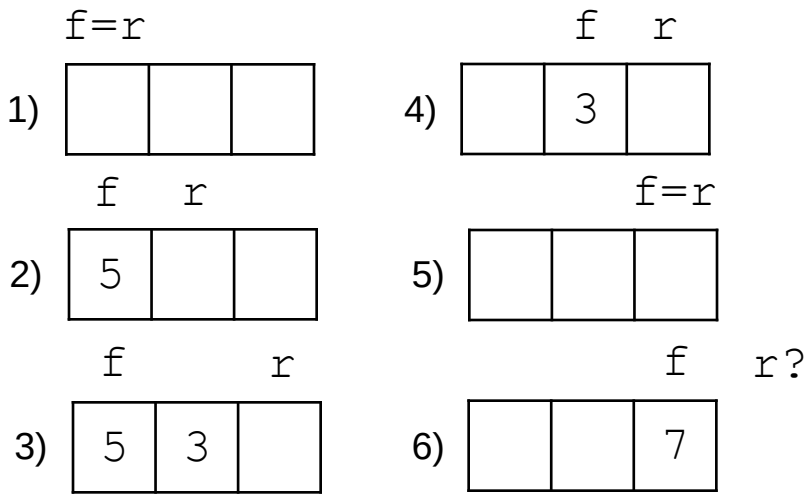
`Elemento` **`cola_rear`**`(Cola s)`: accede al dato que ocupa el rear de la cola sin extraerlo

Contenidos

- El TAD Cola
- Estructura de datos y primitivas de Cola
- **Estructura de datos de Cola como array circular**
- Implementación en C de Cola
  - Implementación con front y rear de tipo entero
- Anexo
  - Implementación con front y rear de tipo puntero

Estructura de datos de Cola como array circular

- Ejemplo de ejecución de operaciones en una cola
  1) cola_inicializar(q)
  2) cola_insertar(q, 5)
  3) cola_insertar(q, 3)
  4) cola_extraer(q)
  5) cola_extraer(q)
  6) cola_insertar(q, 7)



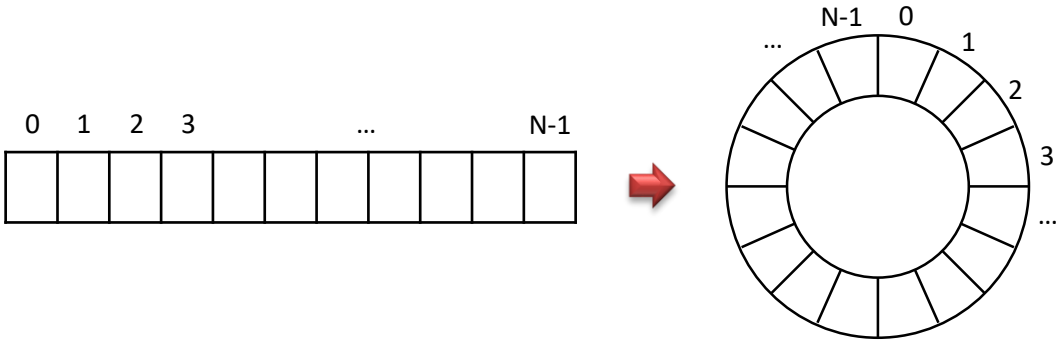
- Problemas
  - Limitación del número máximo de elementos
  - Desperdicio de espacio

Estructura de datos de Cola como array circular

- Soluciones al desperdicio de espacio
  1) Cada vez que se extrae un elemento, se desplazan todos los datos una posición en el array
     - Ineficiente
  2) Cuando *rear* llega al final del array, se desplazan todos los elementos para que estén situados desde el comienzo del array
     - (menos) Ineficiente
  3) Implementación de la cola como un **array circular**
     - Más eficiente

Estructura de datos de Cola como array circular

- **Cola circular**



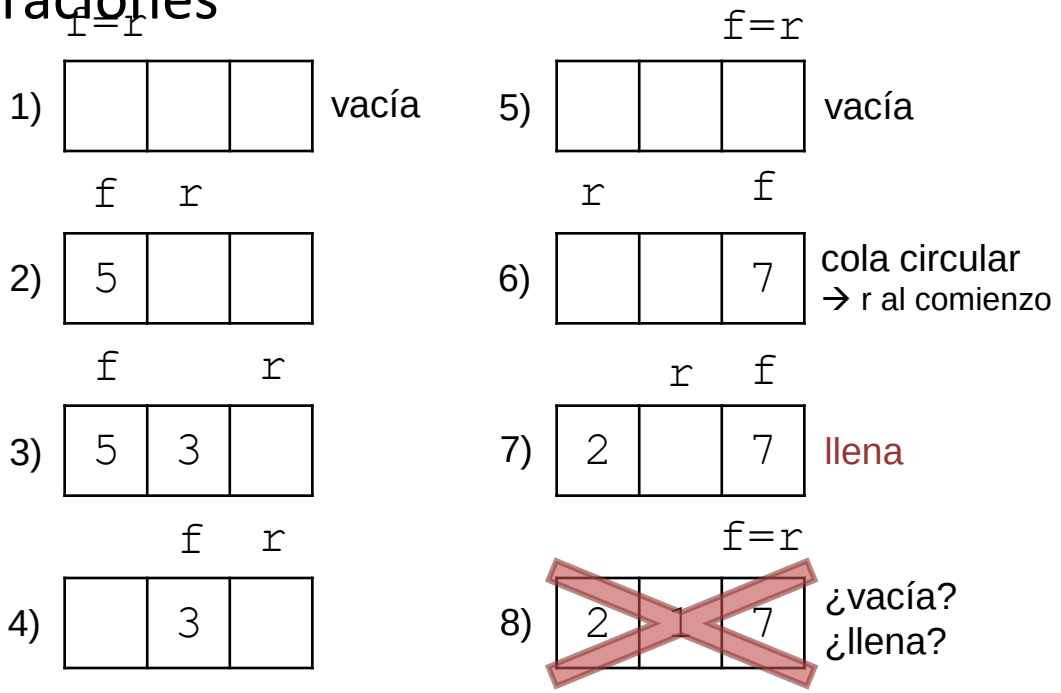
- ¿Cómo implementarla?
  - Incrementando *front* y *rear* módulo COLA_MAX

```
front = (front+1) % COLA_MAX
rear = (rear+1) % COLA_MAX
```

- Problema vigente
  - Limitación del número máximo de elementos

Estructura de datos de Cola como array circular

- Ejemplo de ejecución de operaciones
  1) cola_inicializar(q)
  2) cola_insertar(q, 5)
  3) cola_insertar(q, 3)
  4) cola_extraer(q, e)
  5) cola_extraer(q, e)
  6) cola_insertar(q, 7)
  7) cola_insertar(q, 2)
  8) cola_insertar(q, 1)



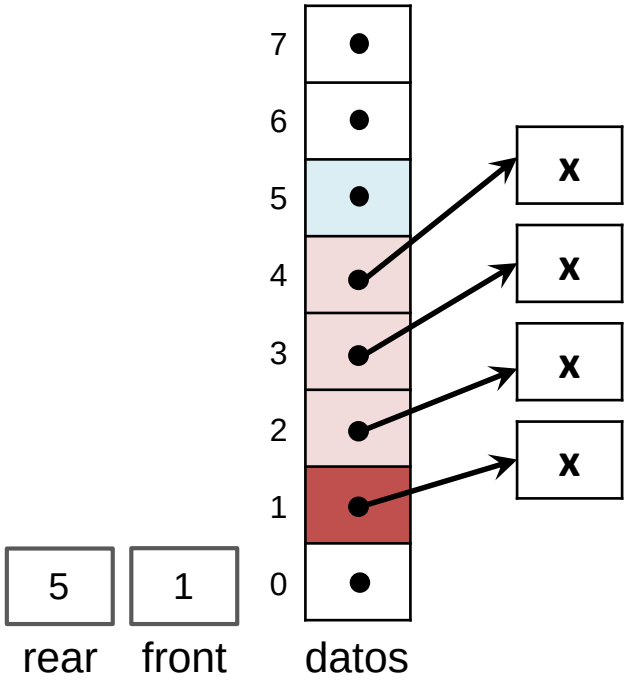
- Conflicto cola llena/vacía
  - front == rear → ¿Cola vacía o llena?
  - **Solución**: sacrificar un hueco libre en el array
    → Prohibir la inserción cuando sólo queda un hueco (7) sería cola llena)
    → Una cola de tamaño COLA_MAX tiene espacio para COLA_MAX -1 elementos

Contenidos

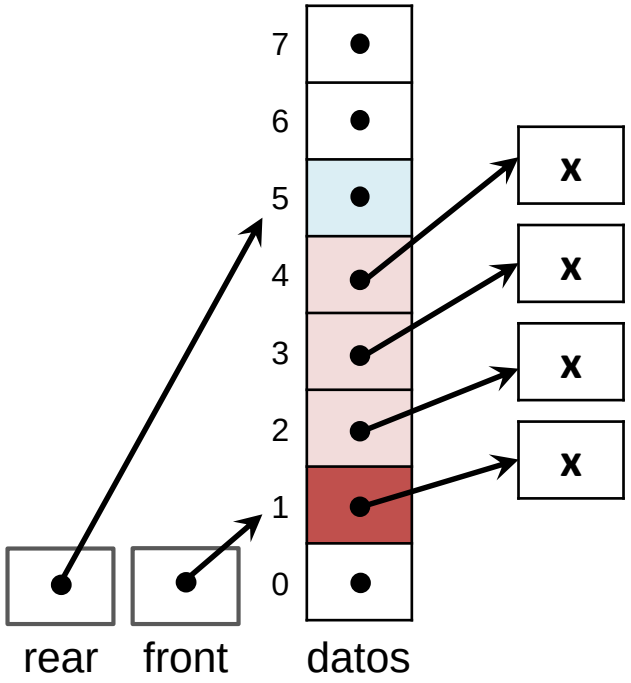
- El TAD Cola
- Estructura de datos y primitivas de Cola
- Estructura de datos de Cola como array circular
- **Implementación en C de Cola**
  - **Implementación con front y rear de tipo entero**
- Anexo
  - Implementación con front y rear de tipo puntero

Estructura de datos y primitivas de Cola

- **EdD en C**
  - **datos**: en este tema será un array de punteros: `Elemento *datos[];`
  - **front**, **rear**: en este tema se declarará de 2 maneras (versiones) distintas
    - Como enteros: `int front, rear;`
    - Como punteros a elemento del array: `Elemento **front, **rear;`



Versión 1

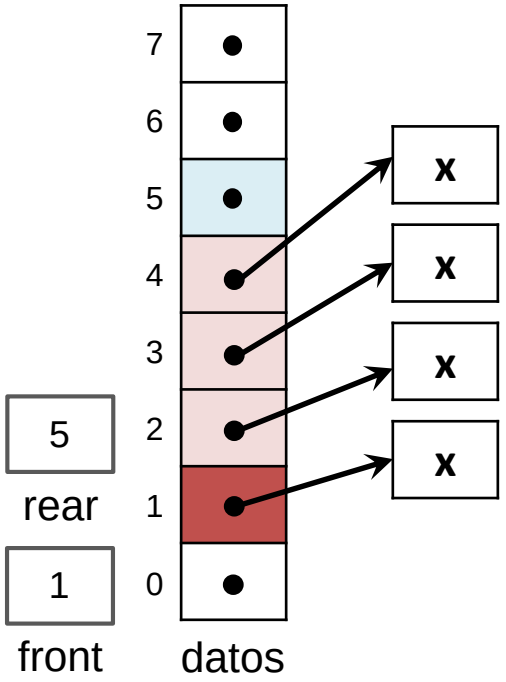


Versión 2

Implementación en C de Cola

- **Implementación con front y rear de tipo entero**
  - Se asume la existencia del TAD Elemento
    - Array es de punteros a Elemento
  - **EdD de Cola mediante un array**

```
// En cola.h
typedef struct _Cola Cola;

// En cola.c
#define COLA_MAX 8
struct _Cola {
   Elemento *datos[COLA_MAX];
   int      front;   // Primer elemento
   int      rear;    // Primer hueco tras
                     // el último elemento
};
```



Implementación en C de Cola

- **Implementación con front y rear de tipo entero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

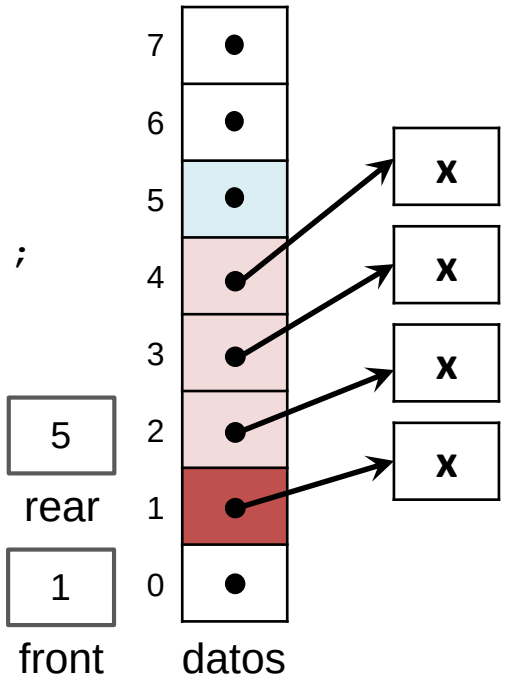
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en cola.h)**

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
boolean cola_vacia(const Cola *pq);
boolean cola_llena(const Cola *pq);
status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
```

  - **Estructura de datos (en cola.c)**

```
struct _Cola {
   Elemento *datos[COLA_MAX];
   int      front;
   int      rear
};
```



Implementación en C de Cola

- **Implementación con front y rear de tipo entero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

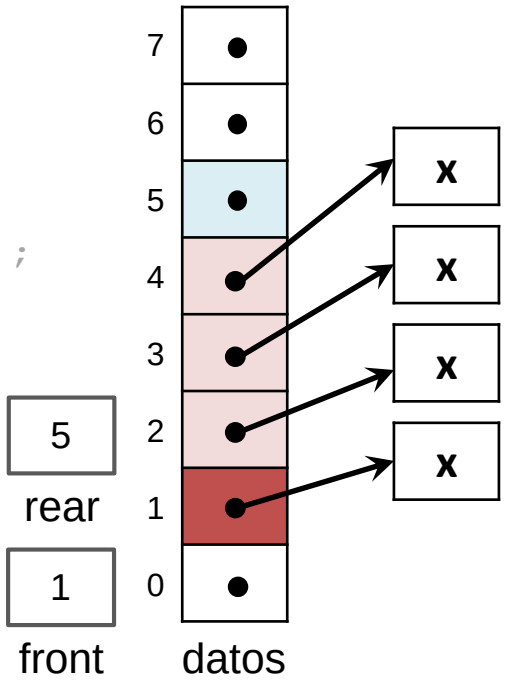
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en cola.h)**

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
boolean cola_vacia(const Cola *pq);
boolean cola_llena(const Cola *pq);
status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
```

  - **Estructura de datos (en cola.c)**

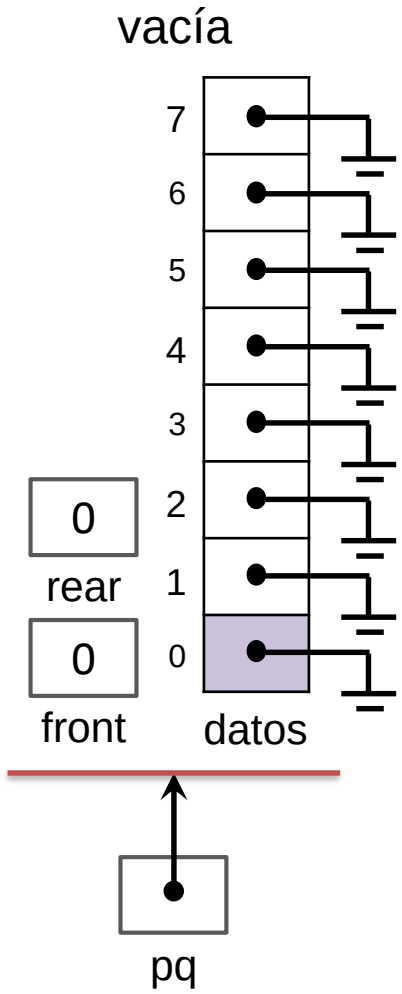
```
struct _Cola {
   Elemento *datos[COLA_MAX];
   int      front;
   int      rear
};
```



Implementación en C de Cola

- **Implementación con front y rear de tipo entero**

```
Cola *cola_crear() {
    Cola *pq = NULL;
    int i;

    pq = (Cola *) malloc(sizeof(Cola));
    if (pq==NULL) {
        return NULL;
    }

    pq->front = 0;
    pq->rear = 0;

    for(i=0;i<COLA_MAX;i++){
        pq->datos[i]=NULL;
    }

    return pq;
}
```

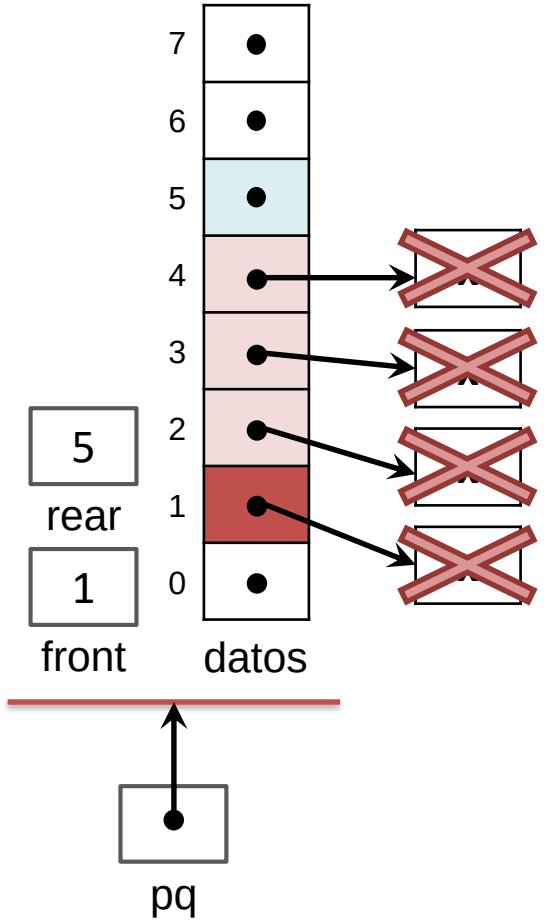


Implementación en C de Cola

- **Implementación con front y rear de tipo entero**

Existe: `void` **`elemento_liberar`**`(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    int i;

    if (pq!=NULL) {
        i = pq->front;
        while (i!=pq->rear) {
            elemento_liberar(pq->datos[i]);
            i = (i+1) % COLA_MAX;
        }

    }
}
```

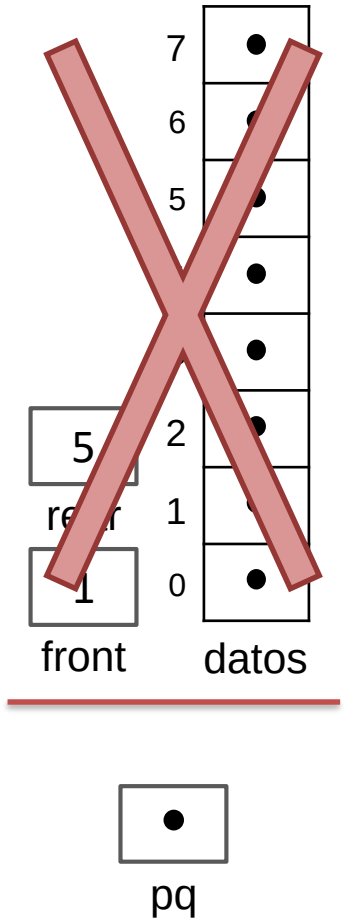


Implementación en C de Cola

- **Implementación con front y rear de tipo entero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    int i;

    if (pq!=NULL) {
        i = pq->front;
        while (i!=pq->rear) {
            elemento_liberar(pq->datos[i]);
            i = (i+1) % COLA_MAX;
        }
        free(pq);
    }
}
```



Implementación en C de Cola

- **Implementación con front y rear de tipo entero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

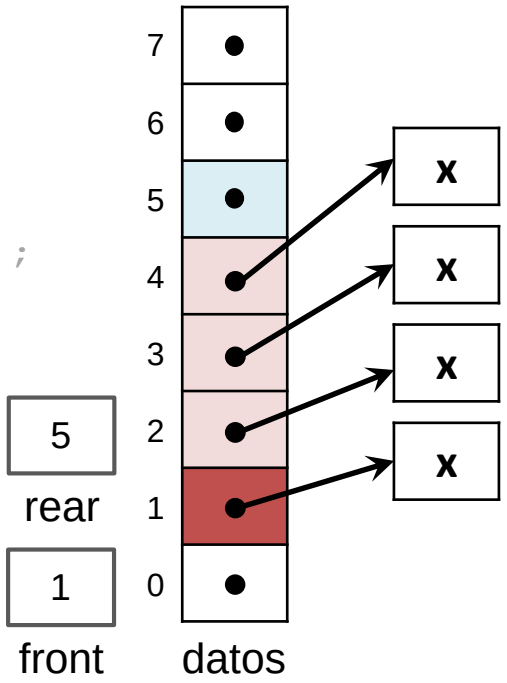
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en cola.h)**

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
boolean cola_vacia(const Cola *pq);
boolean cola_llena(const Cola *pq);
status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
```

  - **Estructura de datos (en cola.c)**

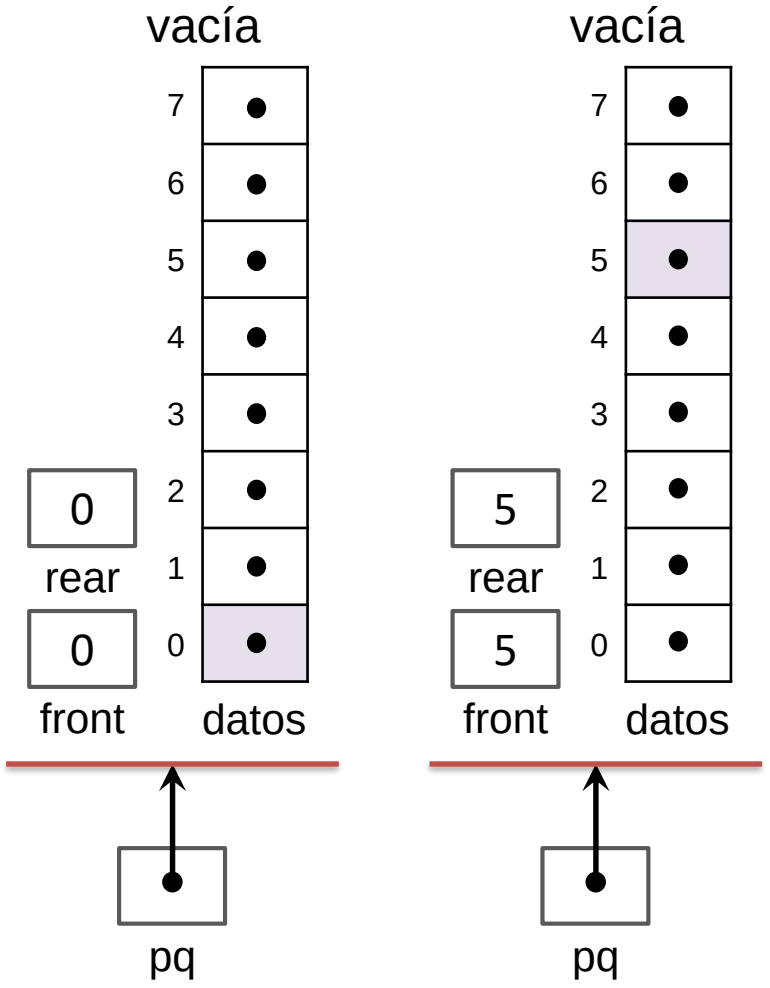
```
struct _Cola {
   Elemento *datos[COLA_MAX];
   int      front;
   int      rear
};
```



Implementación en C de Cola

- **Implementación con front y rear de tipo entero**

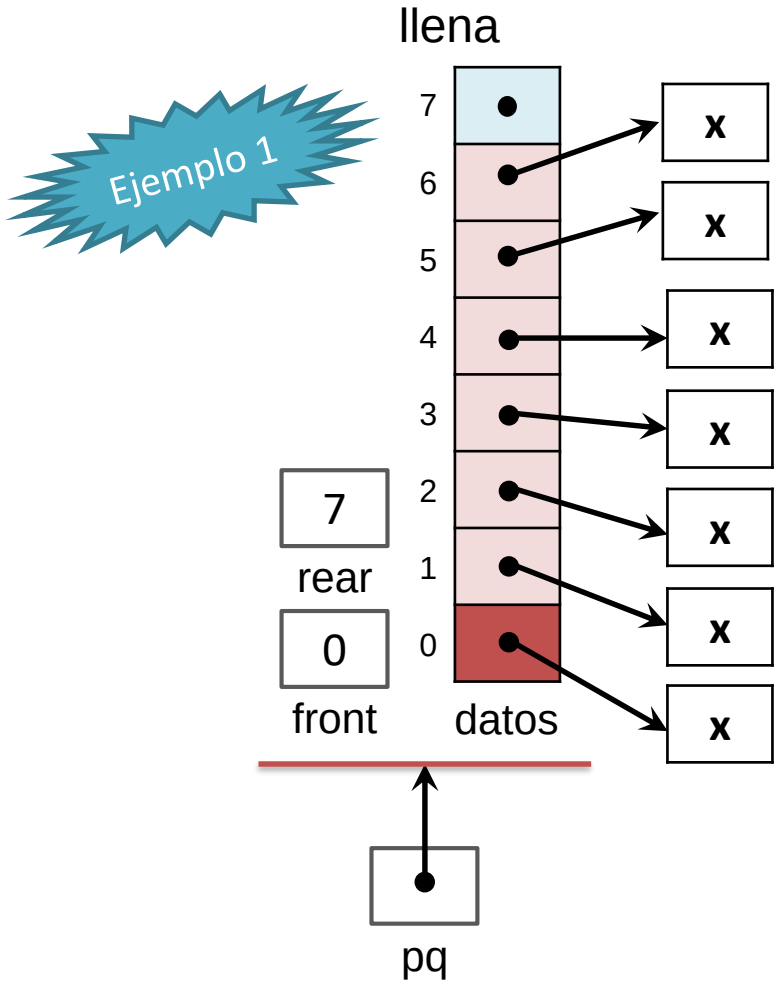
```
boolean cola_vacia(const Cola *pq) {
    if (pq == NULL) {
        return TRUE;
    }

    if (pq->front == pq->rear) {
        return TRUE;
    }
    return FALSE;
}
```



Implementación en C de Cola

- **Implementación con front y rear de tipo entero**

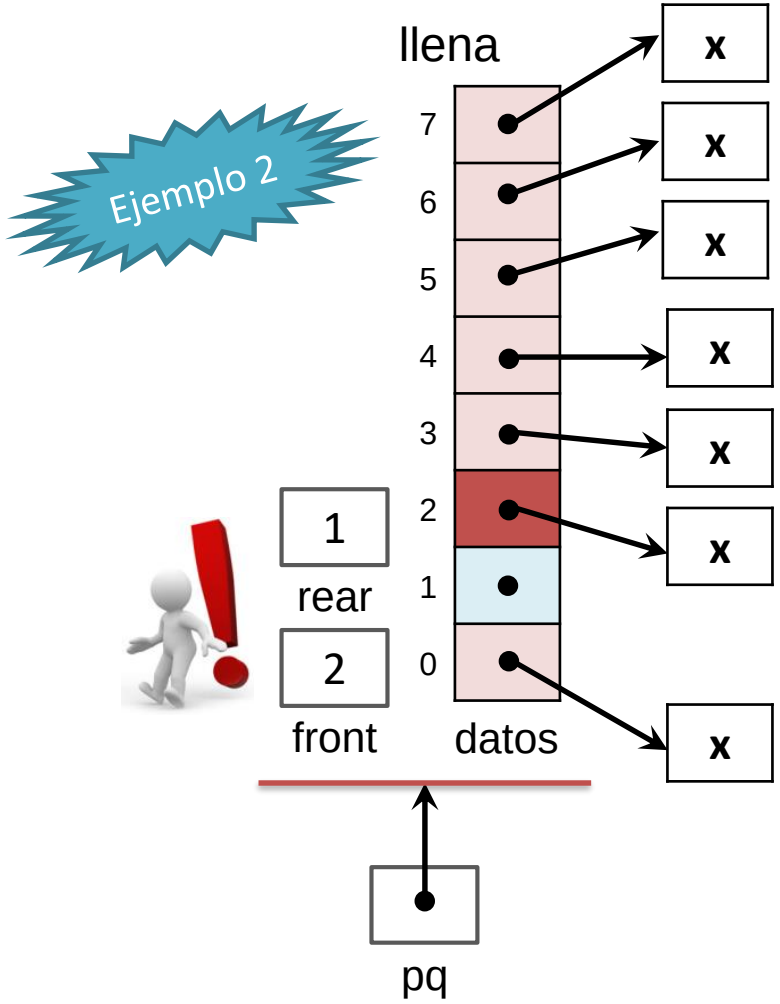
```
boolean cola_llena(const Cola *pq) {
    if (pq == NULL) {
        return TRUE;
    }

    if (pq->front == (pq->rear+1)%COLA_MAX) {
        return TRUE;
    }
    return FALSE;
}
```



Implementación en C de Cola

- **Implementación con front y rear de tipo entero**

```
boolean cola_llena(const Cola *pq) {
    if (pq == NULL) {
        return TRUE;
    }

    if (pq->front == (pq->rear+1)%COLA_MAX) {
        return TRUE;
    }
    return FALSE;
}
```



Implementación en C de Cola

- **Implementación con front y rear de tipo entero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

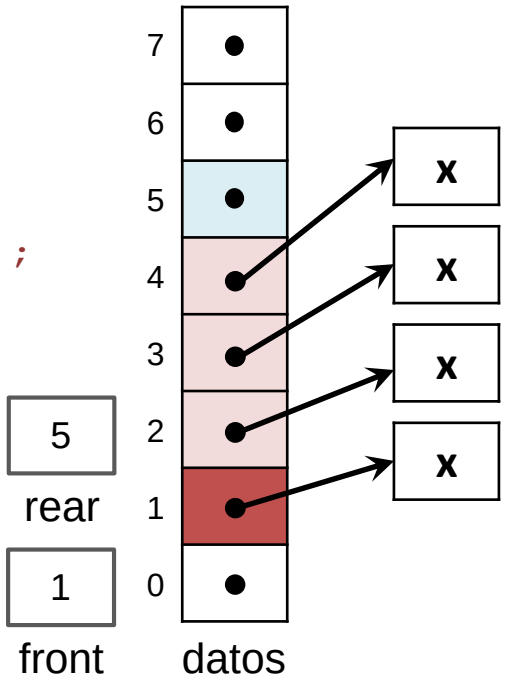
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en cola.h)**

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
boolean cola_vacia(const Cola *pq);
boolean cola_llena(const Cola *pq);
status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
```

  - **Estructura de datos (en cola.c)**

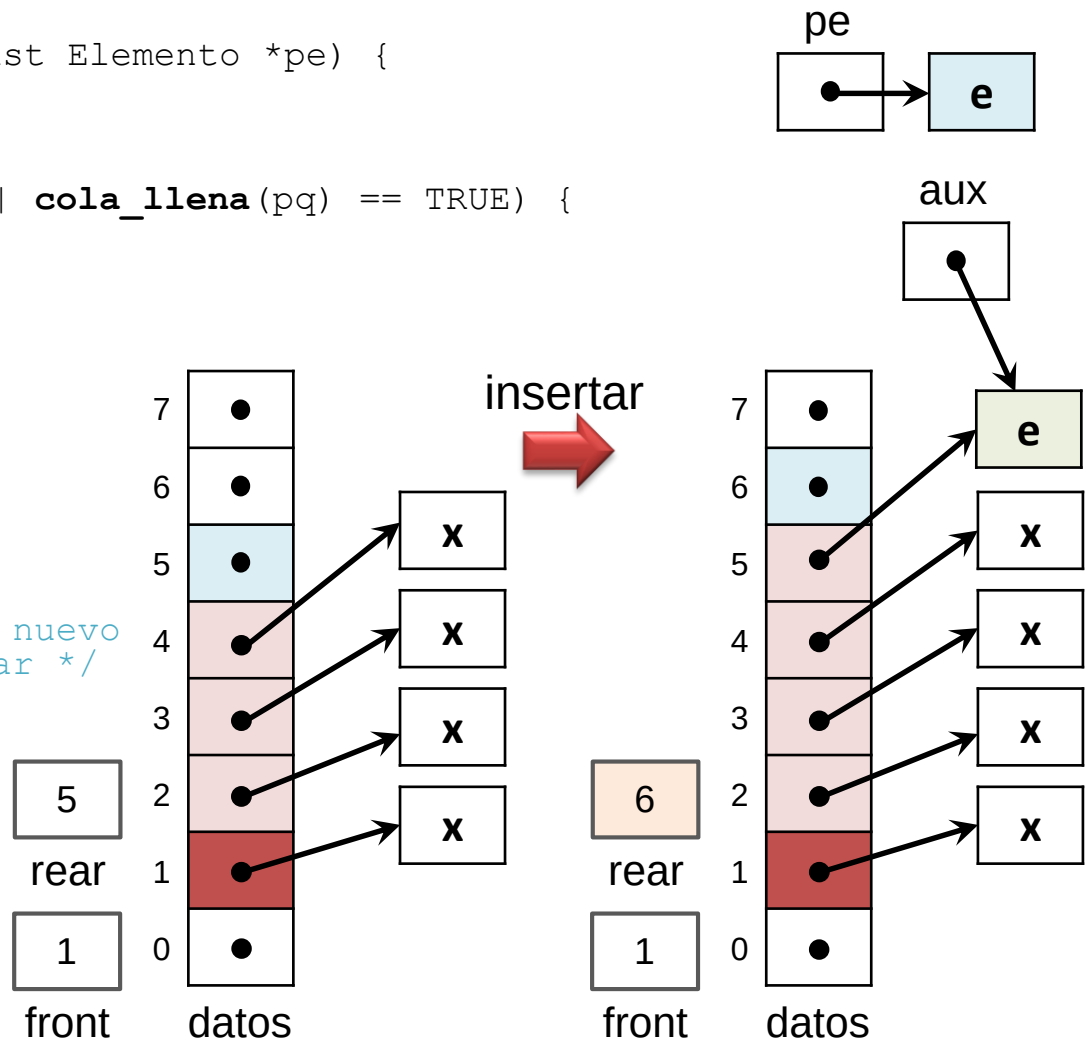
```
struct _Cola {
   Elemento *datos[COLA_MAX];
   int      front;
   int      rear
};
```



Implementación en C de Cola

- **Implementación con front y rear de tipo entero**

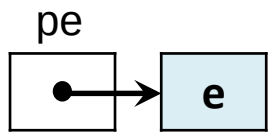
```
status cola_insertar(Cola *pq, const Elemento *pe) {
    Elemento *aux = NULL;

    if (pq == NULL || pe == NULL || cola_llena(pq) == TRUE) {
        return ERROR;
    }

    aux = elemento_copiar(pe);
    if (aux == NULL) {
        return ERROR;
    }

    /* Guarda el ptero al elemento nuevo
    en la posición indicada por rear */
    pq->datos[pq->rear] = aux;

    /* Actualiza el rear */
    pq->rear=(pq->rear+1)%COLA_MAX;

    return OK;
}
```

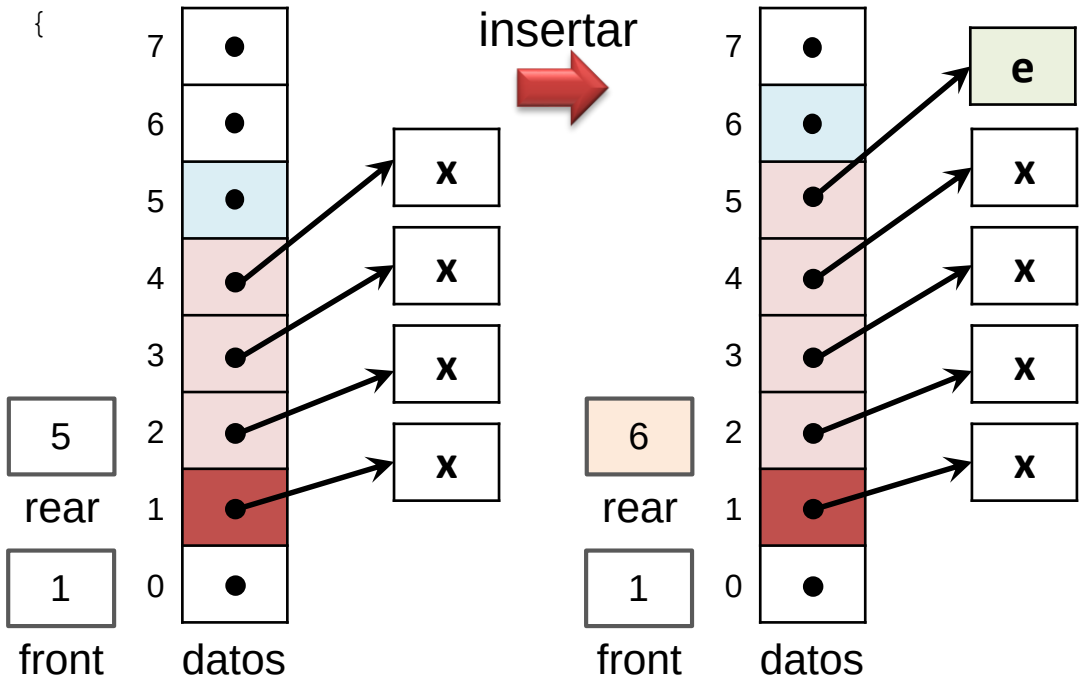


Implementación en C de Cola

- **Implementación con front y rear de tipo entero (versión sin la variable aux)**

```
status cola_insertar(Cola *pq, const Elemento *pe) {
    if (pq == NULL || pe == NULL || cola_llena(pq) == TRUE) {
        return ERROR;
    }
     /* Hacemos copia del dato y lo guardamos en la posición que indica el rear */
    pq->datos[pq->rear]= elemento_copiar(pe);
    if (pq->datos[pq->rear]== NULL) {
        return ERROR;
    }
    /* Actualizamos el rear */
    pq->rear=(pq->rear+1)%COLA_MAX;

    return OK;
}
```





Implementación en C de Cola

- **Implementación con front y rear de tipo entero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

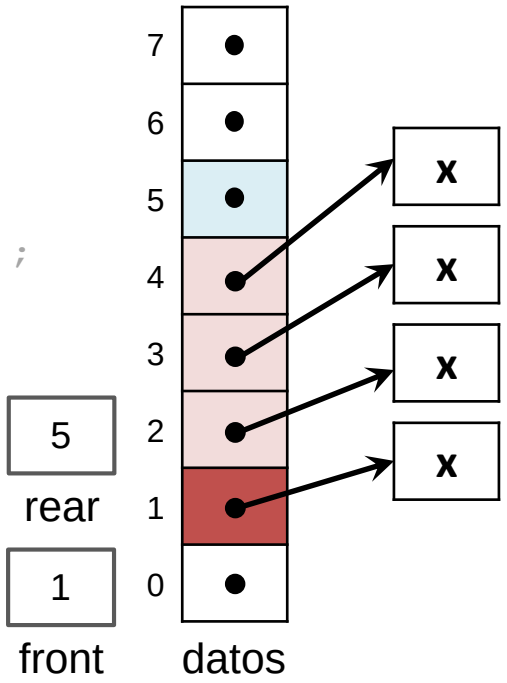
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en cola.h)**

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
boolean cola_vacia(const Cola *pq);
boolean cola_llena(const Cola *pq);
status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
```

  - **Estructura de datos (en cola.c)**

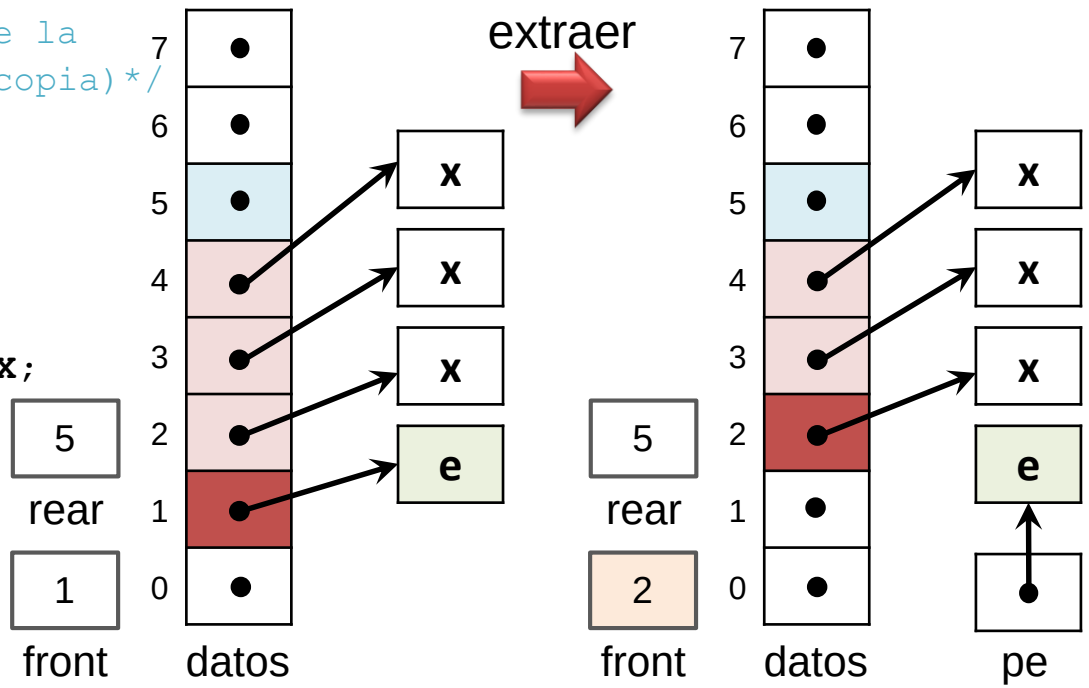
```
struct _Cola {
   Elemento *datos[COLA_MAX];
   int      front;
   int      rear
};
```



Implementación en C de Cola

- **Implementación con front y rear de tipo entero**

```
Elemento *cola_extraer(Cola *pq){
    Elemento *pe = NULL;

    if (pq == NULL || cola_vacia(pq) == TRUE) {
        return NULL;
    }

    /* Devolverá el ptero a Elemto de la
    posición indicada por front (no copia)*/
    pe = pq->datos[pq->front];

   pq->datos[pq->front]=NULL;

    /* Actualiza el front */
    pq->front = (pq->front+1) % COLA_MAX;

    return pe;
}
```



Contenidos

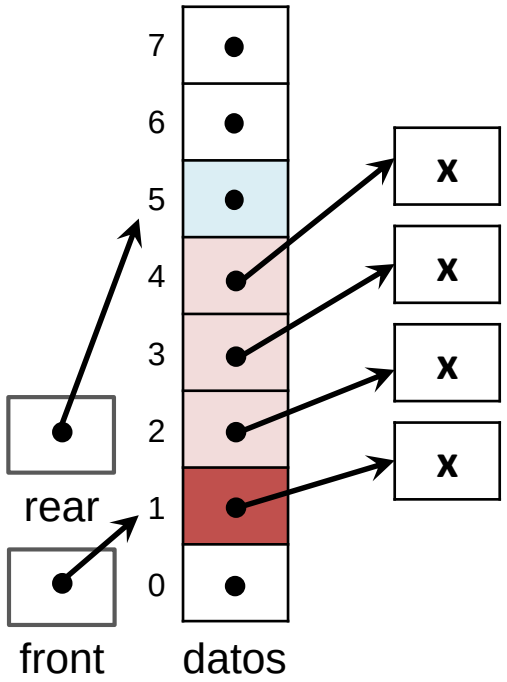
- El TAD Cola
- Estructura de datos y primitivas de Cola
- Estructura de datos de Cola como array circular
- Implementación en C de Cola
  - Implementación con front y rear de tipo entero

- **Anexo**
  - **Implementación con front y rear de tipo puntero**

Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**
  - Se asume la existencia del TAD Elemento
  - **EdD de Cola mediante un array**

```
// En cola.h
typedef struct _Cola Cola;

// En cola.c
#define COLA_MAX 8
struct _Cola {
   Elemento *datos[COLA_MAX];
   Elemento **front; // Primer elemento
   Elemento **rear;  // Ultimo elemento
};
```



Implementación en C de Cola

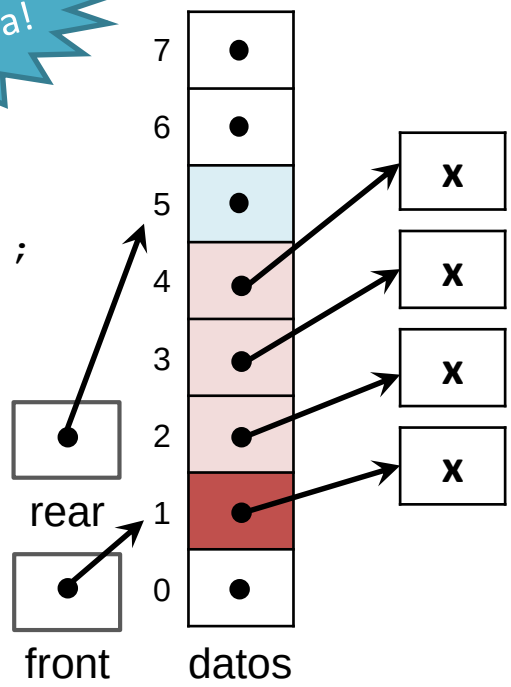
- **Implementación con front y rear de tipo puntero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en cola.h)**

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
boolean cola_vacia(const Cola *pq);
boolean cola_llena(const Cola *pq);
status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
```

¡La interfaz NO cambia!



  - **Estructura de datos (en cola.c)**

```
struct _Cola {
   Elemento *datos[COLA_MAX];
   Elemento **front;
   Elemento **rear;
};
```

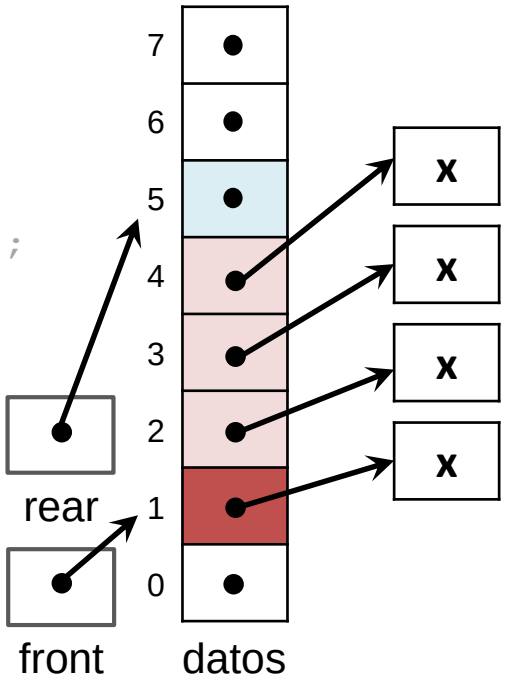
Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en cola.h)**

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
boolean cola_vacia(const Cola *pq);
boolean cola_llena(const Cola *pq);
status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
```



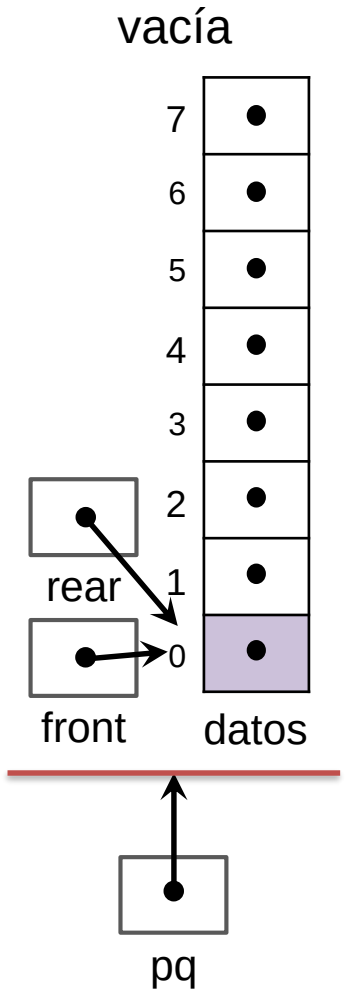
  - **Estructura de datos (en cola.c)**

```
struct _Cola {
   Elemento *datos[COLA_MAX];
   Elemento **front;
   Elemento **rear;
};
```

Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

```
Cola *cola_crear() {
    Cola *pq = NULL;
    int i;

    pq = (Cola *) malloc(sizeof(Cola));
    if (pq==NULL) {
        return NULL;
    }

    pq->rear = pq->datos;    //pq->rear = &(pq->datos[0]);
    pq->front= pq->datos;   //pq->front = &(pq->datos[0]);

    return pq;
}
```

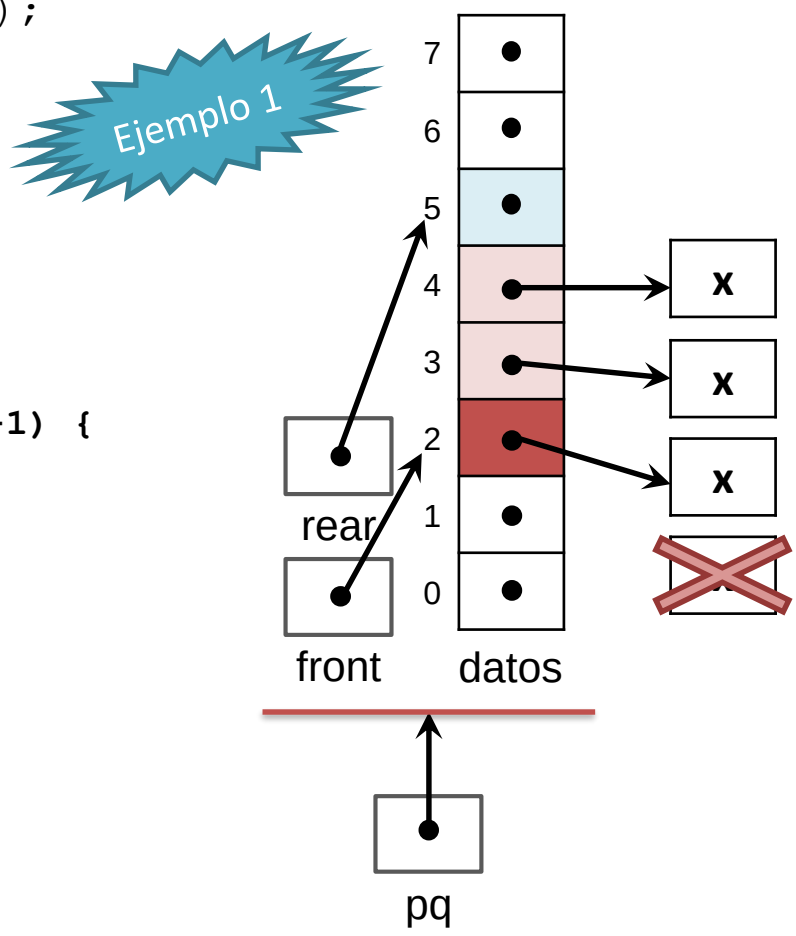


Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    if (pq != NULL) {
        while (pq->front != pq->rear) {

            elemento_liberar(*(pq->front));

            if (pq->front != pq->datos+COLA_MAX-1) {
                pq->front = pq->front+1;
            }
            else {
                pq->front = pq->datos;
            }
        }

    }
}
```

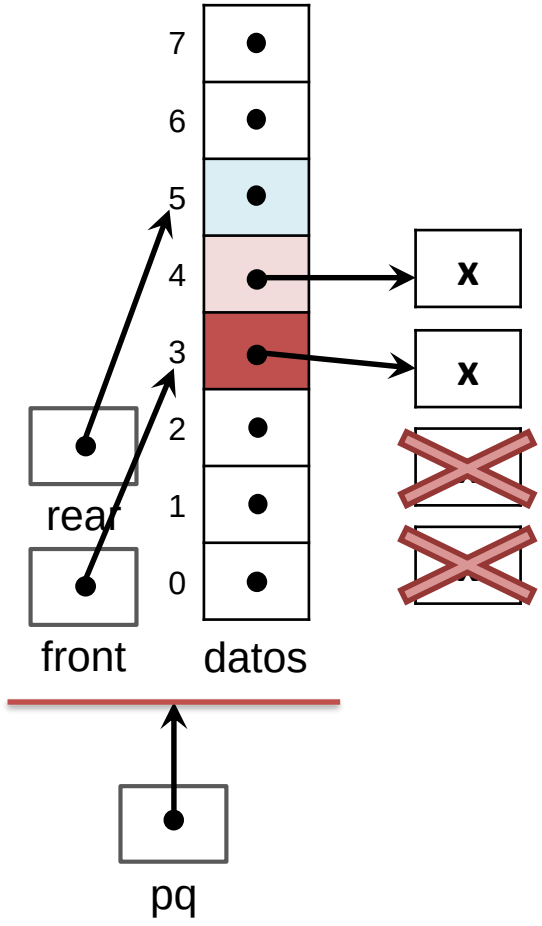


Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    if (pq != NULL) {
        while (pq->front != pq->rear) {

            elemento_liberar(*(pq->front));

            if (pq->front != pq->datos+COLA_MAX-1) {
                pq->front = pq->front+1;
            }
            else {
                pq->front = pq->datos;
            }
        }

    }
}
```

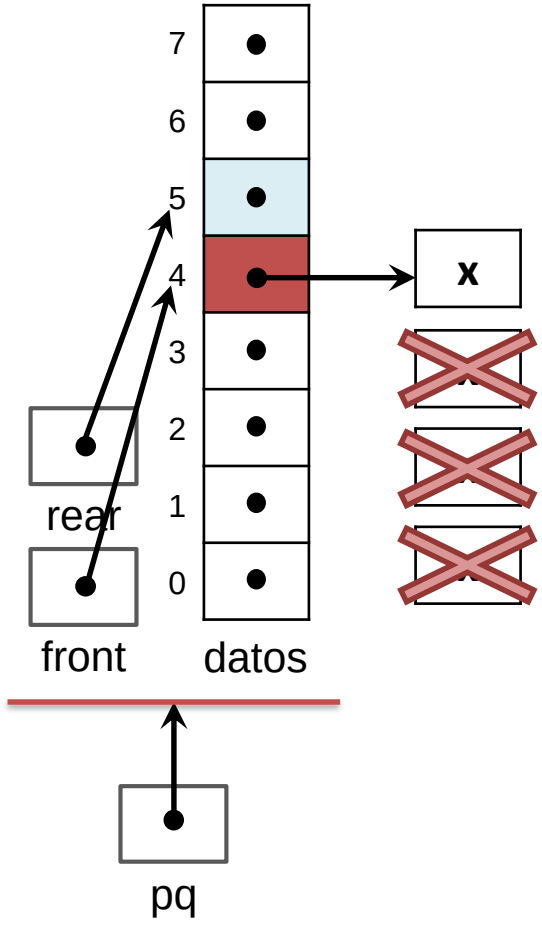


Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    if (pq != NULL) {
        while (pq->front != pq->rear) {

            elemento_liberar(*(pq->front));

            if (pq->front != pq->datos+COLA_MAX-1) {
                pq->front = pq->front+1;
            }
            else {
                pq->front = pq->datos;
            }
        }

    }
}
```

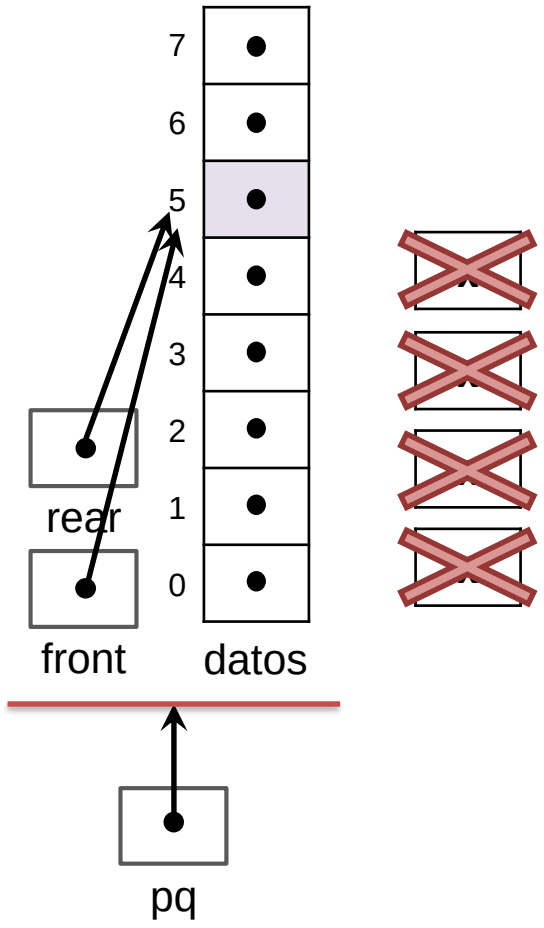


Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    if (pq != NULL) {
        while (pq->front != pq->rear) {

            elemento_liberar(*(pq->front));

            if (pq->front != pq->datos+COLA_MAX-1) {
                pq->front = pq->front+1;
            }
            else {
                pq->front = pq->datos;
            }
        }

    }
}
```

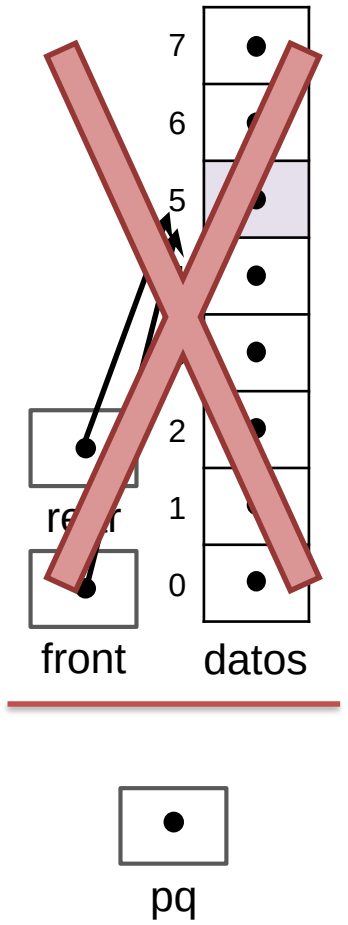


Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    if (pq != NULL) {
        while (pq->front != pq->rear) {

            elemento_liberar(*(pq->front));

            if (pq->front != pq->datos+COLA_MAX-1) {
                pq->front = pq->front+1;
            }
            else {
                pq->front = pq->datos;
            }
        }
        free(pq);
    }
}
```

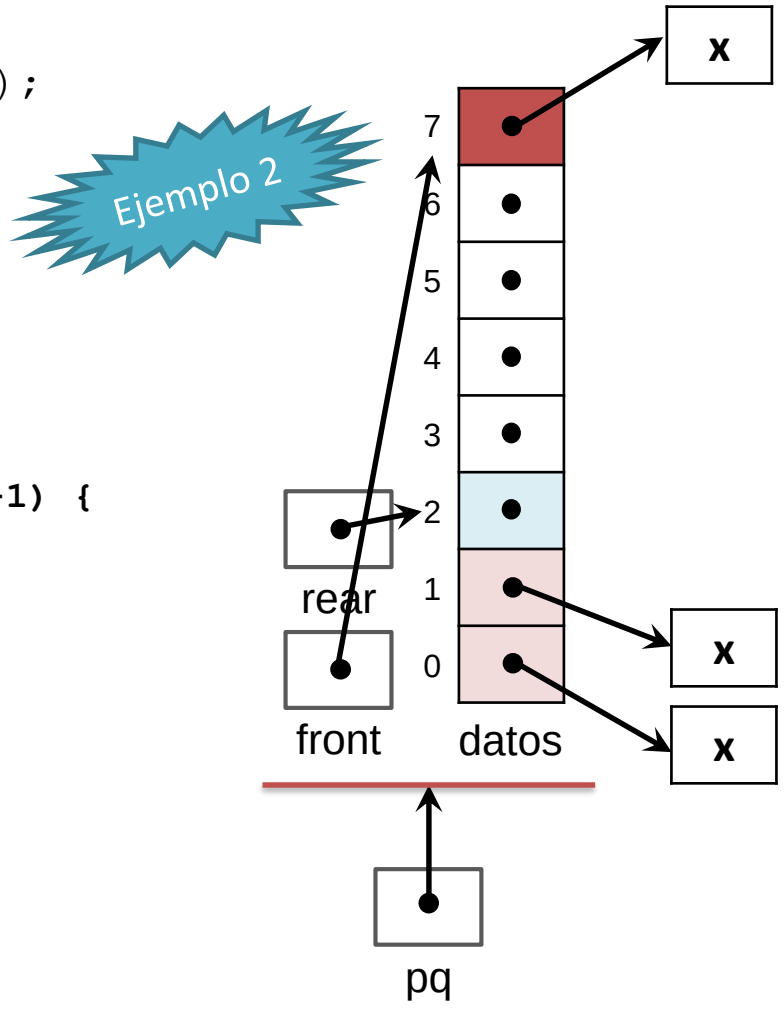


Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

Existe: `void` **`elemento_liberar`**`(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    if (pq != NULL) {
        while (pq->front != pq->rear) {

            elemento_liberar(*(pq->front));

            if (pq->front != pq->datos+COLA_MAX-1) {
                pq->front = pq->front+1;
            }
            else {
                pq->front = pq->datos;
            }
        }
        free(pq);
    }
}
```

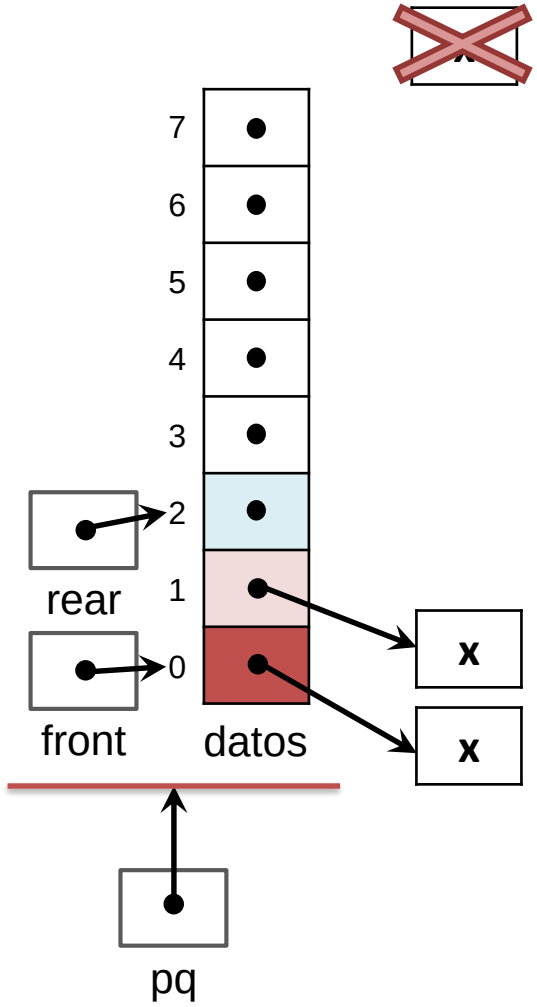


Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    if (pq != NULL) {
        while (pq->front != pq->rear) {

            elemento_liberar(*(pq->front));

            if (pq->front != pq->datos+COLA_MAX-1) {
                pq->front = pq->front+1;
            }
            else {
                pq->front = pq->datos;
            }
        }
        free(pq);
    }
}
```

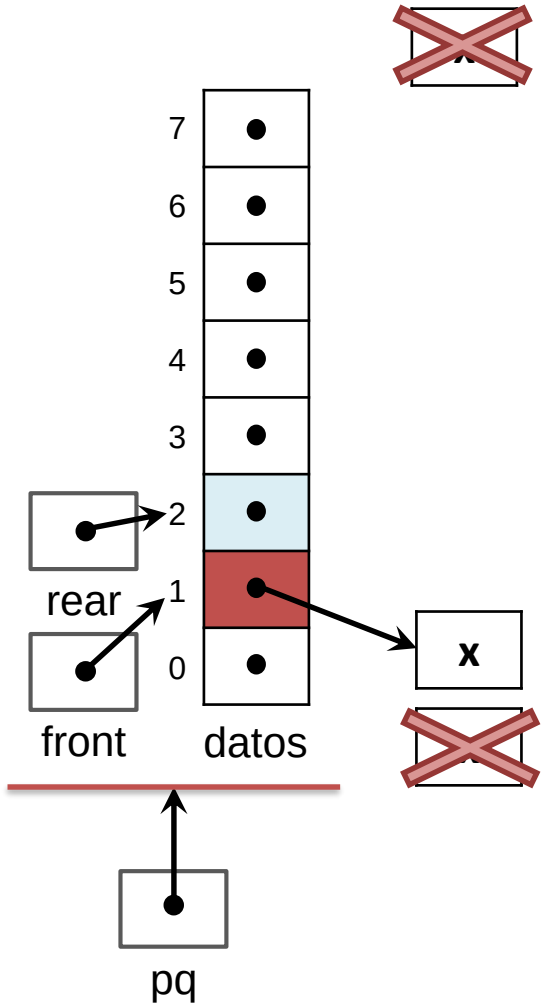


Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    if (pq != NULL) {
        while (pq->front != pq->rear) {

            elemento_liberar(*(pq->front));

            if (pq->front != pq->datos+COLA_MAX-1) {
                pq->front = pq->front+1;
            }
            else {
                pq->front = pq->datos;
            }
        }
        free(pq);
    }
}
```

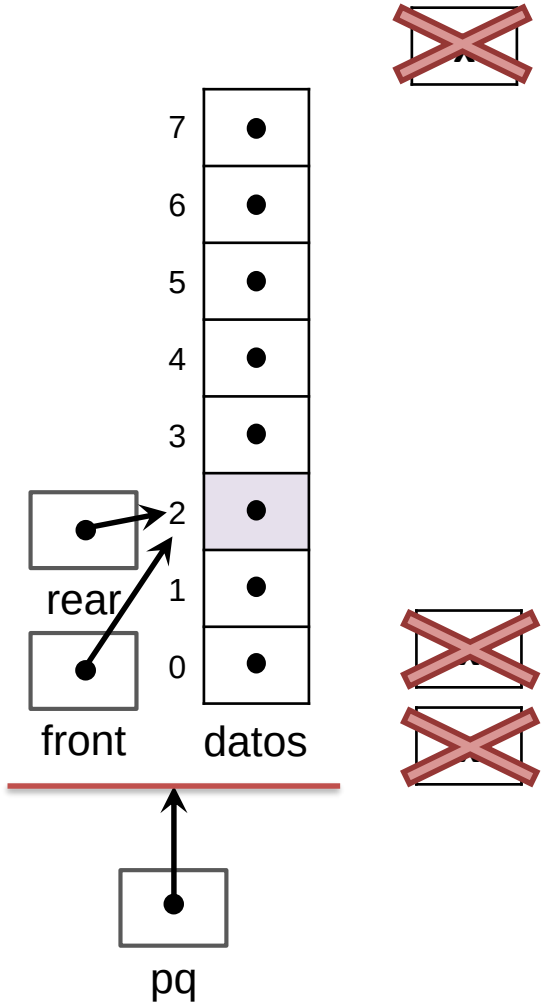


Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    if (pq != NULL) {
        while (pq->front != pq->rear) {

            elemento_liberar(*(pq->front));

            if (pq->front != pq->datos+COLA_MAX-1) {
                pq->front = pq->front+1;
            }
            else {
                pq->front = pq->datos;
            }
        }
        free(pq);
    }
}
```

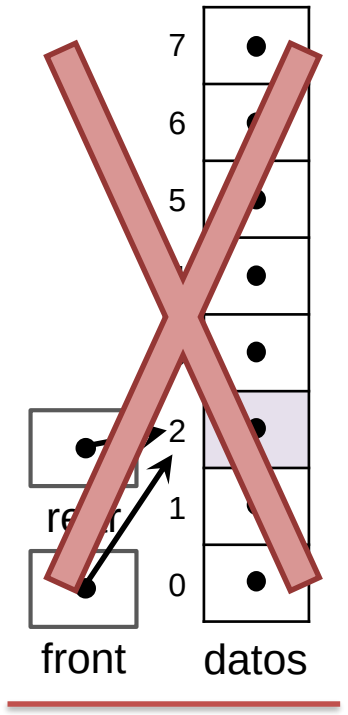


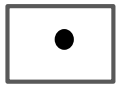
Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

Existe: `void` **`elemento_liberar`**`(Elemento *pe);`

```
void cola_liberar(Cola *pq) {
    if (pq != NULL) {
        while (pq->front != pq->rear) {

            elemento_liberar(*(pq->front));

            if (pq->front != pq->datos+COLA_MAX-1) {
                pq->front = pq->front+1;
            }
            else {
                pq->front = pq->datos;
            }
        }
        free(pq);
    }
}
```





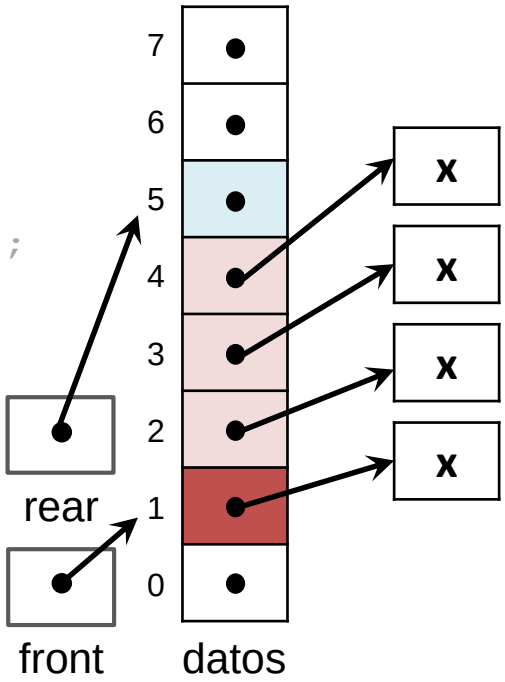
Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en cola.h)**

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
boolean cola_vacia(const Cola *pq);
boolean cola_llena(const Cola *pq);
status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
```



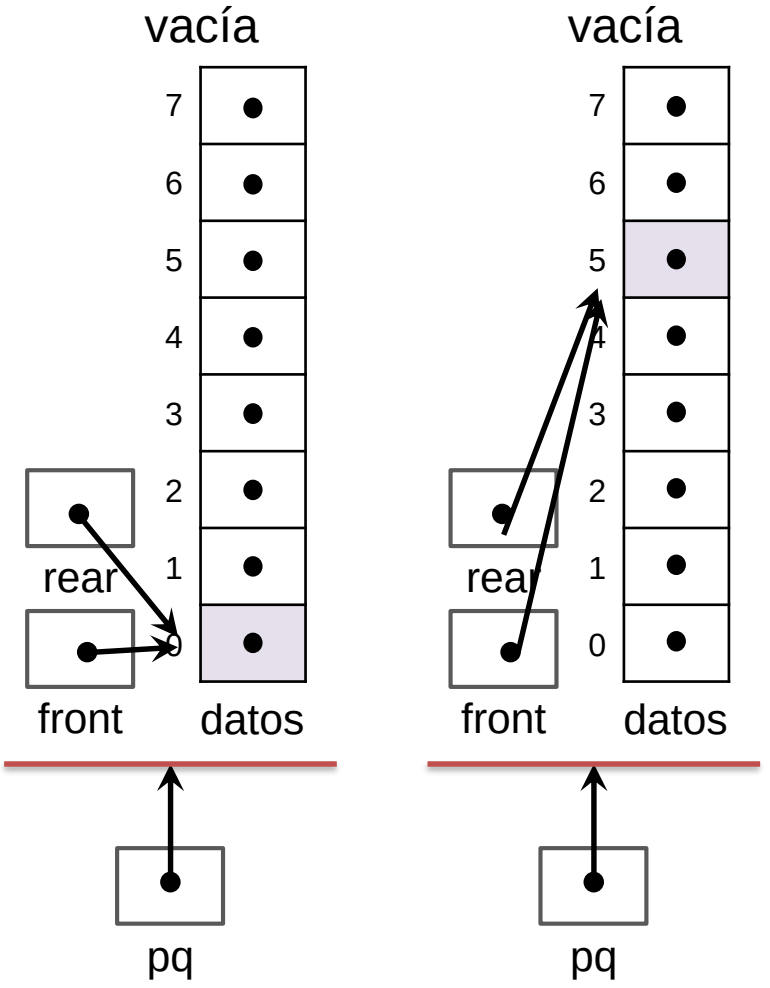
  - **Estructura de datos (en cola.c)**

```
struct _Cola {
   Elemento *datos[COLA_MAX];
   Elemento **front;
   Elemento **rear;
};
```

Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

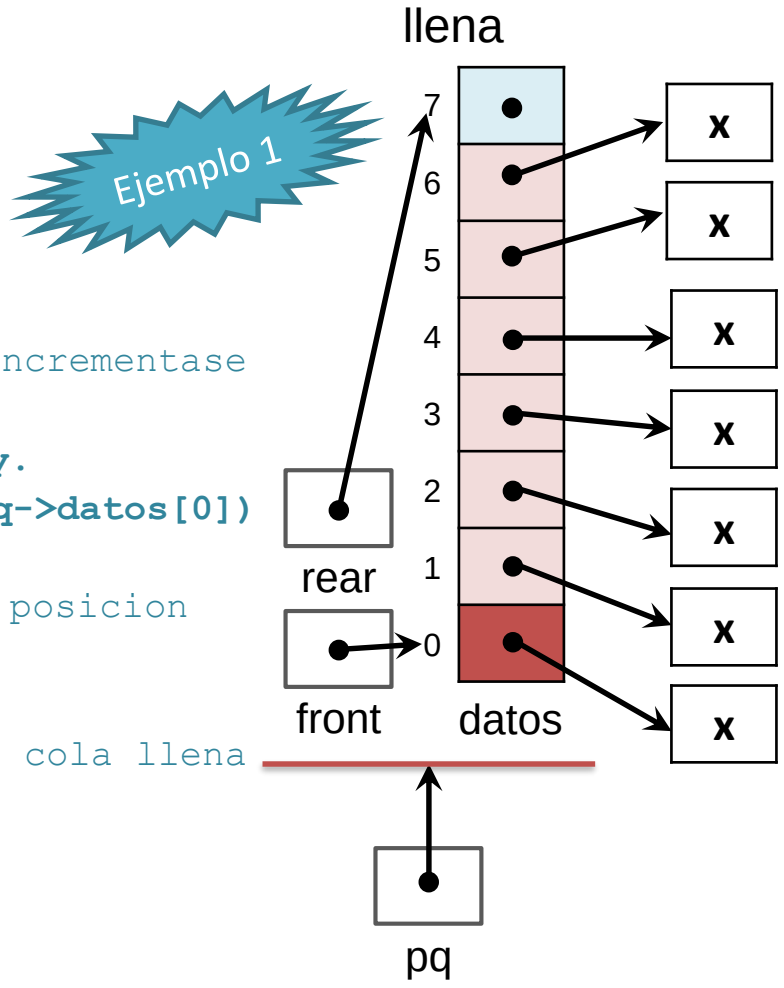
```
boolean cola_vacia(const Cola *pq) {
    if (pq == NULL) {
        return TRUE;
    }

    if (pq->rear == pq->front) {
        return TRUE;
    }
    return FALSE;
}
```



Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

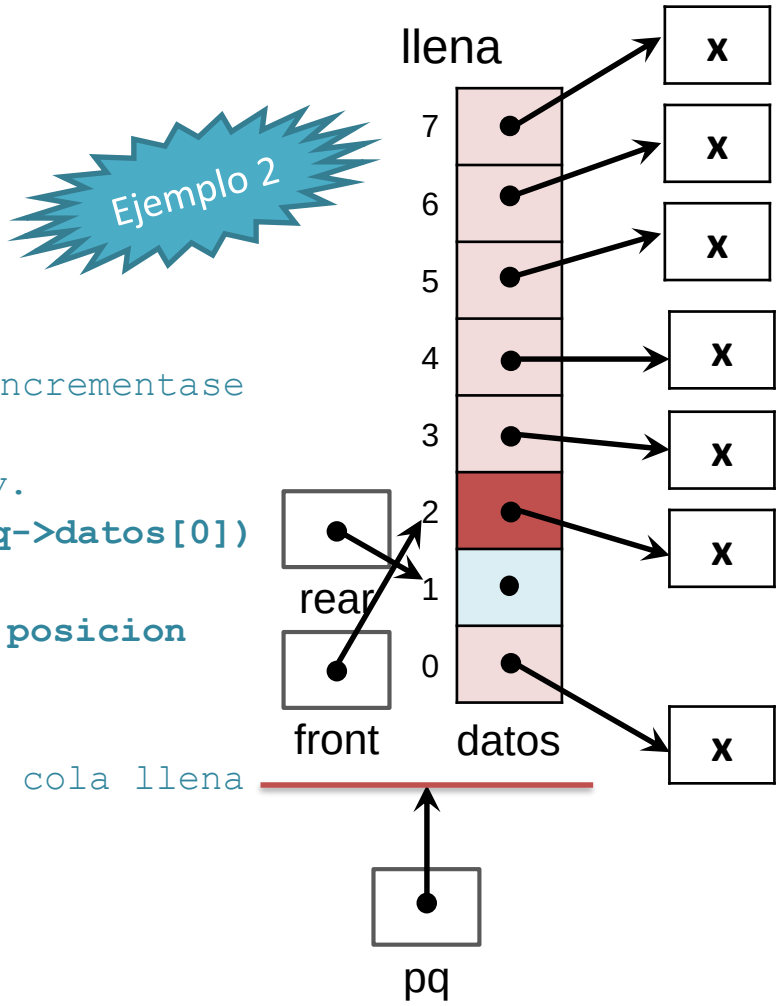
```
boolean cola_llena(const Cola *pq) {
    Elemento **aux = NULL;

    if (pq == NULL) {
        return TRUE;
    }

    // Apuntamos aux donde avanzaría rear si se incrementase
    if (pq->rear==pq->datos+COLA_MAX-1) {
        aux = pq->datos;  // Al comienzo del array.
    }                     // Equivale a aux = &(pq->datos[0])
    else {
        aux = pq->rear + 1;     // A la siguiente posicion
    }

    // Si aux (que es rear+1) coincide con front, cola llena
    if (aux == pq->front) {
        return TRUE;
    }
    return FALSE;
}
```



Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

```
boolean cola_llena(const Cola *pq) {
    Elemento **aux = NULL;

    if (pq == NULL) {
        return TRUE;
    }

    // Apuntamos aux donde avanzaría rear si se incrementase
    if (pq->rear==pq->datos+COLA_MAX-1) {
        aux = pq->datos;  // Al comienzo del array.
    }                     // Equivale a aux = &(pq->datos[0])
    else {
        aux = pq->rear + 1;     // A la siguiente posicion
    }

    // Si aux (que es rear+1) coincide con front, cola llena
    if (aux == pq->front) {
        return TRUE;
    }
    return FALSE;
}
```



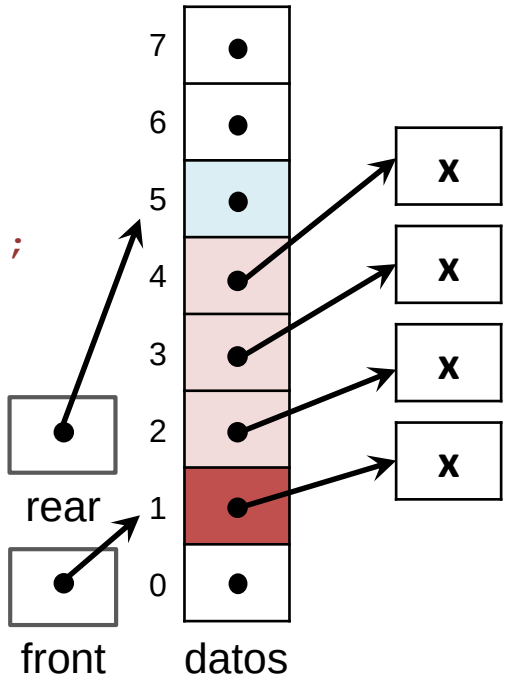
Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en cola.h)**

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
boolean cola_vacia(const Cola *pq);
boolean cola_llena(const Cola *pq);
status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
```



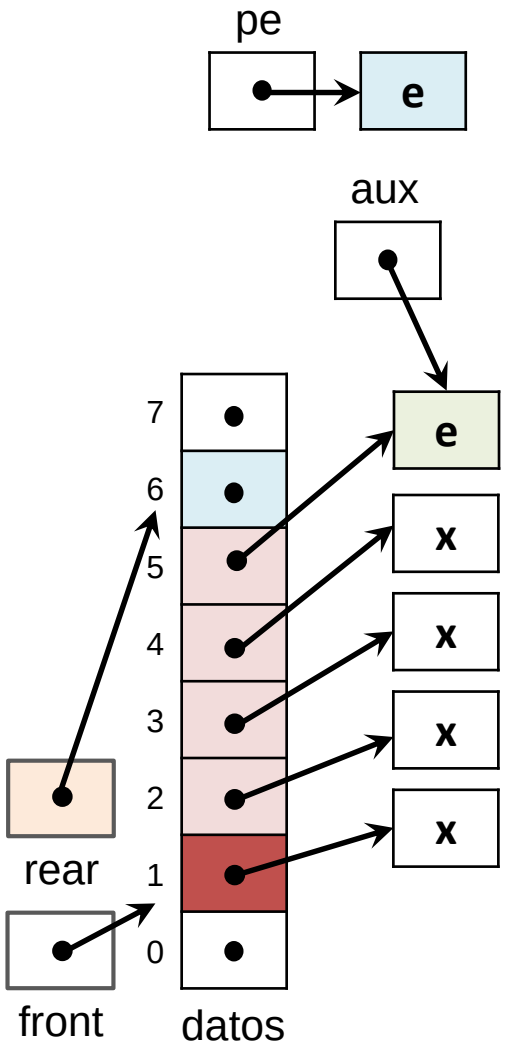
  - **Estructura de datos (en cola.c)**

```
struct _Cola {
   Elemento *datos[COLA_MAX];
   Elemento **front;
   Elemento **rear;
};
```

Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

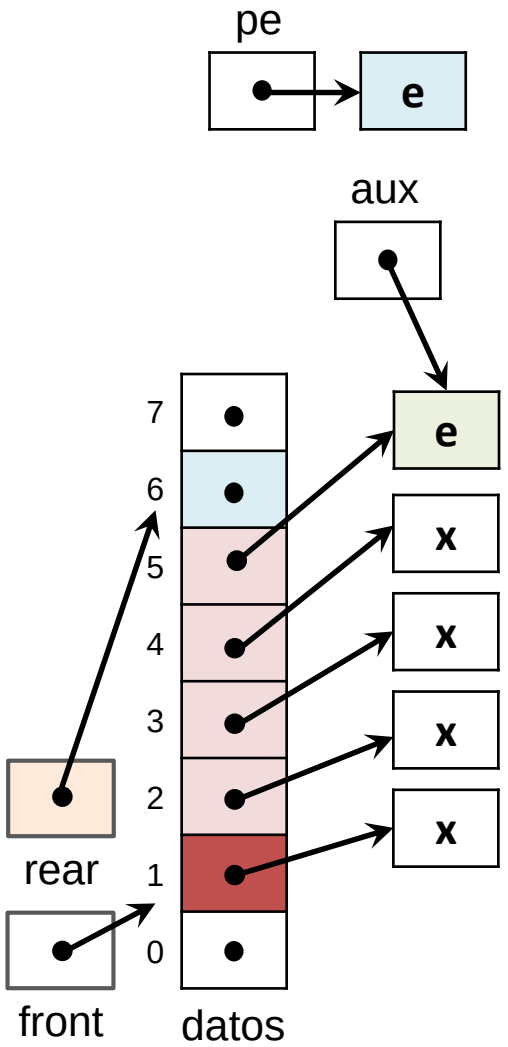
```
status cola_insertar(Cola *pq, const Elemento *pe){
    Elemento *aux = NULL;

    if (pq == NULL || pe == NULL || cola_llena(pq) == TRUE) {
        return ERROR;
    }

    aux = elemento_copiar(pe);
    if (aux == NULL) {
        return ERROR;
    }

    /* Guardamos el dato en el lugar apuntado por rear */
    *(pq->rear) = aux;

    /* Actualizamos el rear */
    if (pq->rear == pq->datos+COLA_MAX-1) {
        pq->rear = pq->datos;  // pq->rear = &(pq->datos[0])
    }
    else {
        pq->rear++;
    }

    return OK;
}
```



Implementación en C de Cola

- **Implementación con front y rear de tipo puntero (versión sin variable aux)**

```
status cola_insertar(Cola *pq, const Elemento *pe){
    if (pq == NULL || pe == NULL || cola_llena(pq) == TRUE) {
        return ERROR;
    }
    /* Guardamos el dato en el lugar apuntado por rear */
    *(pq->rear) = elemento_copiar(pe);
    if ( *(pq->rear) == NULL) {
        return ERROR;
    }
    /* Actualizamos el rear */
    if (pq->rear == pq->datos+COLA_MAX-1) {
        pq->rear = pq->datos;  // pq->rear = &(pq->datos[0])
    }
    else {
        pq->rear++;
    }

    return OK;
}
```



Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

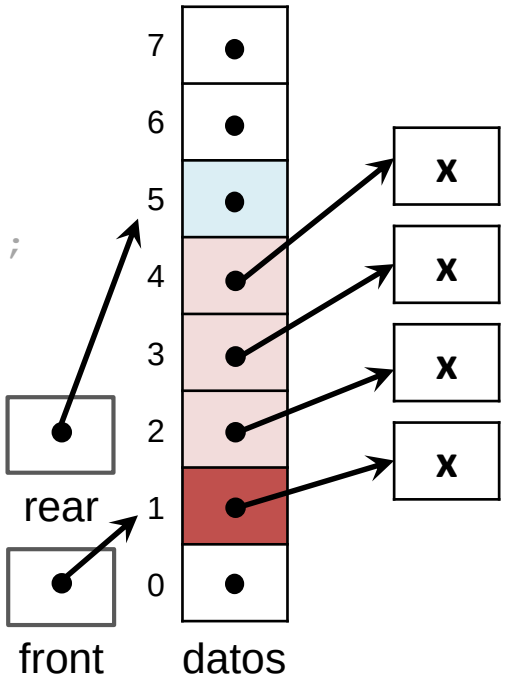
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en cola.h)**

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
boolean cola_vacia(const Cola *pq);
boolean cola_llena(const Cola *pq);
status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
```

  - **Estructura de datos (en cola.c)**

```
struct _Cola {
   Elemento *datos[COLA_MAX];
   Elemento **front;
   Elemento **rear;
};
```



Implementación en C de Cola

- **Implementación con front y rear de tipo puntero**

```
Elemento *cola_extraer(const Cola *pq){
    Elemento *pe = NULL;

    if (pq == NULL || cola_vacia(pq) == TRUE) {
        return NULL;
    }

    /* Recuperamos el dato del lugar apuntado por el front */
    pe = *(pq->front);

    /* Actualizamos el front */
    if (pq->front == pq->datos+COLA_MAX-1) {
        pq->front = pq->datos;  //pq->front= &(pq->datos[0])
    }
    else {
        pq->front++;
    }

    return pe;
}
```