Analista Programador Universitario

Estructura de Datos

Unidad IV: TDA Cola

Facultad de Ingeniería
Universidad Nacional de Jujuy

¿Qué es una cola o fila?

Índice

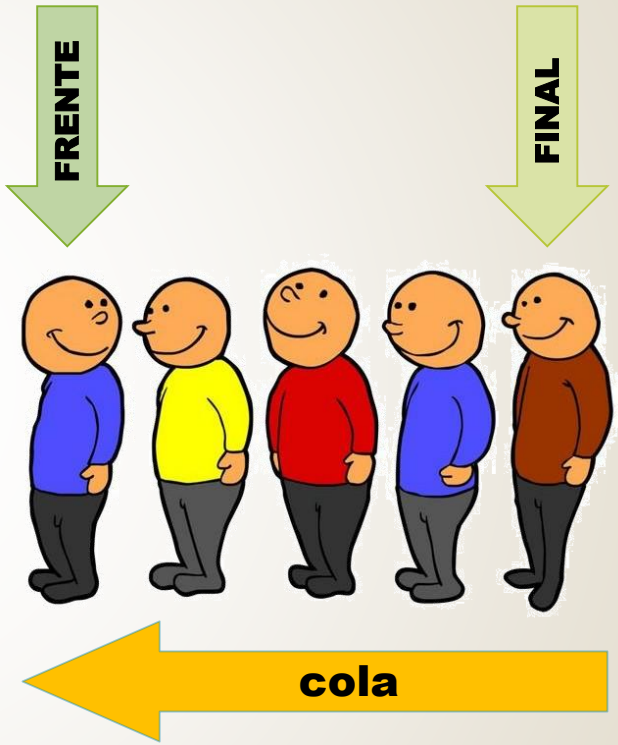
- Definición del TDA cola o fila
- Operaciones fundamentales
- Implementación
- Aplicaciones

Definición (1)

- Una cola o fila es una colección ordenada de elementos, con 3 características:
  - contiene elementos del mismo tipo (estructura homogénea),
  - la recuperación de elementos se realiza según el orden de almacenamiento (acceso FIFO) y
  - la cantidad de elementos almacenados varía durante la ejecución (estructura dinámica).

Definición (2)

- En una computadora, los elementos de una cola pueden almacenarse a partir de una dirección de memoria ocupando ordenadas y posiciones consecutivas o no consecutivas según la implementación.
- La recuperación de elementos se realiza por el frente y la inserción por el final.



Operaciones Fundamentales (1)

- Sobre el TDA cola se definen las siguientes operaciones:
  - Iniciar cola (*Init_Queue*)
  - Agregar elemento (*Push_Queue*)
  - Extraer elemento (*Pop_Queue*)
  - Determinar cola vacía (*Empty_Queue*)
  - Determinar cola llena (*Full_Queue*)
  - Consulta primer elemento (*Top_Queue*)
  - Consulta último elemento (*Bottom_Queue*)

Operaciones Fundamentales (2)

- *Init_Queue (iniciar cola)*
  - Propósito: inicializar la fila o cola (esto genera una cola vacía).
  - Entrada: cola de datos.
  - Salida: cola de datos inicializada (cola vacía).
  - Restricciones: ninguna.

Operaciones Fundamentales (3)

- *Push_Queue (encolar)*
  - Propósito: agregar un elemento al final de la fila o cola de datos.
  - Entrada: cola de datos y un nuevo elemento.
  - Salida: cola de datos con un nuevo elemento al final.
  - Restricciones: cola inicializada y contenedor de datos no completo.
- *Full_Queue (cola llena)*
  - Propósito: determinar si el contenedor de datos de la cola está completo.
  - Entrada: cola de datos.
  - Salida: valor lógico *true* si el contenedor está completo o *false* en caso contrario.
  - Restricciones: cola inicializada.

Operaciones Fundamentales (4)

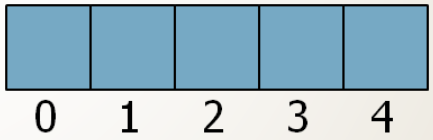
- *Pop_Queue (desencolar)*
  - Propósito: quitar el elemento del frente de la fila o cola de datos.
  - Entrada: cola de datos.
  - Salida: cola de datos con un elemento menos, se modifica el frente.
  - Restricciones: cola inicializada y contenedor de datos no vacío.
- *Empty_Queue (cola vacía)*
  - Propósito: determinar si el contenedor de datos de la cola está vacío.
  - Entrada: cola de datos.
  - Salida: valor lógico *true* si el contenedor está vacío o *false* en caso contrario.
  - Restricciones: cola inicializada.

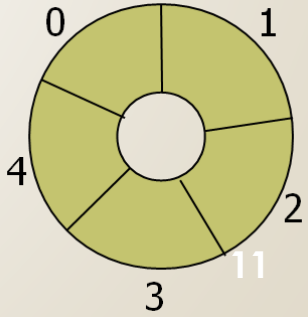
Operaciones Fundamentales (5)

- *Top_Queue (frente o primero de la cola)*
  - Propósito: consultar el primer elemento de la cola de datos (frente).
  - Entrada: cola de datos.
  - Salida: primer elemento de la cola de datos.
  - Restricciones: cola inicializada y contenedor de datos no vacío.
- *Bottom_Queue (final o último de la cola)*
  - Propósito: consultar último elemento de la cola de datos (final).
  - Entrada: cola de datos.
  - Salida: último elemento de la cola de datos.
  - Restricciones: cola inicializada y contenedor de datos no vacío.

Alternativas de Implementación (1)

- Un contenedor de datos y un índice. El frente coincide con la primer posición del contenedor, mientras que el índice apunta al elemento final de la cola.
- Un contenedor y 2 índices. Los índices se utilizan para apuntar, respectivamente, al frente y final de la cola.
  - almacenamiento lineal
  - almacenamiento circular





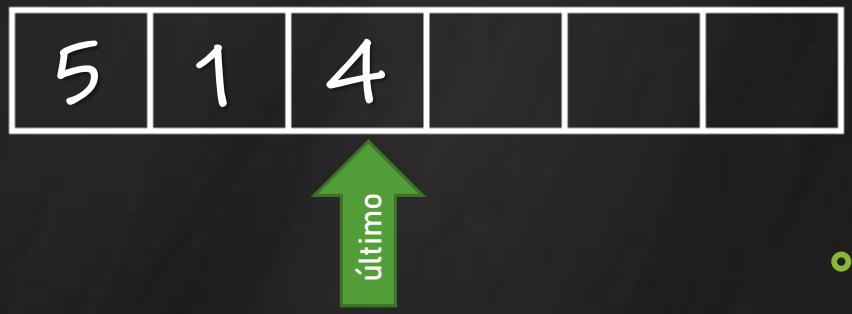
Alternativas de implementación con arreglos

Almacenamiento Lineal (único índice)

- Agregar

- Quitar

8



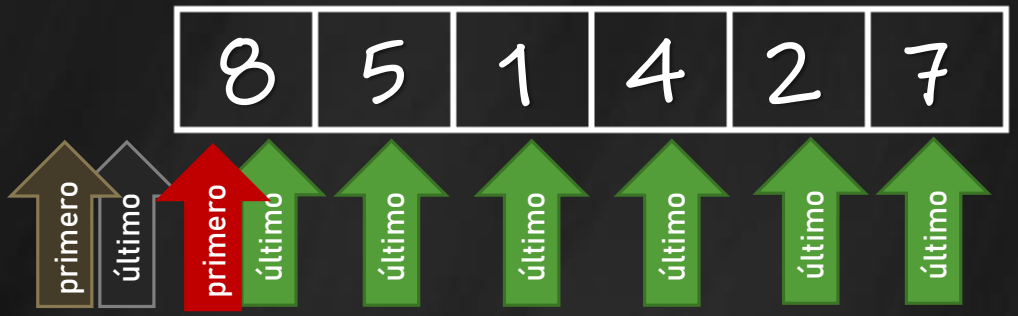
5

¿Qué ocurre si la cola tiene 1000?

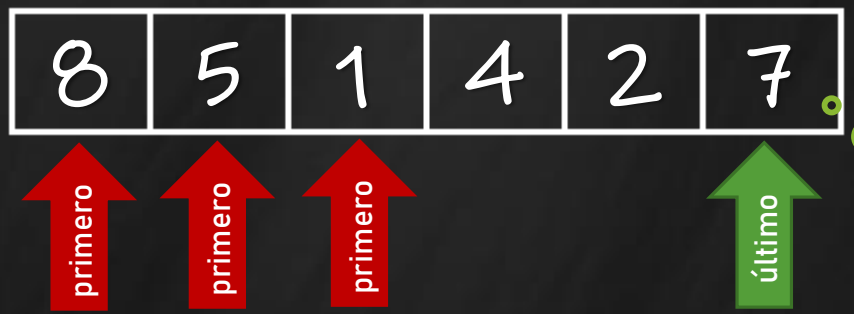
Alternativas de implementación con arreglos

Almacenamiento Lineal (2 índices)

- Agregar



- Quitar

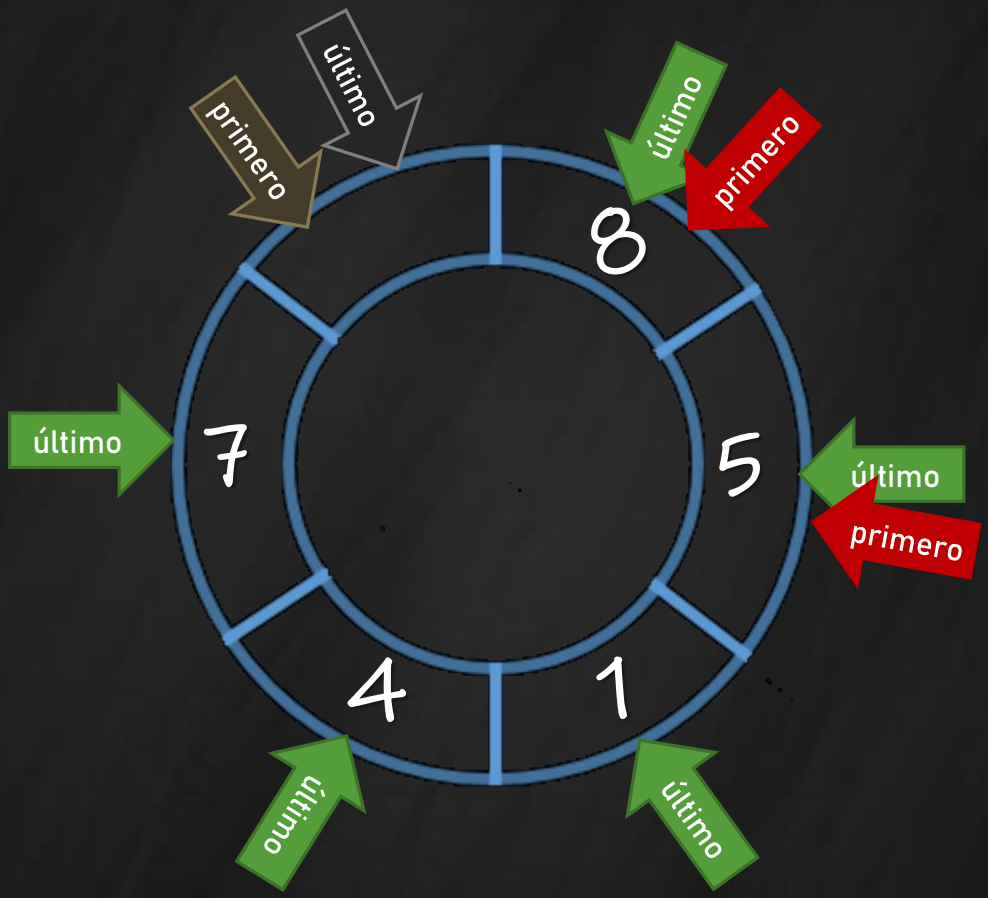


¿Es posible agregar más valores a la cola?

Alternativas de implementación con arreglos

Almacenamiento Circular (2 índices)

- Agregar
- Quitar



Alternativas de implementación con arreglos

Almacenamiento Circular (2 índices)

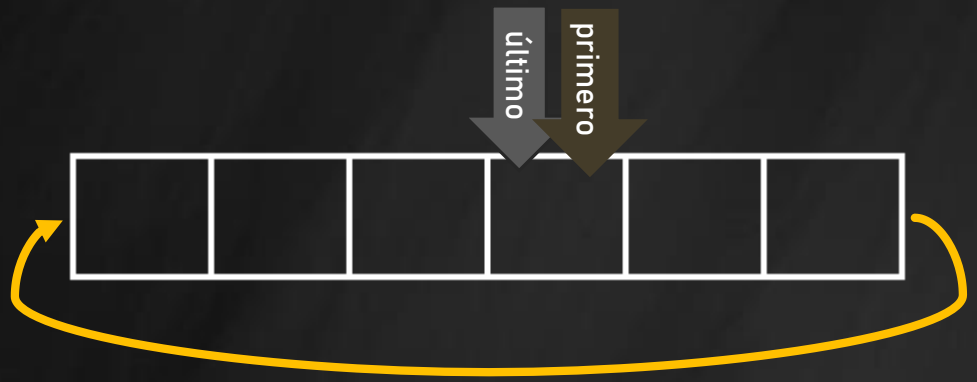
¿Cuándo está vacía la estructura?

¿Cuándo está llena la estructura?

Alternativas de implementación con arreglos

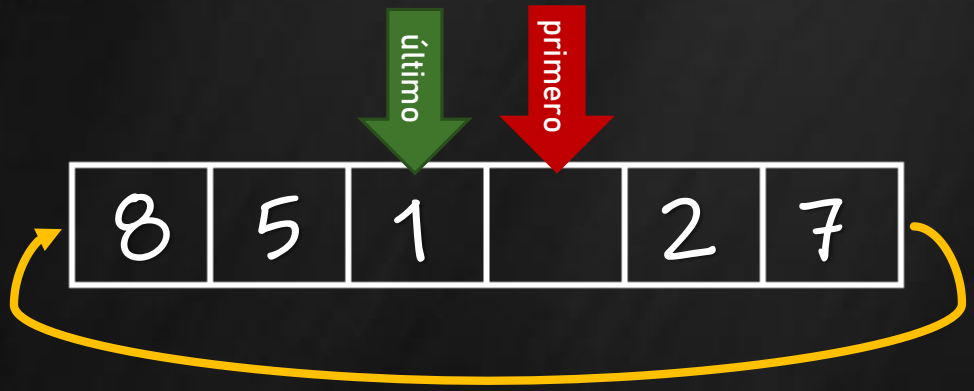
Almacenamiento Circular (2 índices)

¿Cola vacía?



Cuando primero y último apunten a la misma posición del arreglo asumiremos que la cola está vacía.

¿Cola llena?

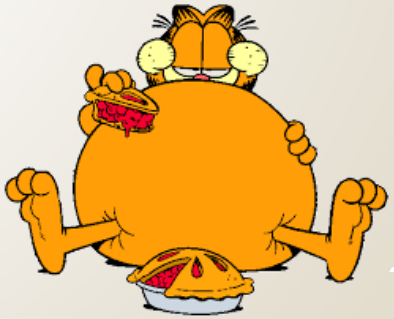


Cuando último apunte a la posición inmediata anterior a primero asumiremos que la cola está llena.

Alternativas de Implementación (2)

- El TDA cola, implementado utilizando un almacenamiento circular, puede presentar 2 variantes:
  - Implementación que prioriza la velocidad de procesamiento.

  - Implementación que prioriza el espacio de almacenamiento.



Implementación (1)

- TDA cola: Implementación que prioriza velocidad de procesamiento.

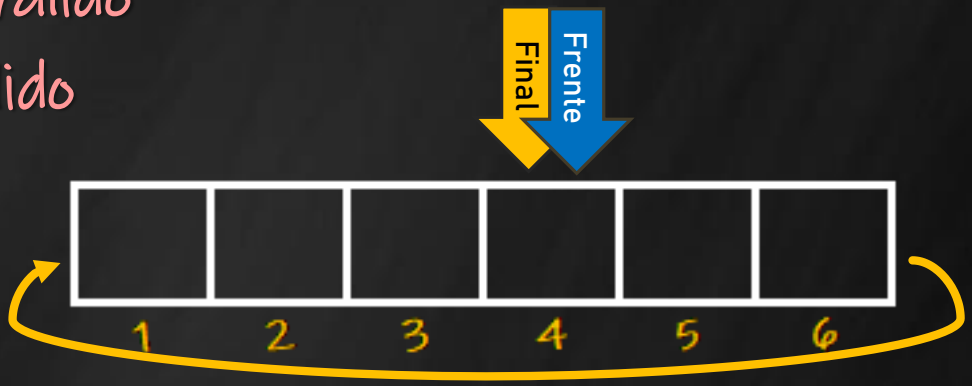
```
contenedor=ARREGLO [1..MAX] de ELEMENTOS
tcola=REGISTRO
        datos:contenedor
        frente,final:ENTERO
    FIN_REGISTRO
```

  - *frente*: indica el último elemento que se extrajo de la cola.
  - *final*: indica el último elemento que se agregó a la cola.

Operación Iniciar Cola

Permite crear una cola vacía.

```
PROCEDIMIENTO iniciar_cola (E/S cola: tcola)
INICIO
        cola.frente←índice_válido
        cola.final←índice_válido
FIN
```

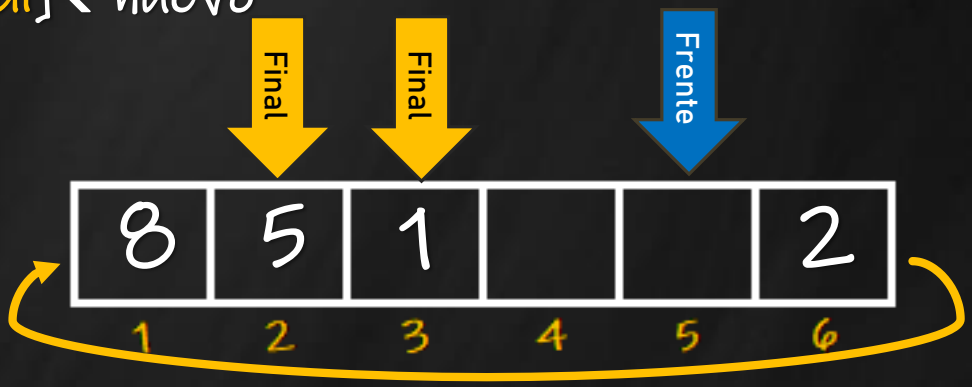



```
contenedor=ARREGLO [1..MAX] de ELEMENTOS
tcola=REGISTRO
        datos:contenedor
        frente:ENTERO
        final:ENTERO
    FIN_REGISTRO
```

Operación Agregar Cola

Permite agregar un elemento a la cola siempre que exista espacio

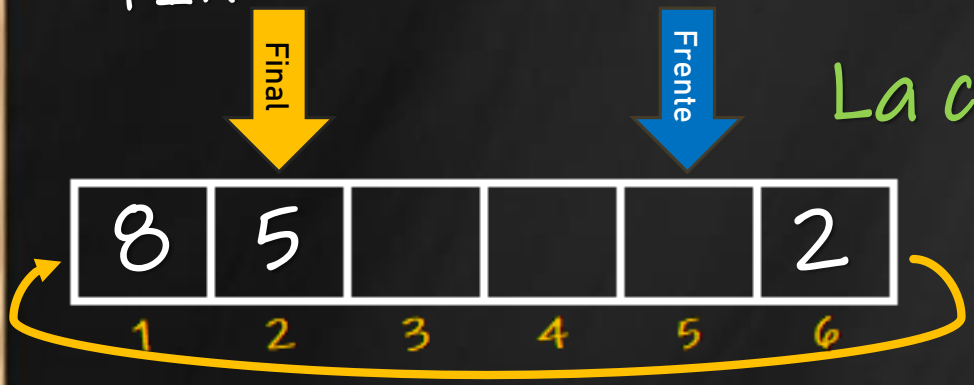
```
PROCEDIMIENTO agregar_cola (E/S cola: tcola, E nuevo: entero)
INICIO
    SI (cola_llena(cola)=VERDADERO) ENTONCES
        ESCRIBIR "COLA LLENA"
    SINO
        cola.final←siguiente(cola.final)
        cola.datos[cola.final]←nuevo
    FIN_SI
FIN
```



Operación Cola Llena

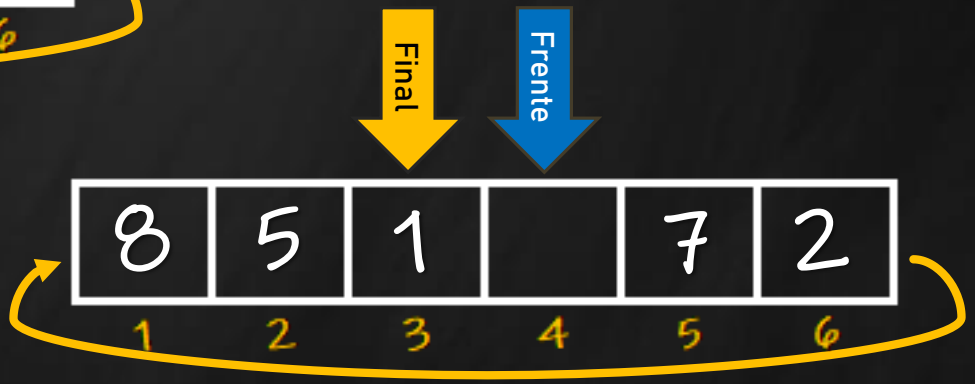
Permite determinar si una cola está llena o no.

```
FUNCIÓN cola_llena (E cola: tcola): LÓGICO
INICIO
        cola_llena←siguiente(cola.final)=cola.frente
FIN
```



La cola no está llena

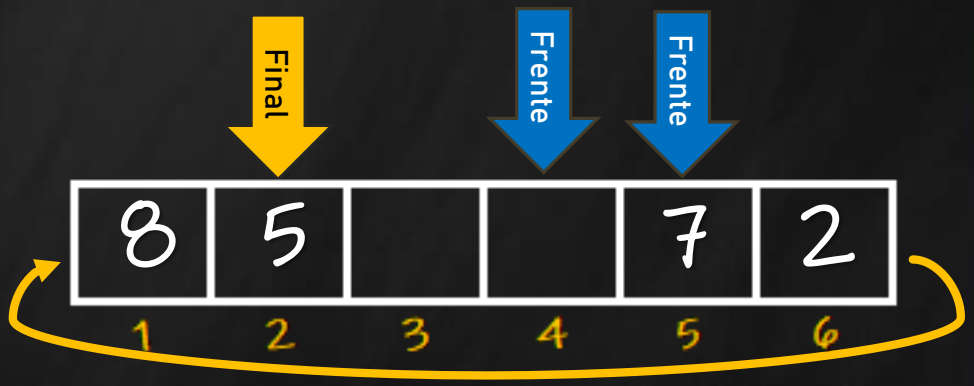
La cola está llena



Operación Quitar Cola

Permite extraer un elemento de la cola siempre que no esté vacía

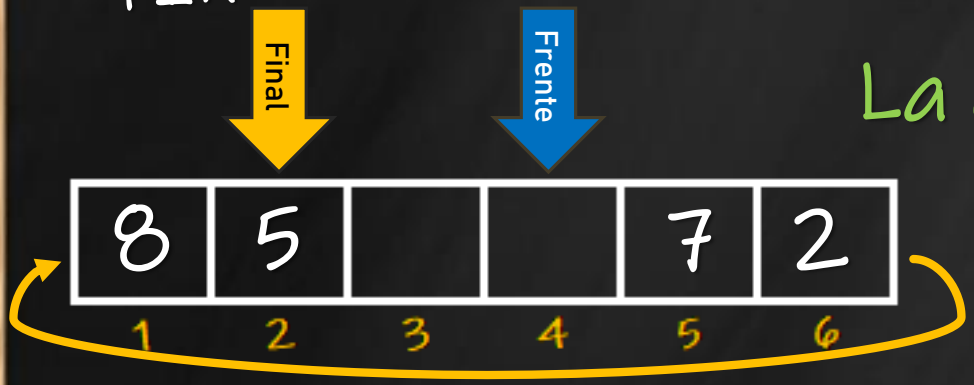
```
FUNCIÓN quitar_cola (E/S cola: tcola,): ENTERO
VARIABLES
      extraido: ENTERO
INICIO
      SI (cola_vacía(cola)=VERDADERO) ENTONCES
            extraido←valor_arbitrario
      SINO
            cola.frente←siguiente(cola.frente)
            extraido←cola.datos[cola.frente]
      FIN_SI
      quitar_cola←extraido
FIN
```



Operación Cola Vacía

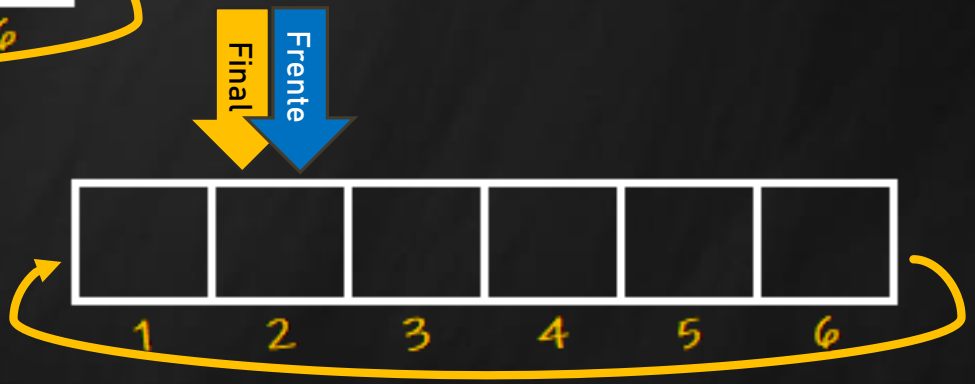
Permite determinar si una cola está vacía o no.

```
FUNCIÓN cola_vacía (E cola: tcola): LÓGICO
INICIO
        cola_vacía←cola.final=cola.frente
FIN
```



La cola no está vacía

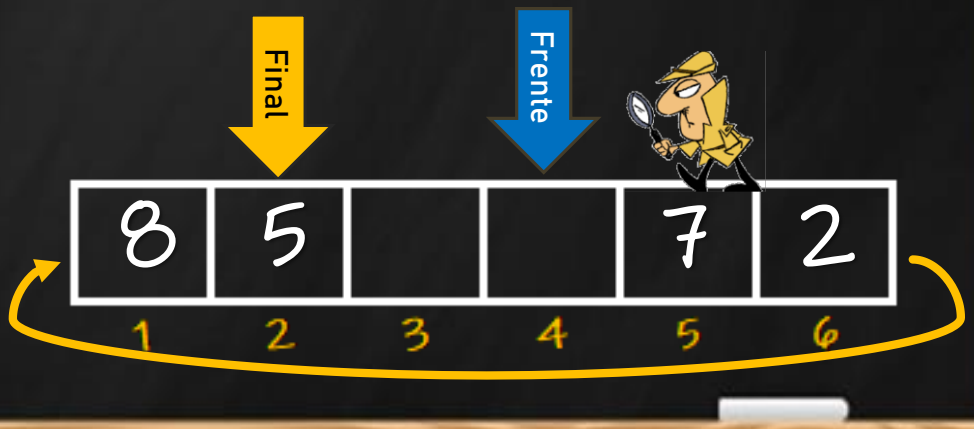
La cola está vacía



Operación Primero Cola

Permite consultar el primer elemento de la cola

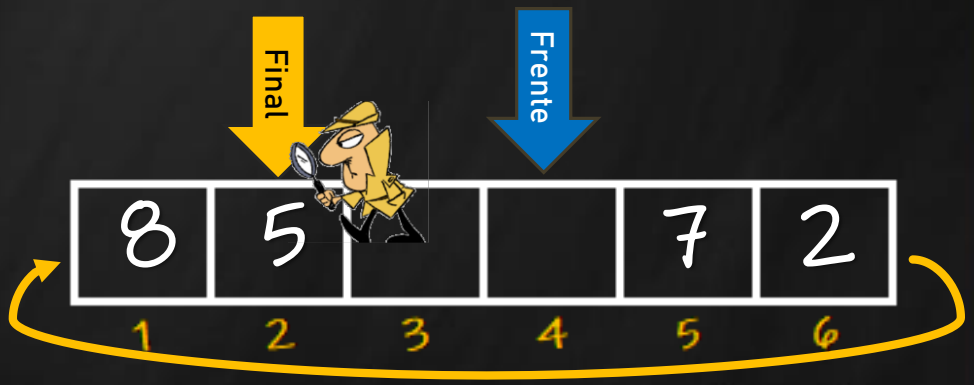
```
FUNCIÓN primero_cola (E cola: tcola,): ENTERO
VARIABLES
      primero: ENTERO
INICIO
      SI (cola_vacía(cola)=VERDADERO) ENTONCES
            primero←valor_arbitrario
      SINO
            primero←cola.datos[siguiente(cola.frente)]
      FIN_SI
      primero_cola←primero
FIN
```



Operación Último Cola

Permite consultar el último elemento de la cola

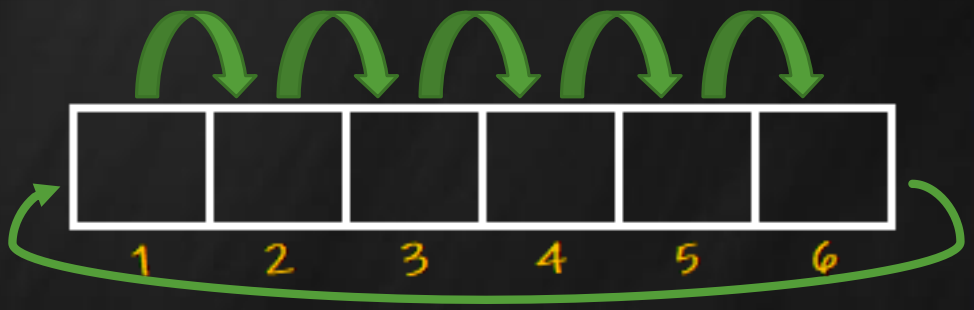
```
FUNCIÓN último_cola (E cola: tcola,): ENTERO
VARIABLES
      último: ENTERO
INICIO
      SI (cola_vacía(cola)=VERDADERO) ENTONCES
            último←valor_arbitrario
      SINO
            último←cola.datos[cola.final]
      FIN_SI
      último_cola←último
FIN
```



Operación Siguiente

Permite calcular el próximo valor para un índice.

```
FUNCIÓN siguiente (E indice: entero): ENTERO
INICIO
      SI (índice=MAX) ENTONCES
          índice←1
       SINO
          índice←índice + 1
       FIN_SI
       siguiente←índice
FIN
```



Implementación (10)

- TDA cola: Implementación que prioriza espacio de almacenamiento en memoria.

```
contenedor=ARREGLO [1..MAX] de ELEMENTOS
tcola=REGISTRO
        datos:contenedor
        frente,final:ENTERO
        cantidad:ENTERO
    FIN_REGISTRO
```

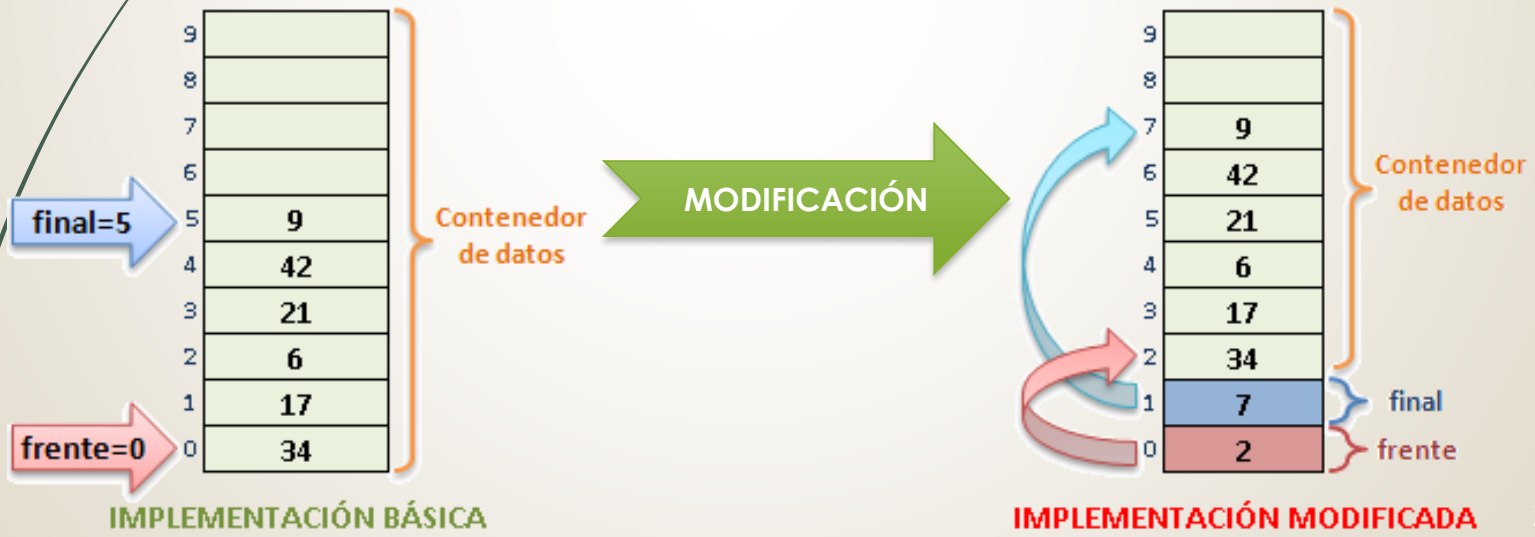
- ¿Cómo se modifican las operaciones básicas?

Implementación (11)

- Modificaciones en las operaciones del TDA cola:
  - *iniciar_cola*: se inicia el contador de elementos.
  - *agregar_cola*: se actualiza la cantidad de elementos de la cola (incremento).
  - *cola_llena*: se verifica si el contador alcanzó la máxima capacidad del contenedor.
  - *quitar_cola*: se actualiza la cantidad de elementos de la cola (decremento).
  - *cola_vacia*: se verifica si el contador alcanzó el valor de inicialización (cero).
  - *primero_cola* y *ultimo_cola*: no se modifican.

Implementación Modificada (1)

- Modifique la implementación básica del TDA cola (que prioriza velocidad de proceso) de forma que el contenedor de datos y los índices de la cola se almacenen en un único arreglo.
- ¿Cómo se modifican las operaciones de cola para esta implementación?



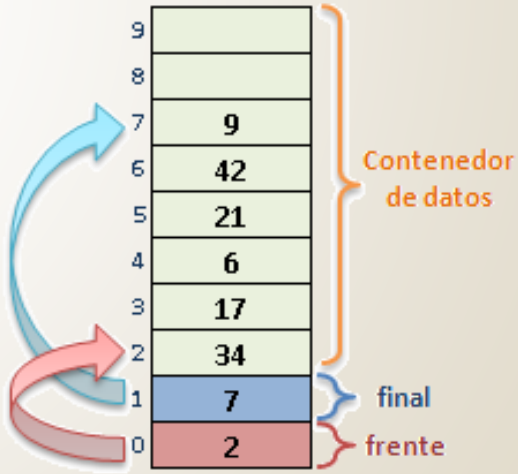
Implementación Modificada (2)

- TDA Cola modificado

```
const int MAX=10;
typedef int tcola[MAX];
```

- Operaciones modificadas: *iniciar_cola*

```
void iniciar_cola (tcola &q)
{
     q[0]←MAX-1; // frente
     q[1]←MAX-1; // final
}
```



IMPLEMENTACIÓN MODIFICADA

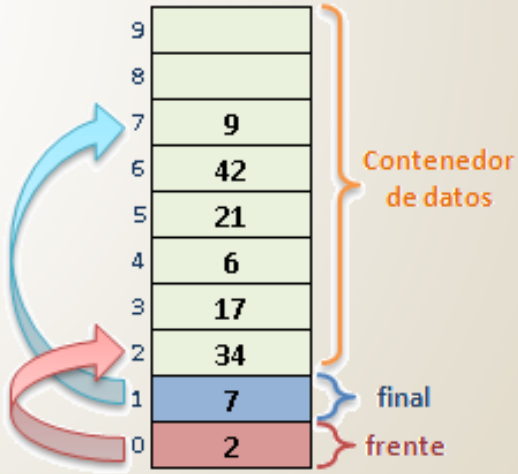
Implementación Modificada (3)

- TDA Cola modificado

```
const int MAX=10;
typedef int tcola[MAX];
```

- Operaciones modificadas: *cola_vacia*

```
bool cola_vacia (tcola q)
{
    return q[1]==q[0];
}
```

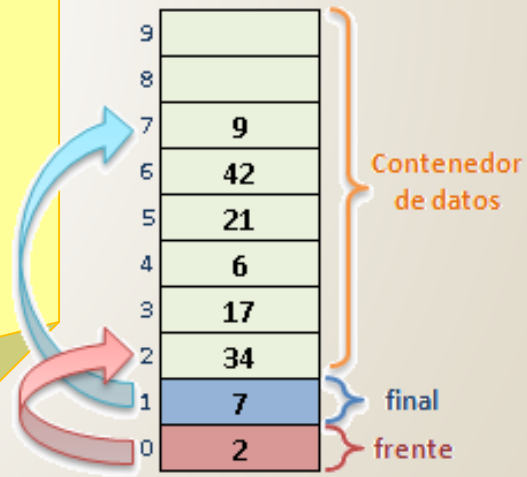


IMPLEMENTACIÓN MODIFICADA

Implementación Modificada (4)

- Operaciones modificadas: *agregar_cola*

```
void agregar_cola(tcola &q,int nuevo)
{
      if (cola_llena(q)==true)
        cout << "COLA LLENA" << endl;
      else
        { q[1]=siguiente(q[1]);
          q[q[1]]=nuevo;
        }
}
```

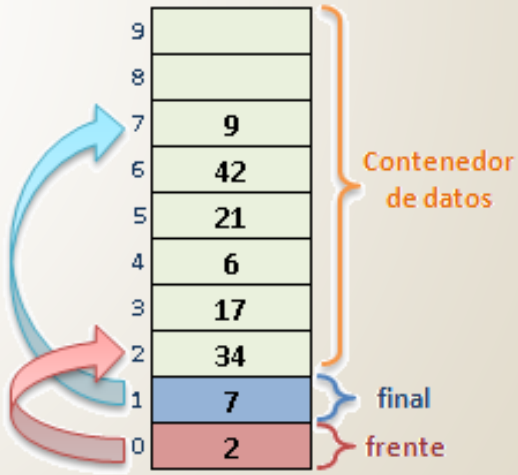


IMPLEMENTACIÓN MODIFICADA

Implementación Modificada (5)

- Operaciones modificadas: *siguiente*

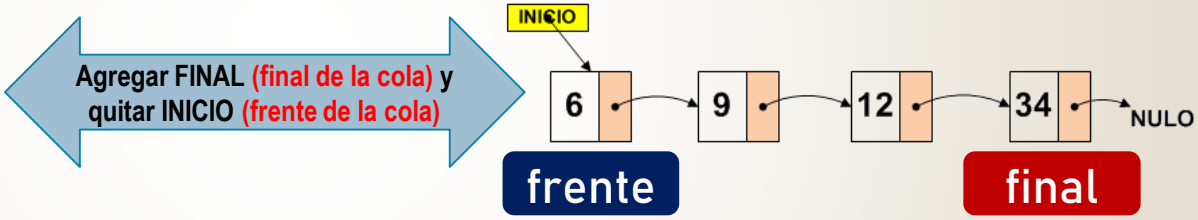
```
int siguiente(int indice)
{
   if (indice==MAX-1)
      indice=2;
   else
      indice++;
   return indice;
}
```



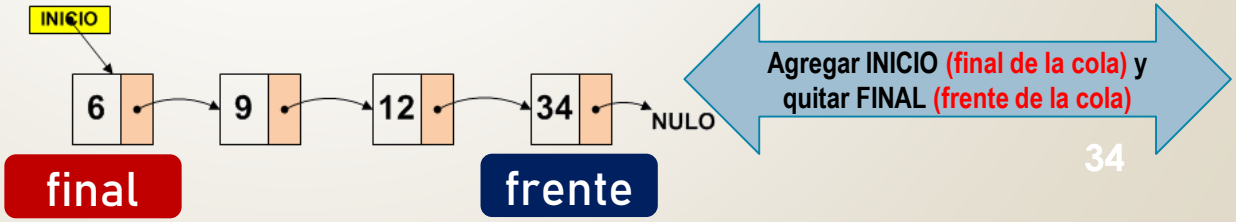
IMPLEMENTACIÓN MODIFICADA

Implementación: Listas (1)

- Implementación del TDA Cola mediante listas simples
  - Alternativa 1:
    - Utilizar las operaciones *agregar_final* y *quitar_inicio* para representar el comportamiento de la cola.

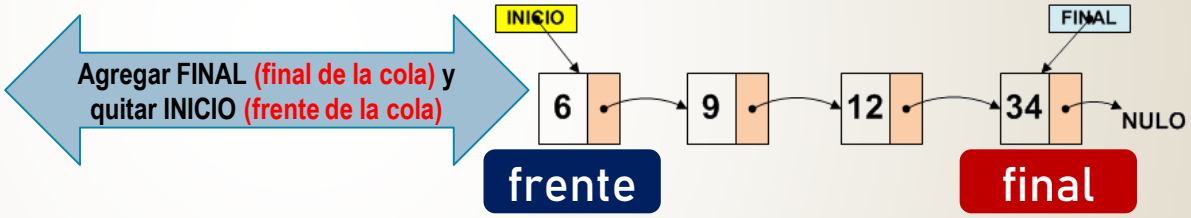


  - Alternativa 2:
    - Utilizar las operaciones *agregar_inicio* y *quitar_final* para representar el comportamiento de la cola.

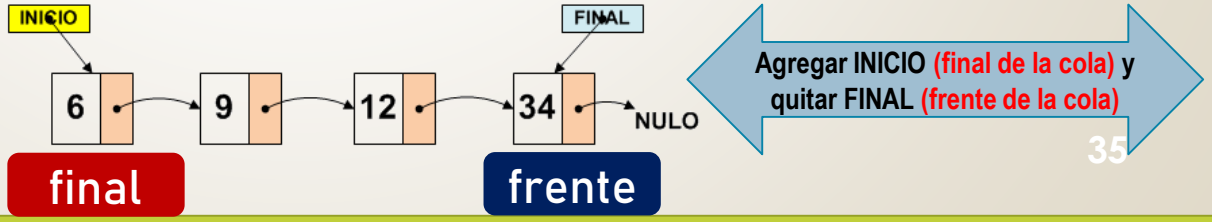


Implementación: Listas (2)

- Implementación del TDA Cola mediante listas simples
  - Alternativa 3:
    - Utilizar las operaciones *agregar_final* y *quitar_inicio* para representar el comportamiento de la cola.

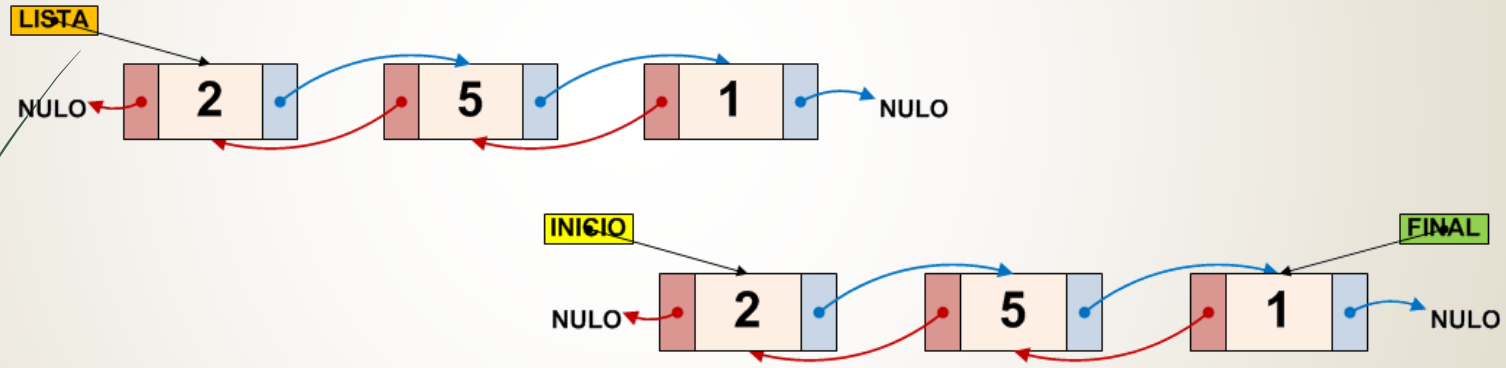


  - Alternativa 4:
    - Utilizar las operaciones *agregar_inicio* y *quitar_final* para representar el comportamiento de la cola.



Implementación: Listas (3)

- Implementación del TDA Cola mediante listas dobles
  - ¿Cuáles son las alternativas de implementación al utilizar listas dobles?



  - ¿Cuáles son las operaciones de listas dobles que pueden utilizarse para implementar las operaciones de cola para cada alternativa?

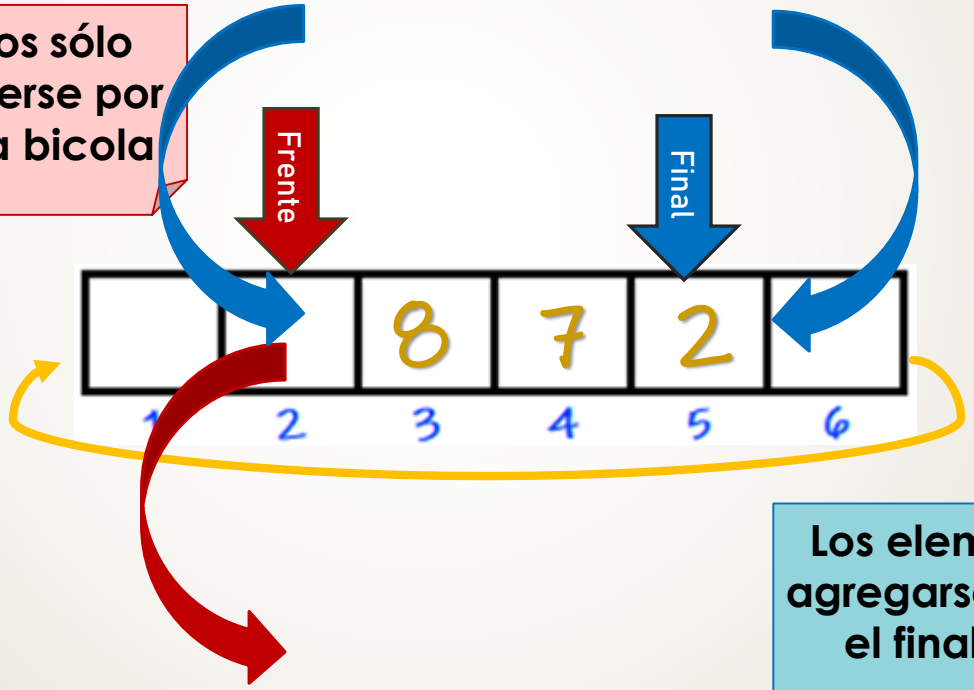
Bicolas (1)

- Una variante del TDA cola es la *bicola* o *cola doble* que permite la inserción y eliminación de elementos por ambos extremos del contenedor de datos.
- De acuerdo al extremo (frente, final) que admita la inserción/eliminación de elementos la *bicola* puede ser:
  - Con entrada restringida (se permite eliminar por *frente* y *final*; y agregar sólo por *final*)
  - Con salida restringida (se permite agregar por *frente* y *final;* y eliminar sólo por *frente*;)

Bicolas (2)

- Bicola con salida restringida

Los elementos sólo pueden extraerse por el frente de la bicola



Los elementos pueden agregarse por el frente o el final de la bicola

Bicolas (3)

- Bicola con salida restringida

```
const int MAX=10;
typedef int contenedor[MAX];
typedef struct tcola {
                    contenedor datos;
                    int frente;
                    int final;
                    };
```

Para implementar la bicola se utilizan las mismas estructuras de datos que para la cola estándar

Bicolas (4)

- Bicola con salida restringida

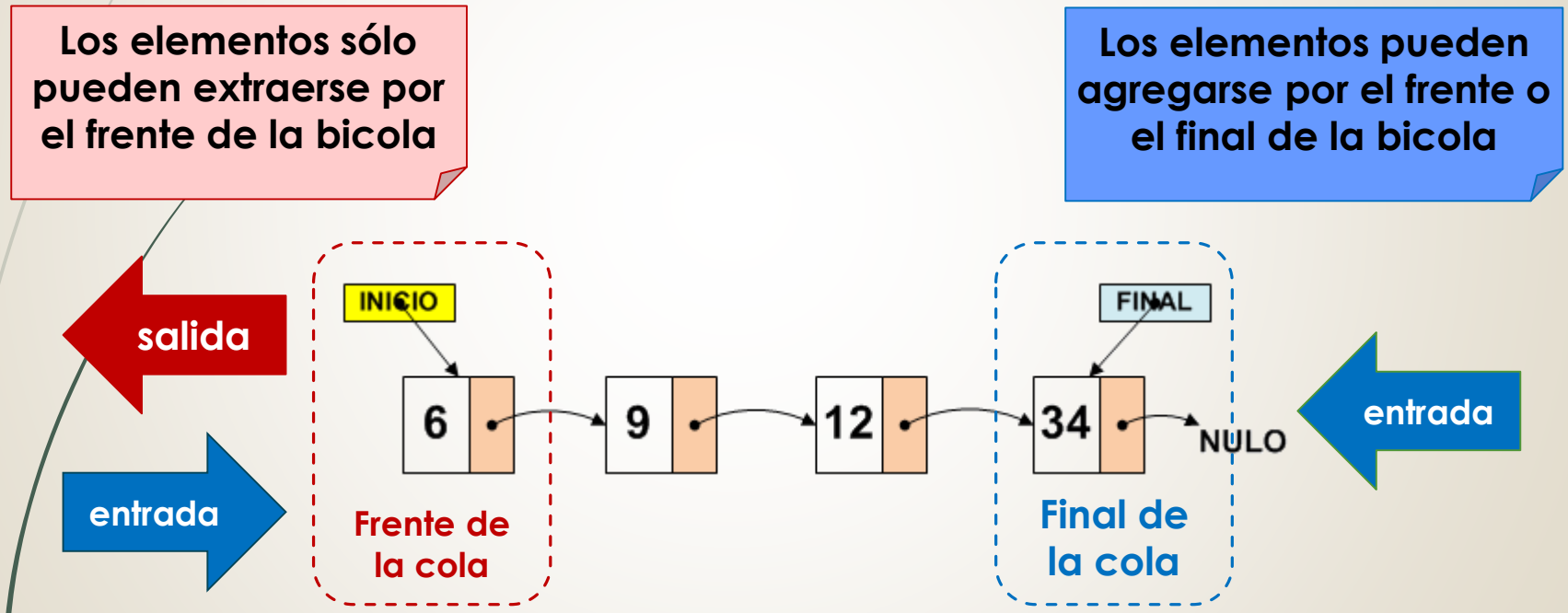
```
void agregar_bicola(tcola &q,int nuevo,bool ultimo)
{ if (cola_llena(q)==true)
        cout << "No hay espacio" << endl;
   else
      if (ultimo==true)
        {q.final=siguiente(q.final);
         q.datos[q.final]=nuevo;}
      else
        {q.datos[q.frente]=nuevo;
         q.frente=anterior(q.frente);}
}
```

Agrega elementos al final de la fila

Agrega elementos al frente de la fila

Bicolas con Lista Simples (1)

- Bicola con salida restringida



Bicolas con Lista Simples (2)

- Bicola con salida restringida

```
typedef struct tnodo *pnodo;
typedef struct tnodo {
                    int dato;
                    pnodo sig;
                    };
typedef struct tbicola {
                    pnodo inicio; // frente
                    pnodo final; // final
                    };
```

Para implementar la bicola pueden utilizarse listas simples o dobles, con uno o 2 punteros a la estructura.

Bicolas con Lista Simples (3)

- Bicola con salida restringida

```
void agregar_bicola(tbicola &q,pnodo nuevo,bool ultimo)
{ if (q.inicio==NULL)
      { q.inicio=nuevo;
        q.final=nuevo; }
   else
     if (ultimo==true)
       {q.final->sig=nuevo;
        q.final=nuevo;}
     else
       {nuevo->sig=q.inicio;
        q.inicio=nuevo;}
}
```

Agrega elementos al final de la cola

Agrega elementos por el frente de la cola

Aplicaciones

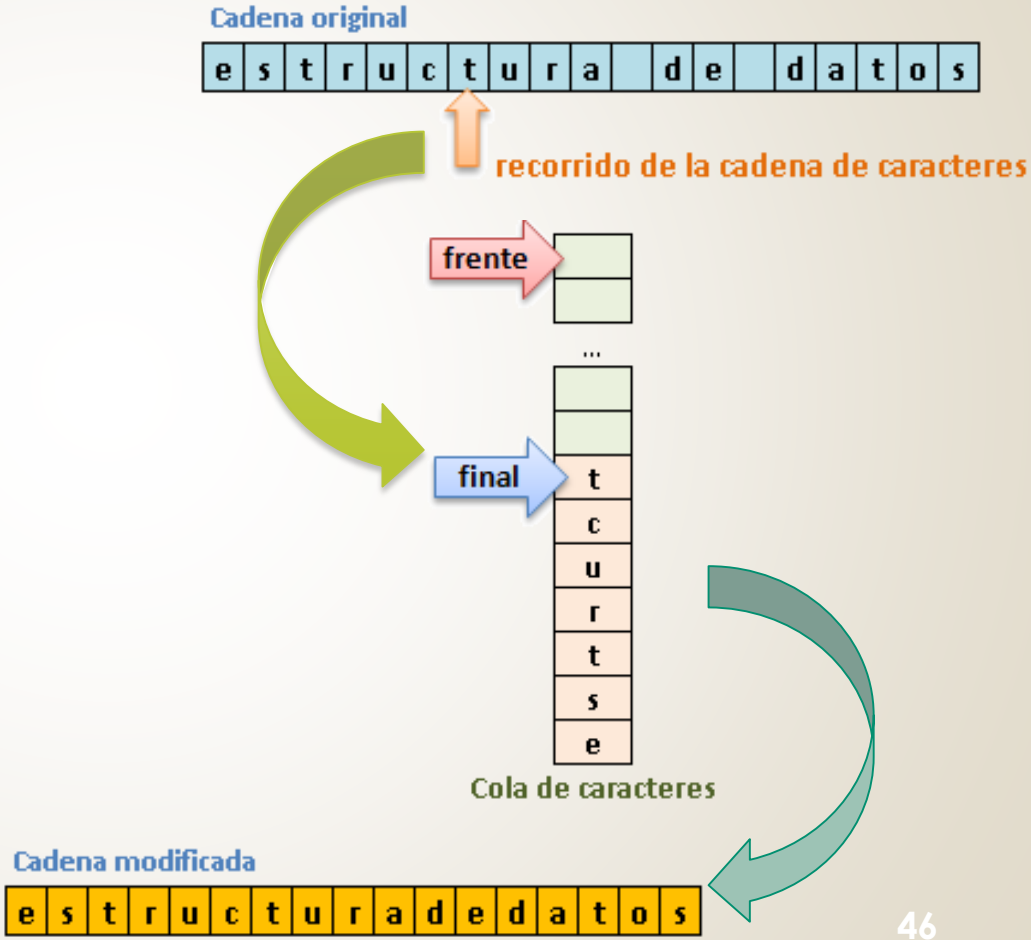
- El concepto de cola puede aplicarse para resolver:
  - simulación (teoría de colas)
  - algoritmos de reemplazo
  - colas de impresión,
  - acceso (escritura) almacenamiento secundario
  - sistemas de tiempo compartido
  - uso de la unidad central de proceso (UCP)

Ejemplo de Aplicación (1)

- Diseñe un algoritmo que elimine los espacios en blanco de una cadena de caracteres. Utilice el TDA cola en la propuesta de solución.
- Propuesta de Solución
  - El algoritmo recorrerá, carácter a carácter, la cadena de entrada guardando estos caracteres en una cola, excepto los espacios en blanco.
  - Luego, el contenido de la cola sobre-escribirá la cadena original, obteniéndose una cadena sin espacios.

Ejemplo de Aplicación (2)

- Conforme se recorre la cadena de entrada se guardan los caracteres leídos en una cola, salvo aquellos que sean espacios en blanco.
- Finalizado el recorrido, se inicia el vaciado de la cola, guardándose cada valor extraído en la cadena original. Se obtiene así una cadena sin espacios en blanco.



Ejemplo de Aplicación (3)

- Algoritmo para eliminar espacios usando TDA cola.

```
void eliminar_blancos(tcad &frase)
{ tcola cola;
  int i;
  iniciar_cola(cola);
  for(i=0;i<strlen(frase);i++)
     if (frase[i]!=' ')
        agregar_cola(cola,frase[i]);
  for(i=0;cola_vacia(cola)==false;i++)
     frase[i]=quitar_cola(cola);
  frase[i]='\0'; // final de cadena
}
```

Inicialización de la cola

Inserción de datos

Extracción de datos

Bibliografía

- Joyanes Aguilar *et al*. Estructuras de Datos en C++. Mc Graw Hill. 2007.
- De Giusti, Armando *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta. 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Hernández, Roberto *et al*. Estructuras de Datos y Algoritmos. Prentice Hall. 2001.