Estructuras de datos lineales

Cola circular

M.S.C. Jacob Green • 05-11-2020

Cola basada en arreglo

- Podemos representar una cola utilizando arreglos

Cola basada en arreglo

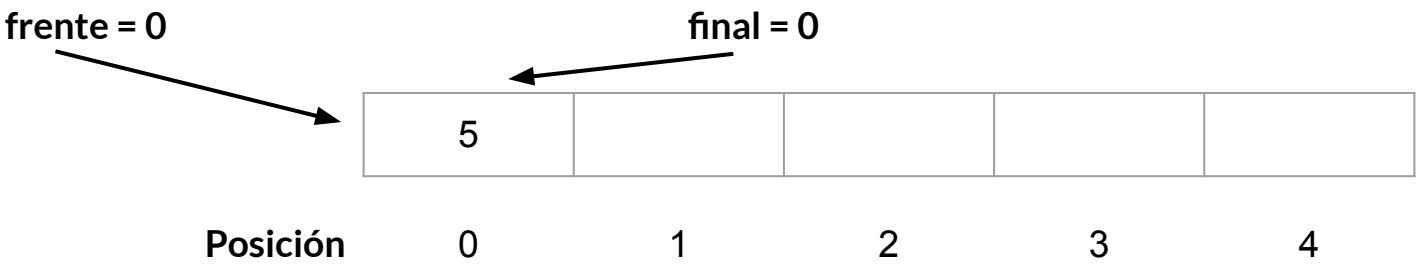
- Podemos representar una cola utilizando arreglos

Posición	0	1	2	3	4

capacidad de almacenamiento de la cola = longitud del arreglo = 5

Cola basada en arreglo

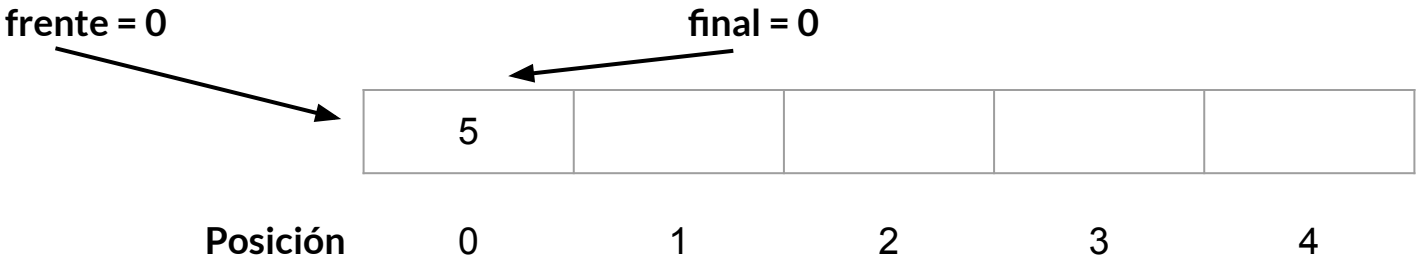
- En este caso, frente y final hacen referencia a la posición:



capacidad de almacenamiento de la cola = longitud del arreglo = 5

Cola basada en arreglo

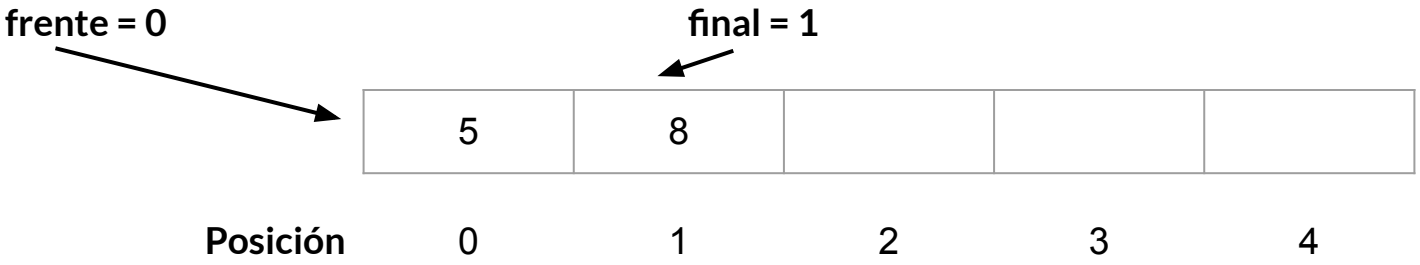
- La operación de inserción (push) consiste en mover final a la derecha:



capacidad de almacenamiento de la cola = longitud del arreglo = 5

Cola basada en arreglo

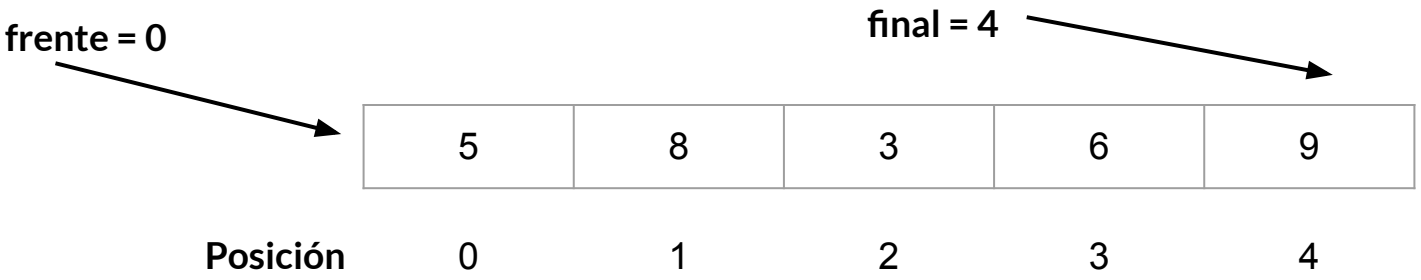
- La operación de inserción (push) consiste en mover final a la derecha:



capacidad de almacenamiento de la cola = longitud del arreglo = 5

Cola basada en arreglo

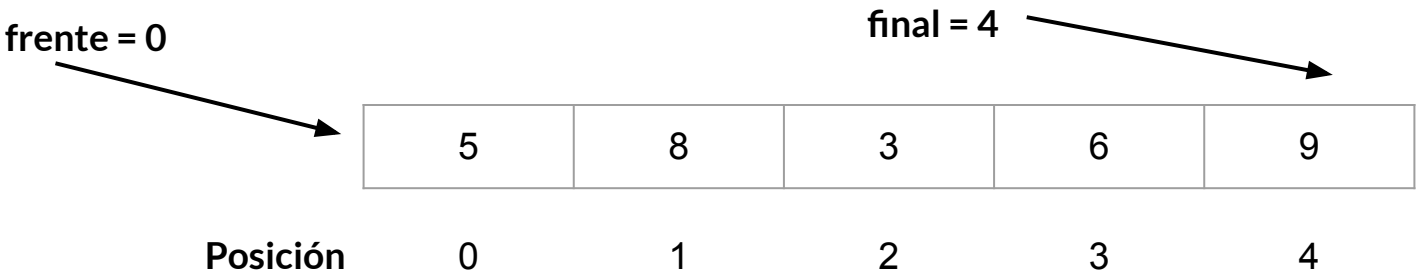
- La operación de inserción (push) consiste en mover final a la derecha:



capacidad de almacenamiento de la cola = longitud del arreglo = 5

Cola basada en arreglo

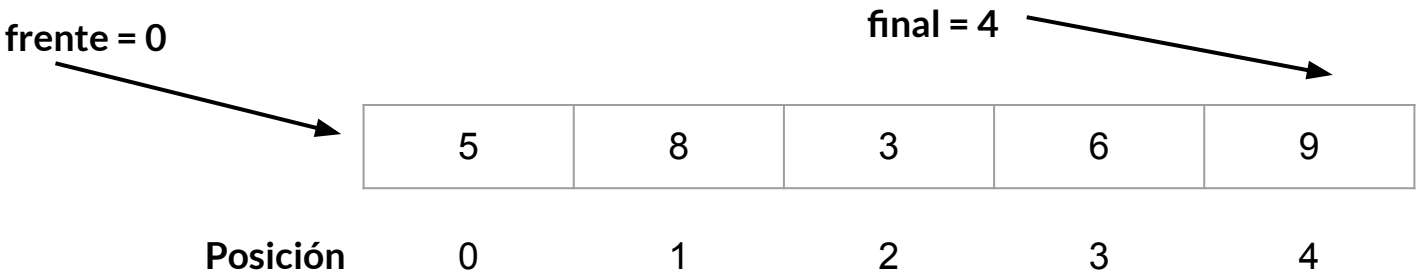
- La cola está llena cuando el arreglo está lleno



cola llena si tamaño de la cola igual a capacidad de la cola

Cola basada en arreglo

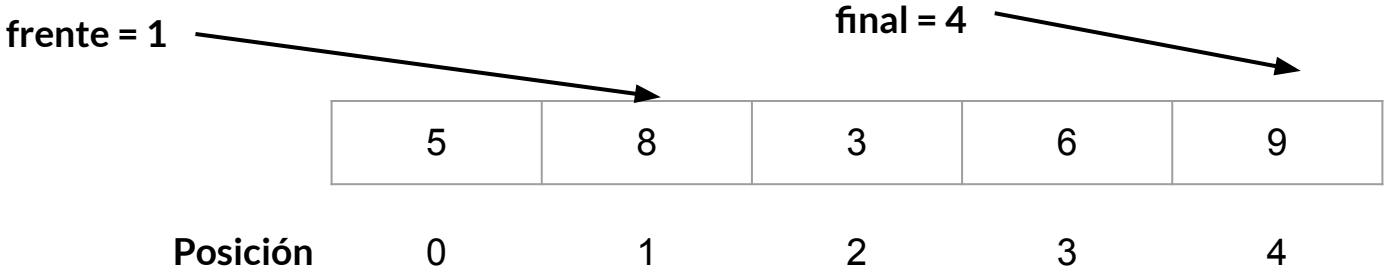
- La operación de eliminación (pop), consiste en mover frente a la derecha:



cola llena si tamaño de la cola igual a capacidad de la cola

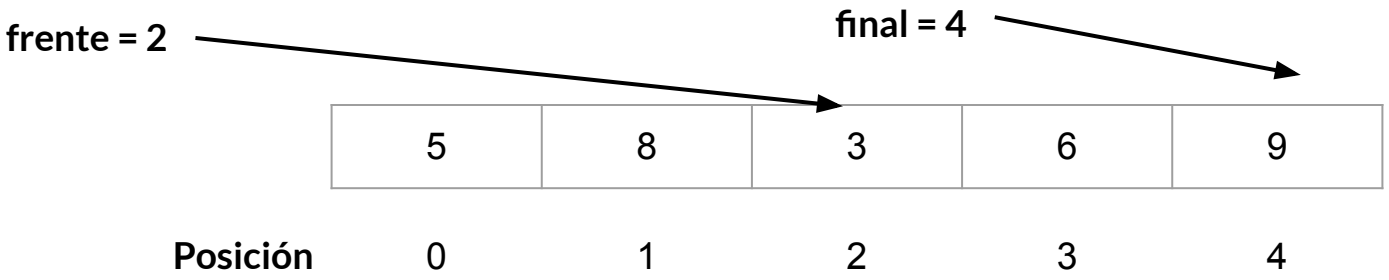
Cola basada en arreglo

- La operación de eliminación (pop), consiste en mover frente a la derecha:



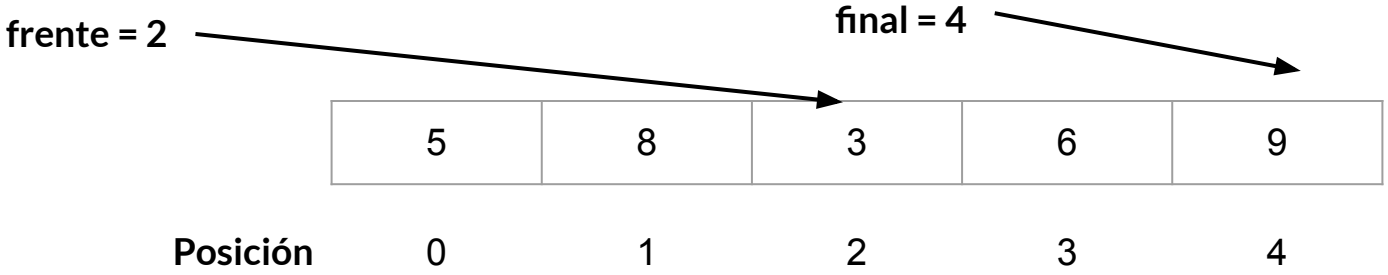
Cola basada en arreglo

- La operación de eliminación (pop), consiste en mover frente a la derecha:



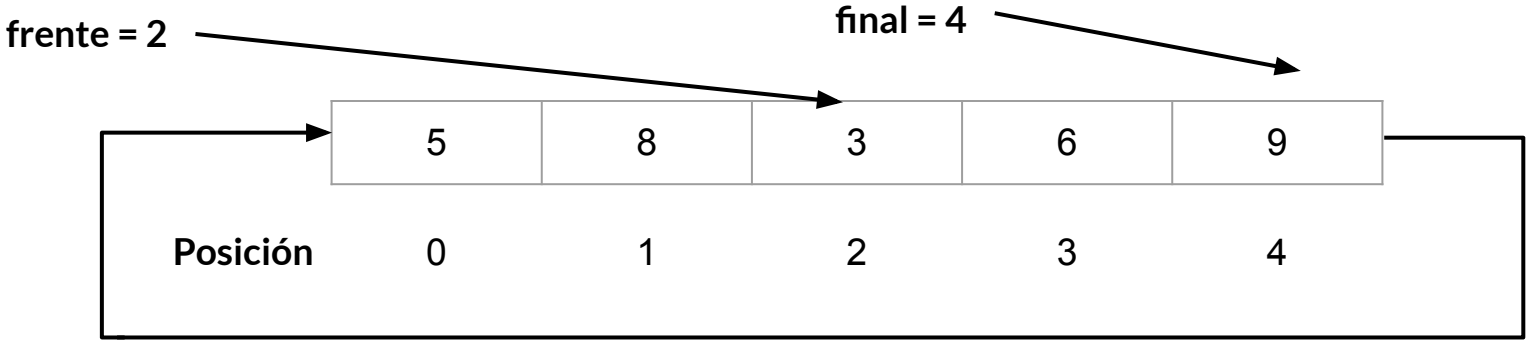
Cola basada en arreglo

- Para aprovechar el espacio en memoria a la izquierda de frente, se une el último elemento del arreglo con el primero:



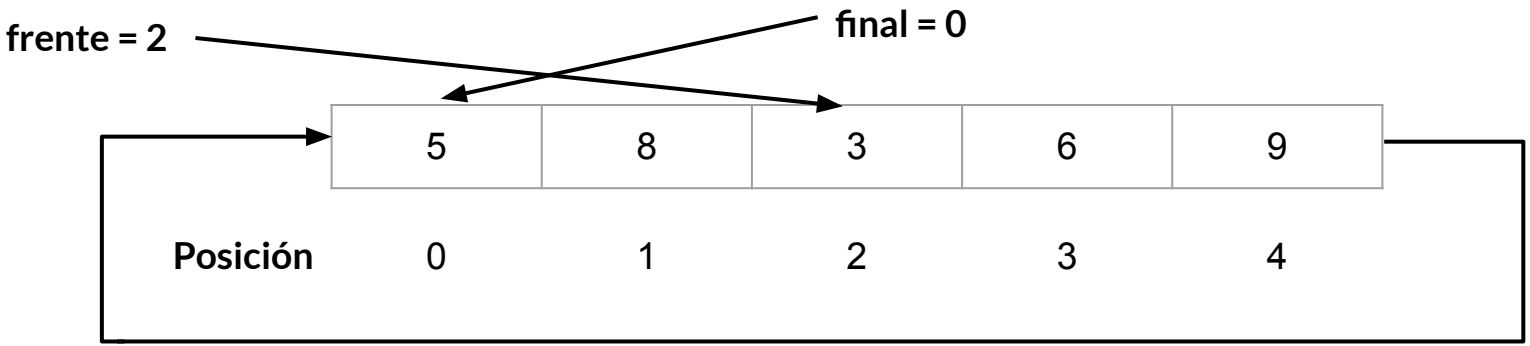
Cola basada en arreglo

- Para aprovechar el espacio en memoria a la izquierda de frente, se une el último elemento del arreglo con el primero:



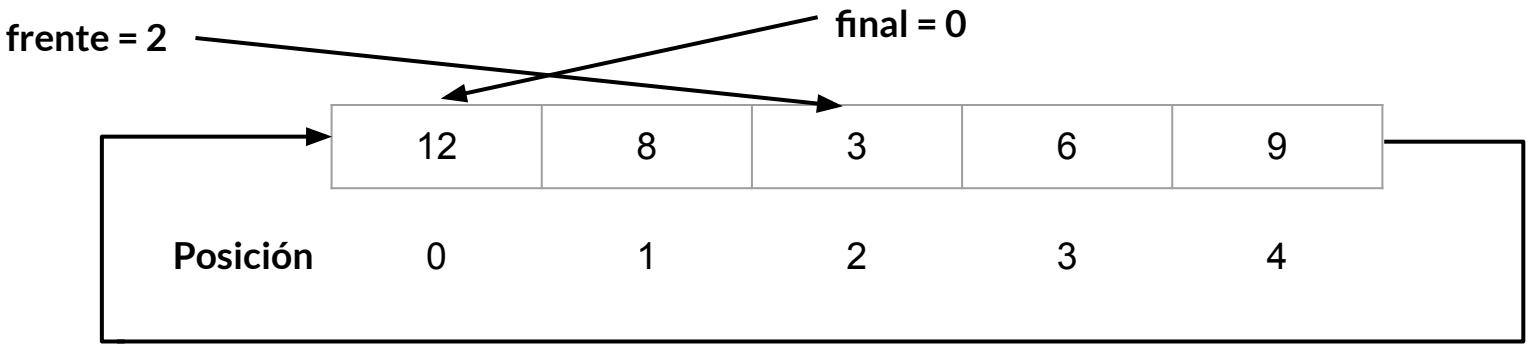
Cola basada en arreglo

- Para aprovechar el espacio en memoria a la izquierda de frente, se une el último elemento del arreglo con el primero:



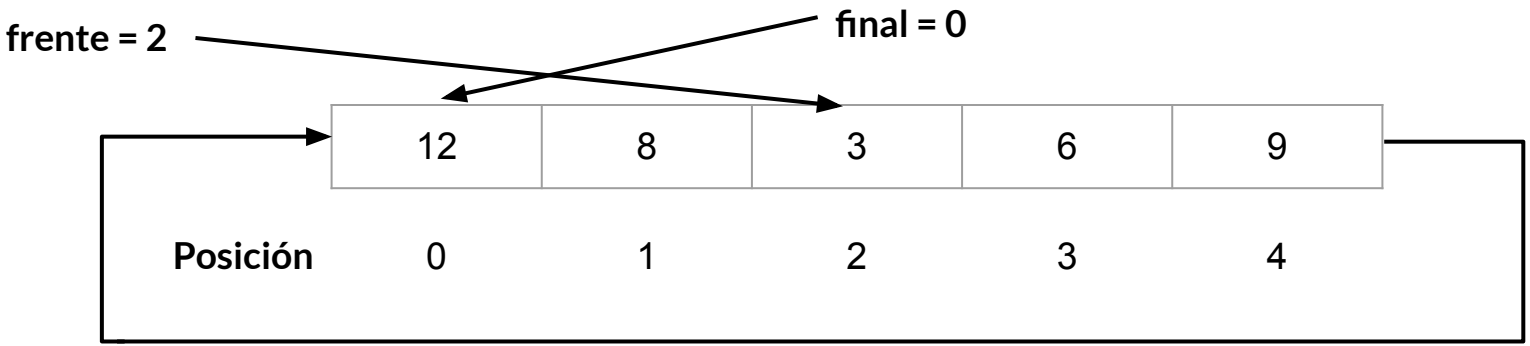
Cola basada en arreglo

- Para aprovechar el espacio en memoria a la izquierda de frente, se une el último elemento del arreglo con el primero:



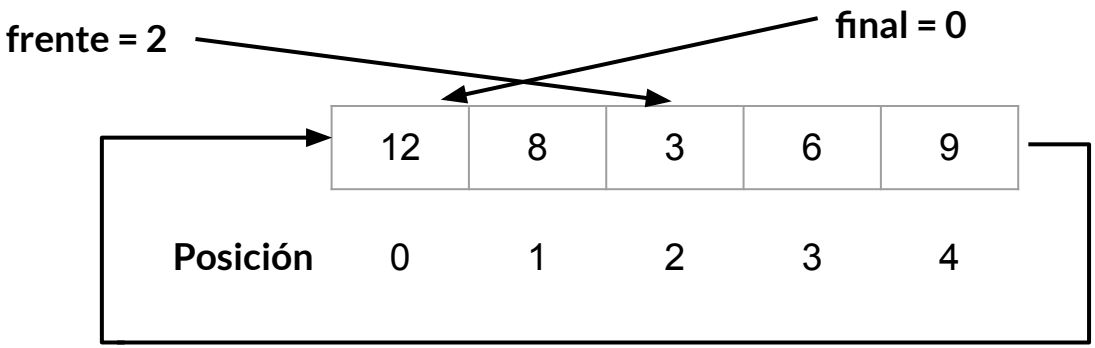
Cola basada en arreglo

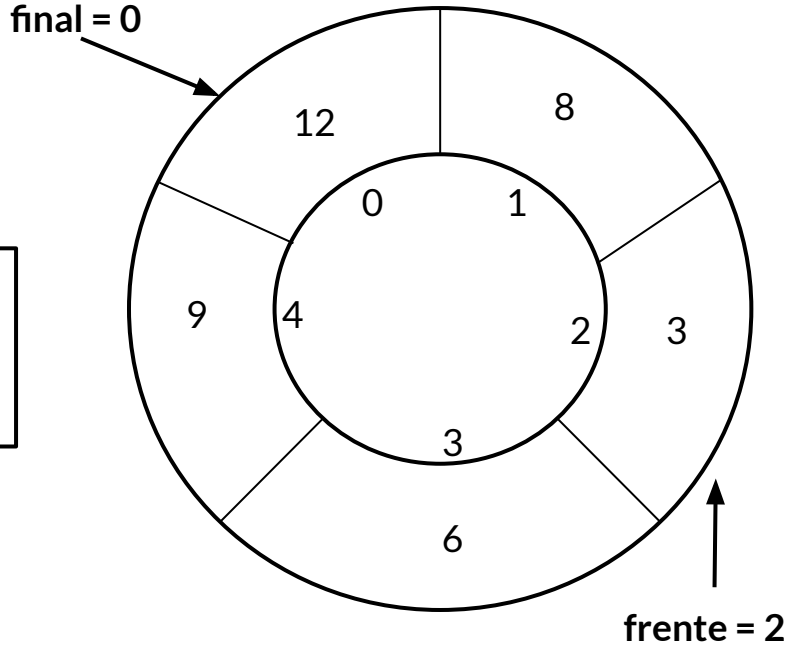
- Para aprovechar el espacio en memoria a la izquierda de frente, se une el último elemento del arreglo con el primero:



A esto le llamamos **cola circular**!!! dado que del final del arreglo se vuelve al inicio

Cola circular





El TDA Cola circular

```
class ColaCircular {
    private:
        int frente;
        int final;
        int tam;
        int capacidad;
        T *nodos;

        void resize();

    protected:
        int siguiente(int r)
        {
            return (r+1) % capacidad;
        }
    public:
        ColaCircular() {
            capacidad = 0;
            frente = 0;
            final = capacidad - 1;
            tam = 0;
            nodos = new T[capacidad];
        }
```

datos

operación privada

operación protegida

constructor

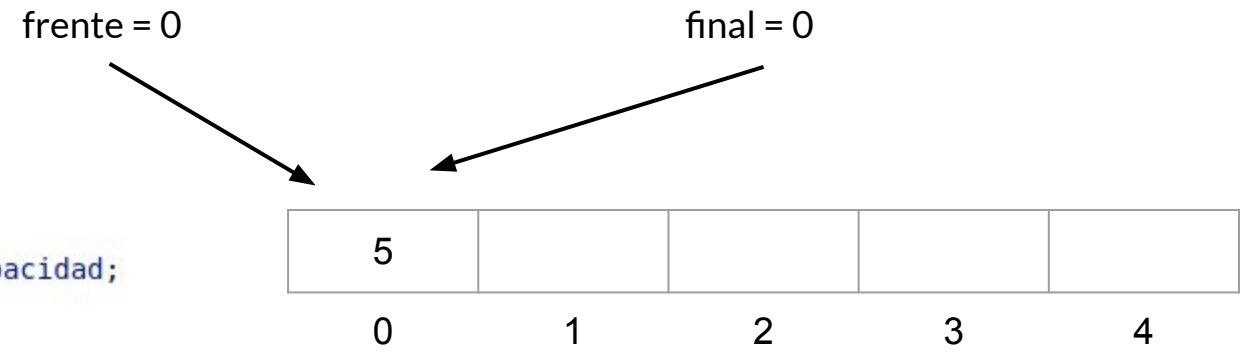
operaciones

```
        void push(T d);
        void pop();
        T front();
        T back();
        bool full();
        bool empty();
        int size();
        void clear();

        ~ColaCircular() {
            clear();
        }
};
```

destructor

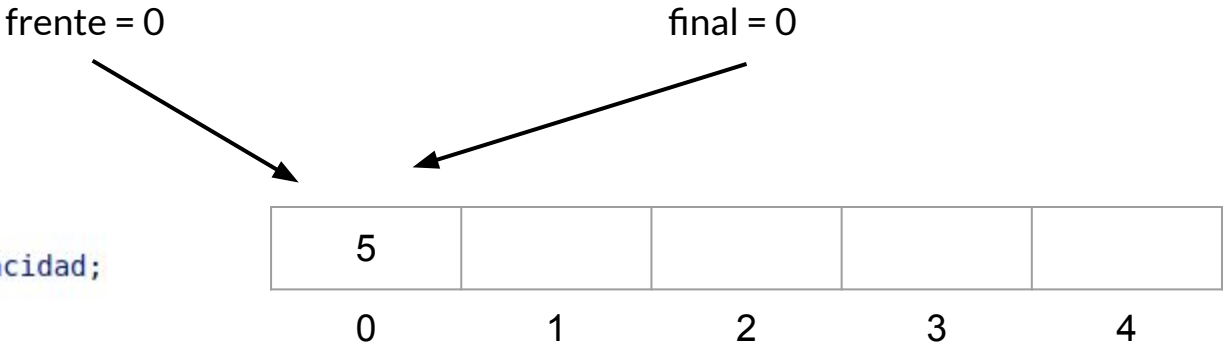
La operación siguiente

```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```



La operación siguiente

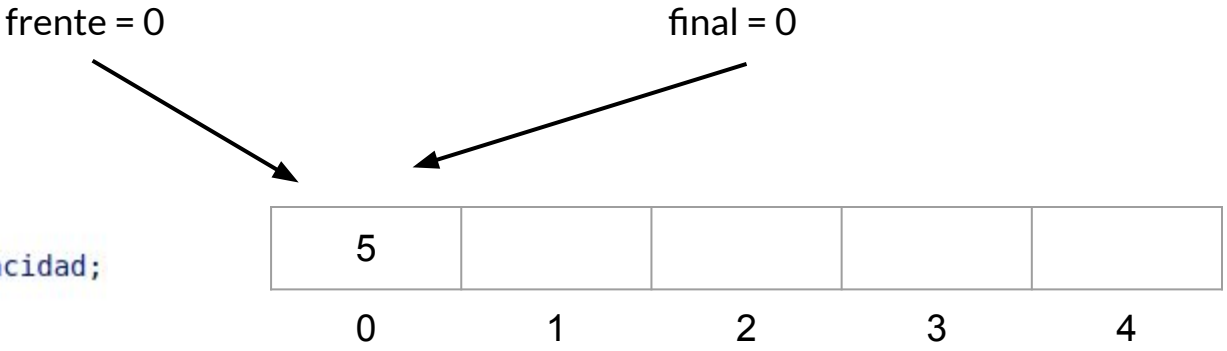
```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```



final = siguiente(final);

La operación siguiente

```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```



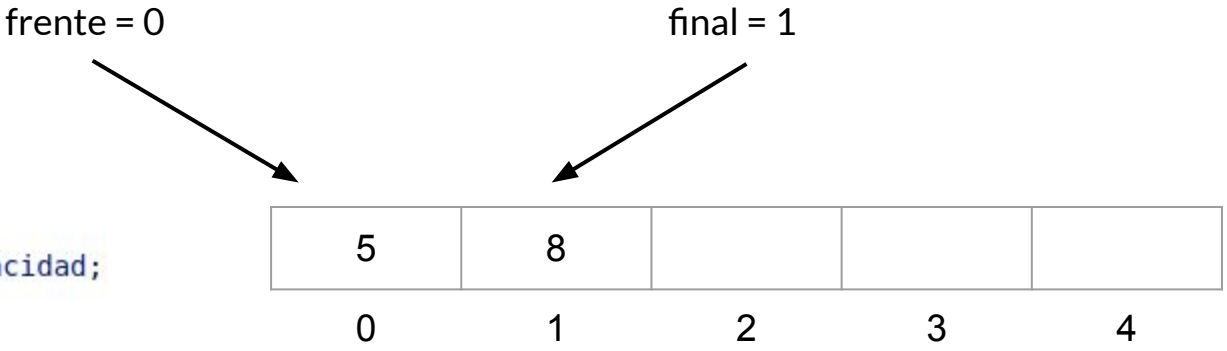
final = siguiente(final);

final = (final + 1) % capacidad;

final = (0 + 1) % capacidad;

La operación siguiente

```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```



final = siguiente(final);

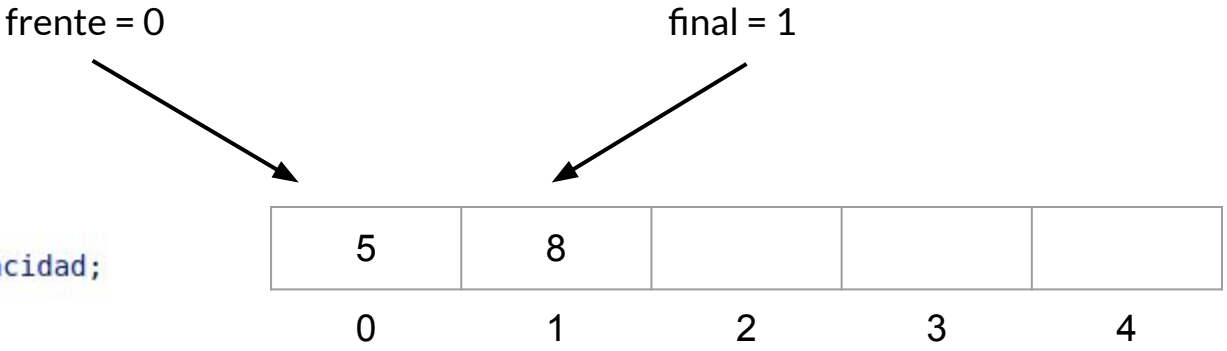
final = (final + 1) % capacidad;

final = (0 + 1) % capacidad;

final = 1;

La operación siguiente

```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```

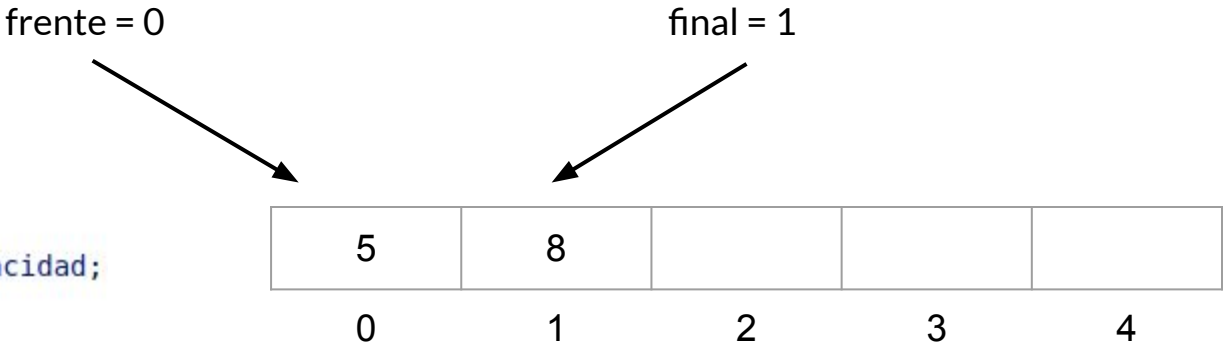


final = siguiente(final);

final = (final + 1) % capacidad;

La operación siguiente

```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```



final = siguiente(final);

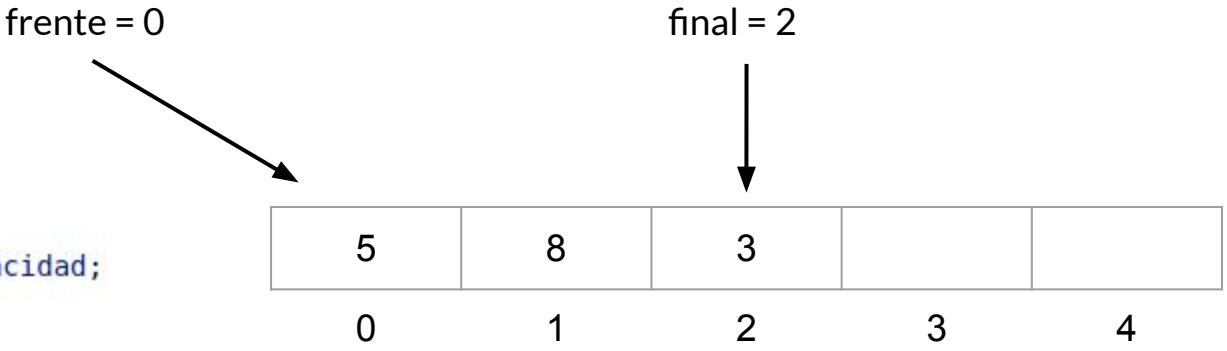
final = (final + 1) % capacidad;

final = (1 + 1) % capacidad;

final = 2;

La operación siguiente

```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```



final = siguiente(final);

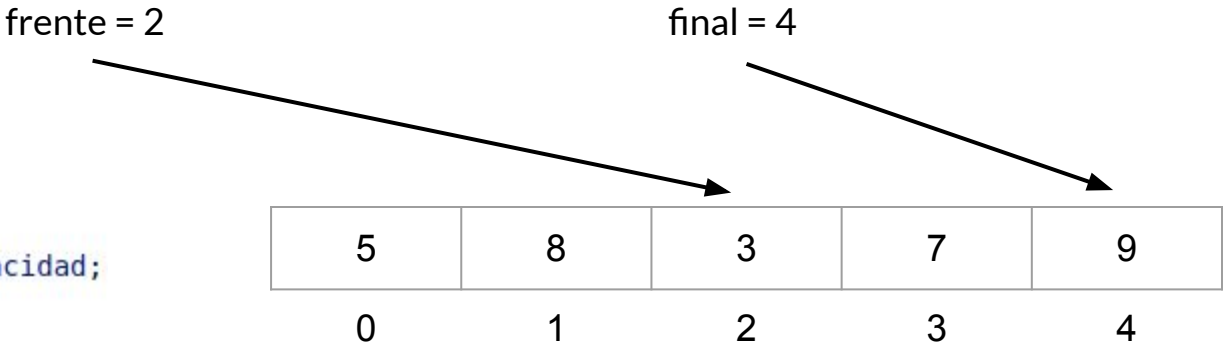
final = (final + 1) % capacidad;

final = (1 + 1) % capacidad;

final = 2;

La operación siguiente

```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```



final = siguiente(final);

final = (final + 1) % capacidad;

La operación siguiente

frente = 2

final = 4

```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```

5	8	3	7	9
0	1	2	3	4

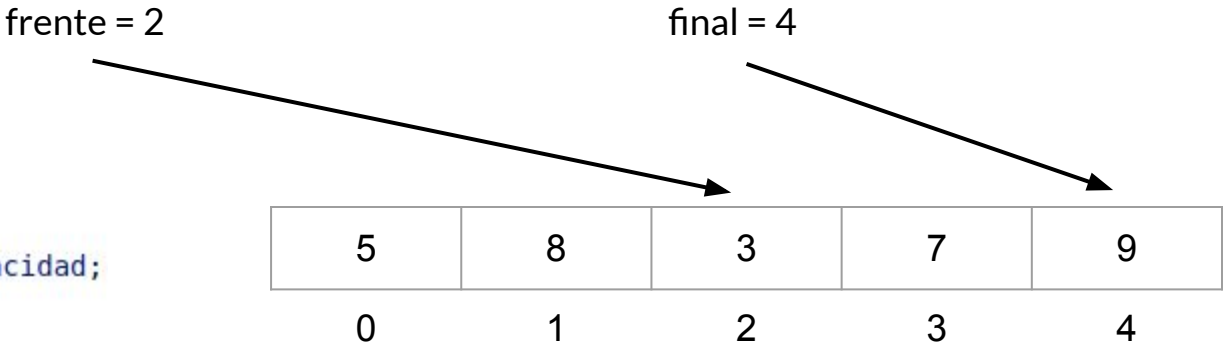
final = siguiente(final);

final = (final + 1) % capacidad;

final = (4 + 1) % capacidad;

La operación siguiente

```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```



final = siguiente(final);

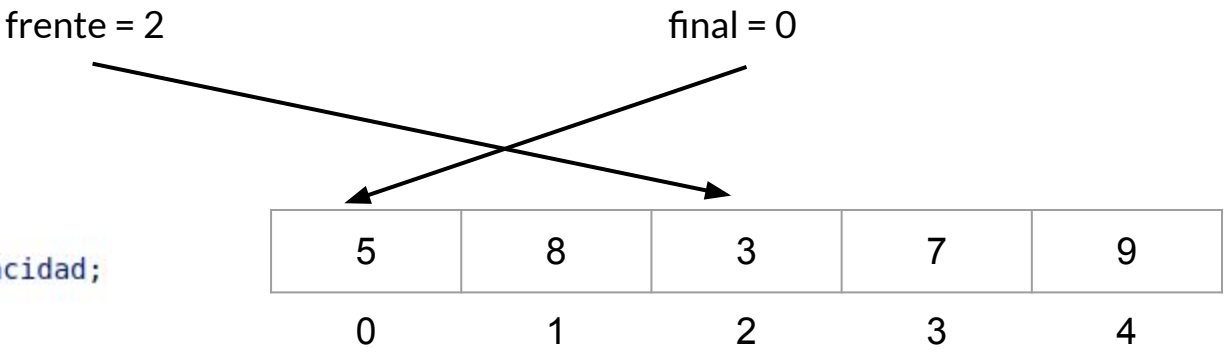
final = (final + 1) % capacidad;

final = (4 + 1) % capacidad;

final = 0;

La operación siguiente

```
int siguiente(int r)
{
    return (r+1) % capacidad;
}
```



final = siguiente(final);

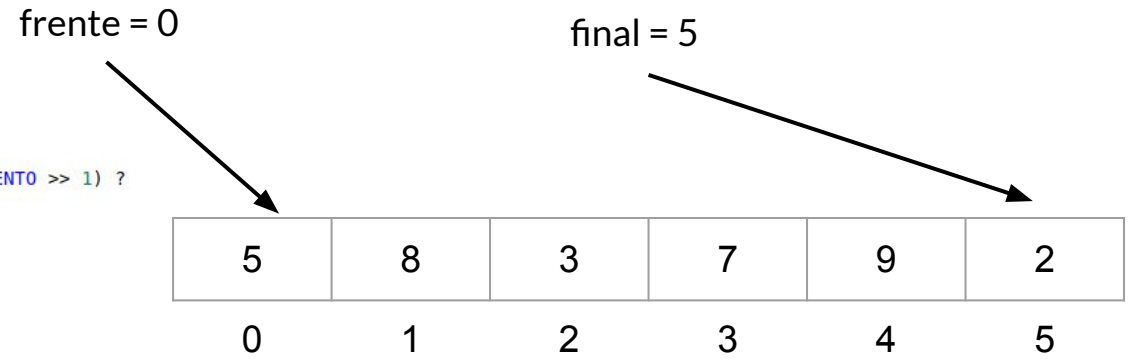
final = (final + 1) % capacidad;

final = (4 + 1) % capacidad;

final = 0;

La operación resize

```
#define CAPACIDAD INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```



La operación resize

```
#define CAPACIDAD INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```

frente = 0

final = 5

5	8	3	7	9	2
0	1	2	3	4	5

Si **6** es menor que **3** entonces **incremento** = **6**, de lo contrario **incremento** = **3**;

La operación resize

```
#define CAPACIDAD INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```

frente = 0

final = 5

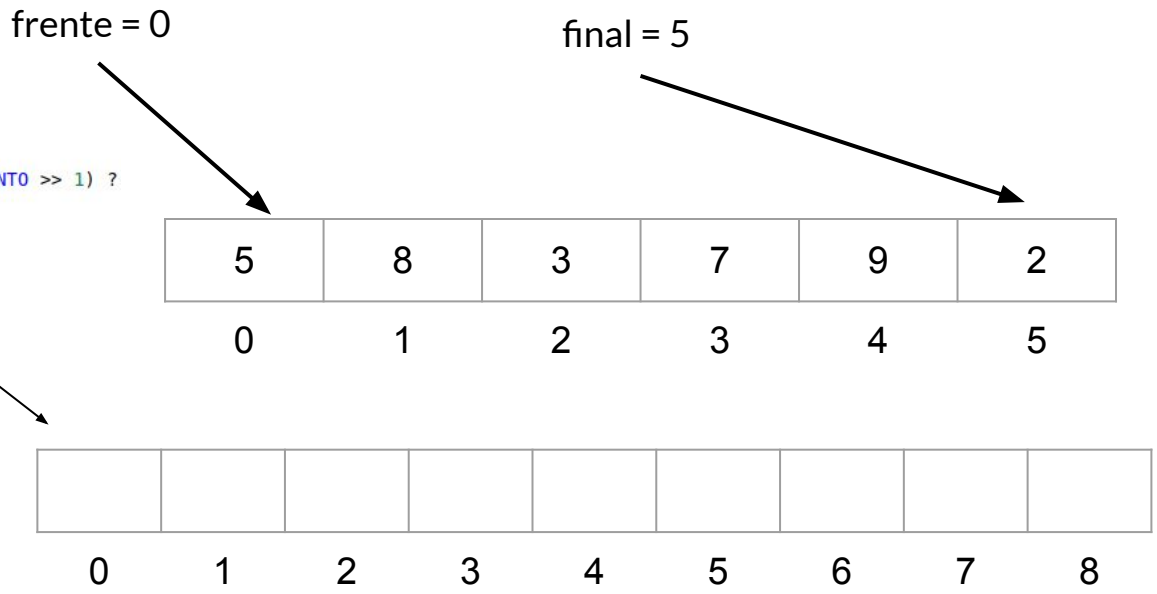
5	8	3	7	9	2
0	1	2	3	4	5

capacidad = 6 + 3;

capacidad = 9;

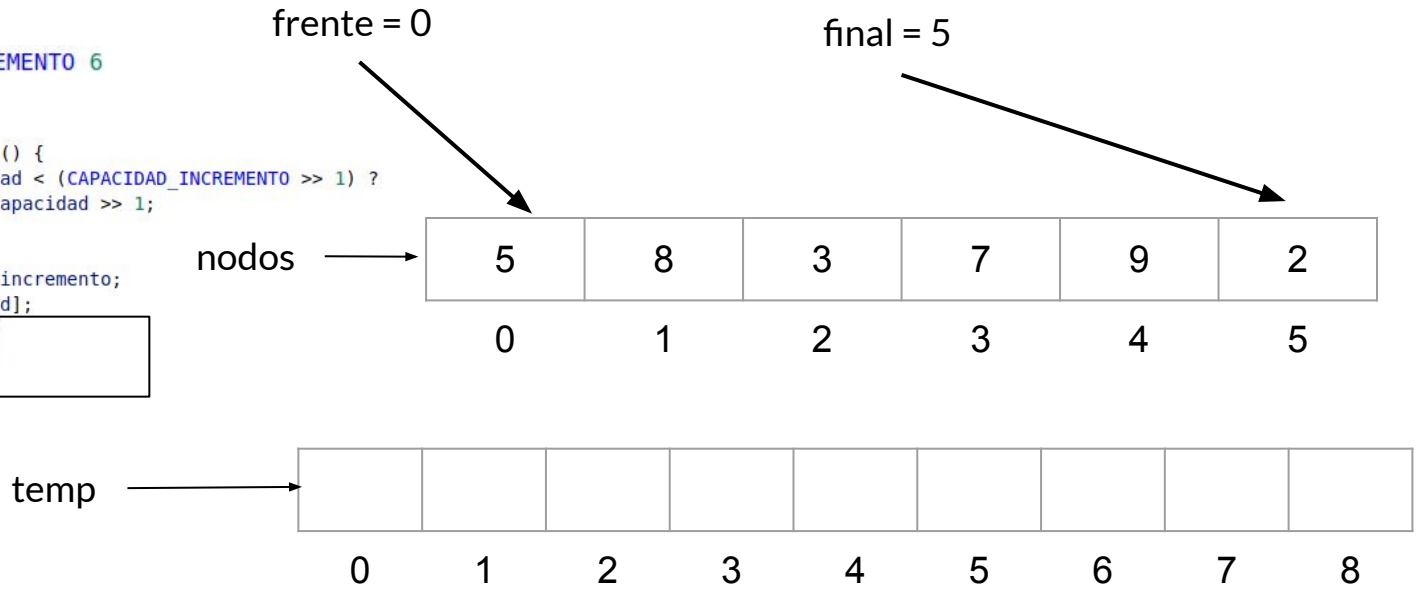
La operación resize

```
#define CAPACIDAD INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```



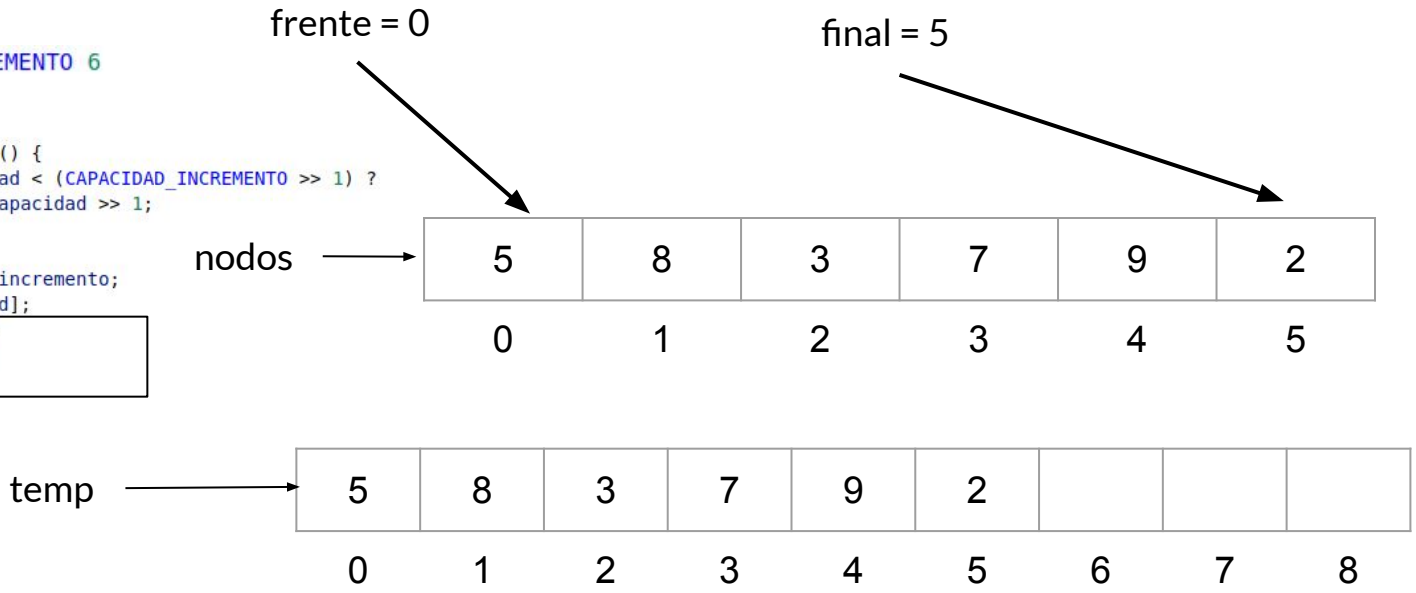
La operación resize

```
#define CAPACIDAD INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```



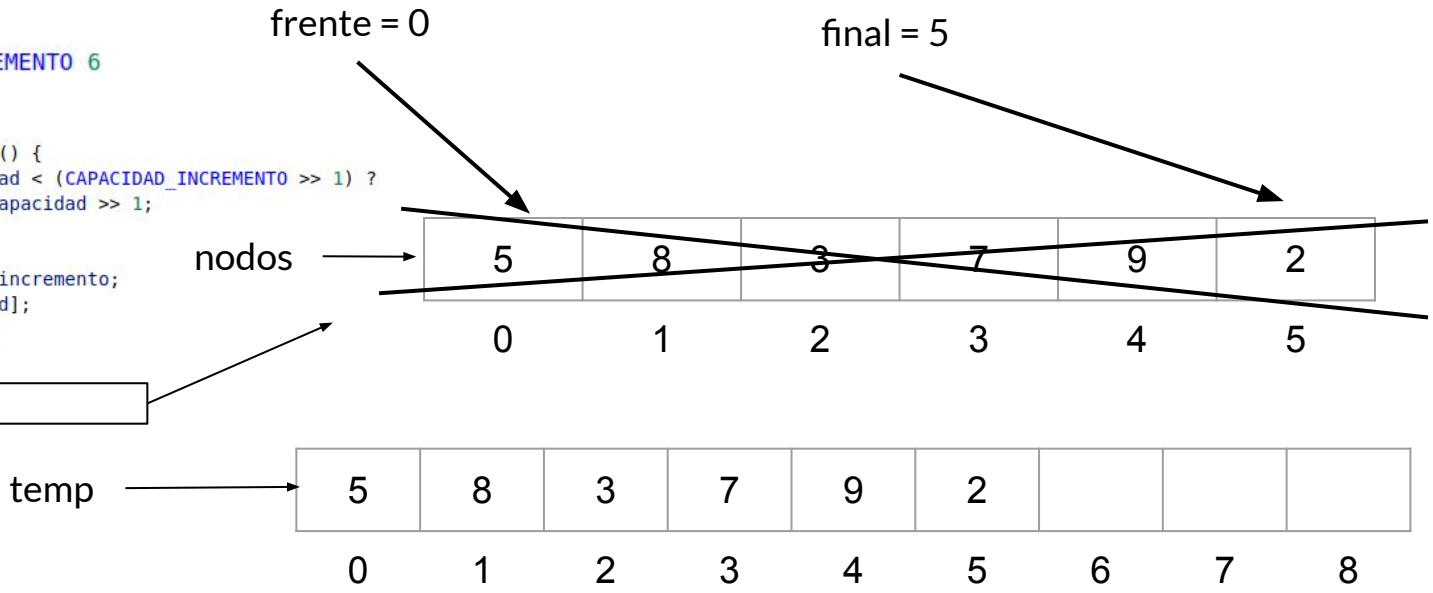
La operación resize

```
#define CAPACIDAD INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```



La operación resize

```
#define CAPACIDAD INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```



La operación resize

```
#define CAPACIDAD_INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```

frente = 0

final = 5

nodos

temp

5	8	3	7	9	2			
0	1	2	3	4	5	6	7	8

La operación resize

```
#define CAPACIDAD INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```

frente = 0

final = 5

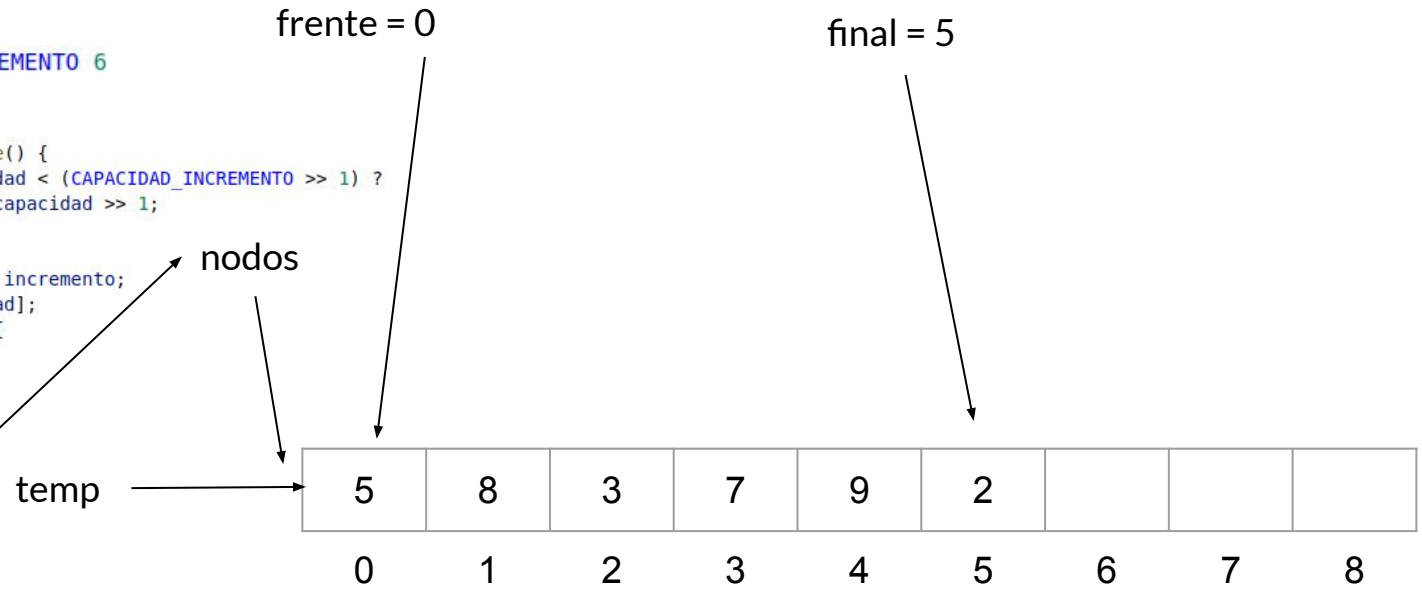
nodos

temp

5	8	3	7	9	2			
0	1	2	3	4	5	6	7	8

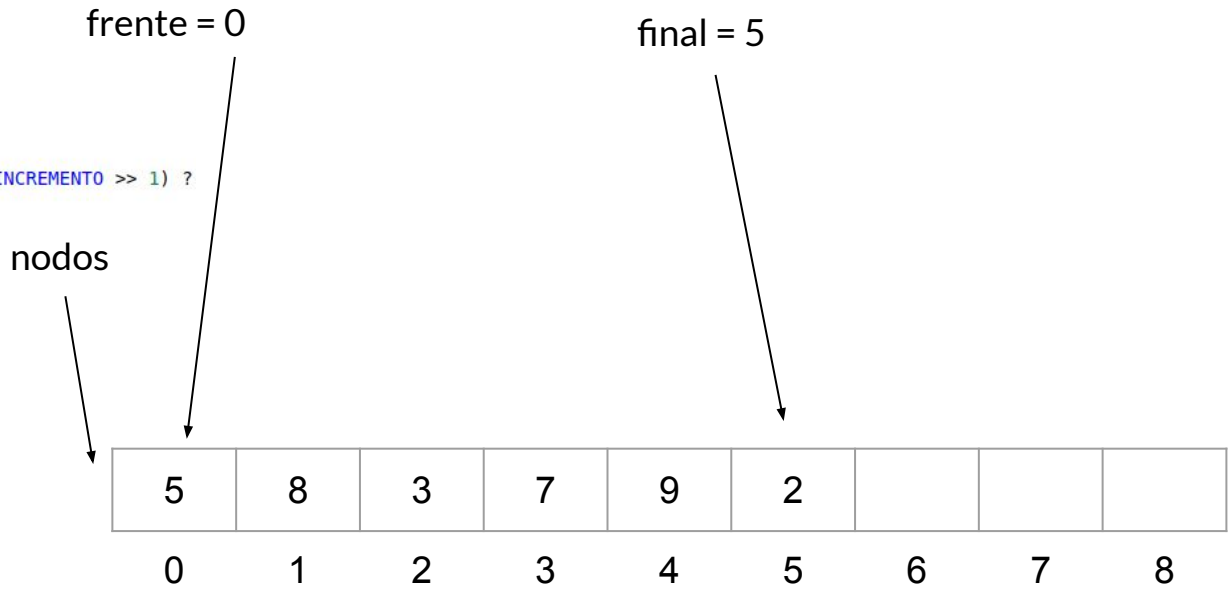
La operación resize

```
#define CAPACIDAD INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```



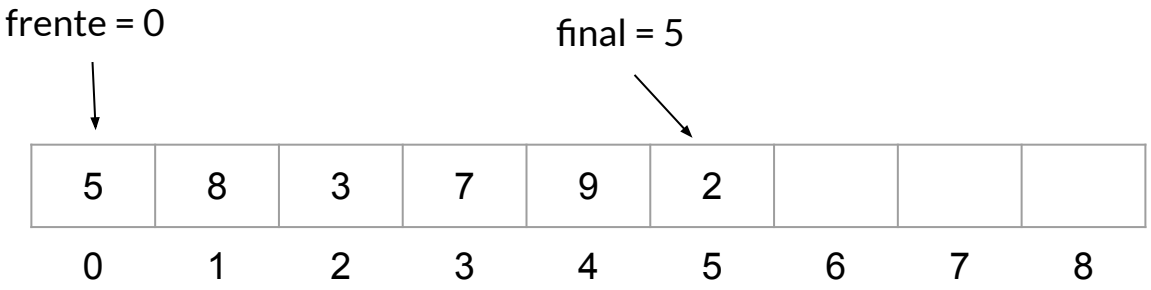
La operación resize

```
#define CAPACIDAD_INCREMENTO 6

template <class T>
void ColaCircular<T>::resize() {
    int incremento = capacidad < (CAPACIDAD_INCREMENTO >> 1) ?
    CAPACIDAD_INCREMENTO : capacidad >> 1;

    int n = capacidad;
    capacidad = capacidad + incremento;
    T *temp = new T[capacidad];
    for(int i=0; i<n; i++) {
        temp[i] = nodos[i];
    }
    delete[] nodos;
    nodos = temp;
}
```



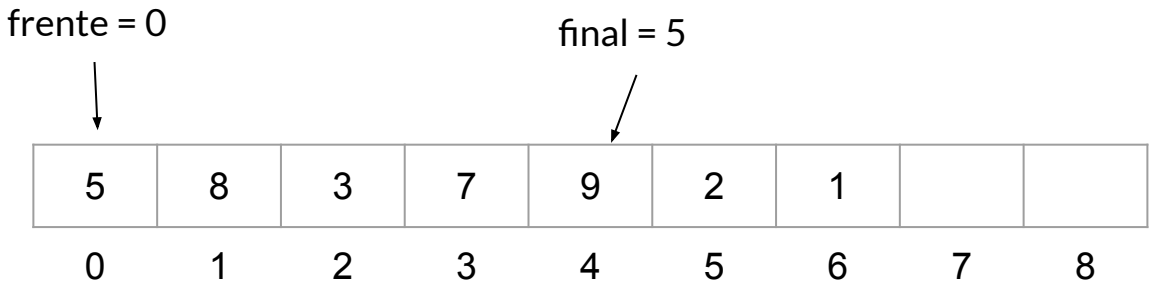
La operación push

```
template <class T>
void ColaCircular<T>::push(T d) {
    if(full()) {
        resize();
    }
    final = siguiente(final);
    nodos[final] = d;
    tam++;
}
```



La operación push

```
template <class T>
void ColaCircular<T>::push(T d) {
    if(full()) {
        resize();
    }
    final = siguiente(final);
    nodos[final] = d;
    tam++;
}
```

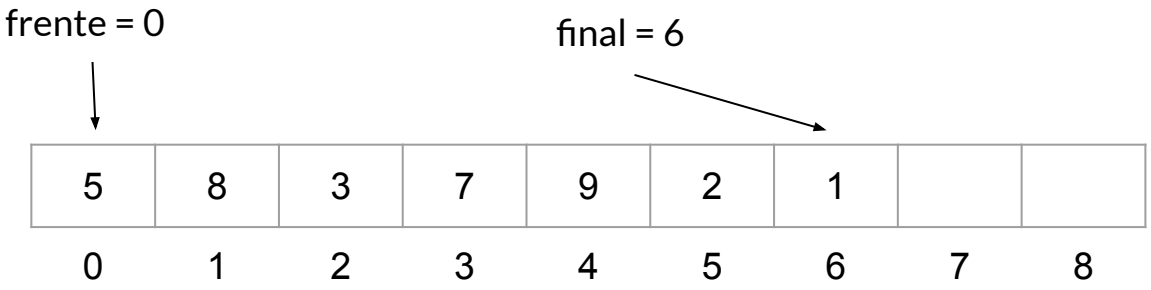


final = siguiente(final);

final = siguiente(5);

La operación push

```
template <class T>
void ColaCircular<T>::push(T d) {
    if(full()) {
        resize();
    }
    final = siguiente(final);
    nodos[final] = d;
    tam++;
}
```



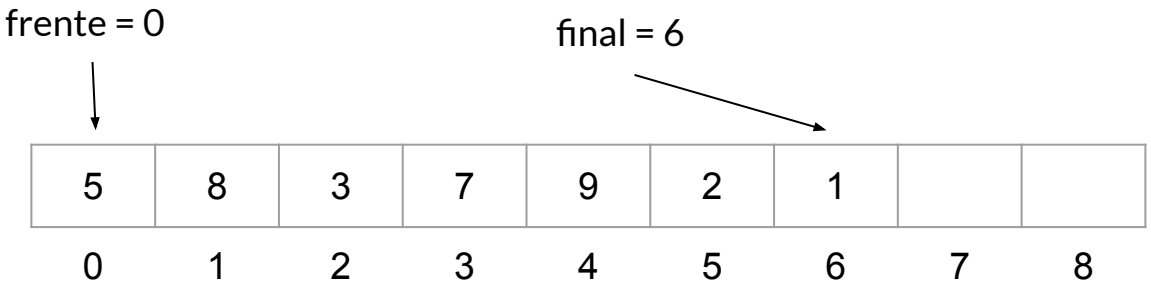
final = siguiente(final);

final = siguiente(5);

final = 6;

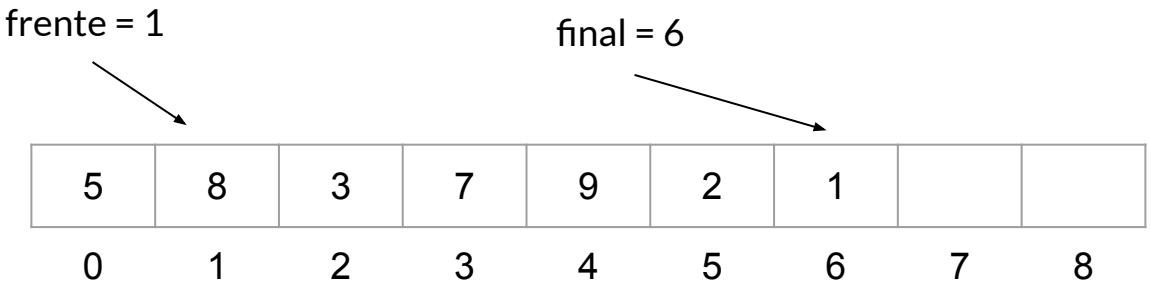
La operación pop

```
template <class T>
void ColaCircular<T>::pop() {
    if(!empty()) {
        frente = siguiente(frente);
        tam--;
    }
}
```

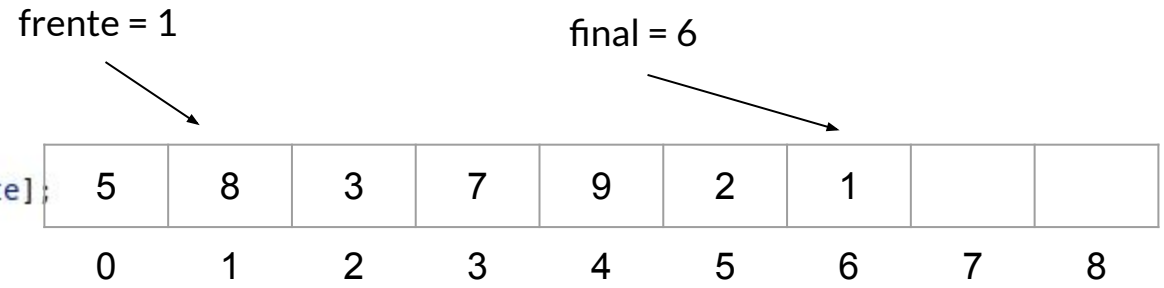


La operación pop

```
template <class T>
void ColaCircular<T>::pop() {
    if(!empty()) {
        frente = siguiente(frente);
        tam--;
    }
}
```



Las operaciones front y back




```
template <class T>
T ColaCircular<T>::front() {
    if(!empty()) return nodos[frente];
    T *t;
    return *t;
}
```

```
template <class T>
T ColaCircular<T>::back() {
    if(!empty()) return nodos[final];
    T *t;
    return *t;
}
```

Otras operaciones

```
template <class T>
bool ColaCircular<T>::empty() {
    return tam == 0;
}
```

```
template <class T>
bool ColaCircular<T>::full() {
    return tam == capacidad;
}
```

```
template <class T>
int ColaCircular<T>::size() {
    return tam;
}
```

```
template <class T>
void ColaCircular<T>::clear() {
    delete[] nodos;
}
```

Código completo

https://github.com/JGreen86/EstructurasDeDatos/blob/master/EstructurasLineales/Colas/ColaCircular.hpp