Grado en Ingeniería Información

Estructura de Datos y Algoritmos

Sesión 6

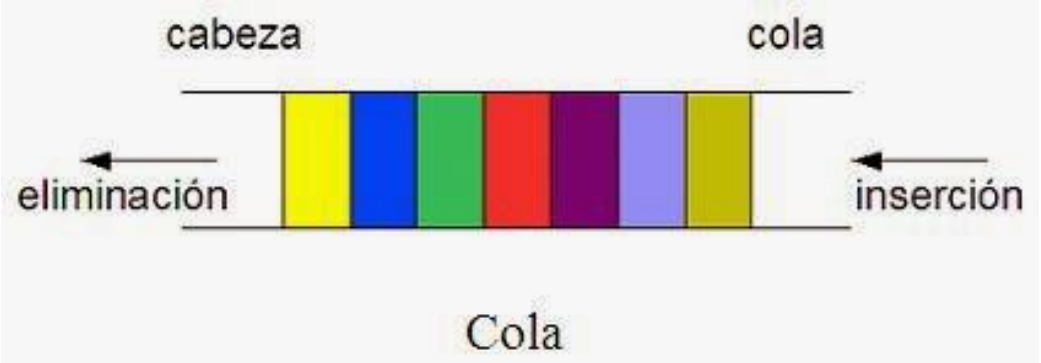
Curso 2022-2023

Marta N. Gómez

UNIVERSIDAD NEBRIJA

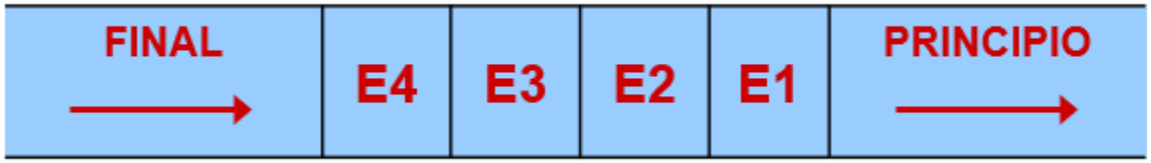
T3. Tipos Abstractos de Datos (TAD)

- Concepto.
- Tipos de datos lineales:
  - Pilas
  - **Colas**
  - Listas



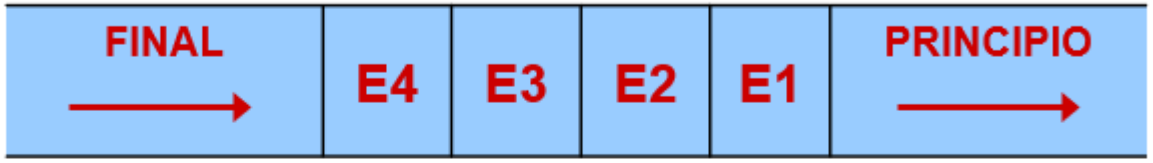
Una **COLA** es un **conjunto ordenado de elementos homogéneos**, en el cual los elementos se eliminan por uno de sus extremos, ***Principio*** o ***Cabeza***, y se añaden por el otro extremo, ***Final***. Su funcionamiento sigue una política ***FIFO***.

Es una estructura de datos **Lineal.**



COLAS

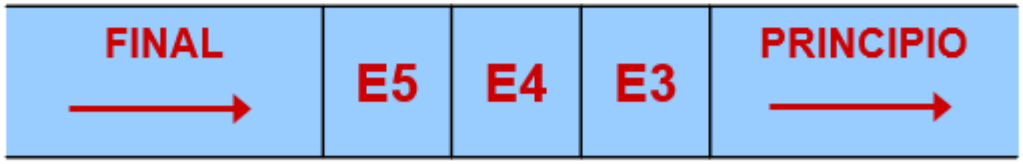
Añadir:



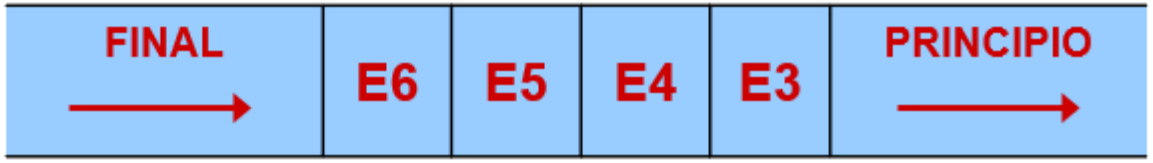
Añadir:

Eliminar:

Eliminar:



Añadir:



Operaciones Básicas de Cola

empty: **Determina si la cola está vacía o no.**

Precondición: Ninguna.

Postcondición: Decide si la cola q tiene elementos o no.

Por tanto, la cola q no se modifica.

front: **Devuelve el elemento que ocupa la primera posición de la cola, siempre que no esté vacía.**

Precondición: La cola q no puede estar vacía.

Postcondición: Obtiene el elemento que ocupa la posición del principio de la cola q sin eliminarlo.

Por tanto, la cola q no se modifica.

Operaciones Básicas de Cola

push: **Inserta un elemento en por el final de la cola y se obtiene la cola con un elemento más.**

Precondición: Ninguna.

Postcondición: Almacena en la cola q el elemento e y devuelve la cola resultante.

Por tanto, la cola q se modifica.

pop: **Elimina el elemento de que ocupa la posición del principio de la cola y devuelve la cola resultante, siempre que la cola no esté vacía.**

Precondición: La cola q no puede estar vacía

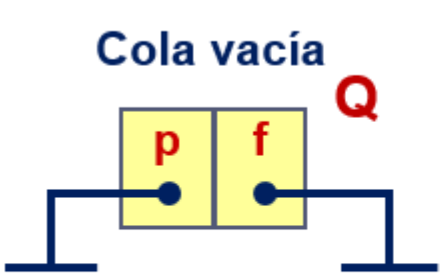
Postcondición: Elimina de la cola q el elemento que ocupa la primera posición de la cola.

Por tanto, la cola q se modifica.

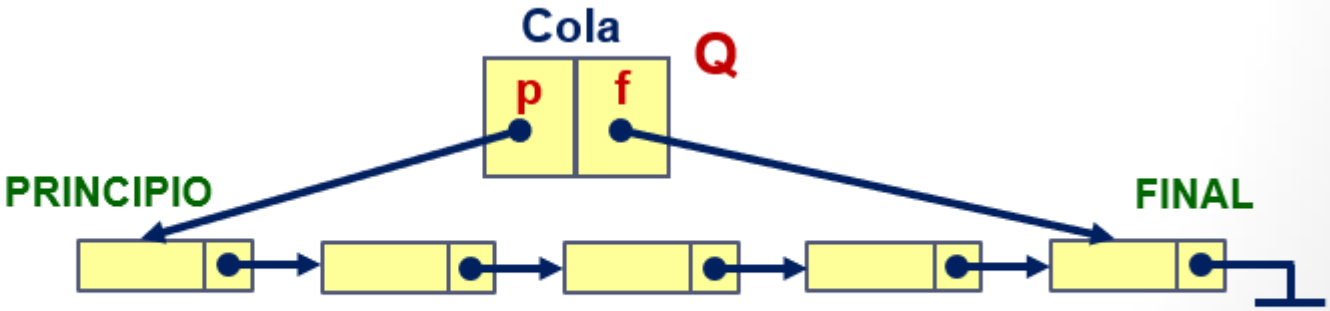
COLAS

```
using namespace std;

struct TipoDato {
    string elem;
    int aa;
};

class Nodo {
    private:
        TipoDato dato;
        shared_ptr<Nodo> next;
    public:
        Nodo():next(nullptr){}
        Nodo(TipoDato const &d, shared_ptr<Nodo> ptr):dato(d),next(ptr){}

        TipoDato getDato() const;
        void setDato(const TipoDato &newDato);
        shared_ptr<Nodo> getNext() const;
        void setNext(const shared_ptr<Nodo> &newNext);
};

class Cola {
    public:
        Cola():first(nullptr), end(nullptr){}

        bool empty() const;
        void push(const TipoDato &dato);
        void pop();
        TipoDato front() const;

    private:
        shared_ptr<Nodo> first, end;
};
```

Operación empty



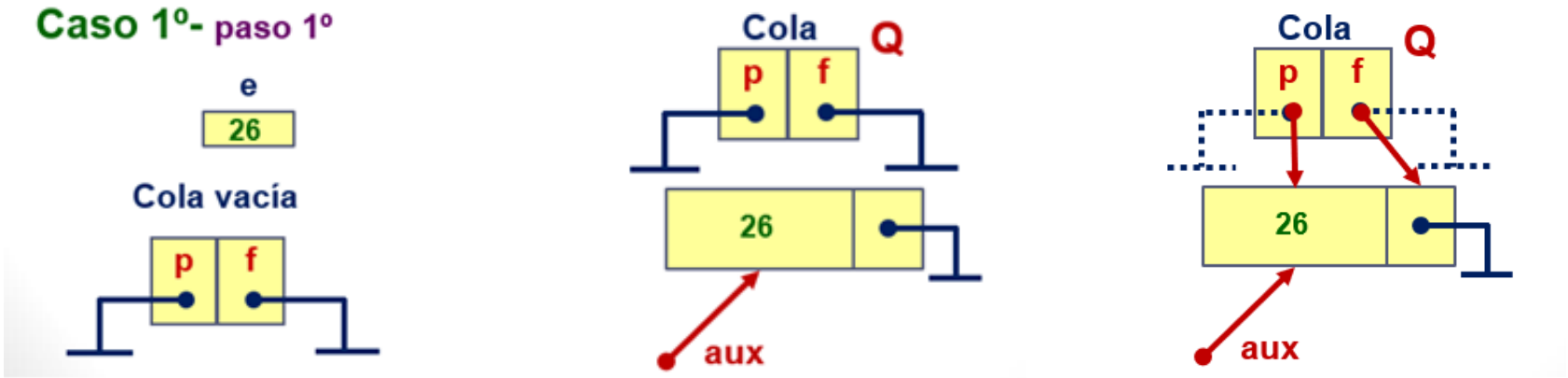
Verdadero



Falso

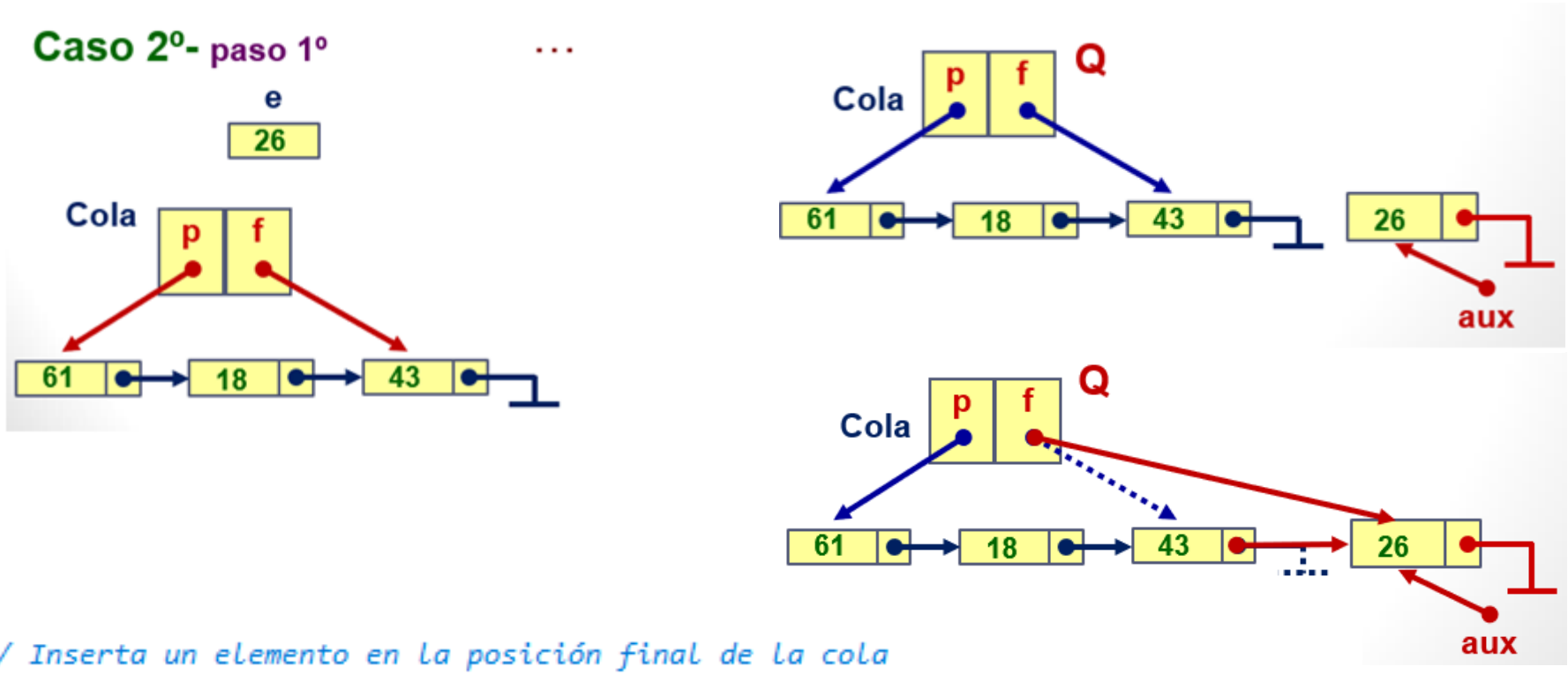
```
// Determina si la cola está vacía o no
bool Cola::empty() const  {
    return (first == nullptr && end == nullptr);
}
```

Operación push




```
// Inserta un elemento en la posición final de la cola
void Cola::push(const TipoDato& dato) {
    shared_ptr<Nodo> ptraux = make_shared<Nodo>(Nodo(dato, nullptr));

    if (this->empty()) {
        first = ptraux;
    }
    else  {
        end->setNext(ptraux);
    }
    end = ptraux;  // El puntero final debe señalar siempre al elemento incluido
}
```

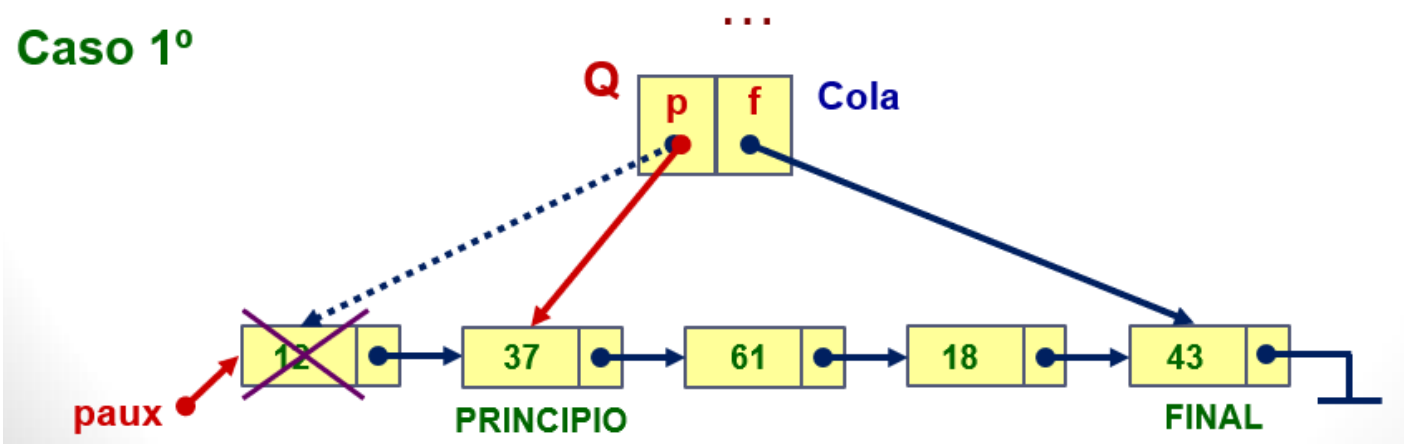
Operación push



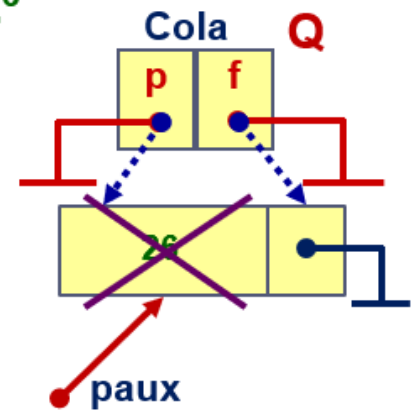

```
// Inserta un elemento en la posición final de la cola
void Cola::push(const TipoDato& dato) {
    shared_ptr<Nodo> ptraux = make_shared<Nodo>(Nodo(dato, nullptr));

    if (this->empty()) {
        first = ptraux;
    }
    else  {
        end->setNext(ptraux);
    }
    end = ptraux;  // El puntero final debe señalar siempre al elemento incluido
}
```

Operación pop

Caso 1º



Caso 2º




```
// Elimina un elemento de la primera posición de la cola
void Cola::pop()  {
    first = first->getNext();
    if (first == nullptr) {
        end = nullptr;  // La cola queda vacía
    }
}
```

Ejercicio: Implementar las funciones miembro de la clase Cola considerando el **TipoDato** un valor de tipo ***int***:

Queue();

bool **empty**() const;

void **push**(const TipoDato &**dato**);

void **pop**();

TipoDato **front**() const;

Escribir un programa en C++11, realizando las funciones necesarias, que permita generar y gestionar una **cola** de números enteros utilizando sólo las operaciones básicas de cola.

El programa deberá de mostrar un **menú** al usuario que le permita hacer los siguientes procesos:

1. **Introducir números enteros en la Cola:**

   Esta opción, ***introducirElemCola***, consistirá en solicitar un número entero al usuario y añadirlo a la Cola de números. Opcionalmente, se puede permitir añadir más valores si el usuario así lo indica.

2. **Mostrar el contenido de la Cola:**

   Esta opción, ***mostrarElemCola*****,** consistirá en mostrar por pantalla los números guardados en la Cola.

   Si la Cola está vacía se debe de mostrar un mensaje para indicárselo al usuario.

3. **Elemento mayor de la Cola:**

   Esta opción, ***elMayorElemCola***, consistirá en mostrar por pantalla un mensaje con el MAYOR de los números almacenados en la Cola.

   Para ello, realizar una función miembro de la clase Cola que determine el valor del elemento mayor de la Cola.

   Si la Cola está vacía se debe de mostrar un mensaje para indicárselo al usuario.

4. **Buscar un número en la Cola:**

   Esta opción, ***buscarElemCola***, consistirá en comprobar si un determinado valor solicitado al usuario está guardado en la Cola.

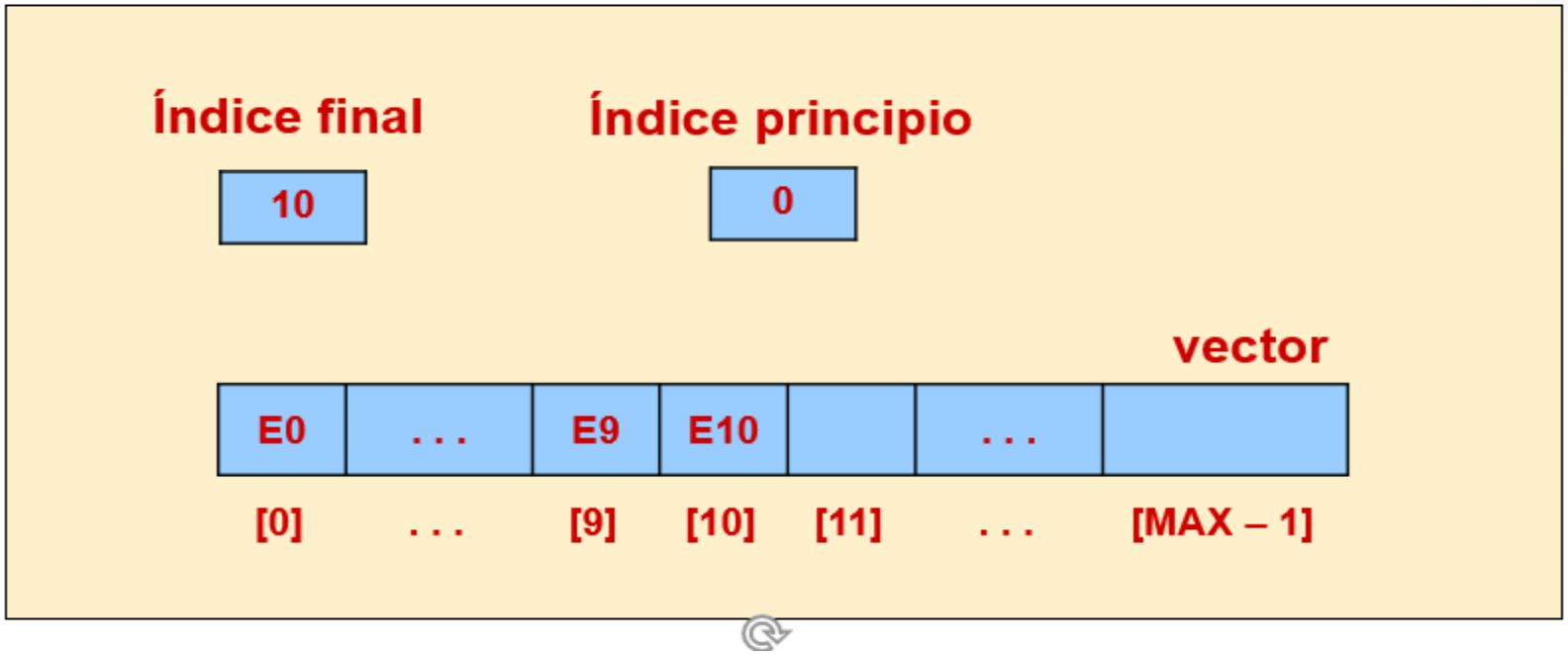
   Si la Cola está vacía se debe de mostrar un mensaje para indicárselo al usuario.

5. **Número de veces que está un número en la Cola:**

   Esta opción, ***vecesElemCola***, consistirá en determinar las veces que un determinado valor solicitado al usuario está en la Cola.

   Si la Cola está vacía se debe de mostrar un mensaje para indicárselo al usuario.

Cola

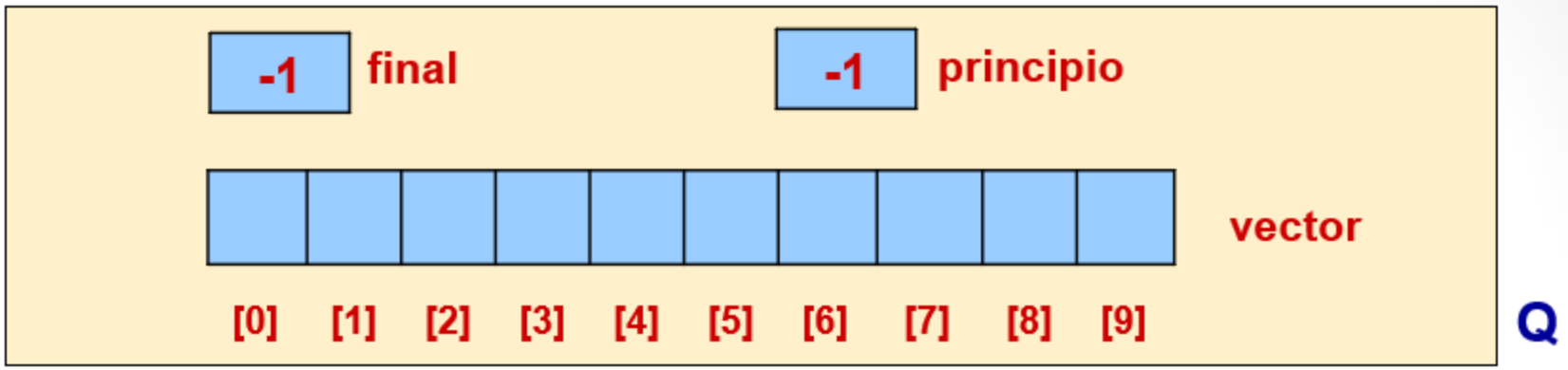


Índice principio, contiene el índice correspondiente a la **posición anterior del elemento más antiguo de la cola**.

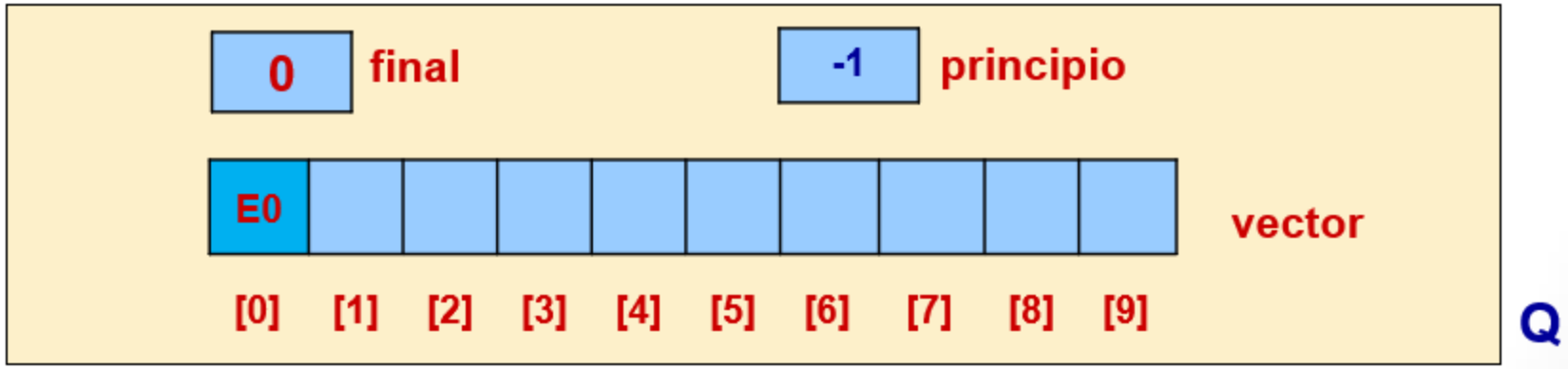
Índice final, contiene el índice correspondiente a la **posición del elemento añadido más recientemente**.

Cola vacía, principio = final = -1

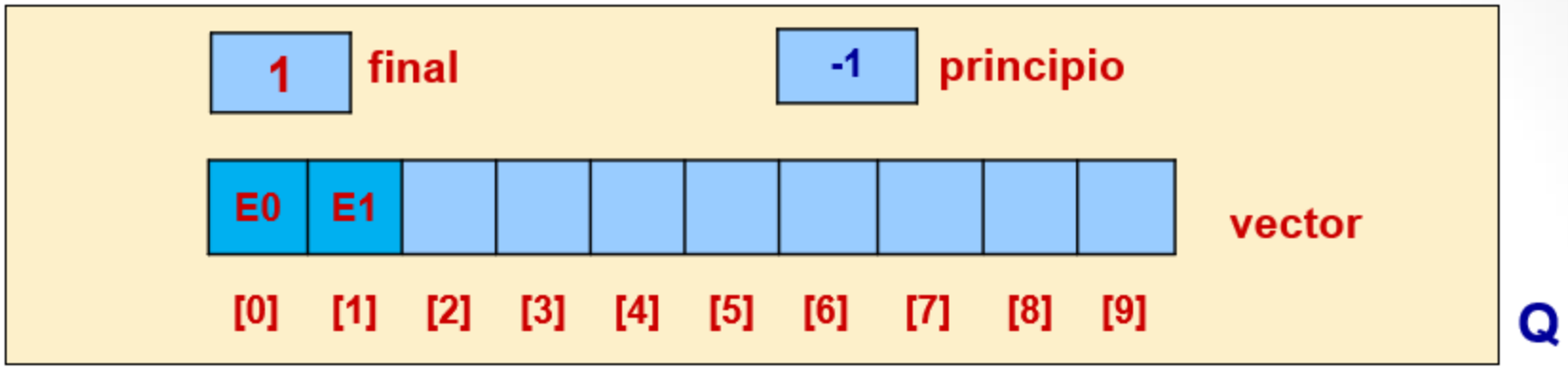
ColaVacia (Q)



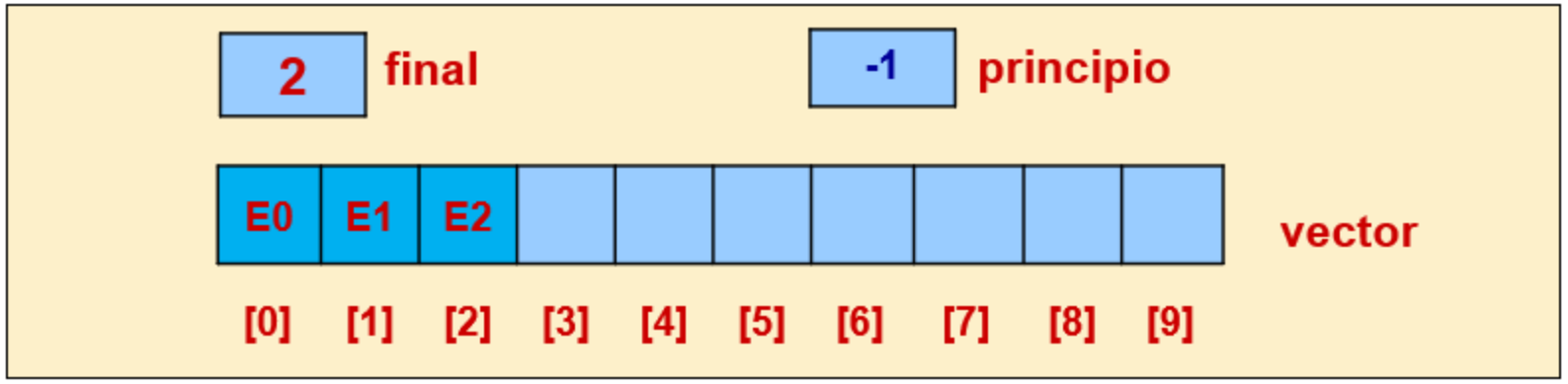
AniadirElemCola (Q, E0)



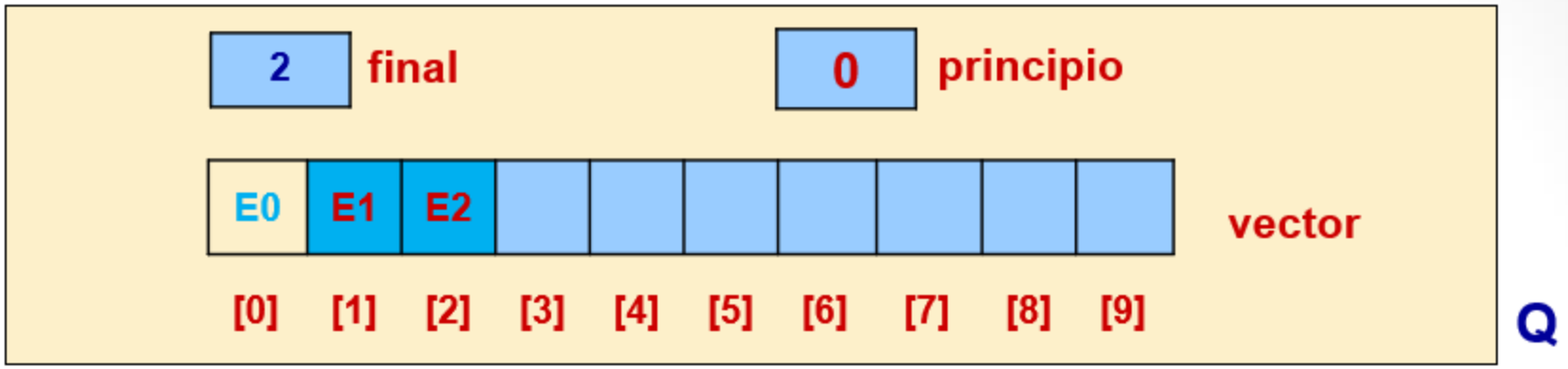
AniadirElemCola (Q, E1)



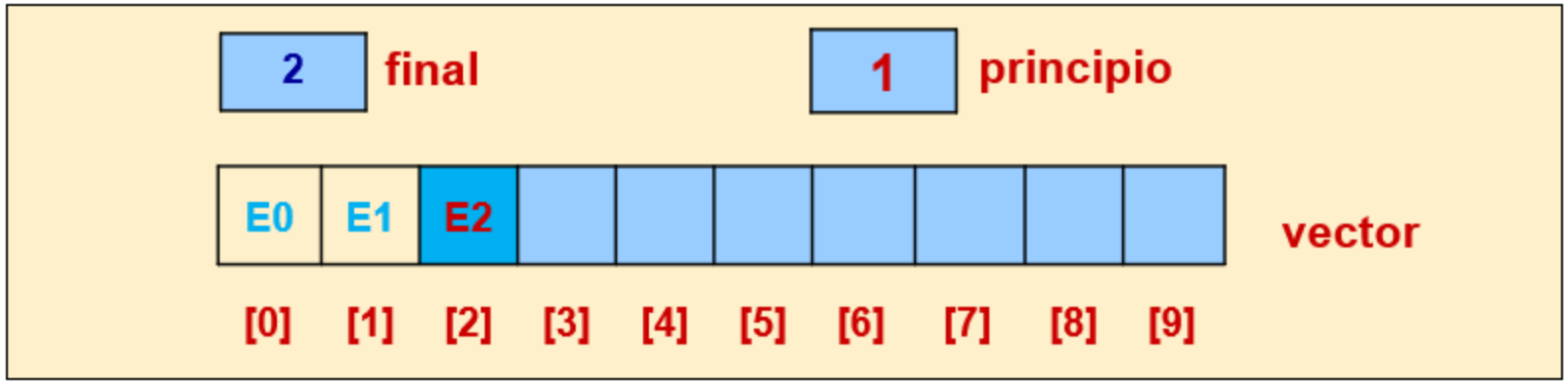
AniadirElemCola (Q, E2)



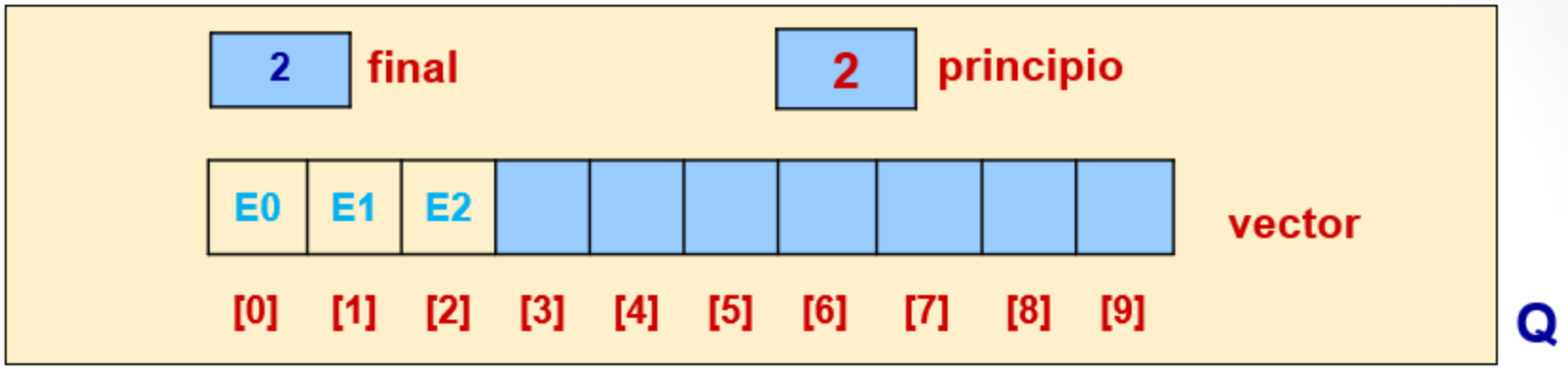
EliminarElemCola (Q)



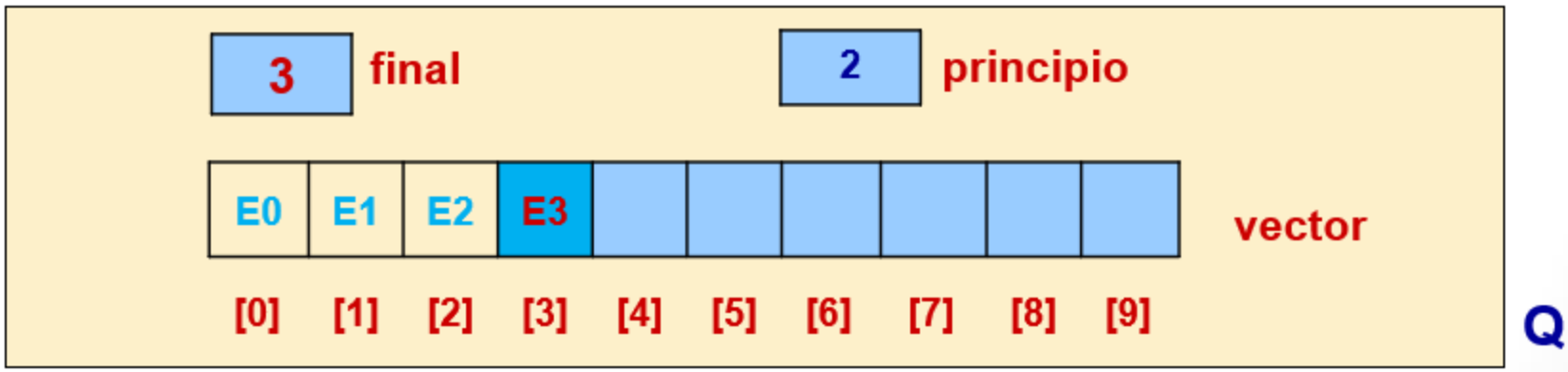
EliminarElemCola (Q)



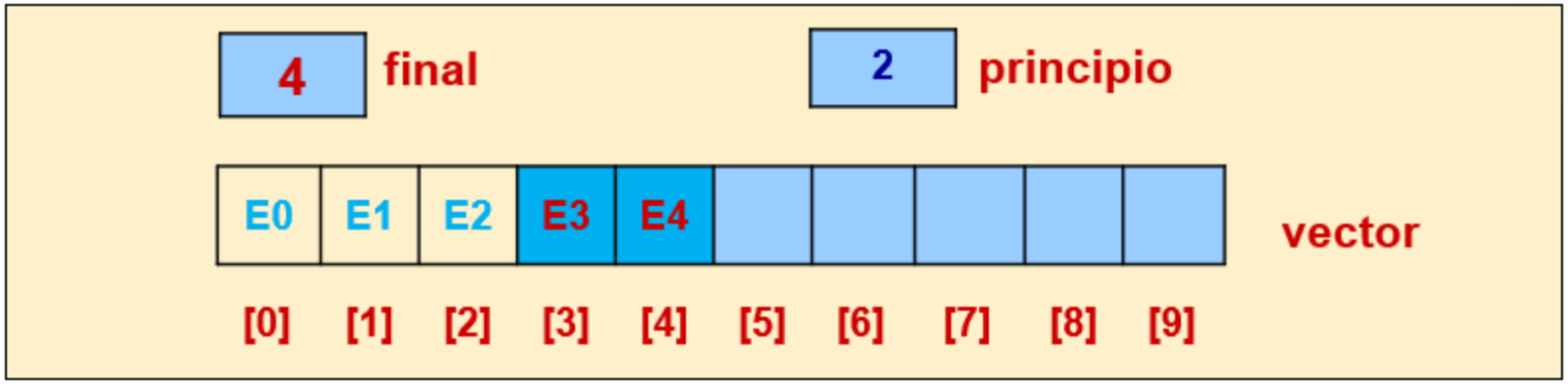
EliminarElemCola (Q)



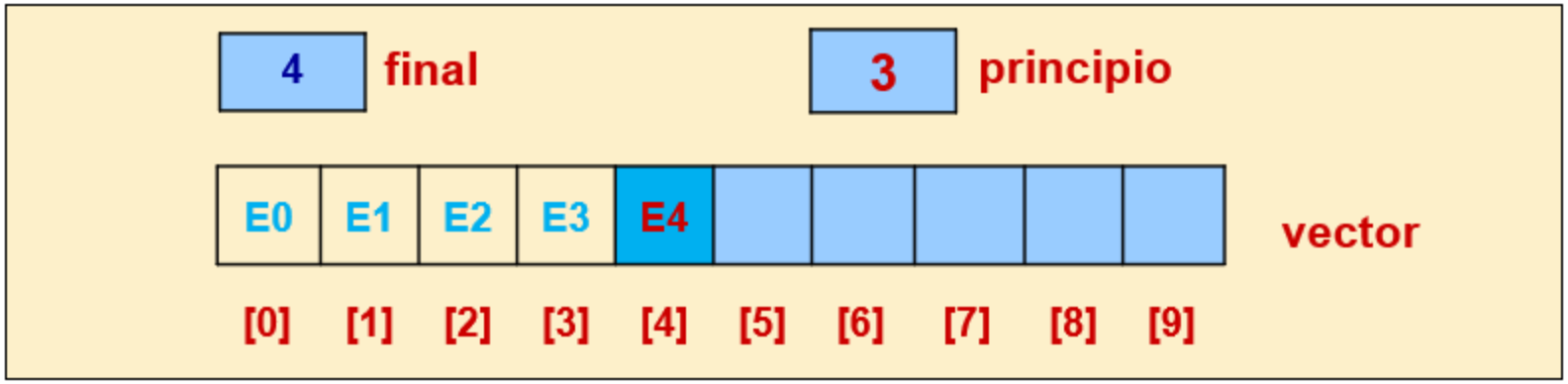
AniadirElemCola (Q, E3)



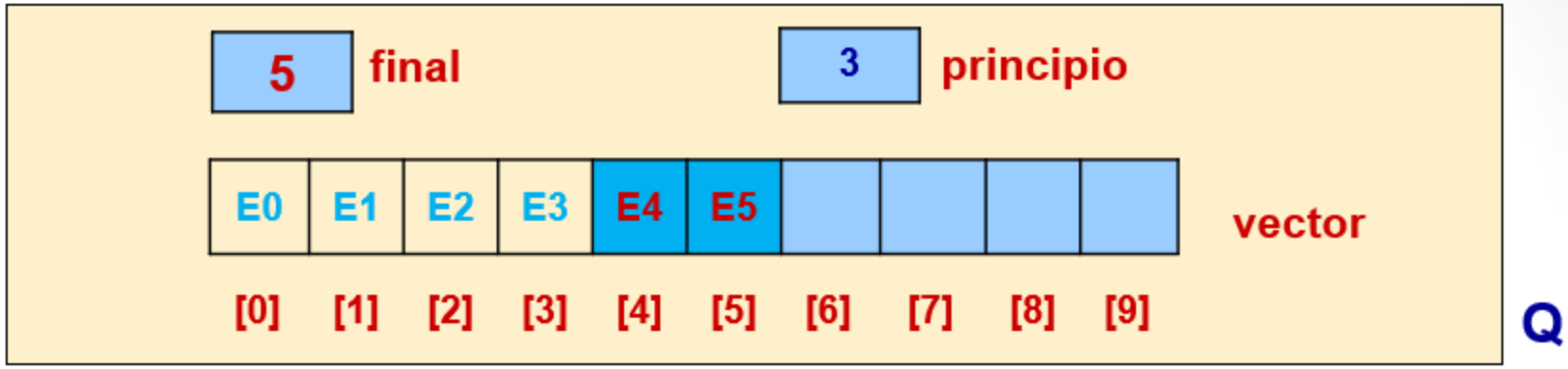
AniadirElemCola (Q, E4)



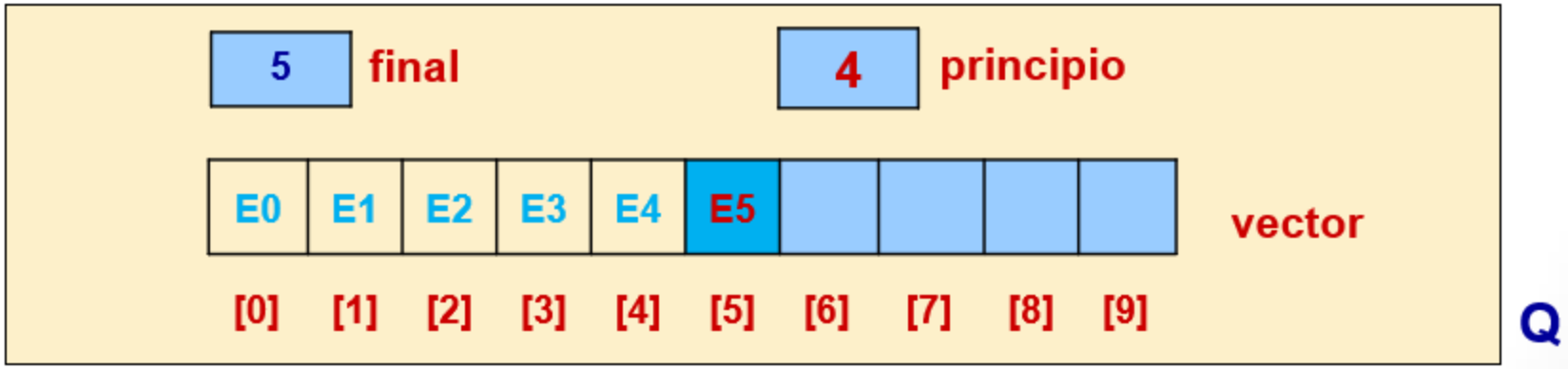
EliminarElemCola (Q)



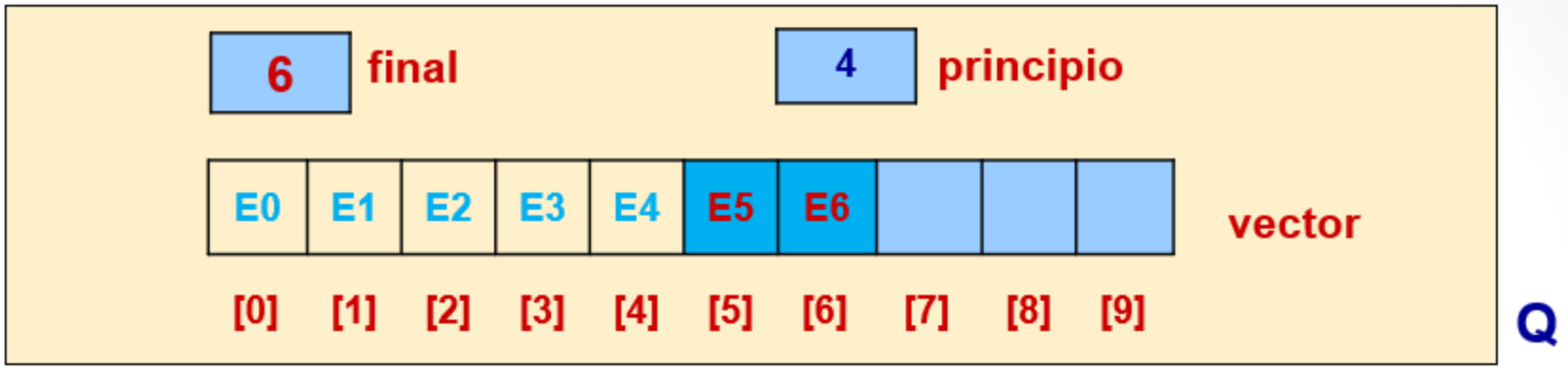
AniadirElemCola (Q, E5)



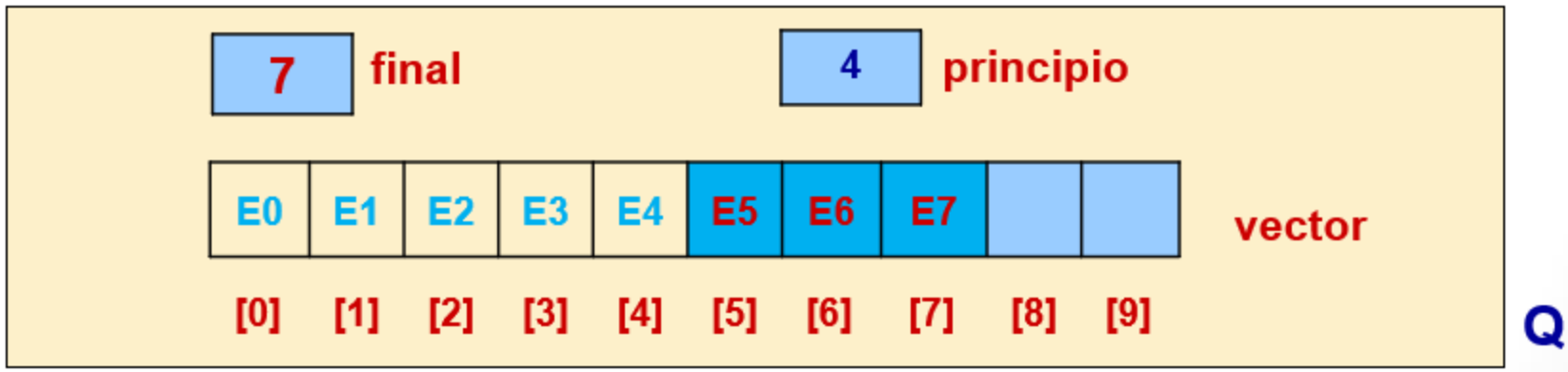
EliminarElemCola (Q)



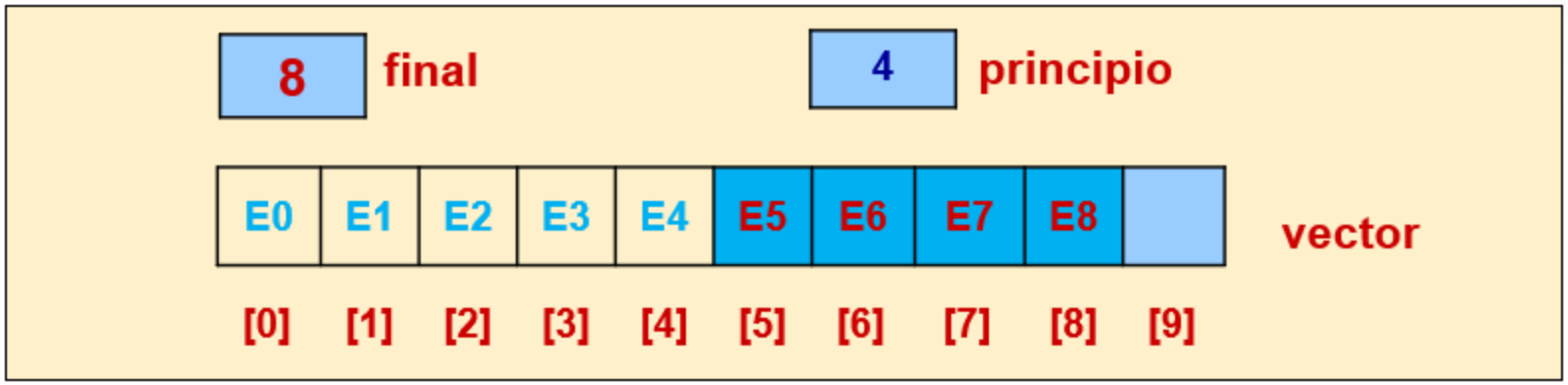
AniadirElemCola (Q, E6)



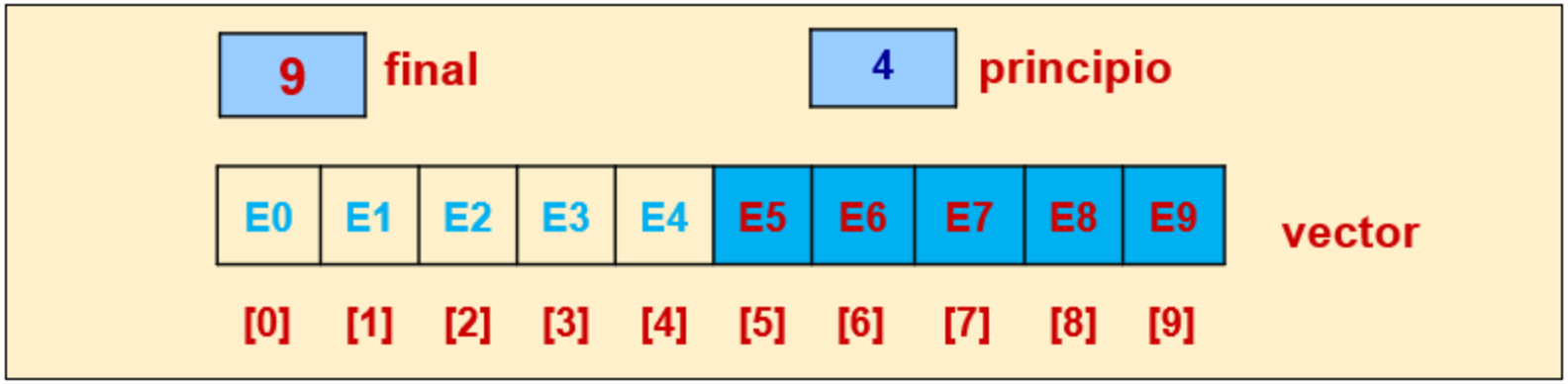
AniadirElemCola (Q, E7)

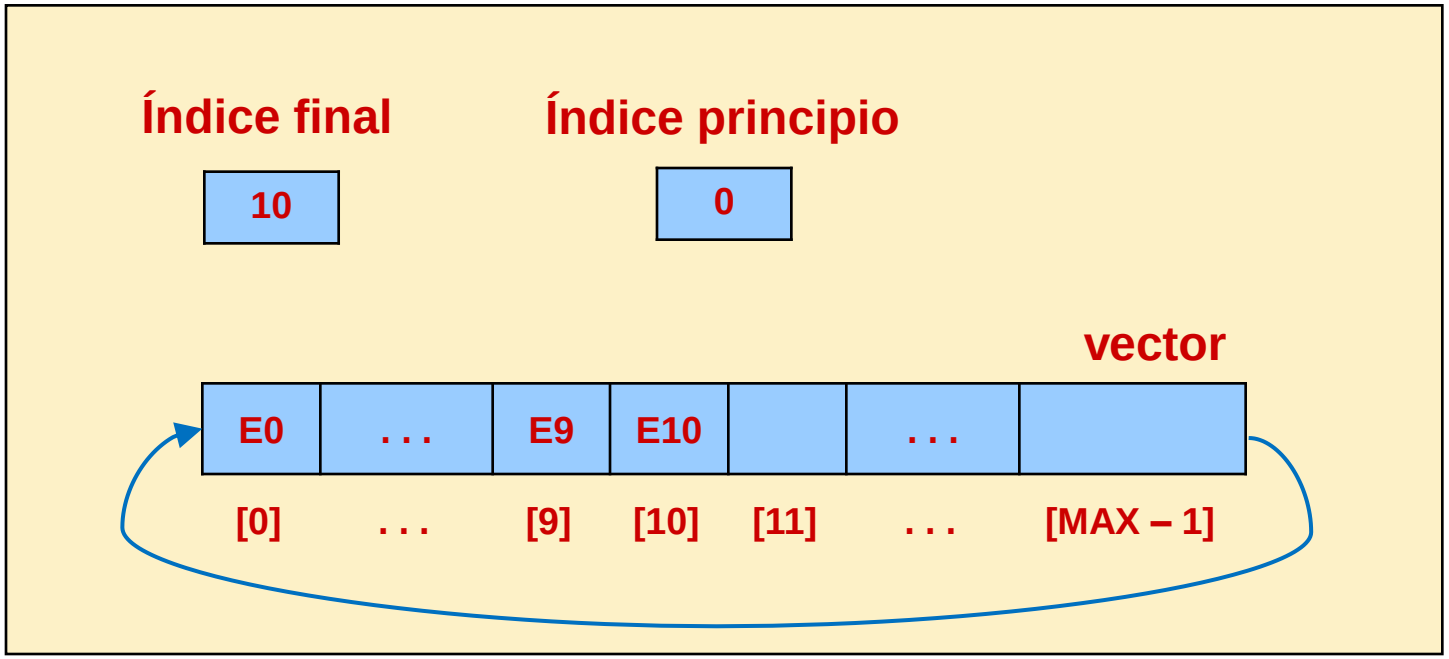


AniadirElemCola (Q, E8)



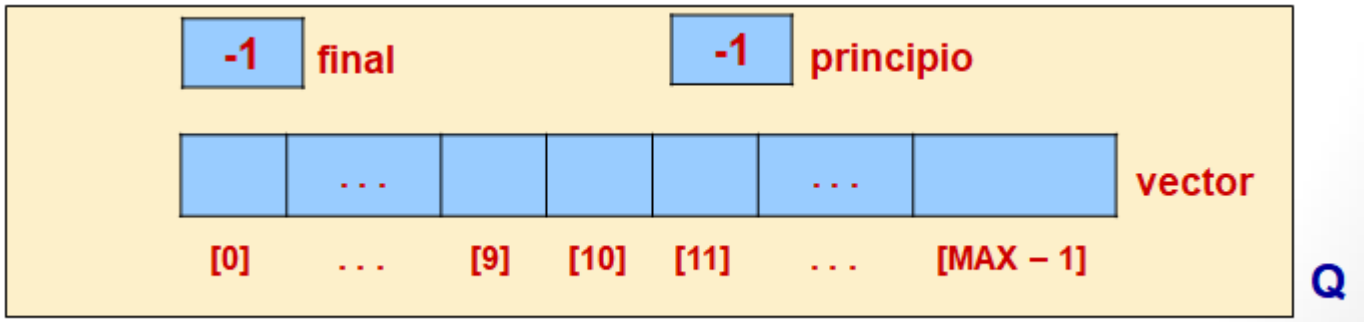
AniadirElemCola (Q, E9)





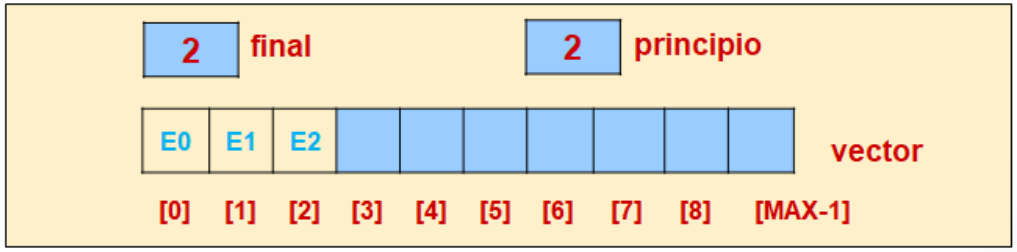
- **La posición siguiente a la última del vector será la primera posición del vector**.
- No siempre la siguiente posición a una dada, **i**, será la posición **i+1**. Habrá que implementar una función, **PosSiguiente**, para determinar la posición correcta.

Constructor Cola()

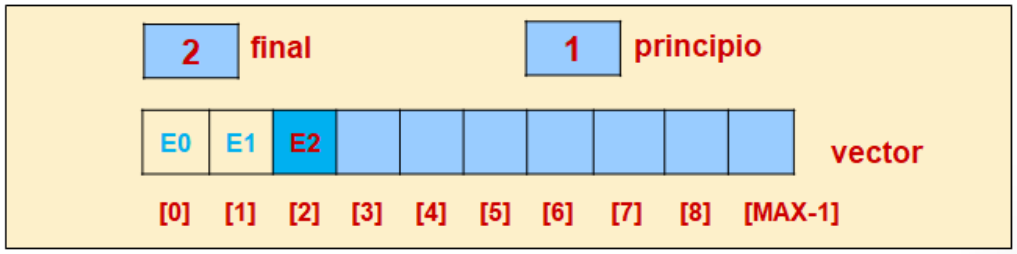


Operación empty

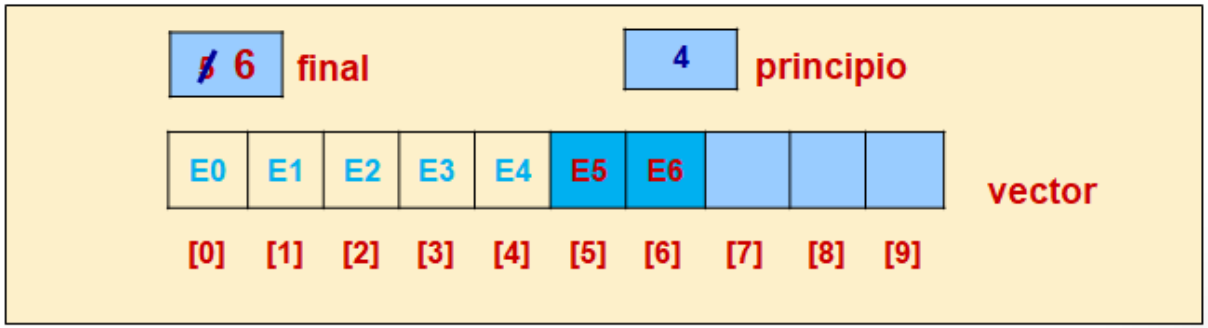
Verdadero



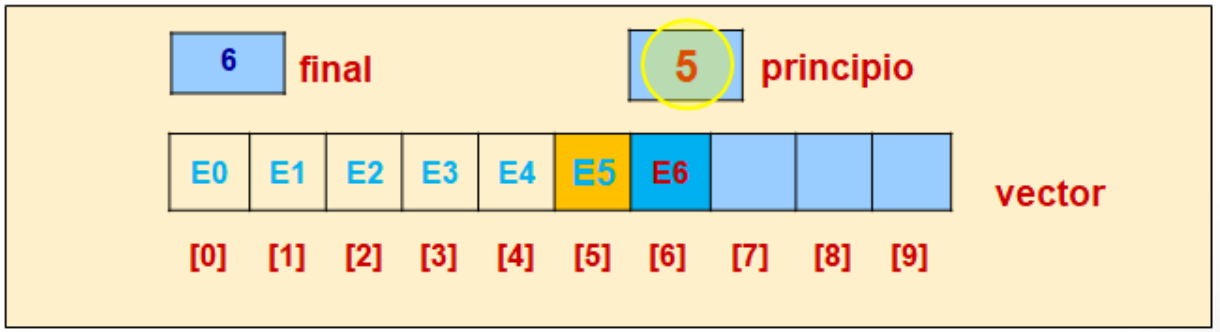
Falso



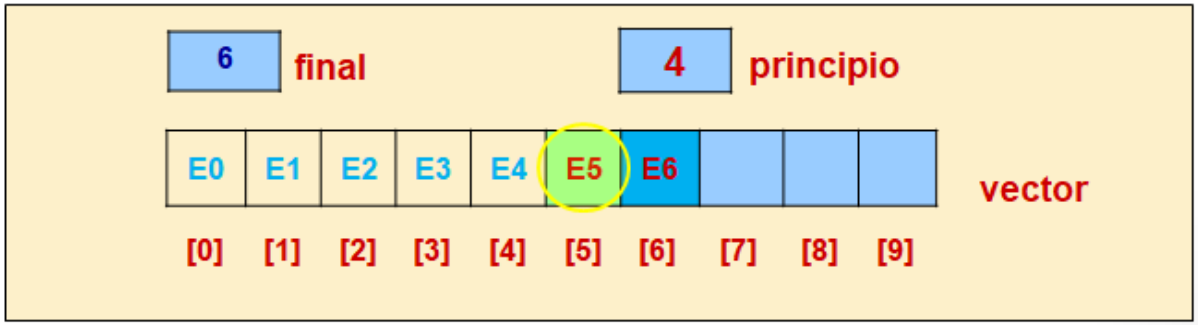
Operación push

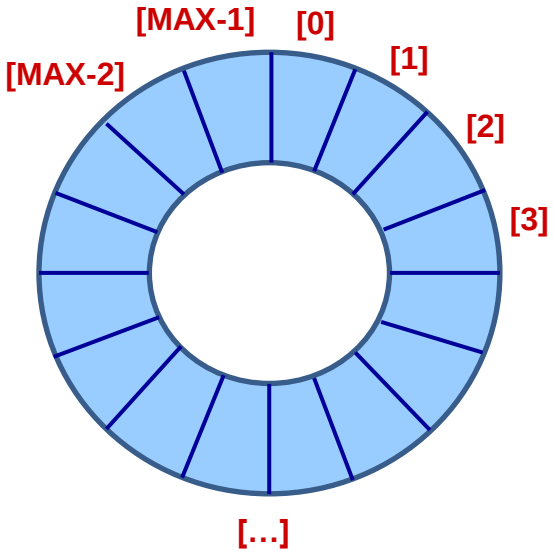


Operación pop



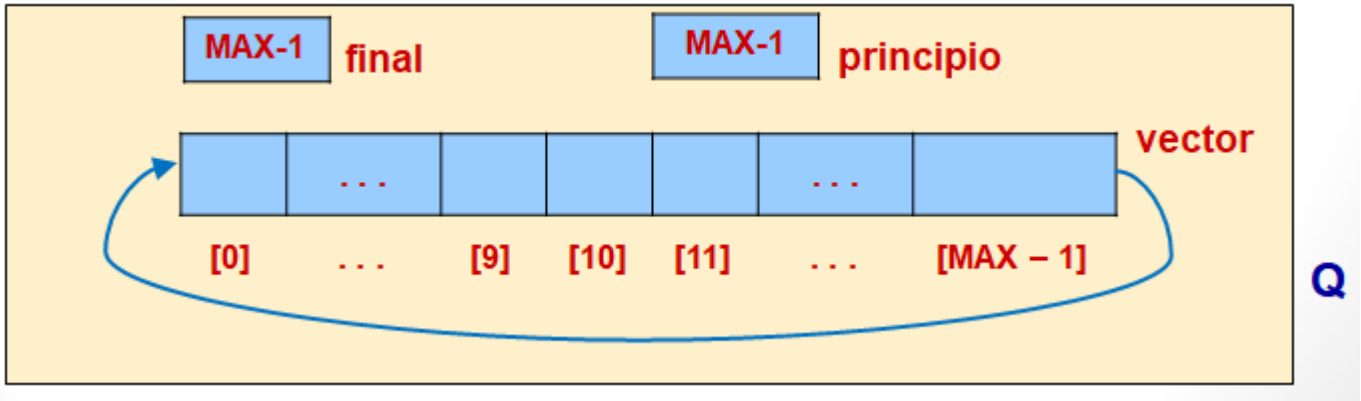
Operación front



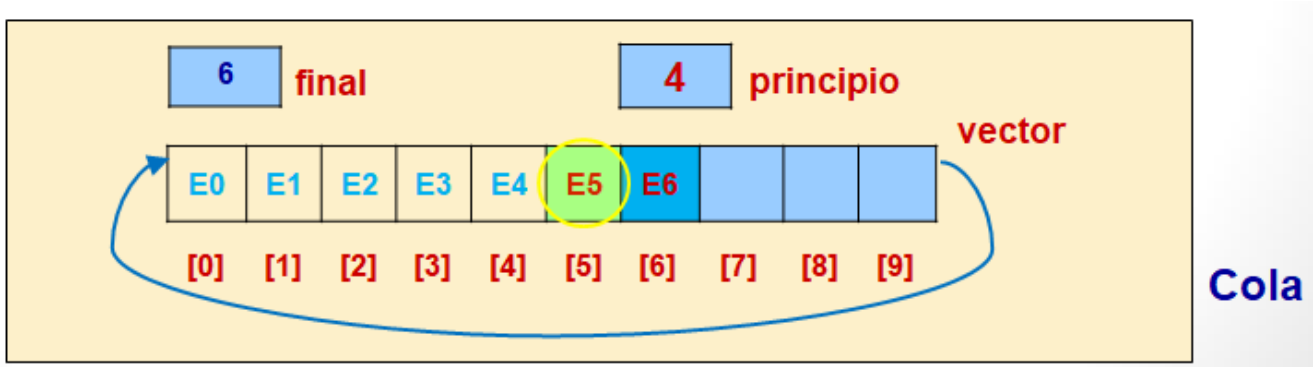


int **PosSiguiente** (int pos)

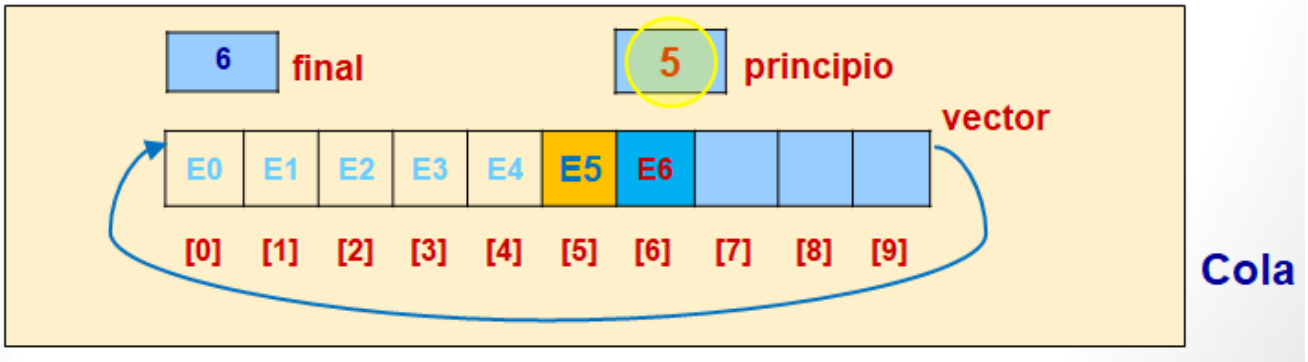
Constructor Cola()



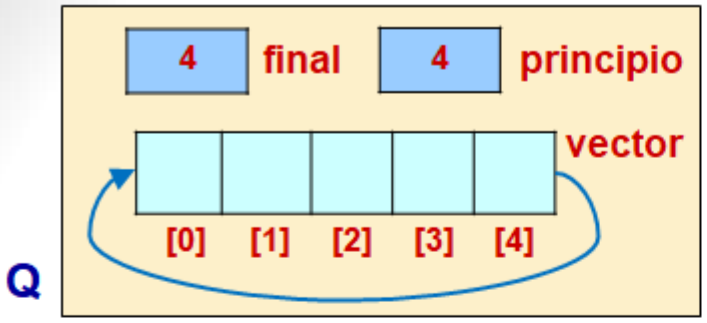
Operación front



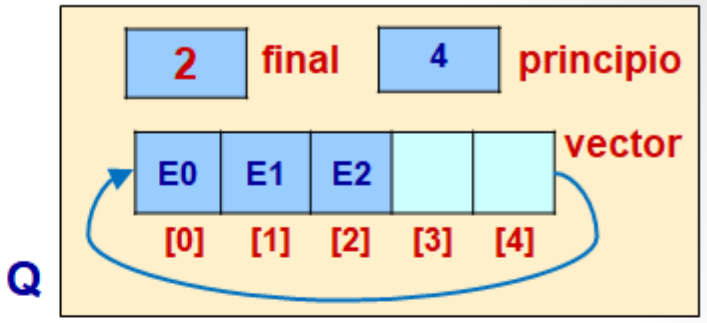
Operación pop



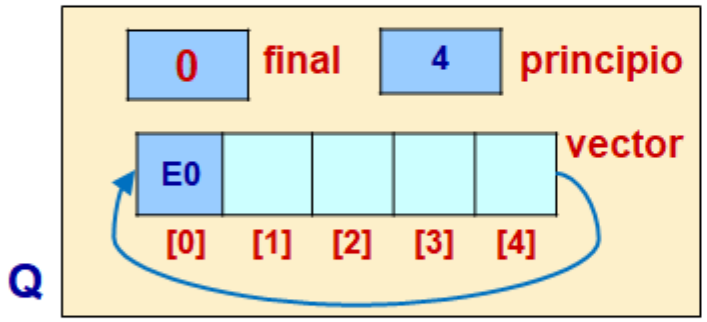
ColaVacia (Q)



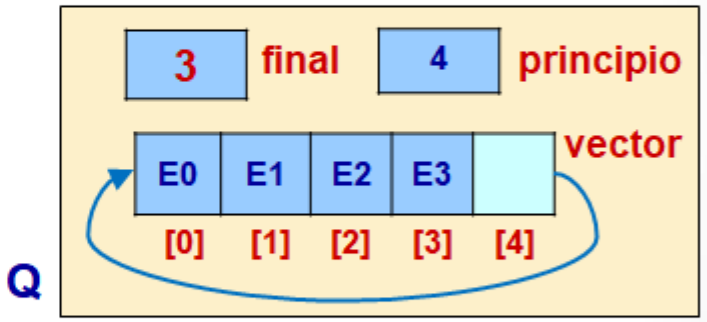
AniadirElemCola (Q, E2)



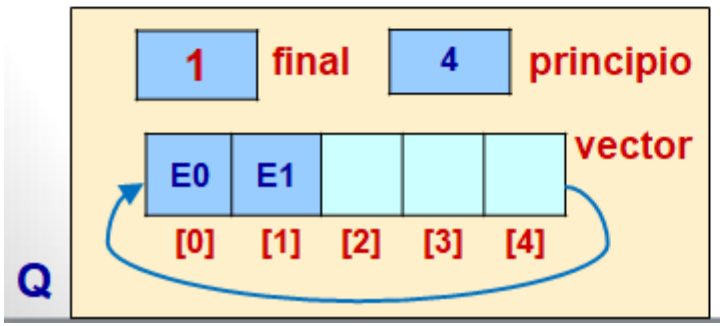
AniadirElemCola (Q, E0)



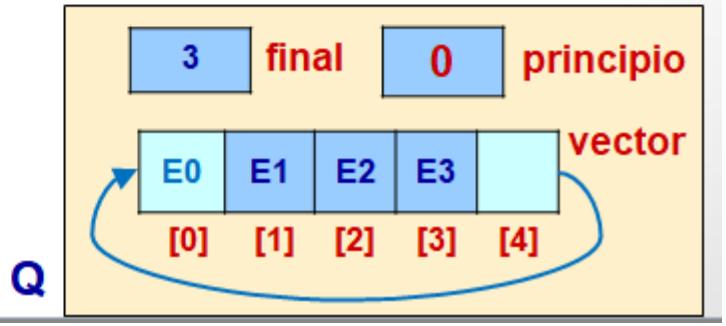
AniadirElemCola (Q, E3)

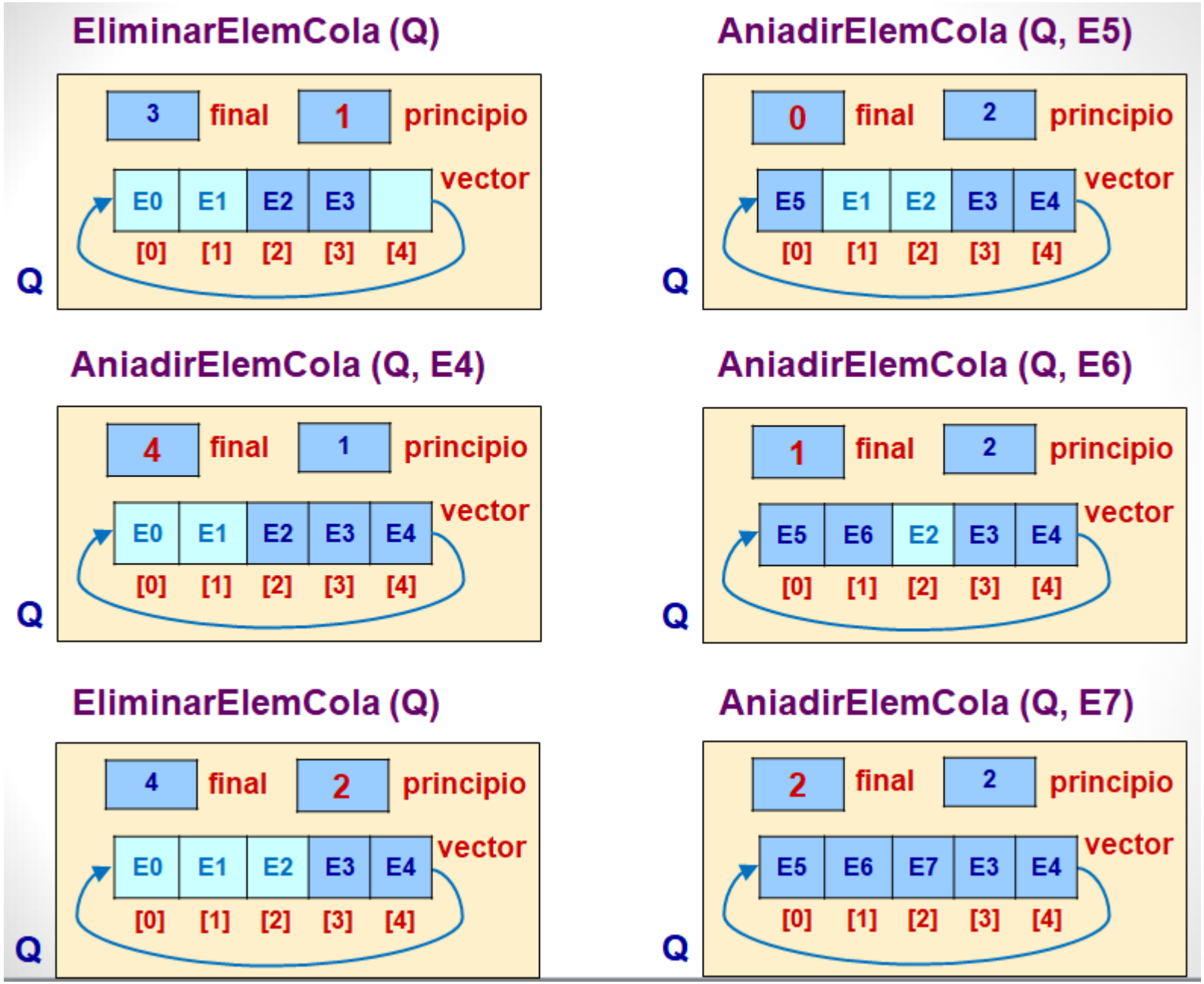


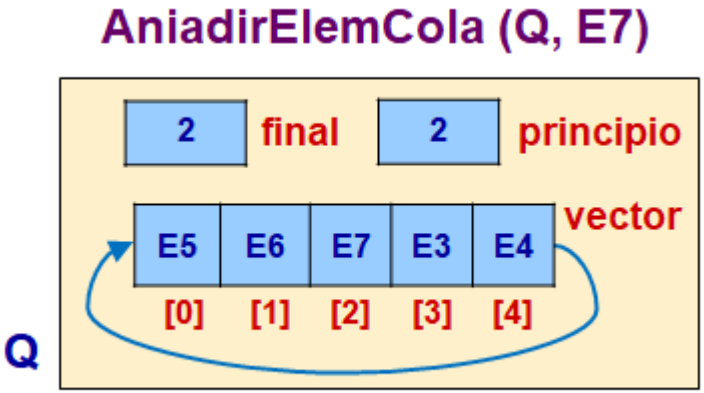
AniadirElemCola (Q, E1)



EliminarElemCola (Q)




EliminarElemCola (Q)
3
final
1
principio
vector
E0
E1
E2
E3
[0]
[1]
[2]
[3]
[4]
Q
AniadirElemCola (Q, E5)
0
final
2
principio
vector
E5
E1
E2
E3
E4
[0]
[1]
[2]
[3]
[4]
Q
AniadirElemCola (Q, E4)
4
final
1
principio
vector
E0
E1
E2
E3
E4
[0]
[1]
[2]
[3]
[4]
Q
AniadirElemCola (Q, E6)
1
final
2
principio
vector
E5
E6
E2
E3
E4
[0]
[1]
[2]
[3]
[4]
Q
EliminarElemCola (Q)
4
final
2
principio
vector
E0
E1
E2
E3
E4
[0]
[1]
[2]
[3]
[4]
Q
AniadirElemCola (Q, E7)
2
final
2
principio
vector
E5
E6
E7
E3
E4
[0]
[1]
[2]
[3]
[4]
Q



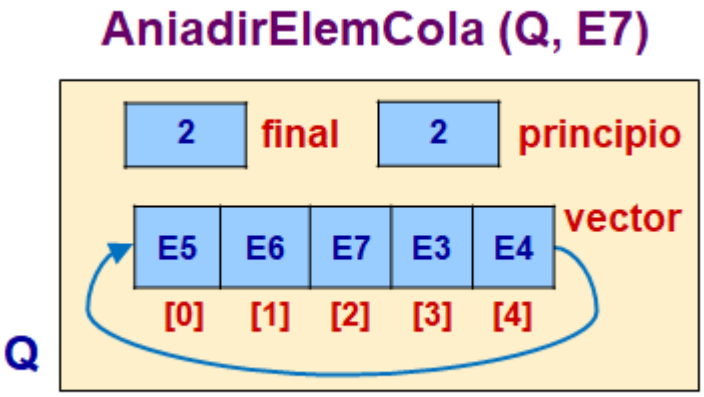
AMBIGÜEDAD!!

- La figura muestra que la Cola está **llena**:

Q.principio == Q.final

- Condición definida para determinar que la Cola está **vacía**:

Q.principio == Q.final



AMBIGÜEDAD!!

- La figura muestra que la Cola está **llena:**

  Q.principio == Q.final

- Condición definida para determinar que la Cola está **vacía**:

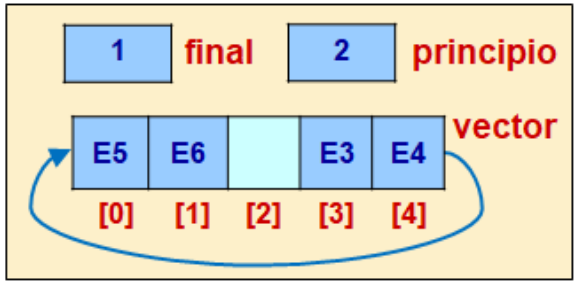
  Q.principio == Q.final

SOLUCIÓN

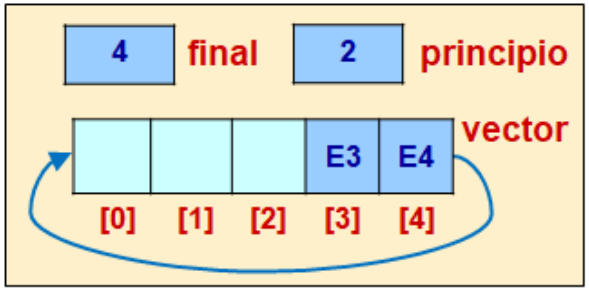
- **Incluir un contador de los elementos en el vector**: Modificar la estructura Tcola y la implementación de las operaciones.
- **Dejar una posición libre en el vector**: La posición que se deja disponible es la señalada por el campo **principio** (posición anterior al elemento más antiguo en la cola).

Operación colaLlena

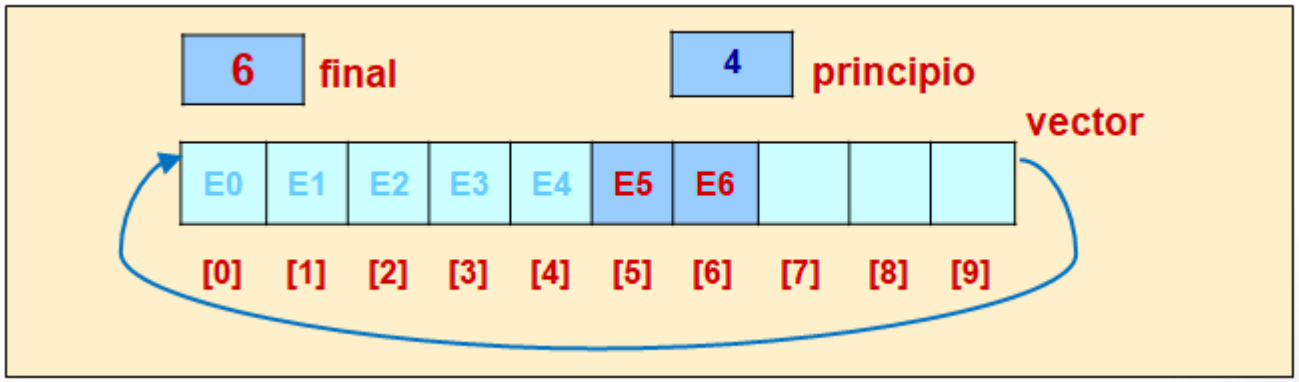
Verdadero



Falso



Operación push



```
#include <iostream>
#include <vector>

using namespace std;

struct TipoDato {
            string elem;
            int aa;
        };

class Cola {
    public:
        Cola():first(-1), end(-1){}
        Cola(int dim):first(dim-1), end(dim-1){datos.resize(dim);}

        bool empty() const;
        bool full();
        int posSiguiente (int p);
        void push(TipoDato const &e);
        void pop();
        TipoDato front();


    private:
        int first, end;
        vector<TipoDato> datos;
};
```

```
// Inserta un elemento en la cola
void Cola::push(TipoDato const &e) {
    end = posSiguiente(end);
    datos.at(end)=e;
}


// Elimina un elemento en la cola
void Cola::pop()  {
    first = posSiguiente(first);
}


// Devuelve el primer elemento de la cola
TipoDato Cola::front()  {
    return datos.at(posSiguiente(first));
}
```

```
// Determina si la cola está vacía o no
bool Cola::empty() const  {
    return (first == end);
}

// Determina si la cola está llena o no
bool Cola::full()   {
    return (posSiguiente(end) == first);
}


int Cola::posSiguiente (int p)
{   int size = datos.size();
    if (p < size-1) {
        return p=p+1;
    }
    else {
        return 0;
    }
}
```