Grado en Ingeniería Información

Estructura de Datos y Algoritmos

Sesión 8

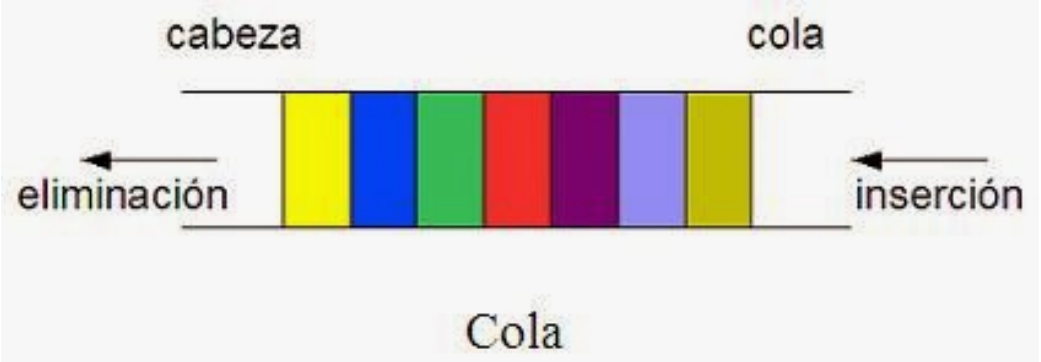
Curso 2023-2024

Marta N. Gómez

UNIVERSIDAD NEBRIJA

T3. Tipos Abstractos de Datos (TAD)

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

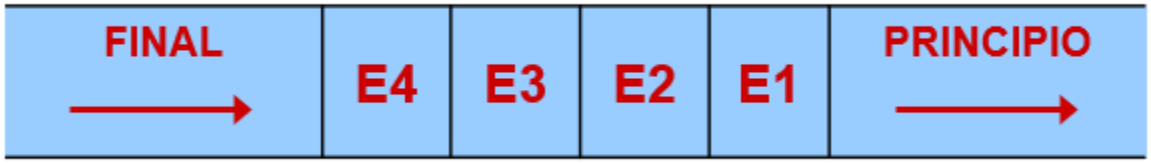


UNIVERSIDAD NEBRIJA

ÍNDICE

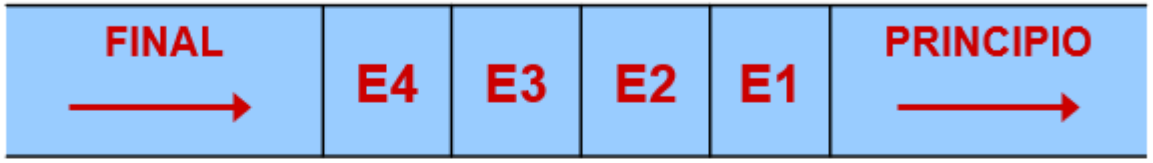
Una **COLA** es un **conjunto ordenado de elementos homogéneos**, en el cual los elementos se eliminan por uno de sus extremos, ***Principio*** o ***Cabeza***, y se añaden por el otro extremo, ***Final***. Su funcionamiento sigue una política ***FIFO***.

Es una estructura de datos **Lineal.**



COLAS

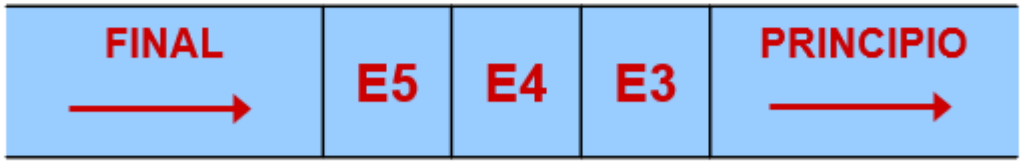
Añadir:



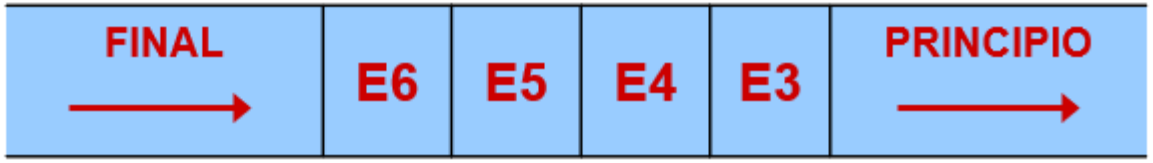
Añadir:

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.

frirst: **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

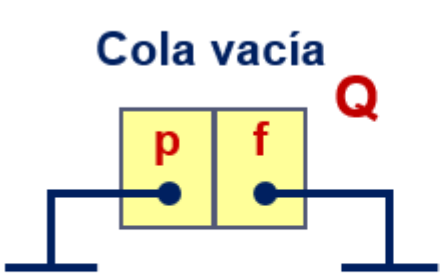
Postcondición: Elimina de la cola q el elemento que ocupa la primera posición de la cola.

Por tanto, la cola q se modifica.

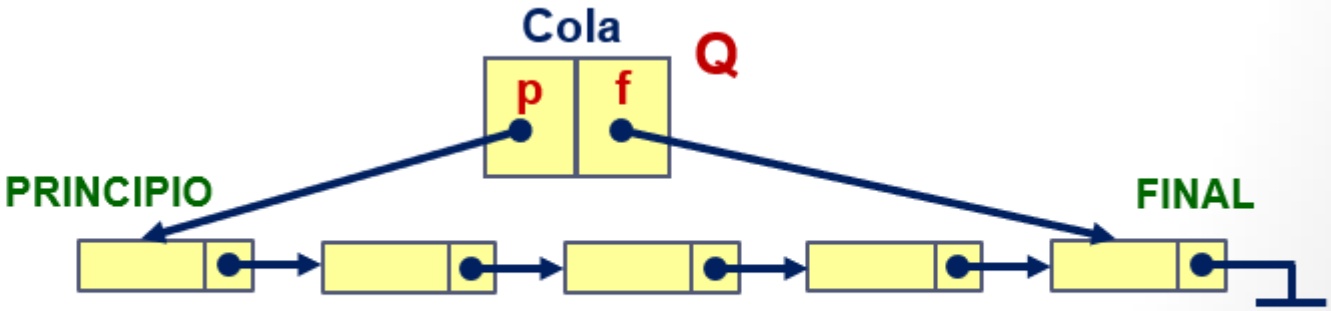
```
struct TipoDato {
    string elem;
    int aa;
};

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():front(nullptr), end(nullptr){}

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

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

Operación empty



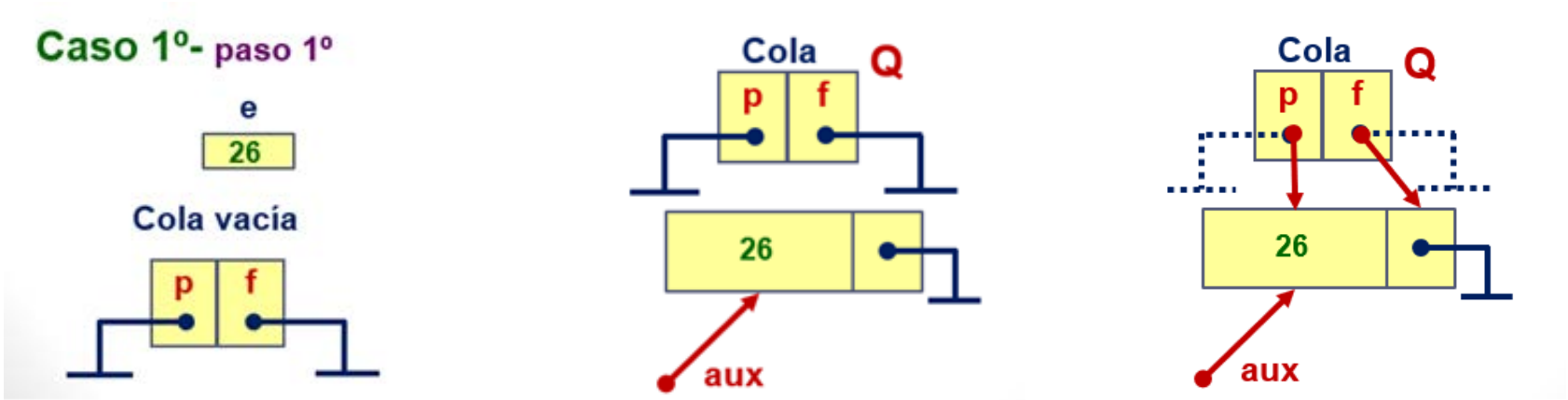
Verdadero



Falso

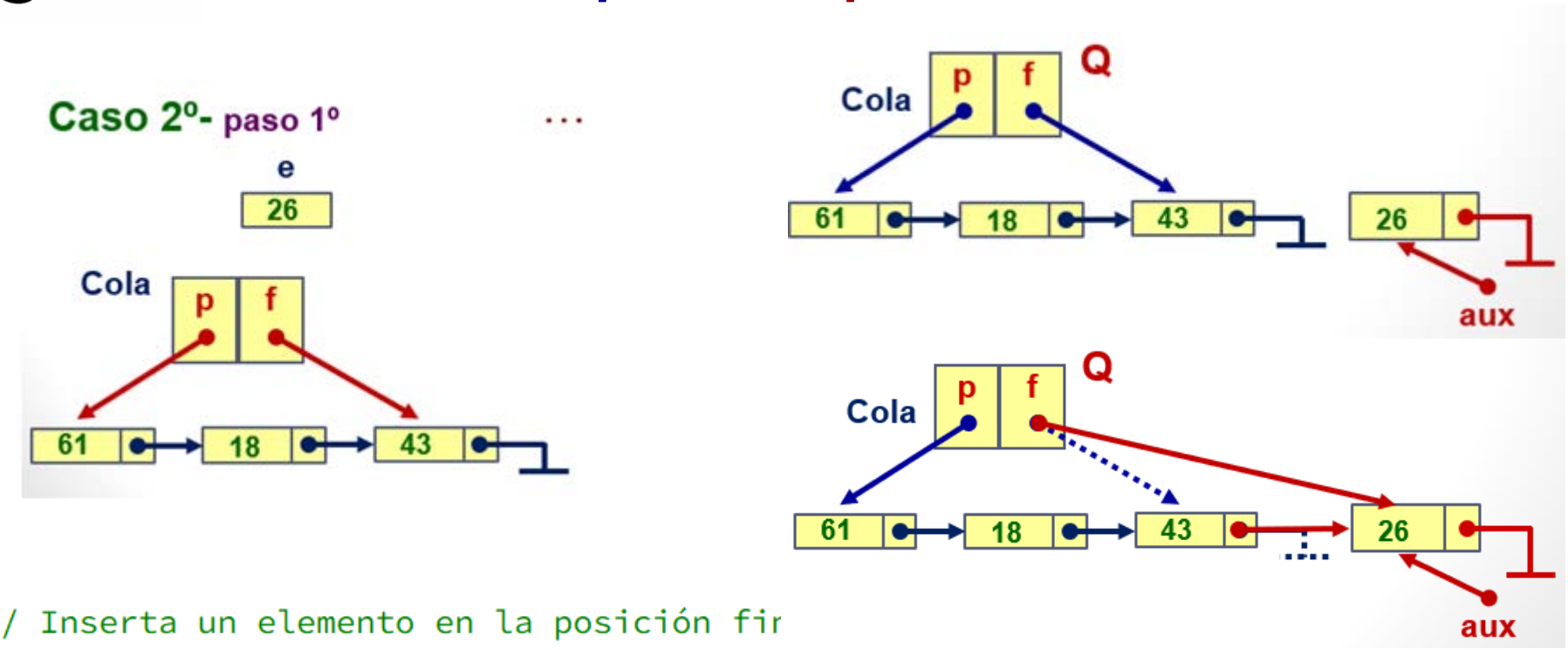
```
// Determina si la cola está vacía o no
bool Cola::empty() const  {
    return (front == nullptr && end == nullptr);
}
```

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()) {
        front = ptraux;
    }
    else  {
        end->setNext(ptraux);
    }
    // El puntero final de la cola debe señalar siempre al elemento incluido
    end = ptraux;
}
```

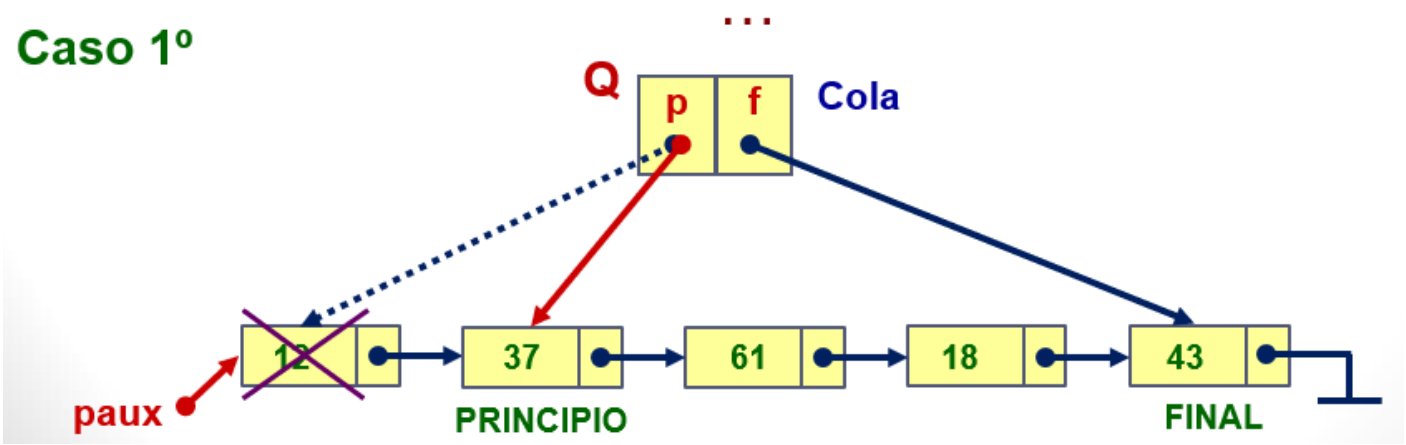
Operación push




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

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

Operación pop

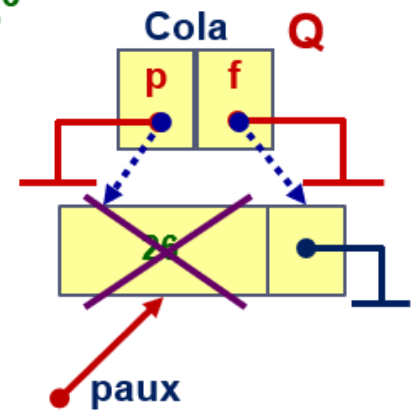
Caso 1º




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

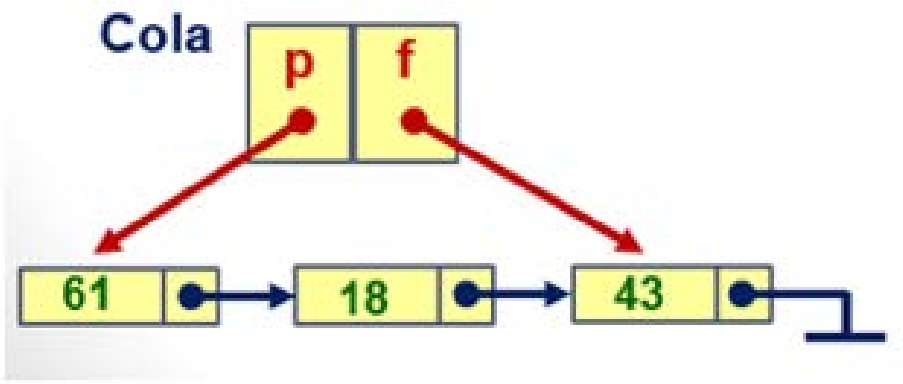
Operación pop

Caso 2º




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

Operación first




```
// Devuelve el elemento de la primera posición de la cola
TipoDato Cola::first() const {
    return (front->getDato());
}
```

```
int main() {
    Cola libros;

    TipoDato infor{"Cinco horas con Mario", 1966};

    libros.push(infor);

    infor.elem = "Don Quijote de la Mancha";
    infor.aa = 1605;
    libros.push(infor);

    infor.elem = "Platero y yo";
    infor.aa = 1914;
    libros.push(infor);

    while (!libros.empty())  {
        cout << "\n\n\tSiguiente libro para leer es: " << libros.first().elem
             << " " << libros.first().aa << endl;
        libros.pop();  // Eliminamos el elemento cima

        if (libros.empty())  {
            cout << "\n\n\tLa cola esta vacia y no contiene mas libros.\n\n";
        }
    }

    return 0;
}
```

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

Cola();

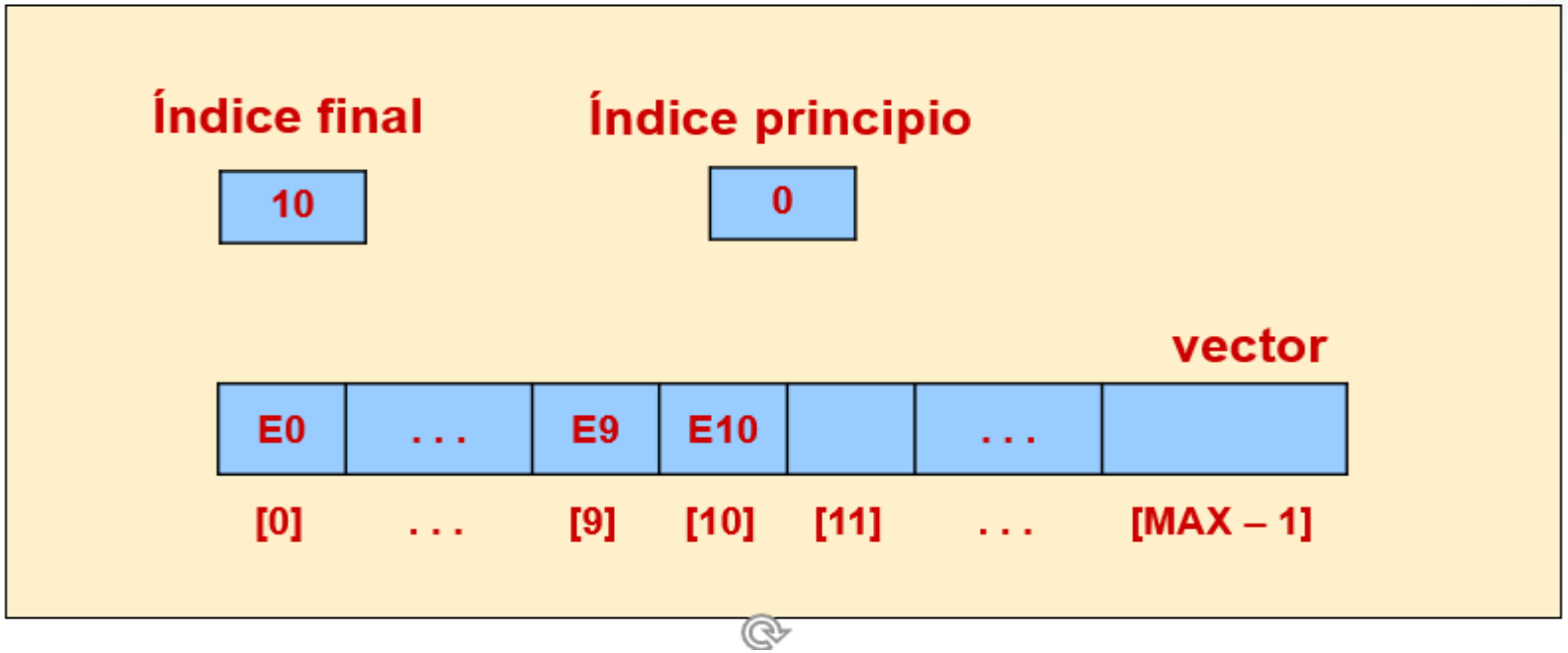
bool **empty**() const;

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

void **pop**();

TipoDato **first**() const;

Cola

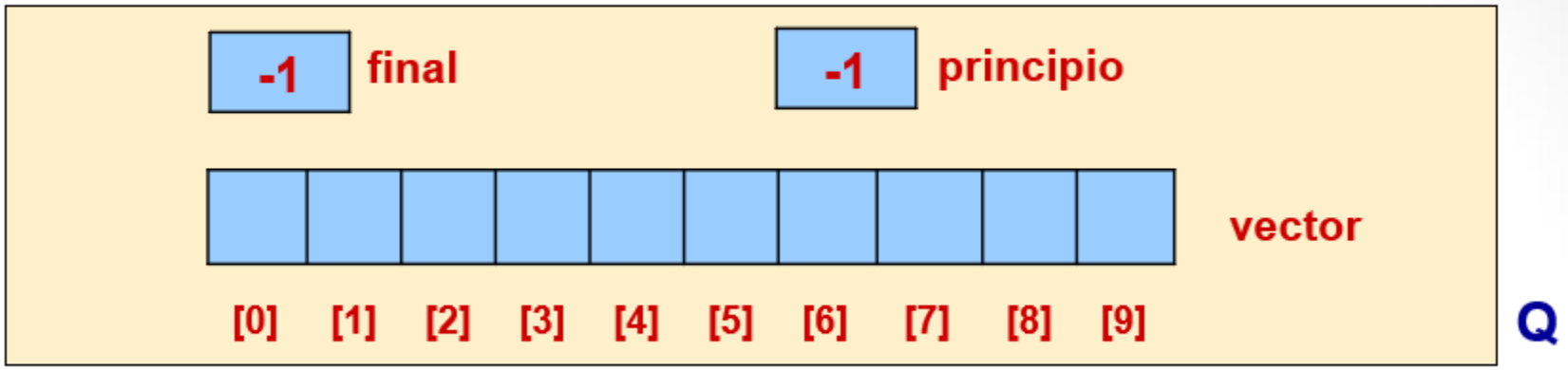


Índice principio, contiene el índice correspondiente a la **posición anterior del elemento más antiguo de la cola**.

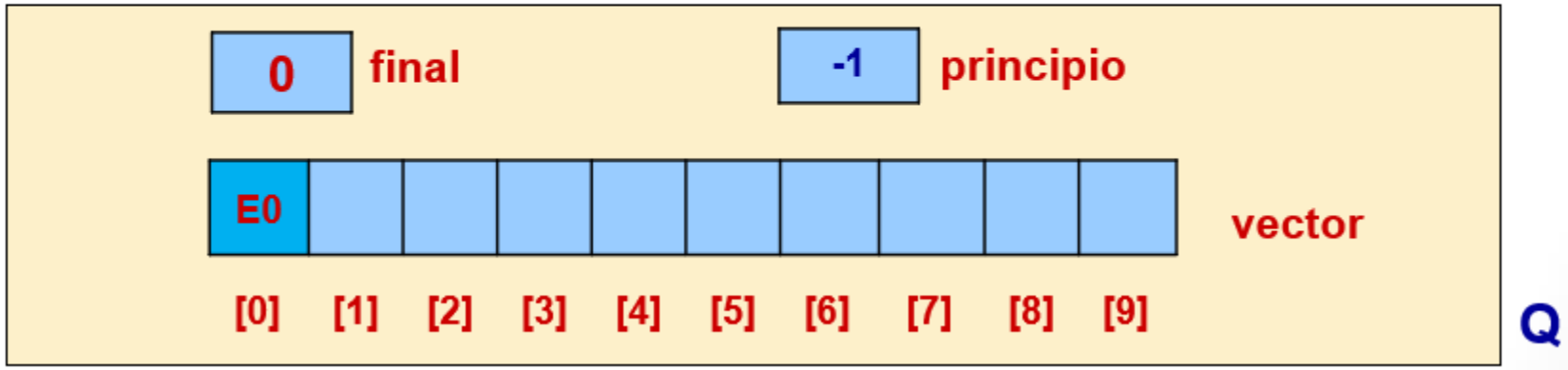
Índice final, contiene el índice correspondiente a la **posición del elemento añadido más recientemente**.

Cola vacía, principio = final = -1

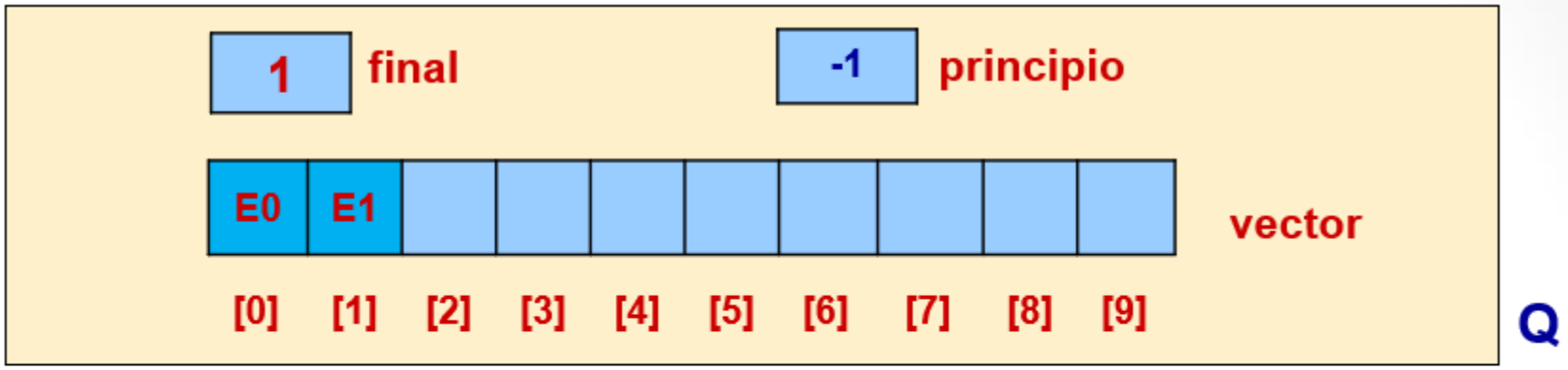
ColaVacia (Q)



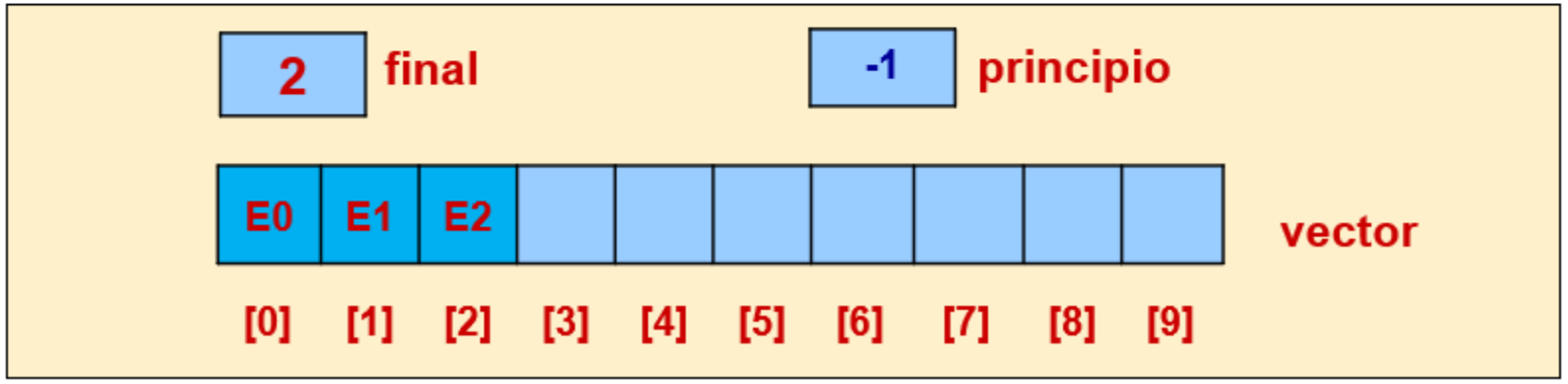
AniadirElemCola (Q, E0)



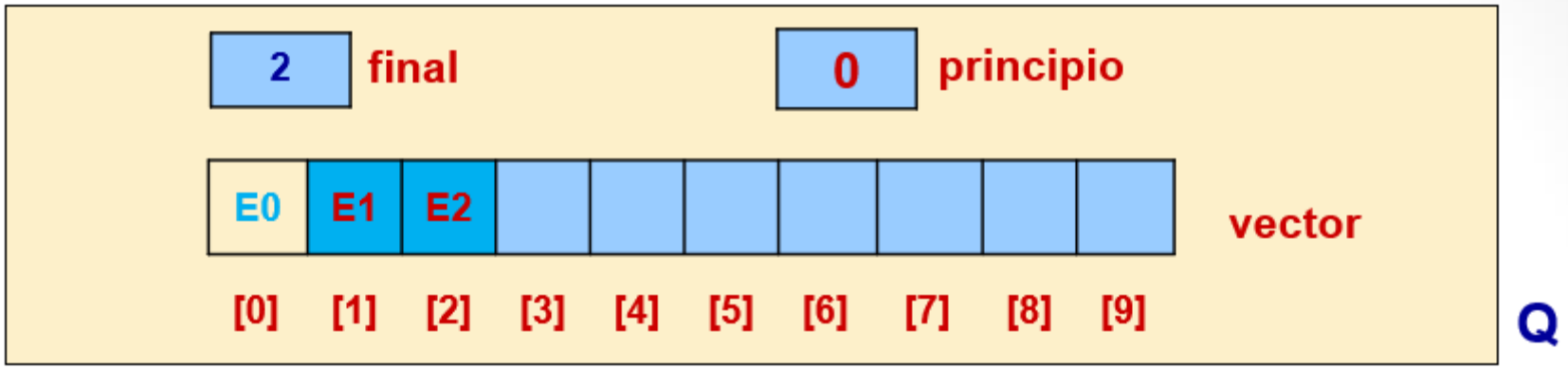
AniadirElemCola (Q, E1)



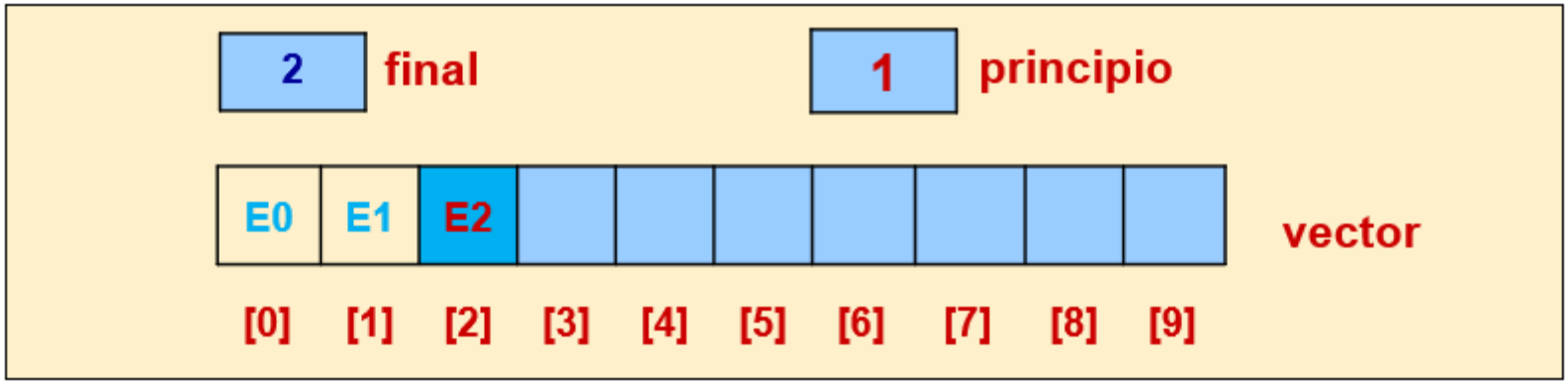
AniadirElemCola (Q, E2)



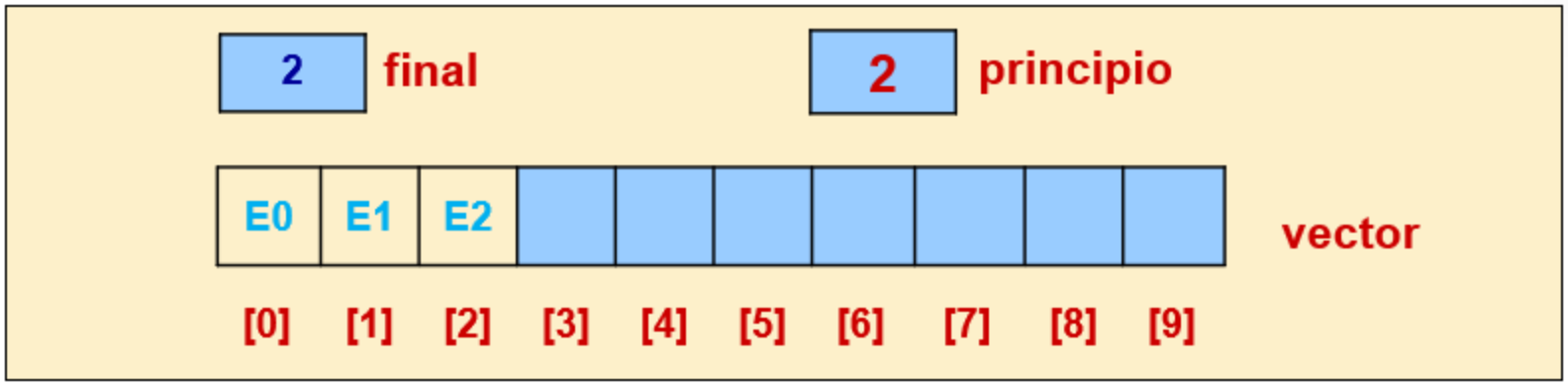
EliminarElemCola (Q)



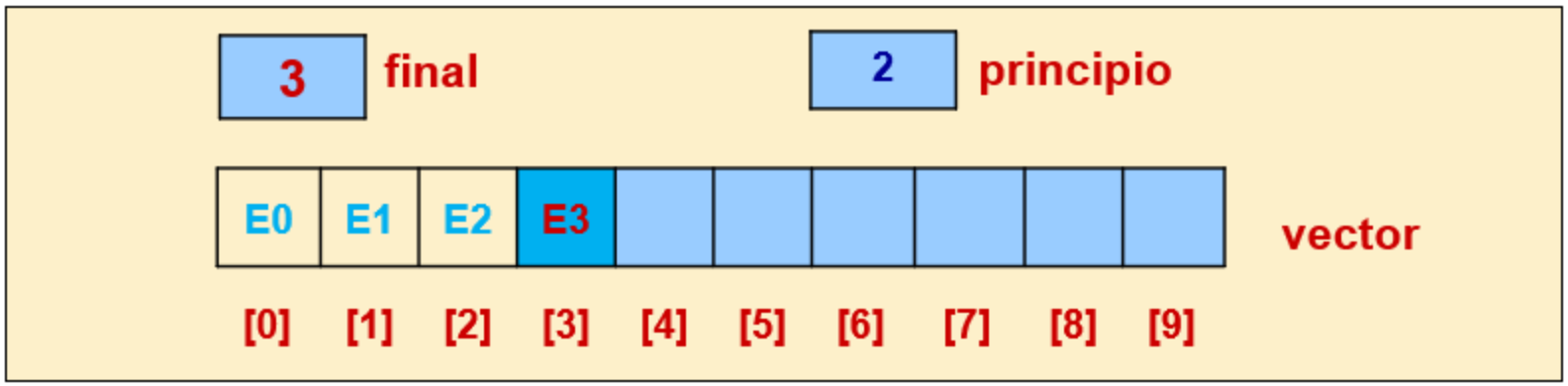
EliminarElemCola (Q)



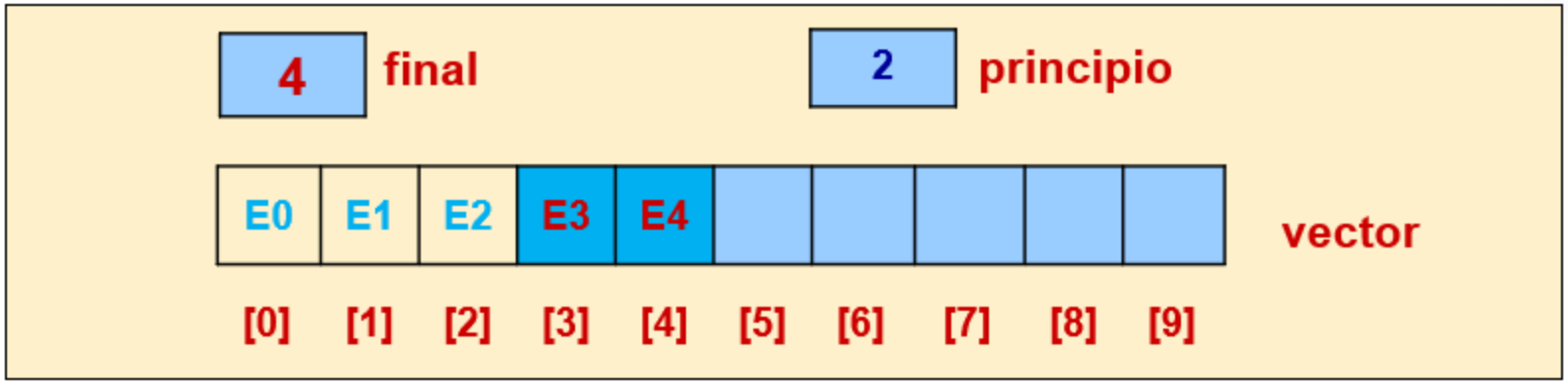
EliminarElemCola (Q)



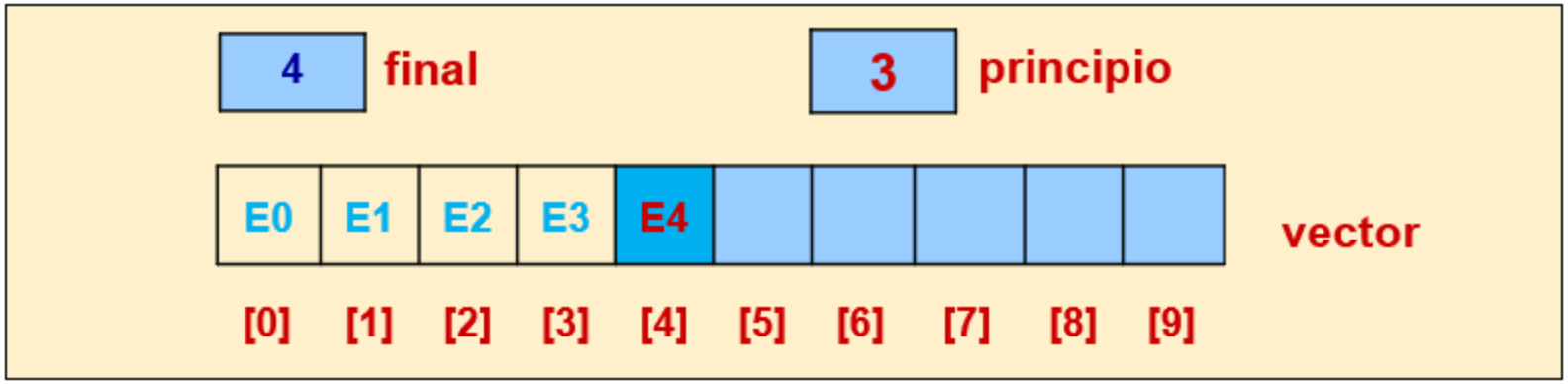
AniadirElemCola (Q, E3)



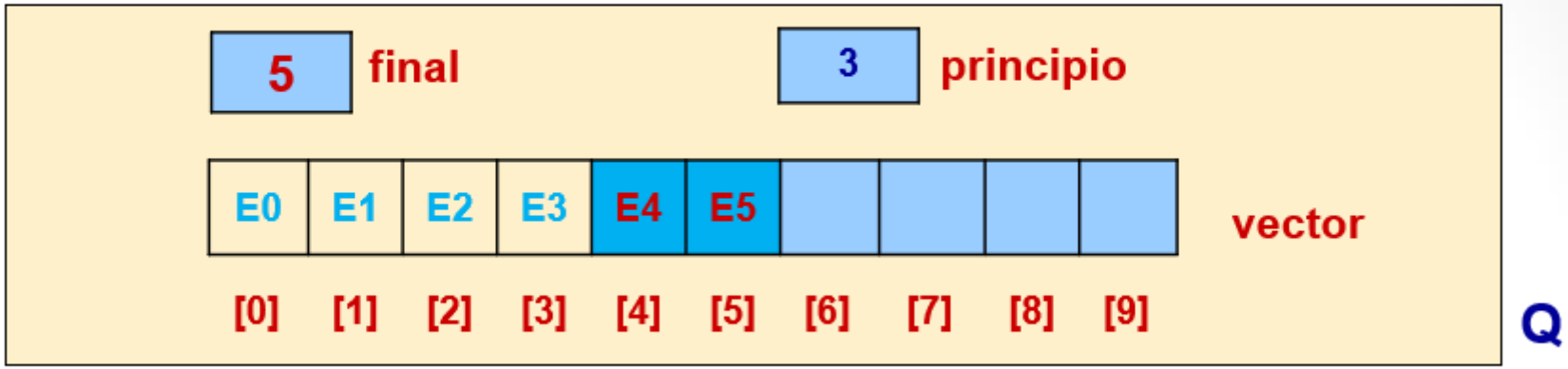
AniadirElemCola (Q, E4)



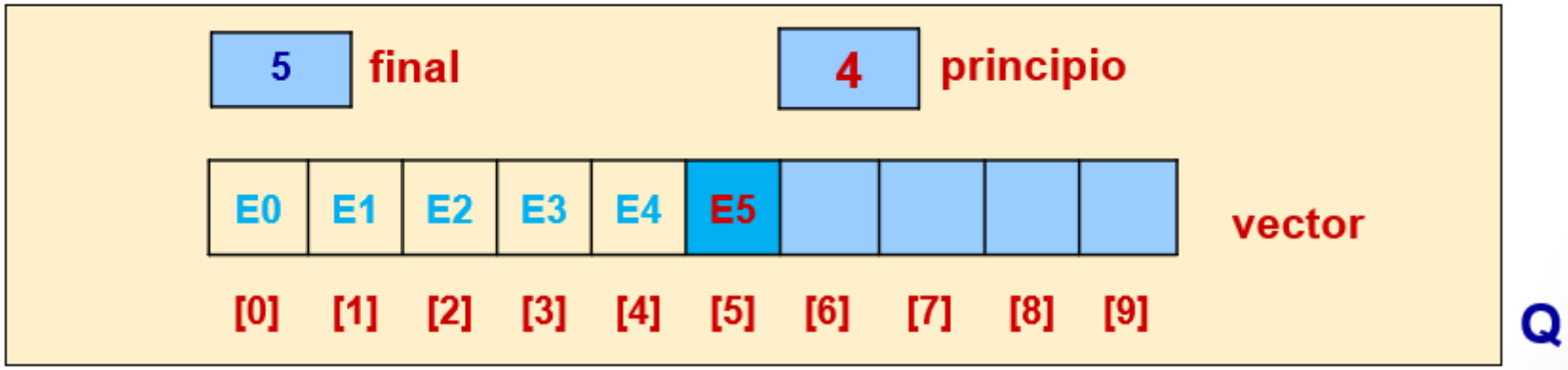
EliminarElemCola (Q)



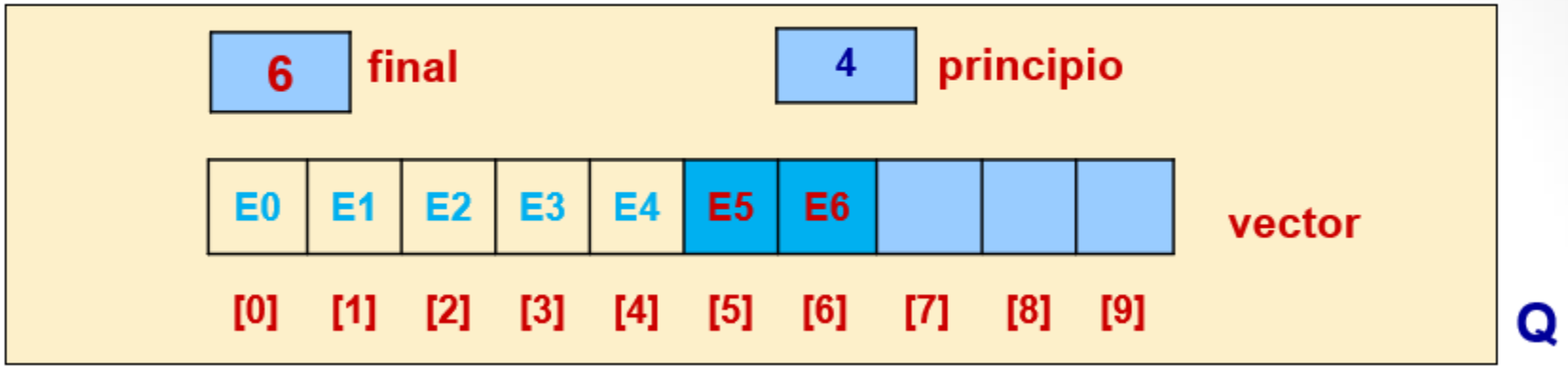
AniadirElemCola (Q, E5)



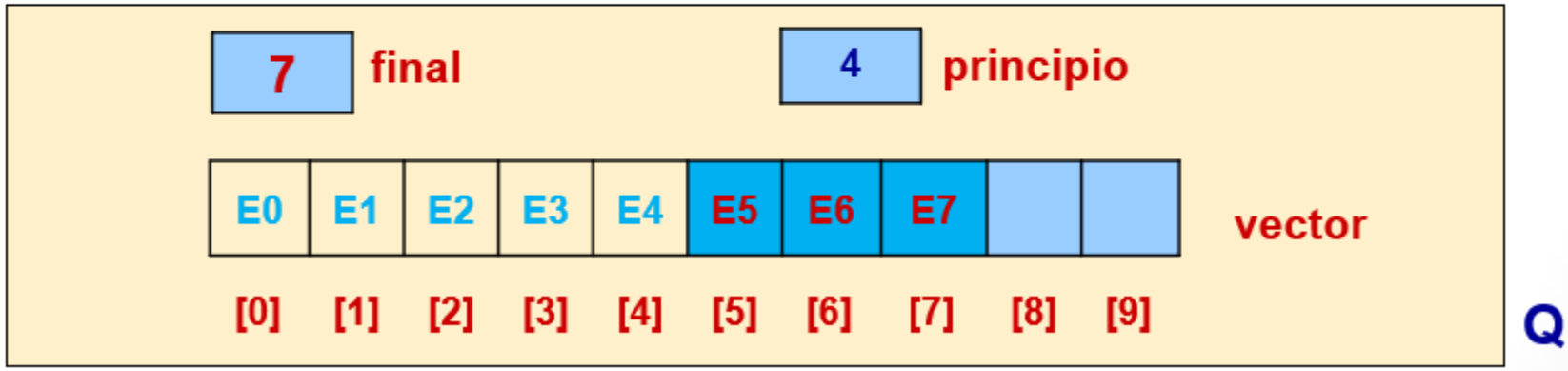
EliminarElemCola (Q)



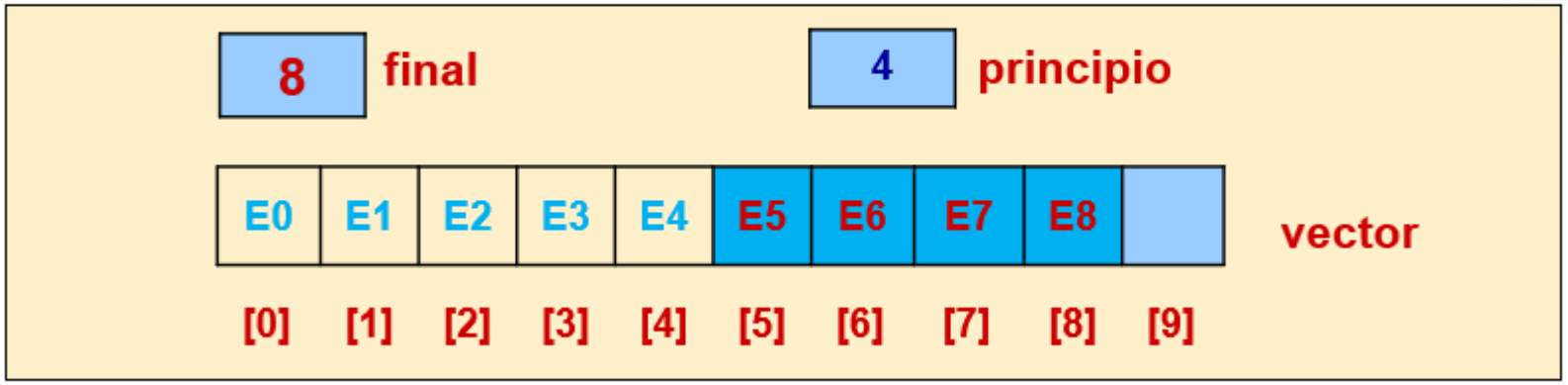
AniadirElemCola (Q, E6)



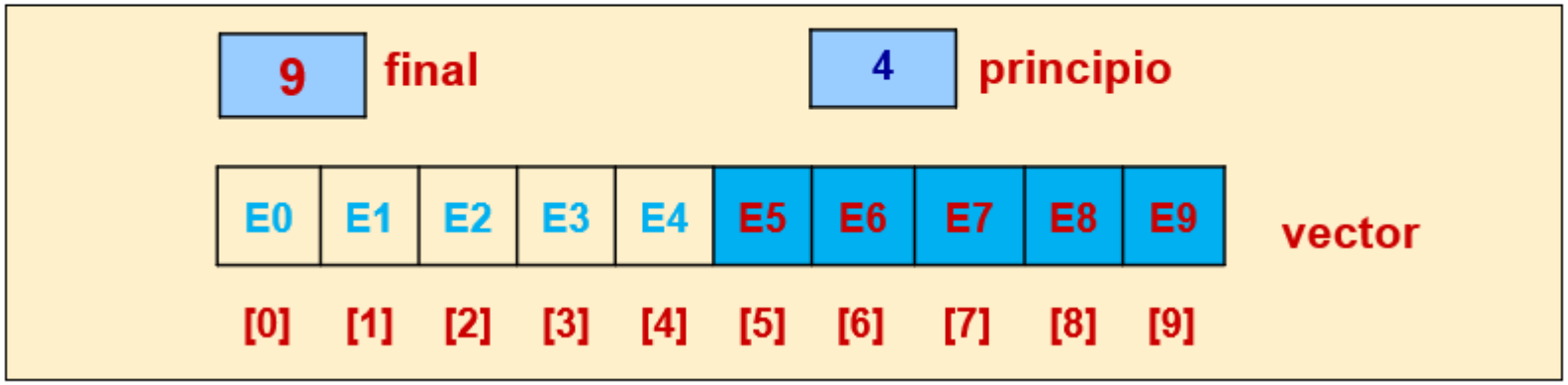
AniadirElemCola (Q, E7)

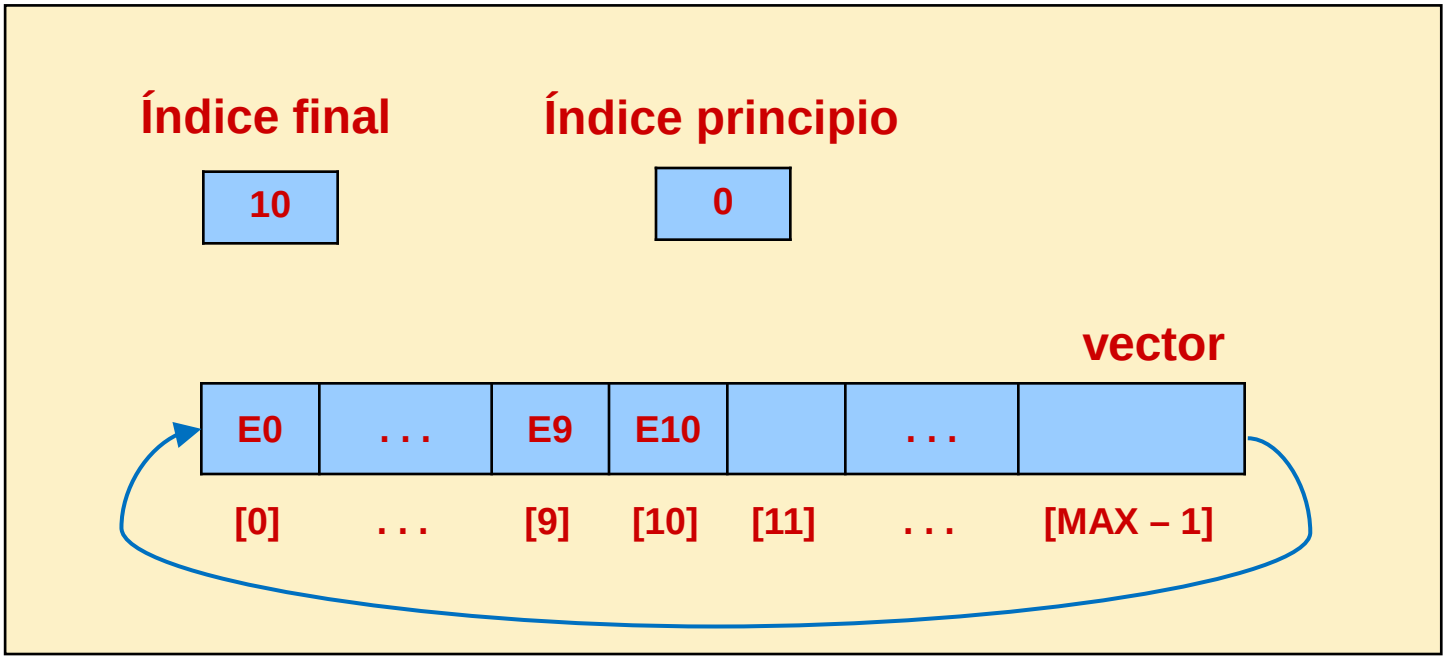


AniadirElemCola (Q, E8)



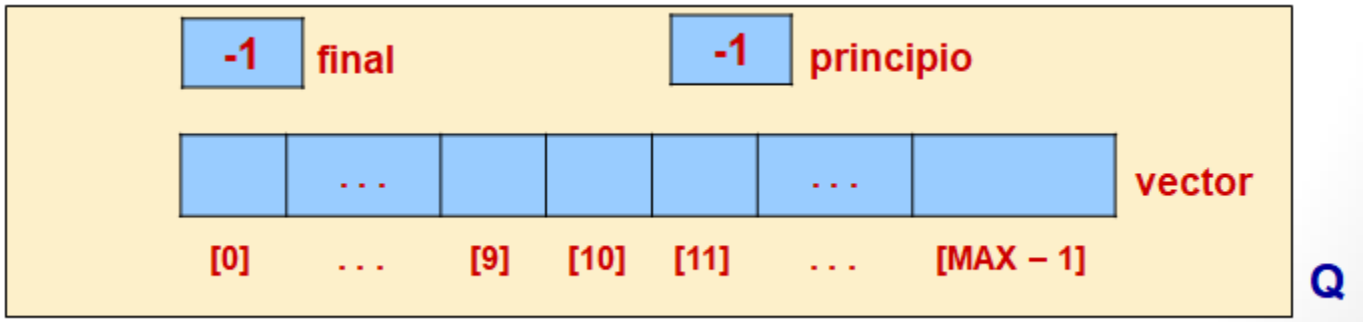
AniadirElemCola (Q, E9)





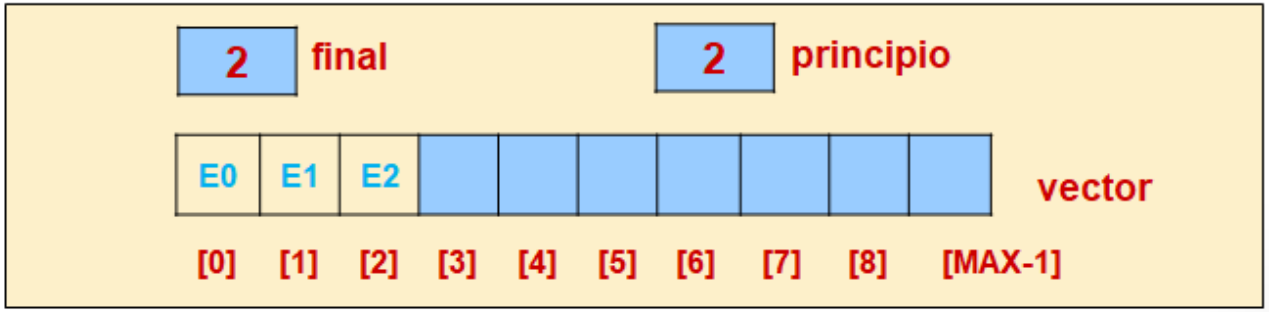
- **La posición siguiente a la última del vector será la primera posición del vector**.
- No siempre la siguiente posición a una dada, **i**, será la posición **i+1**. Habrá que implementar una función, **PosSiguiente**, para determinar la posición correcta.

Constructor Cola()

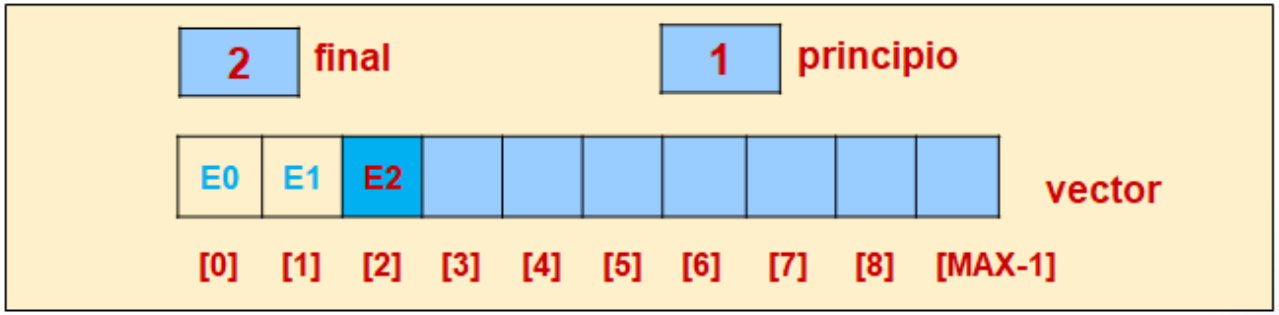


Operación empty

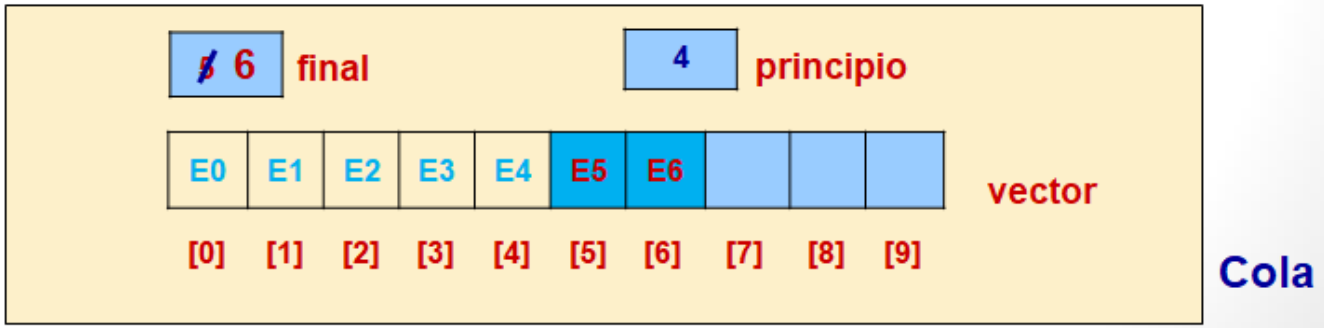
Verdadero



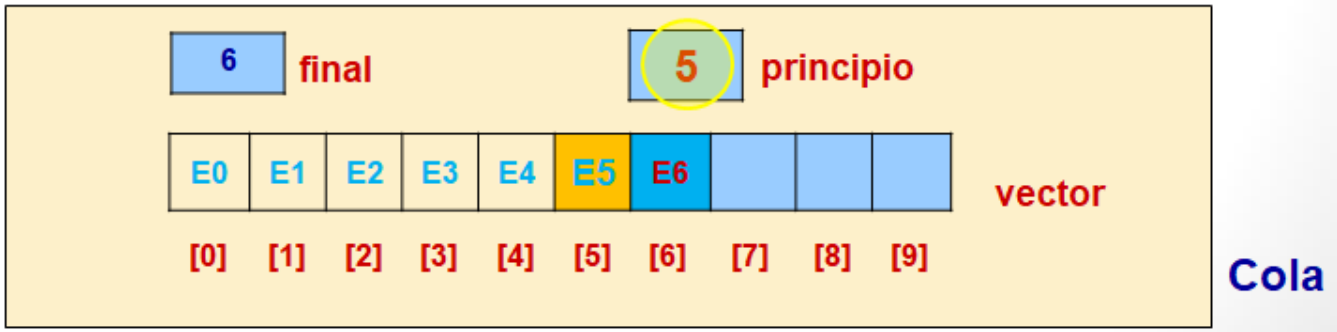
Falso



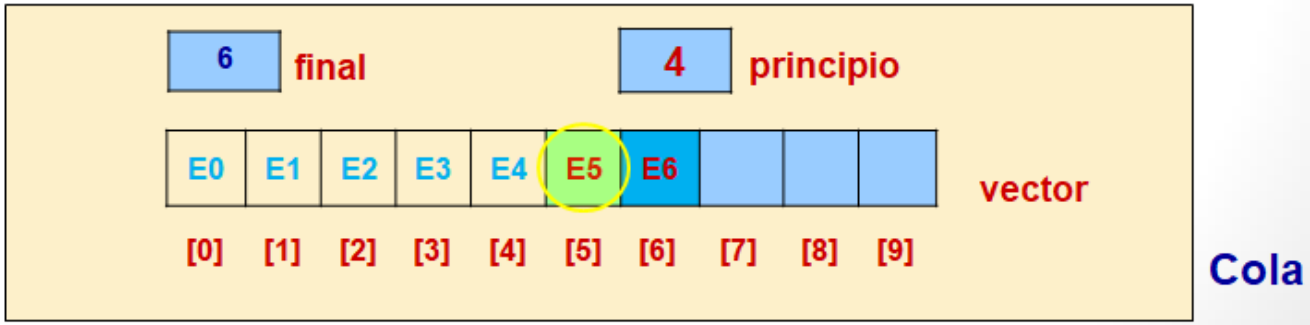
Operación push

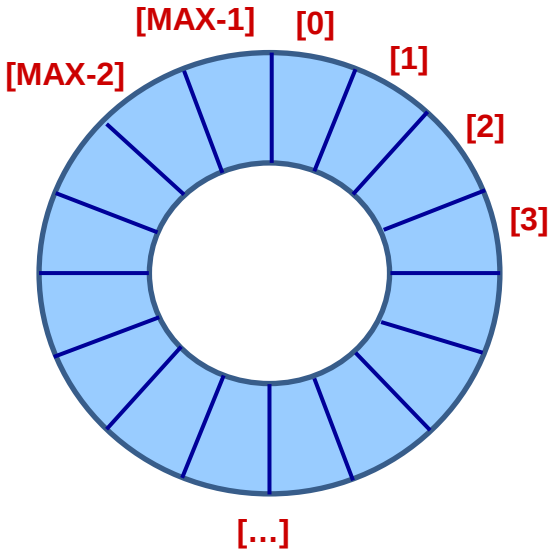


Operación pop



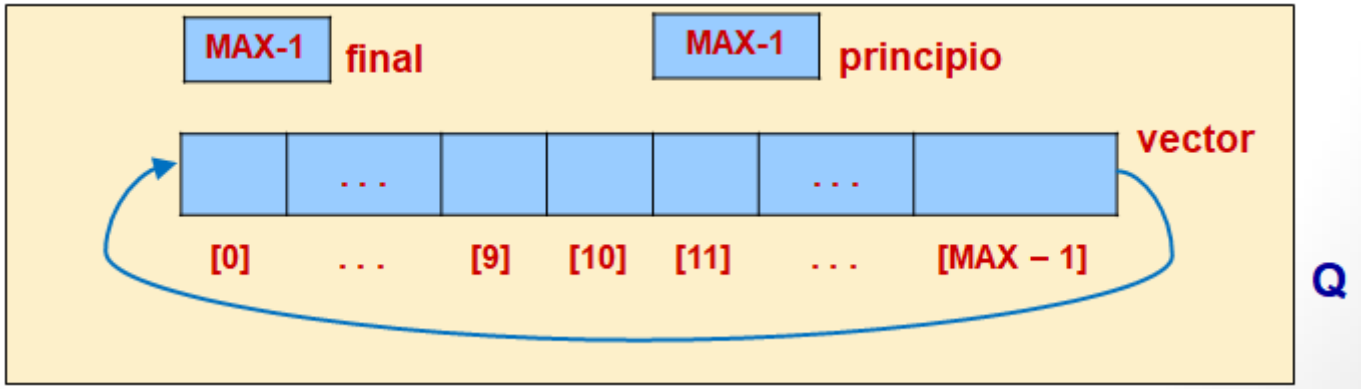
Operación first



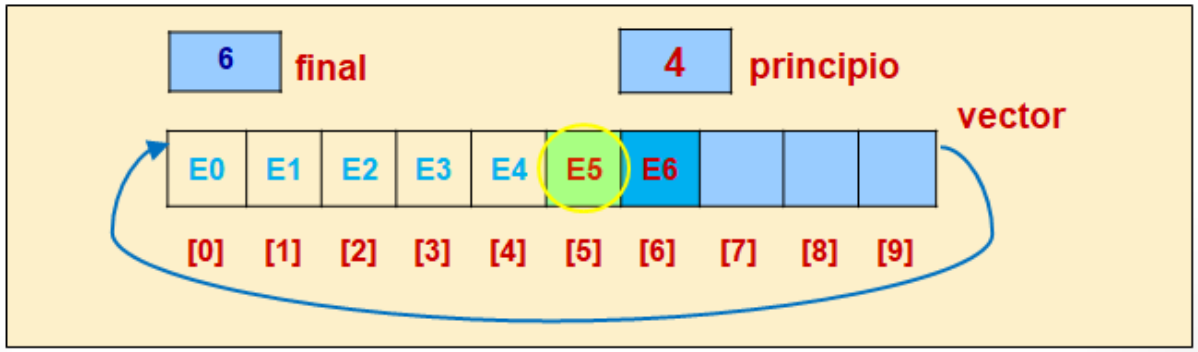


int **PosSiguiente** (int pos)

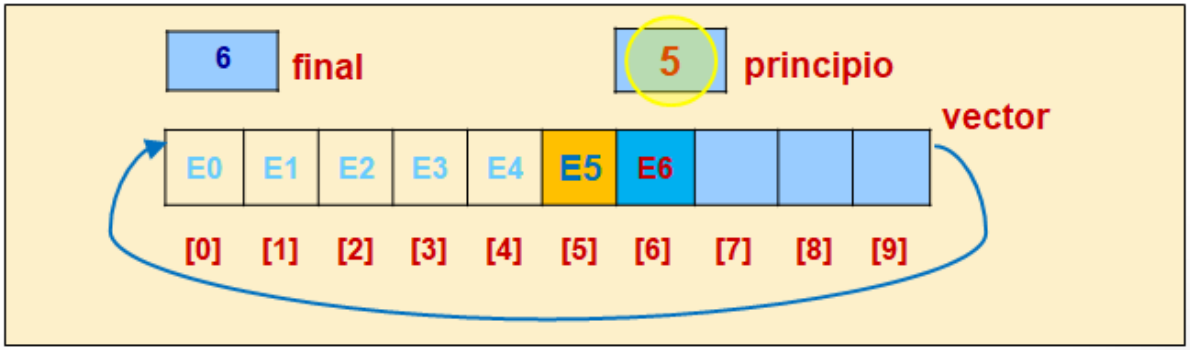
Constructor Cola()



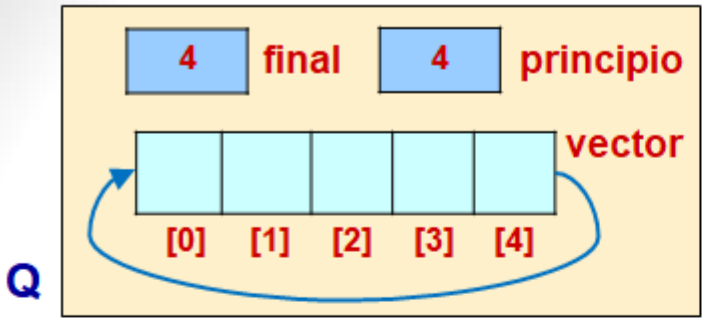
Operación first



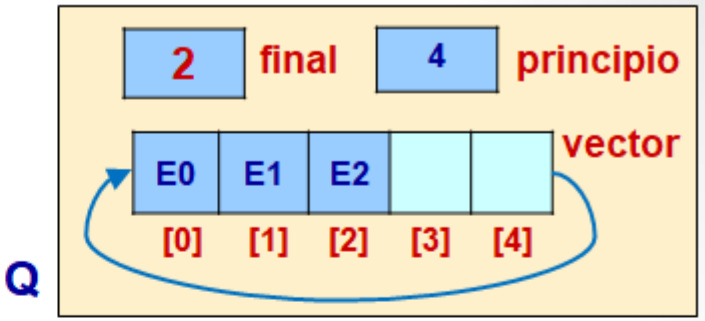
Operación pop



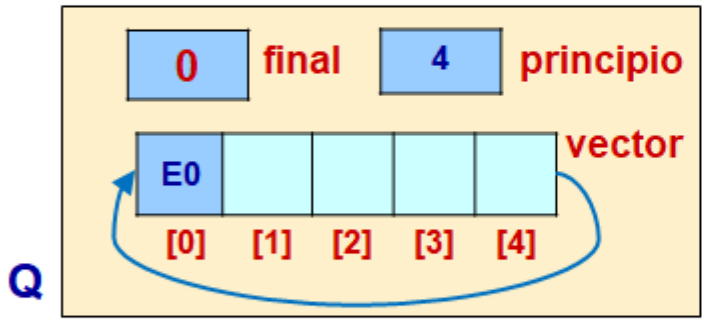
ColaVacia (Q)



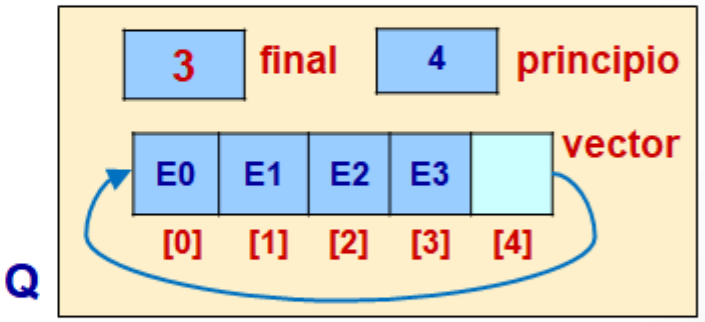
AniadirElemCola (Q, E2)



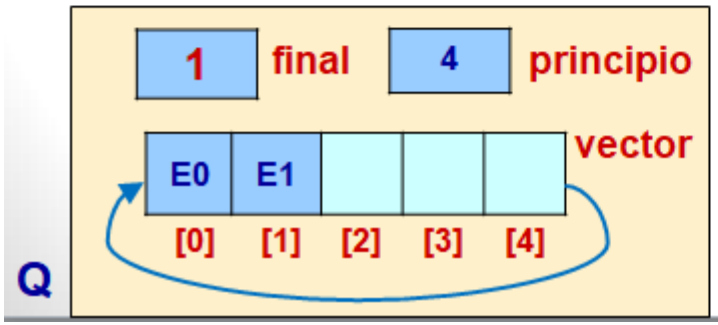
AniadirElemCola (Q, E0)



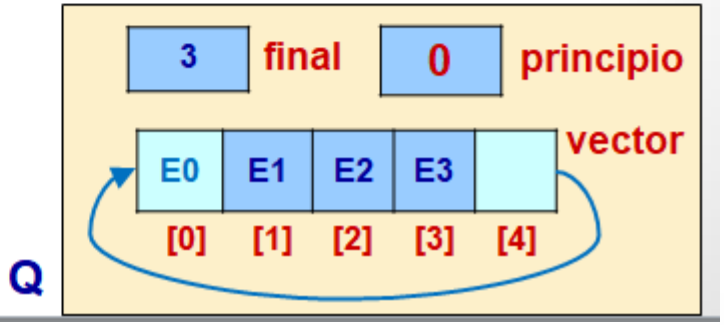
AniadirElemCola (Q, E3)



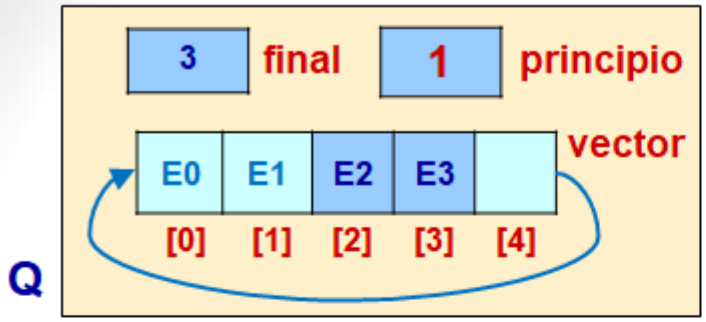
AniadirElemCola (Q, E1)



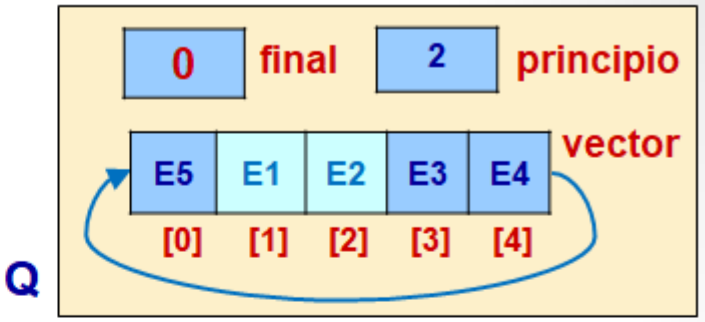
EliminarElemCola (Q)



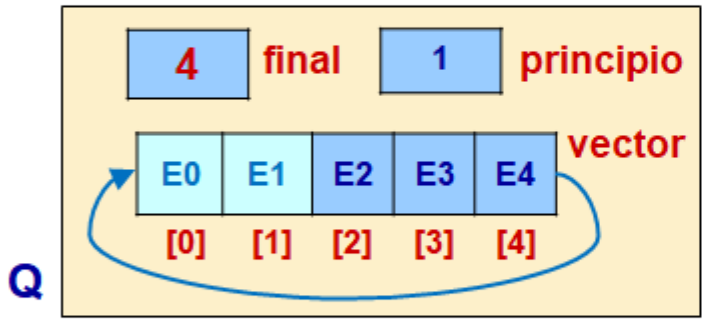
EliminarElemCola (Q)



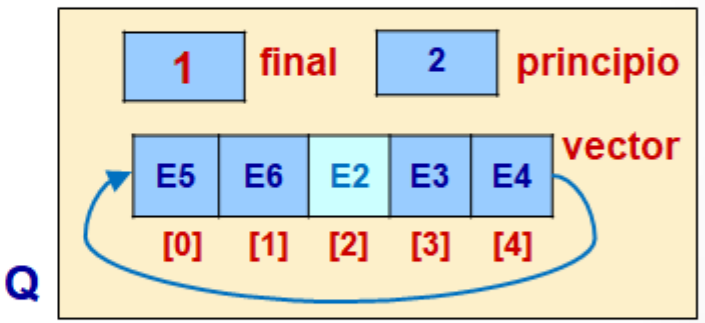
AniadirElemCola (Q, E5)



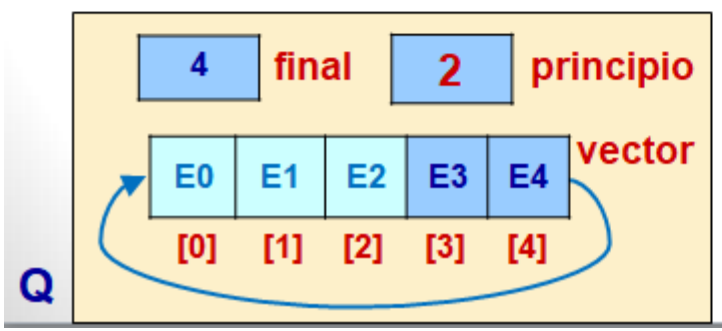
AniadirElemCola (Q, E4)



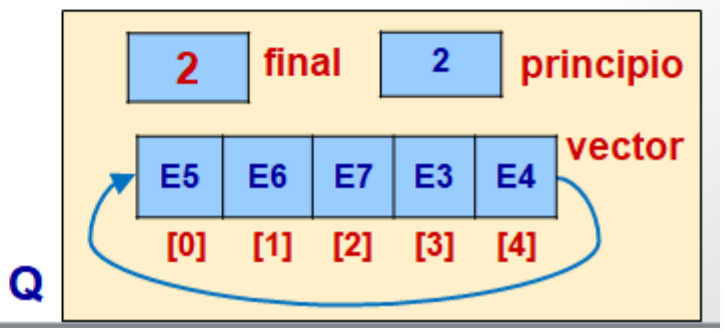
AniadirElemCola (Q, E6)

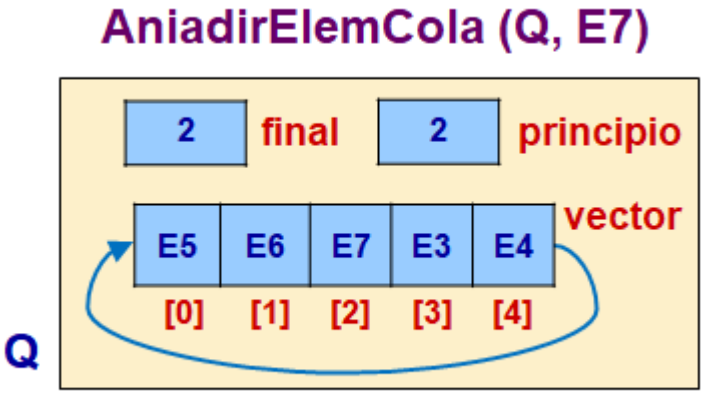


EliminarElemCola (Q)



AniadirElemCola (Q, E7)





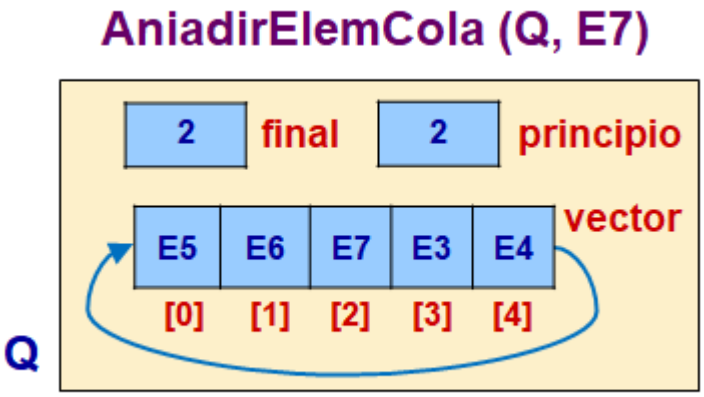
AMBIGÜEDAD!!

- 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**:

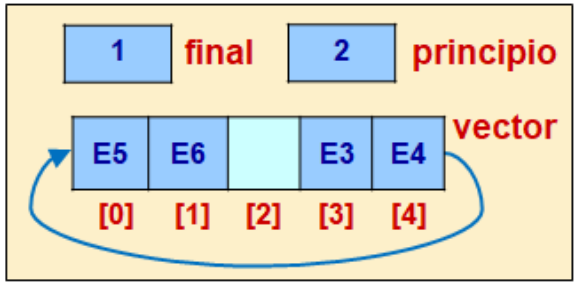
  Q.principio == Q.final

SOLUCIÓN

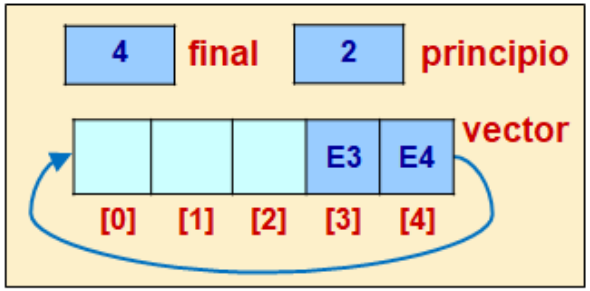
- **Incluir un contador de los elementos en el vector**: Modificar la estructura Tcola y la implementación de las operaciones.
- **Dejar una posición libre en el vector**: La posición que se deja disponible es la señalada por el campo **principio** (posición anterior al elemento más antiguo en la cola).

Operación colaLlena

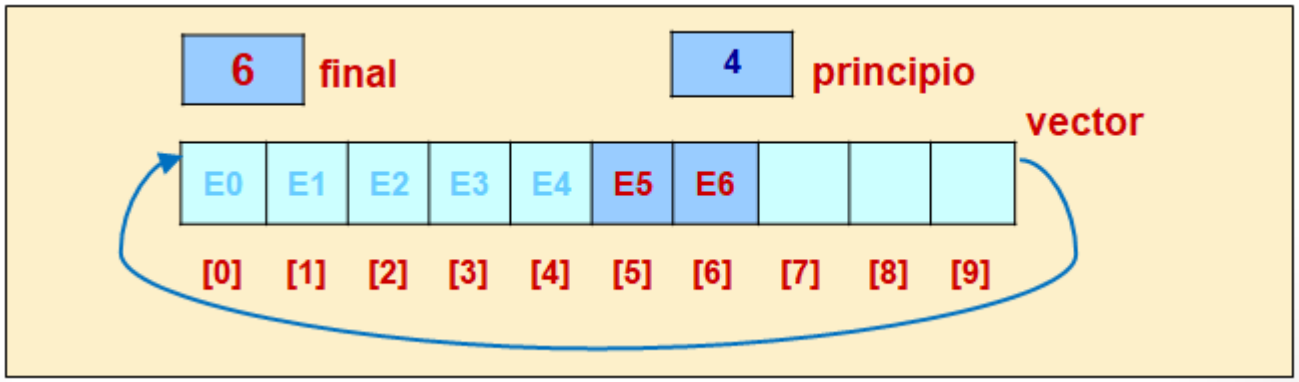
Verdadero



Falso



Operación push



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

using namespace std;

struct TipoDato {
            string elem;
            int aa;
        };

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


    private:
        int front, 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 cima de la cola
void Cola::pop()  {
    front = posSiguiente(front);
}

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

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

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

// Determina la posición siguiente a una dada
int Cola::posSiguiente (int p)
{   int size = datos.size();
    if (p < size-1) {
        return p = p + 1;
    }
    else {
        return 0;
    }
}
```