

Algoritmos y

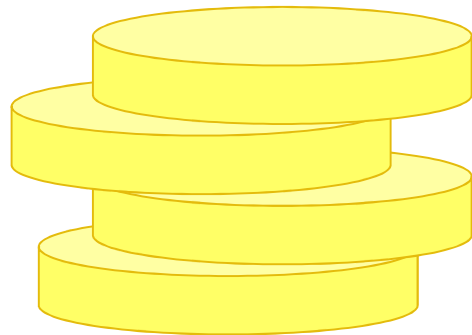
Estructuras

De

Datos

PILAS Y COLAS
IMPLEMENTADAS
EN ARREGLOS

PILAS



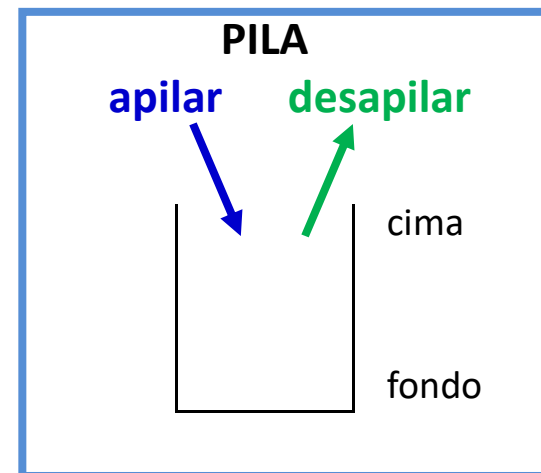
Pilas

- Estructura de datos en la cual el **acceso está limitado** al elemento más recientemente insertado (el del extremo superior de la pila).
- El último elemento añadido a la pila es colocado en la **cima**, donde es accedido directamente, mientras que los elementos que llevan más tiempo son más difíciles de acceder.
- **Operaciones naturales:** insertar - eliminar - buscar
apilar - **desapilar** - **cima**

- Estructura **LIFO**:

“Last In, First Out”

Último en entrar, primero en salir



Pilas

Es una estructura de datos de **entradas ordenadas**

... pues hay un elemento al que se puede acceder primero (el que está encima de la pila), otro elemento al que se puede acceder en segundo lugar (justo el elemento que está debajo de la cima), un tercero, etc.

– Los elementos sólo se pueden agregar y eliminar en el **tope**.

- **Apilar:**

– Agrega un elemento nuevo en el tope de la pila.

- **Desapilar:**

– Elimina un elemento del tope de la pila.

– Los elementos de la pila deben ser eliminados en el orden inverso al que se situaron en la misma.

Pilas – Implementación en estructuras estáticas

Las pilas se pueden implementar guardando los elementos en un **array** de longitud fija. Y utilizando **una variable** para mantener la posición del **último elemento** colocado en la pila

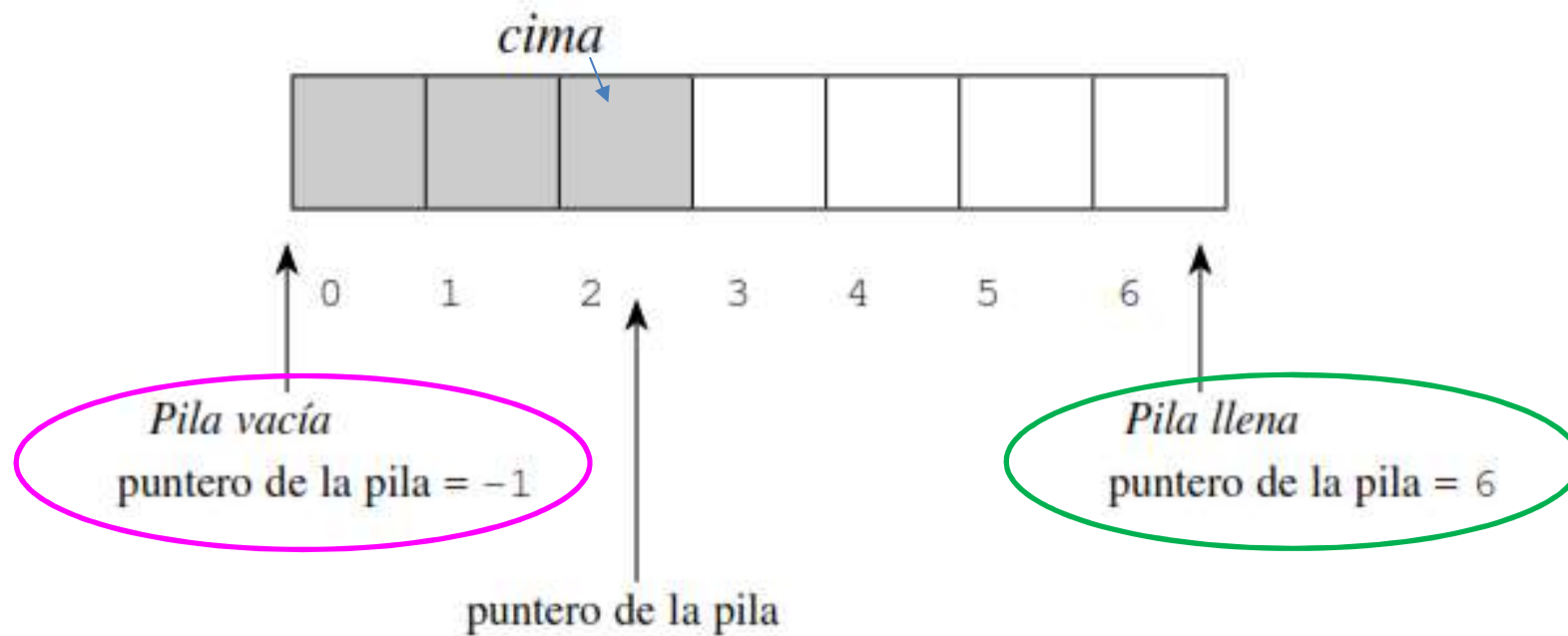
Una pila puede estar **vacía**, o **llena** (en la representación con un array, si se ha llegado al último elemento).

Si un programa intenta sacar un elemento de una pila vacía, se producirá un error por **desbordamiento negativo (underflow)**.

Si un programa intenta poner un elemento en una pila llena se producirá un error por **desbordamiento (overflow) o rebosamiento**.

Para evitarlos se diseñan funciones, que comprueban si la pila está llena o vacía.

Pilas

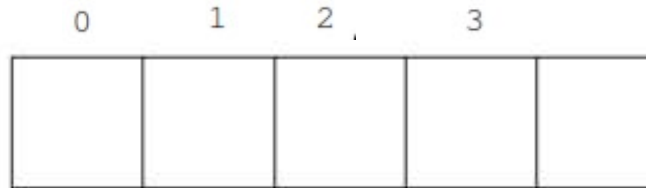


Puntero de la pila = índice del arreglo

Representación gráfica de una pila de 7 elementos

Pilas

Pila vacía
ptrcima = -1



Insertar 50
ptrcima = 0



Insertar 25
ptrcima = 1



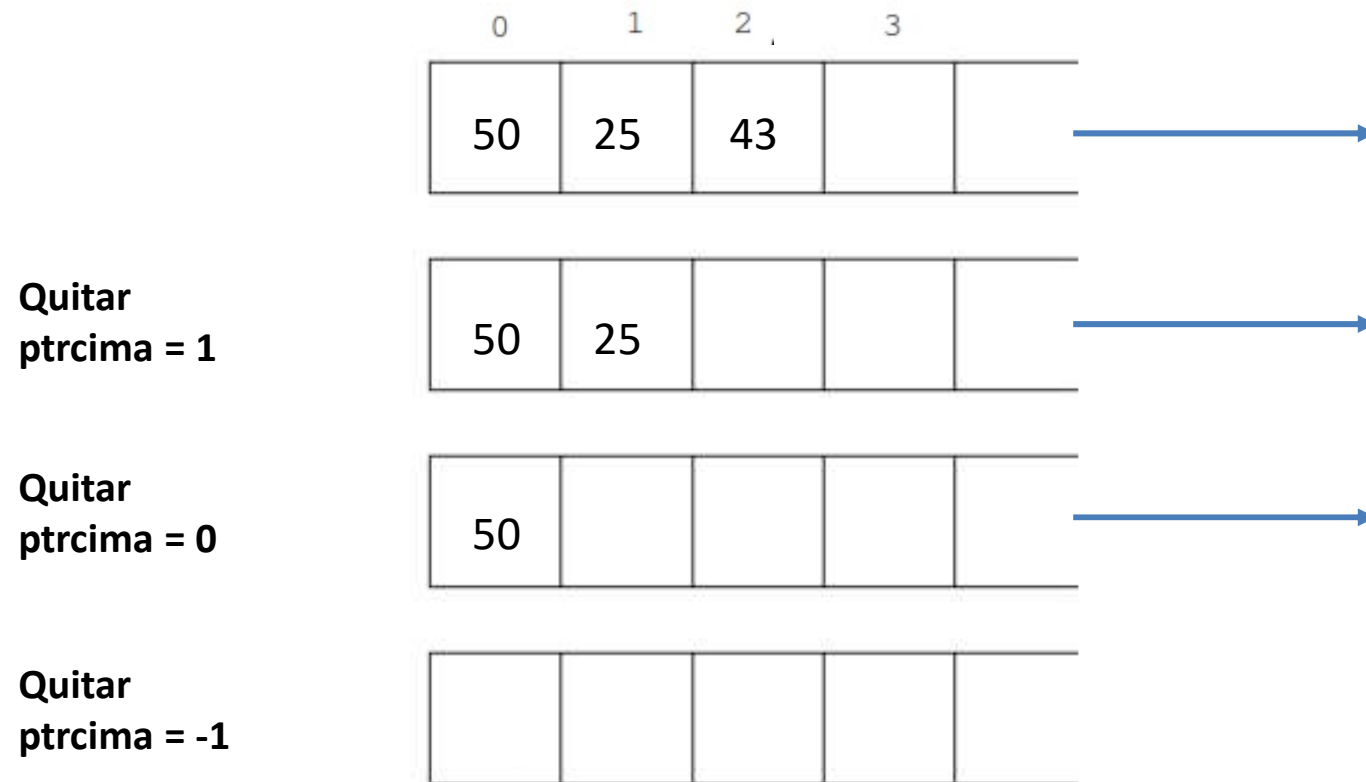
Insertar 43
ptrcima = 2



Inserción de elementos - Apilar

1. Verificar si la pila no está llena.
2. Incrementar en 1 el apuntador (cima) de la pila.
3. Almacenar el elemento en la posición del apuntador de la pila.

Pilas



Eliminación de elementos - Desapilar

1. Si la pila no está vacía.
2. Leer el elemento de la posición del apuntador (cima) de la pila.
3. Decrementar en 1 el apuntador de la pila.

Ejemplo

```
1  /* Pila de enteros implementada sobre un vector */
2  /* TDA básico y otros algoritmos relacionados con pilas */
3  /* Interfase */
4
5  const int TF=100;
6
7  // Definicion de la estructura para la pila
8  struct Pila {
9      int info[TF]; // vector para almacenar los valores
10     int tope;     // subindice del último elemento insertado
11 };
12
13 // Prototipos de Funciones
14 void inicializar(Pila & P);
15 bool agregar( Pila & P, int valor );
16 bool sacar ( Pila & P, int & valor );
17 bool vacia ( const Pila P );
18 void mostrarPila ( const Pila P );
19 void invertirPila ( Pila & P );
20
```

```

#include "pila.h"
int main(){
    Pila P;
    int aux;

    inicializar(P);
    mostrarPila(P);

    if (agregar(P, 3)) cout << "Se inserto 3 " << endl << endl;
    if (agregar(P, 8)) cout << "Se inserto 8 " << endl << endl;
    if (agregar(P, 5)) cout << "Se inserto 5 " << endl << endl;

    mostrarPila(P);
    cout << endl << "Invertir Pila " << endl;
    invertirPila(P);
    mostrarPila(P);

    if (sacar(P, aux)) cout << "Se retiro elemento de la pila = " << aux << endl << endl;
    else cout << "Underflow = Pila vacia " << endl << endl;
    mostrarPila(P);

    if (sacar(P, aux)) cout << "Se retiro elemento de la pila = " << aux << endl << endl;
    else cout << "Underflow = Pila vacia " << endl << endl;
    mostrarPila(P);

    if (sacar(P, aux)) cout << "Se retiro elemento de la pila = " << aux << endl << endl;
    else cout << "Underflow = Pila vacia " << endl << endl;
    mostrarPila(P);

    if (sacar(P, aux)) cout << "Se retiro elemento de la pila = " << aux << endl << endl;
    else cout << "Underflow = Pila vacia " << endl << endl;
    mostrarPila(P);
    return 0;
}

```

```
using namespace std;
```

```
#include "pila.h"
```

```
void inicializar(Pila & P){  
    P.tope=-1;  
}
```

```
bool agregar( Pila & P, int valor ){  
    if (P.tope == TF-1)  
        return false;    //no se puede insertar en una pila llena  
    else  
    { P.tope++;  
      P.info[P.tope]= valor;  
      return true;}  
}
```

```
bool sacar ( Pila & P, int & valor ){  
    if (vacía( P ))  
        return false;    //no se puede sacar de una pila vacía  
    else {  
        valor= P.info[P.tope];  
        P.tope--;  
        return true;  
    }  
}
```

```
bool vacía ( const Pila P ) {  
    return (P.tope==-1);  
}
```

```
- void mostrarPila ( const Pila P ){  
    cout << "Pila = [ ";  
    for (int i=0; i<=P.tope; i++)  
        cout << P.info[i] << " ";  
    cout << "]" , Tope = " << P.tope << endl;  
}
```

```
- void invertirPila ( Pila & P ){  
    Pila Aux;  
    int valor;  
  
    inicializar(Aux);  
  
    while ( not vacia(P) ){  
        sacar(P, valor);  
        agregar(Aux, valor);  
    }  
  
    P=Aux;  
}
```

```
Pila = [ ] , Tope = -1
Se inserto 3

Se inserto 8

Se inserto 5

Pila = [ 3 8 5 ] , Tope = 2

Invertir Pila
Pila = [ 5 8 3 ] , Tope = 2
Se retiro elemento de la pila = 3

Pila = [ 5 8 ] , Tope = 1
Se retiro elemento de la pila = 8

Pila = [ 5 ] , Tope = 0
Se retiro elemento de la pila = 5

Pila = [ ] , Tope = -1
Underflow = Pila vacia

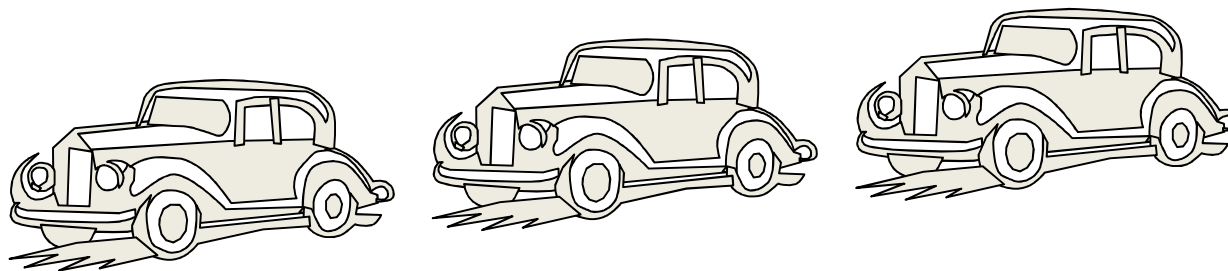
Pila = [ ] , Tope = -1
```

Aplicación de Pilas

Las aplicaciones de las pilas en la programación son numerosas. Se utilizan en:

- compiladores
- sistemas operativos
- programas de aplicaciones

COLAS



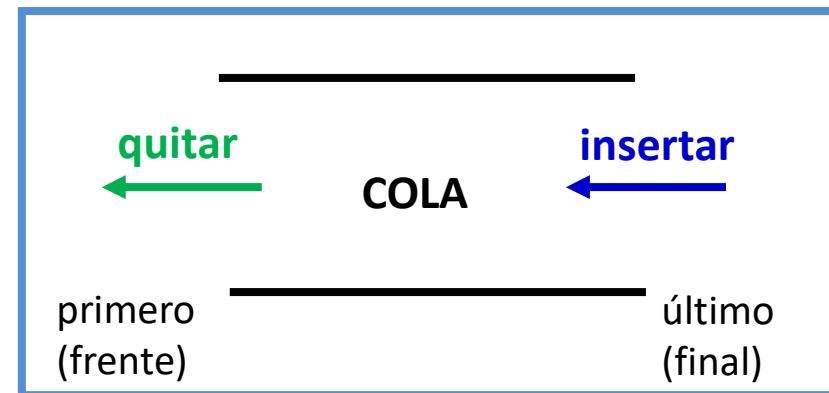
Colas

- Estructura de datos en la cual se puede **eliminar el elemento más antiguo (primero en entrar)** y solo se puede **insertar después del elemento más reciente (último en entrar)**.
- Se puede identificar un **frente** y un **final**.
- Operaciones naturales: **insertar** (al final) - encolar
eliminar (al frente) - desencolar

- Estructura **FIFO**:

“First In, First Out”

Primero en entrar, primero en salir



- Ejemplo: trabajos que se mandan a impresión.

Colas

Similar a una cola de pago en un supermercado

Es una estructura de datos de **entradas ordenadas**

... pues solo se pueden añadir elementos por un extremo (final de la cola) y eliminar por el otro extremo (frente).

- **Operaciones de insertar y eliminar:**
 - **Encolar** (insertar un elemento nuevo al final de la cola).
 - **Desencolar** (eliminar un elemento del frente de la cola).

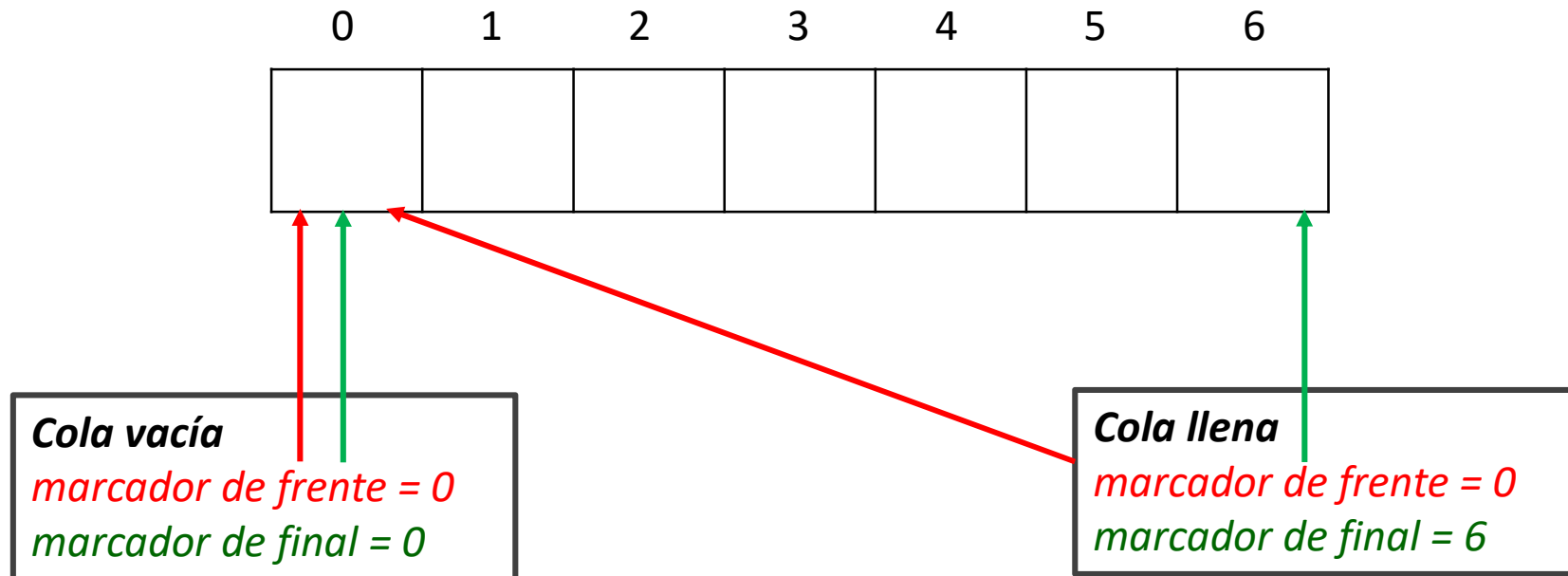
Colas – Implementación en estructuras estáticas

Las colas se pueden implementar guardando los elementos en un **array** de longitud fija. Y utilizando **dos marcadores** para mantener las posiciones **frente** y **final** de la cola.

- Un marcador almacena la posición de la *cabeza* de la cola.
- El otro, almacena el primer espacio vacío que sigue al elemento *final* de la cola.

Al igual que la pila, una cola puede estar **vacía** (sin elementos), o **llena**.

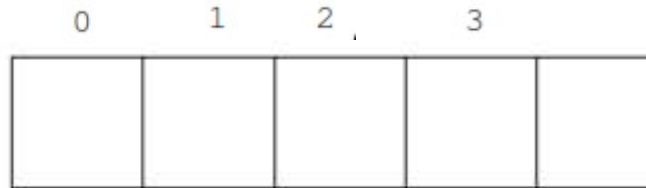
Colas



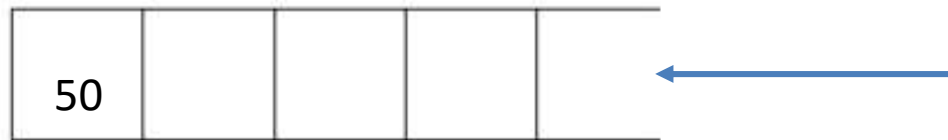
Representación gráfica de una cola de 5 elementos

Colas

Cola vacía
frente = 0
final = 0



Insertar 50
frente = 0
final = 1



Insertar 25
frente = 0
final = 2

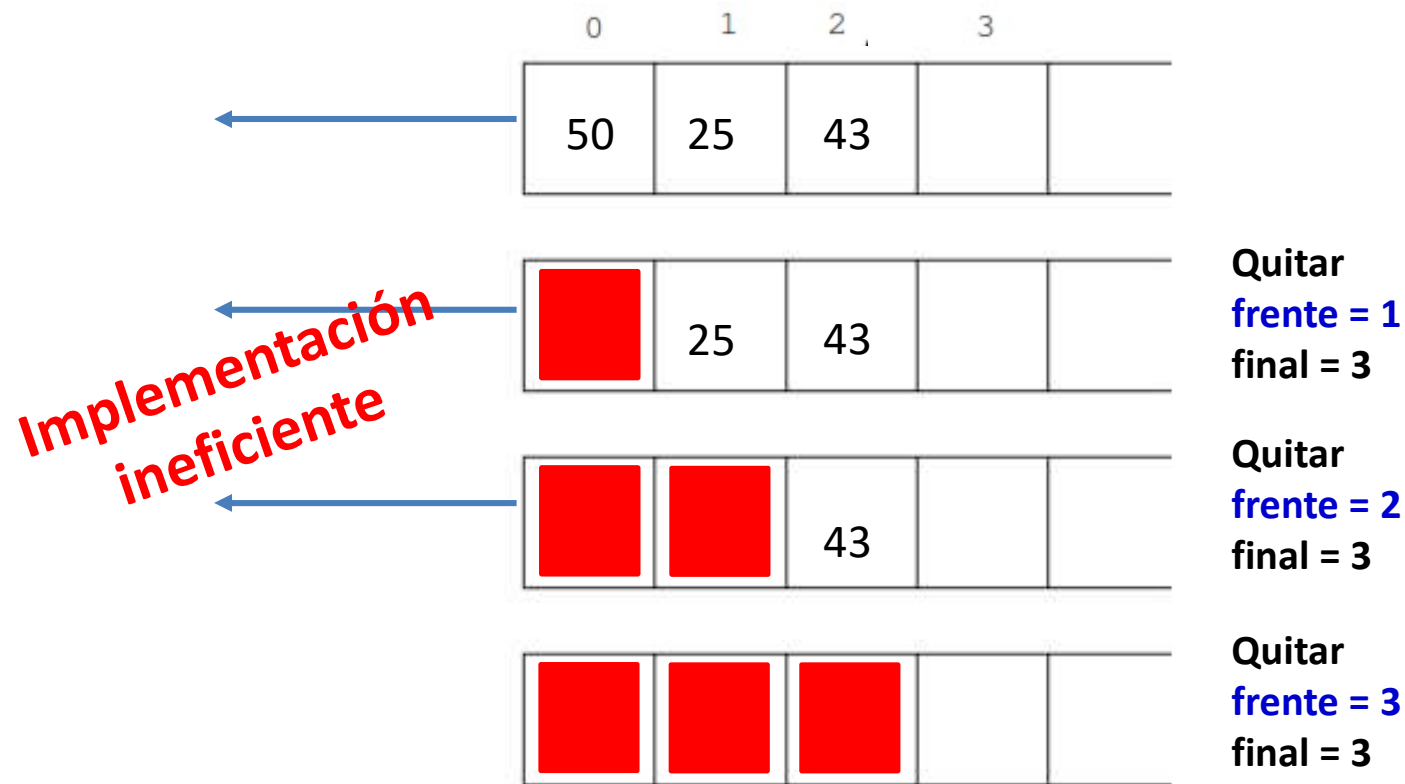


Insertar 43
frente = 0
final = 3



Insertión de elementos - Encolar

Colas



Van quedando huecos por la izquierda del array.

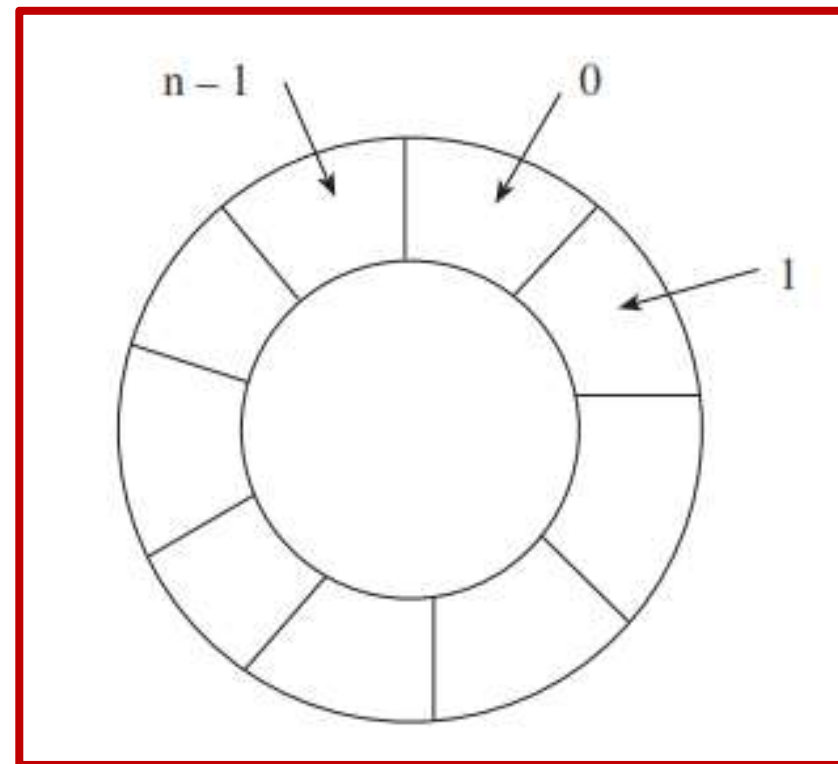
El puntero *final* alcanza la posición máxima del arreglo y no se pueden añadir más elementos.

Sin embargo, hay posiciones libres a la izquierda de *frente*.

Colas Circulares en Array

- Es la forma **más eficiente** de almacenar una cola en un array.
- En las **colas circulares** se *une el extremo final con el extremo de la cabeza*.

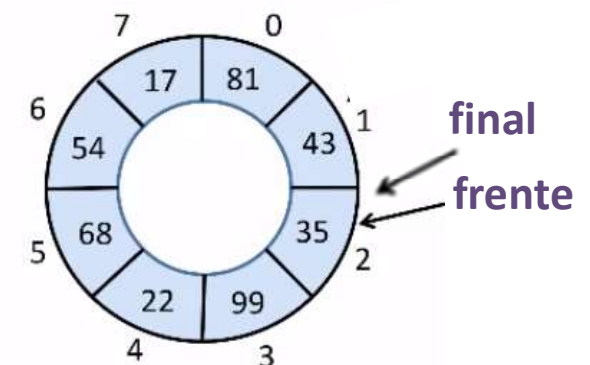
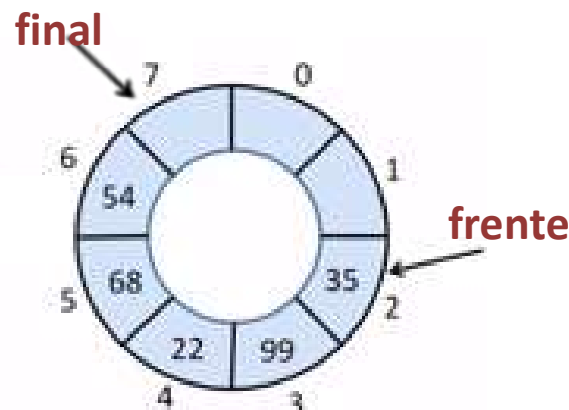
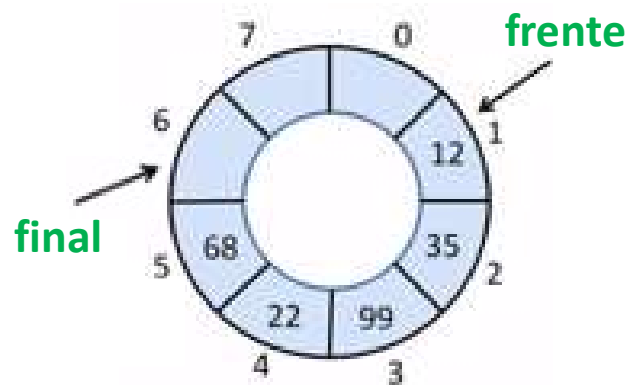
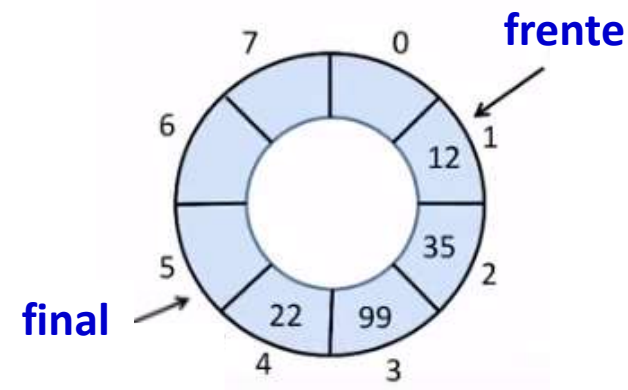
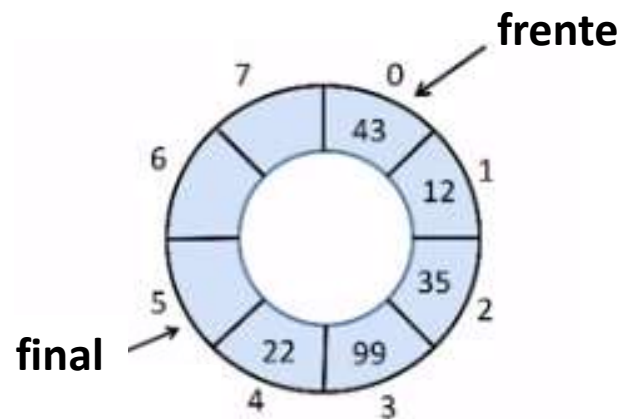
De esta manera, la totalidad de las posiciones del array se utilizan para poder almacenar elementos de la cola **sin necesidad de desplazar elementos**.



Colas Circulares en Array

Las colas circulares se implementan en un **array lineal**.

Se utilizan **dos marcadores** para mantener las posiciones **frente** y **final** de la cola.



Colas Circulares en Array

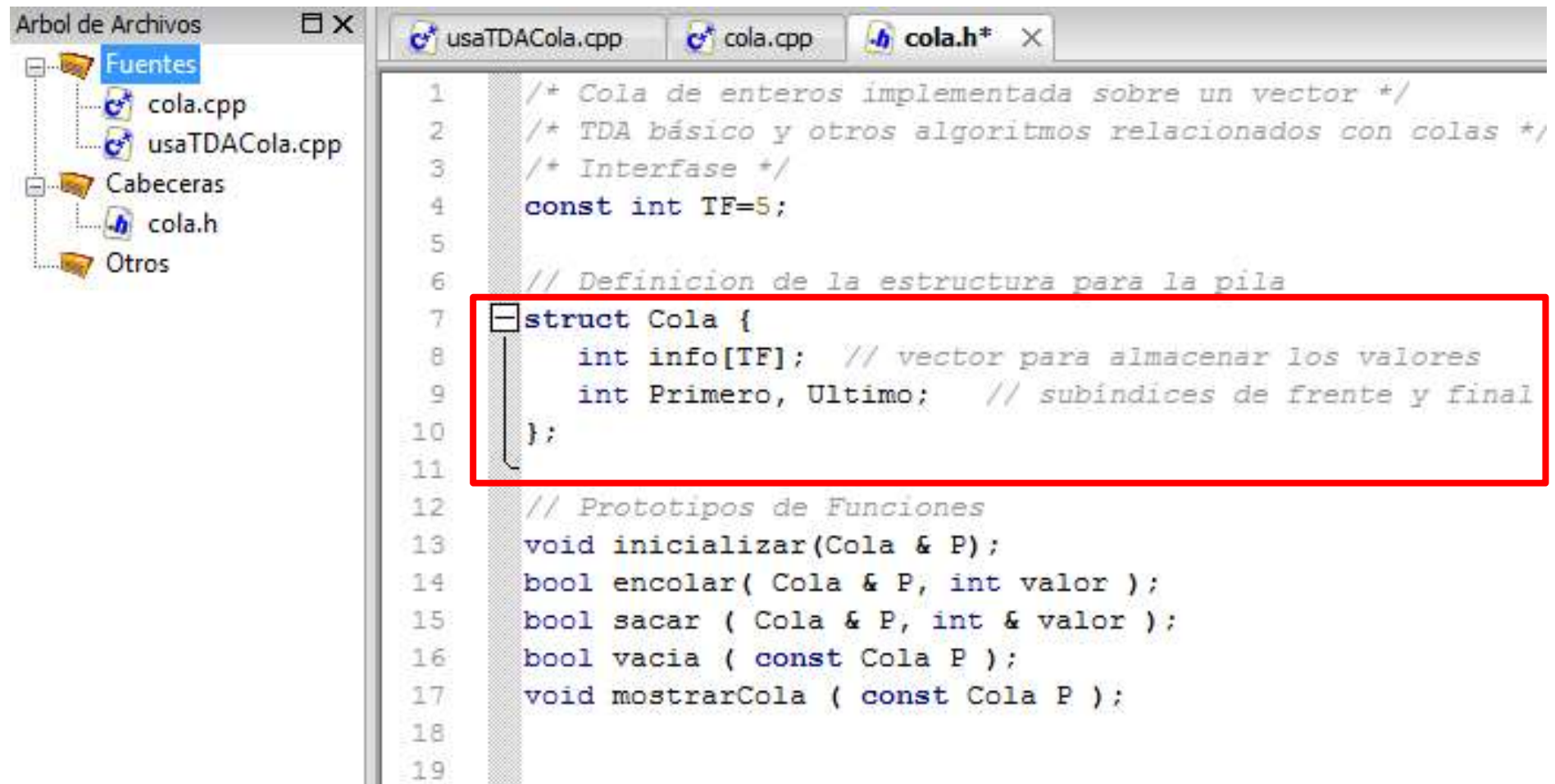
Implementación de los índices

- *final* comenzó en 5, luego se puso en 6, 7
- Es decir, **final = final + 1**
- Pero ... como la cola es circular, entonces *final* varía entre....
- 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, etc...
- Podemos usar la **teoría de los restos** para generar índices entre 0 y el **máximo tamaño de la cola - 1**:

$$\begin{aligned}\text{final} &= (\text{final} + 1) \% \text{MAXTAMQ} \\ \text{frente} &= (\text{frente} + 1) \% \text{MAXTAMQ}\end{aligned}$$

- **MAXTAMQ** indica el tamaño físico del arreglo
- Va a quedar:
 - 5 % MAXTAMQ = 5
 - 6 % MAXTAMQ = 6
 - 7 % MAXTAMQ = 7
 - 8 % MAXTAMQ = 0

Ejemplo



The image shows a screenshot of a C++ IDE. On the left, the 'Arbol de Archivos' (File Tree) displays a project structure with 'Fuentes' (Sources) containing 'cola.cpp' and 'usaTDACola.cpp', and 'Cabeceras' (Headers) containing 'cola.h'. The main editor window shows the code in 'cola.h'. The code includes comments in Spanish and defines a 'Cola' structure and several functions. A red rectangle highlights the 'struct Cola' definition, which includes an array 'info' and two integer variables 'Primero' and 'Ultimo'.

```
1  /* Cola de enteros implementada sobre un vector */
2  /* TDA básico y otros algoritmos relacionados con colas */
3  /* Interfase */
4  const int TF=5;
5
6  // Definicion de la estructura para la pila
7  struct Cola {
8      int info[TF]; // vector para almacenar los valores
9      int Primero, Ultimo; // subíndices de frente y final
10 };
11
12 // Prototipos de Funciones
13 void inicializar(Cola & P);
14 bool encolar( Cola & P, int valor );
15 bool sacar ( Cola & P, int & valor );
16 bool vacia ( const Cola P );
17 void mostrarCola ( const Cola P );
18
19
```

```

#include <iostream>
#include <cstdlib>
using namespace std;

#include "cola.h"

int main(){
    Cola C;
    int aux;

    inicializar(C);
    mostrarCola(C);
    if (encolar(C, 3)) cout << "Se inserto 3 " << endl << endl;
    else cout << "Overflow" << endl;
    mostrarCola(C);
    if (encolar(C, 8)) cout << "Se inserto 8 " << endl << endl;
    else cout << "Overflow" << endl;
    mostrarCola(C);
    if (encolar(C, 5)) cout << "Se inserto 5 " << endl << endl;
    else cout << "Overflow" << endl;
    mostrarCola(C);
    if (encolar(C, 1)) cout << "Se inserto 1 " << endl << endl;
    else cout << "Overflow" << endl;
    mostrarCola(C);
    if (encolar(C, 6)) cout << "Se inserto 6 " << endl << endl;
    else cout << "Overflow" << endl;
    mostrarCola(C);
    if (sacar(C, aux)) cout << "Se retiro de la cola el elemento = " << aux << endl << endl;
    else cout << "Underflow = Cola vacia " << endl << endl;
    mostrarCola(C);
    return 0;
}

```

```
#include <iostream>
using namespace std;

#include "cola.h"
```

```
void inicializar(Cola & C){
    C.Primeros = C.Ultimo = 0;    // P.Ultimo indica el lugar que ocupará
    }                             // el próximo valor que se encole
```

```
bool encolar(Cola & C, int valor){
    if ( (C.Ultimo+1)%TF == C.Primeros) //no se puede insertar en una cola llena
        return false;
    else {
        C.info[C.Ultimo]= valor;
        C.Ultimo= (C.Ultimo+1)%TF;
        return true;
    }
}
```

```
bool sacar (Cola & C, int & valor){  
    if (vacía(C))  
        return false;    //no se puede sacar de una cola vacía  
    else {  
        valor= C.info[C.Primer0];  
        C.Primer0 = (C.Primer0+1) % TF;  
        return true;  
    }  
}
```

```
bool vacía (const Cola C) {  
    return (C.Primer0==C.Ultimo);  
}
```

```
void mostrarCola ( const Cola C ){  
    cout << "Cola = [ ";  
    for (int i=C.Primer0; i<C.Ultimo; i= (i+1)%TF )  
        cout << C.info[i] << " ";  
    cout << " ] , Primer0 = " << C.Primer0 << " , Ultimo = " << C.Ultimo << endl;  
}
```

Cola = [] , Primero = 0 , Ultimo = 0

Se inserto 3

Cola = [3] , Primero = 0 , Ultimo = 1

Se inserto 8

Se inserto 5

Se inserto 1

Overflow

Cola = [3 8 5 1] , Primero = 0 , Ultimo = 4

Se retiro de la cola el elemento = 3

Cola = [8 5 1] , Primero = 1 , Ultimo = 4

Aplicación de Colas

Las aplicaciones de las colas en la programación son numerosas. Se utilizan en:

- colas de mensajes
- colas de tareas a realizar por una impresora
- colas de prioridades

LEER

“Pilas”, pág. 311

“Colas”, pág. 339

Libro: “Estructura de datos en C++” – Luis Joyanes Aguilar