Tema 7. Colas

FUNDAMENTOS DE PROGRAMACIÓN II

Profesor: Baldomero Imbernón Tudela

Escuela Politécnica Superior
Grado en Ingeniería Informática

Contenidos

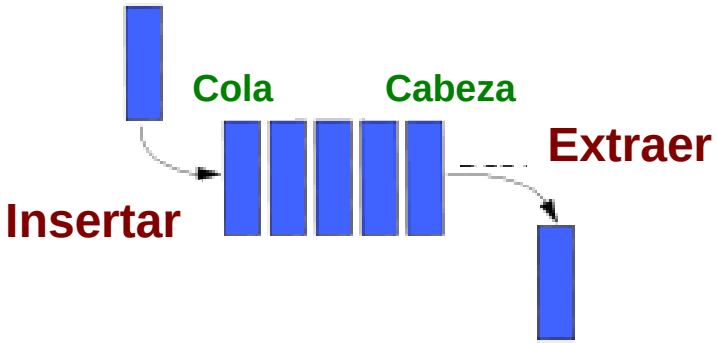
- Introducción
- Funcionamiento de una cola estática
- Funcionamiento de una cola dinámica
- Modalidades de colas

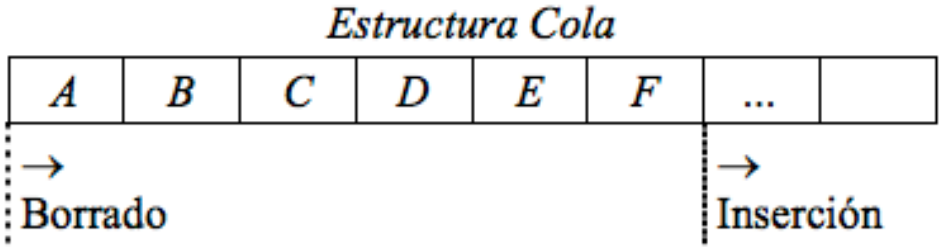
Contenidos

- **Introducción**
- Funcionamiento de una cola estática
- Funcionamiento de una cola dinámica
- Modalidades de colas

Cola

- Secuencia lineal de elementos
- Restricción: primer elemento en ser insertado, primer elemento en ser extraído.
  - Política **FIFO** (First Input, First Output)
- Operaciones de inserción y extracción se realizan sobre los extremos de la estructura.
  - **Inserción**: al “final” de la estructura
  - **Extracción**: al “inicio” de la estructura
- Símil: pensar en una cola de cine





Cola

1.Ejemplo de Cola

15	20	9	.....	18	19

2.Vamos a Insertar el 13 en la Cola.

15	20	9	.....	18	19	13

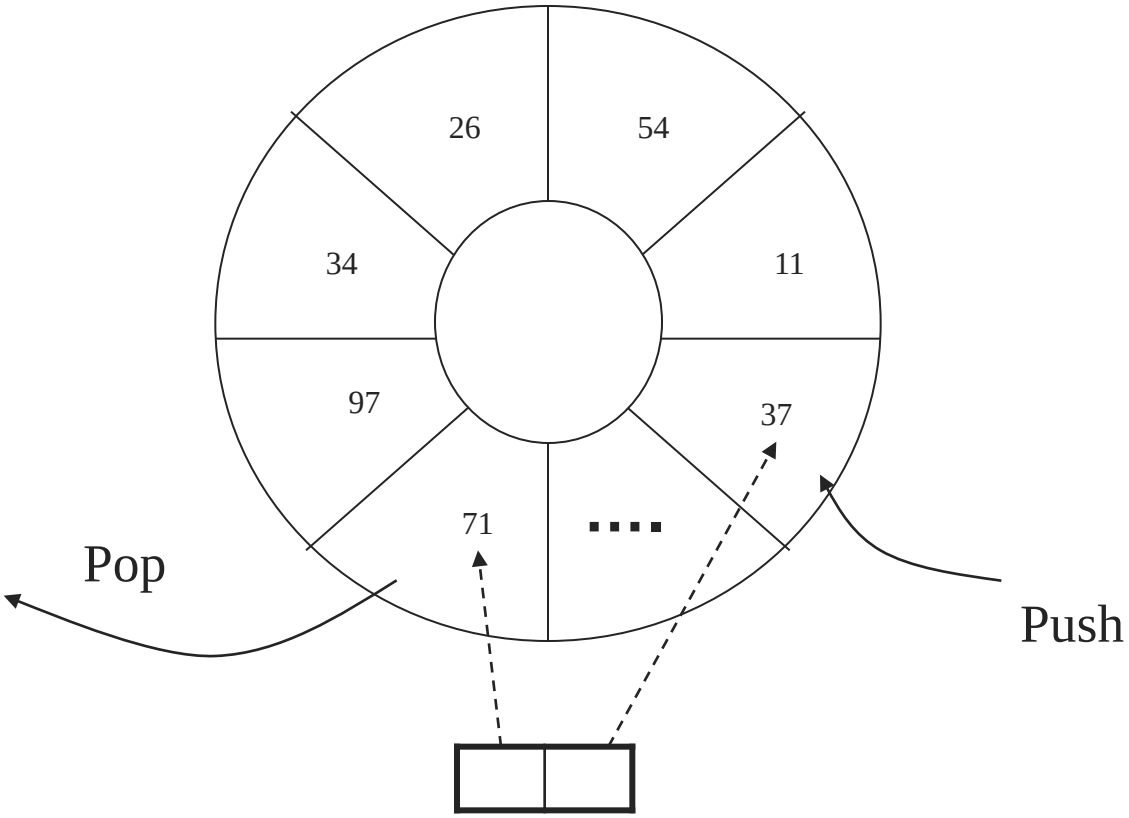
3.Sacamos el frente de la Cola (15)

20	9	.....	18	19	13

Cola

- Muy útil para resolver problemas informáticos:
  - **Ejemplo 1**: *tareas de un ordenador*
    - Trabajos son encolados y se procesan por orden de llegada
    - Si hay prioridad entre tareas, se usa una cola de prioridad.
  - **Ejemplo 2:** *trabajos de impresión en una impresora*
    - Los trabajos son procesados por orden de llegada
- Operaciones básicas para el manejo de colas:
  - **CREA**: Crea una cola vacía.
  - **VACIA**: Devuelve un valor cierto si la cola está vacía, y falso en caso contrario.
  - **PRIMERO**: Devuelve el primer elemento de una cola.
  - **INSERTA (Push)**: Añade un elemento por el extremo final de una cola.
  - **SUPRIME (Pop)**: Suprime el primer elemento de una cola

Cola



Cola

- Especificación Formal
  - **Sintaxis**:
    - crea = Cola
    - vacia(Cola) = booleano
    - primero(Cola) = Elemento
    - inserta(Cola,Elemento) = Cola
    - suprime(Cola) = Cola
  - **Semántica**:
    - vacia(crea) = cierto
    - vacia(inserta(C,E)) = falso
    - primero(crea) = error
    - primero(inserta(C,E)) = si vacia(C) ? E : primero(C)
    - suprime(crea) = error
    - suprime(inserta(C,E)) = si vacia(C) ? crea : inserta(suprime(C),E)

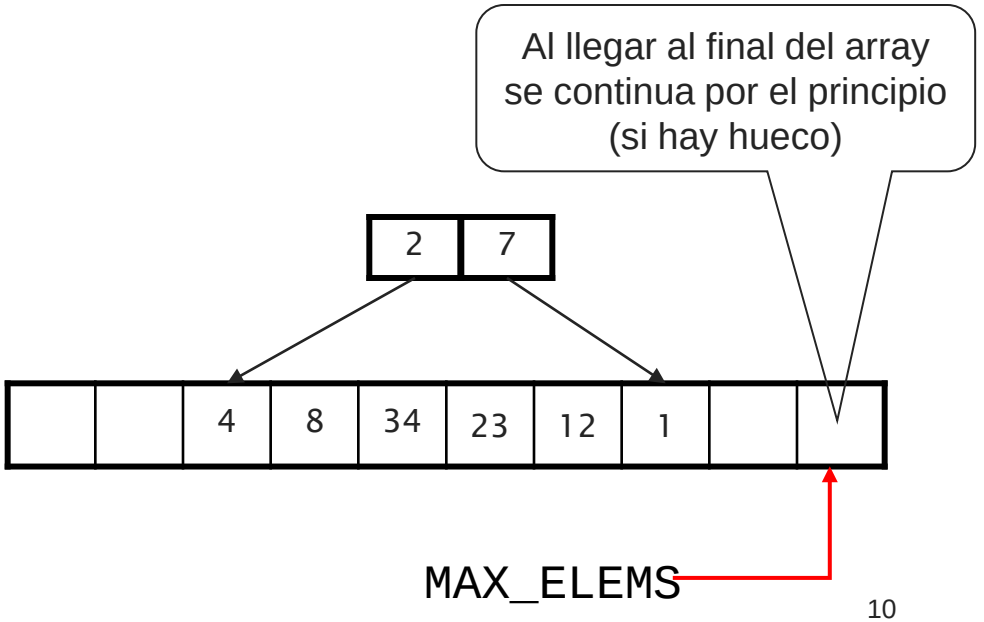
Contenidos

- Introducción
- **Funcionamiento de una cola estática**
- Funcionamiento de una cola dinámica

Cola: versión estática

- Registro que contiene
  - Array de datos
  - Dos enteros:
    - *cabeza*: para extraer
    - *cola*: para introducir

```
#define MAX_ELEMS 10
typedef struct cola{
int cabeza,cola;
int valores[MAX_ELEMS];
}tipo_cola;
```



Cola: versión estática

- Se va a hacer uso de esta función para calcular la posición en el array

```
int suma_uno(int i){
return ((i+1) % MAX_ELEMS);
}
```

Se utiliza el operador % para limitar la posición máxima dentro del array

Por ejemplo:

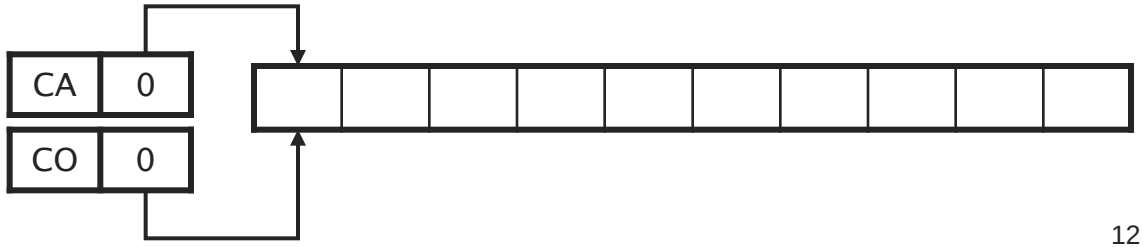
- Si se quiere introducir en la posición 7 => suma_uno(7) = (7+1) % 10 = 8
- Si se quiere introducir en la posición 9 => suma_uno(9) = (9+1) % 10 = 0
- Si se quiere introducir en la posición 15 => suma_uno(15) = (15+1) % 10 = 6

Cola: versión estática

Función iniciar

```
void iniciar(tipo_cola *c){
    c->cabeza = 0;
    c->cola = 0;
}
```

```
tipo_cola cola;
iniciar(&cola);
```



Cola: versión estática

Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

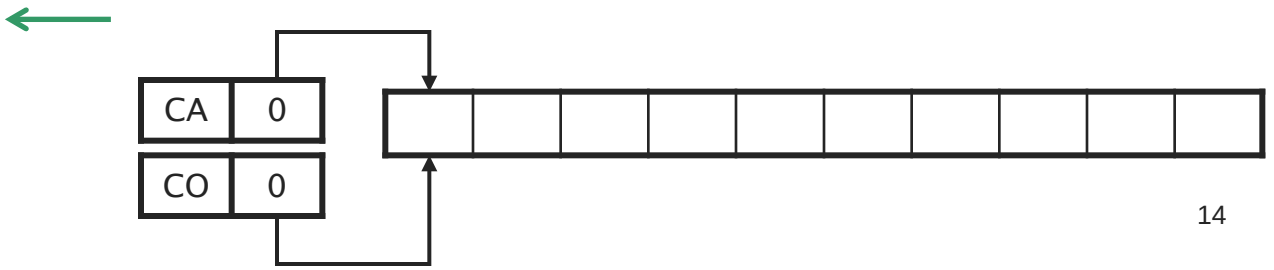
```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```

Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```



Cola: versión estática

- Función push (apilar)

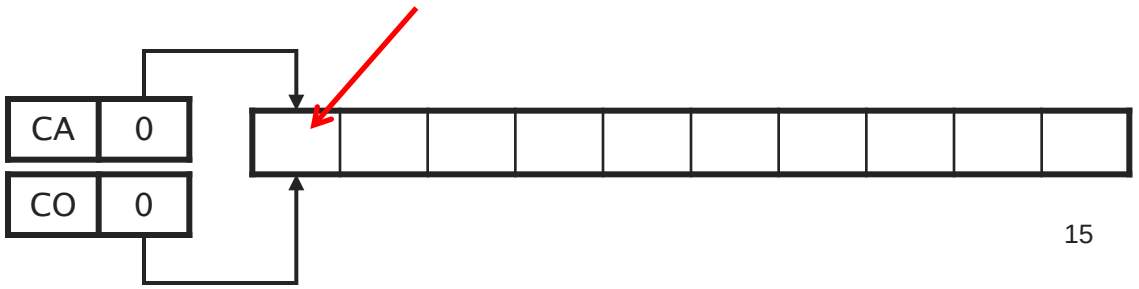
```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

¿suma_uno(0) == 0?

¿1 == 0?

No

```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```

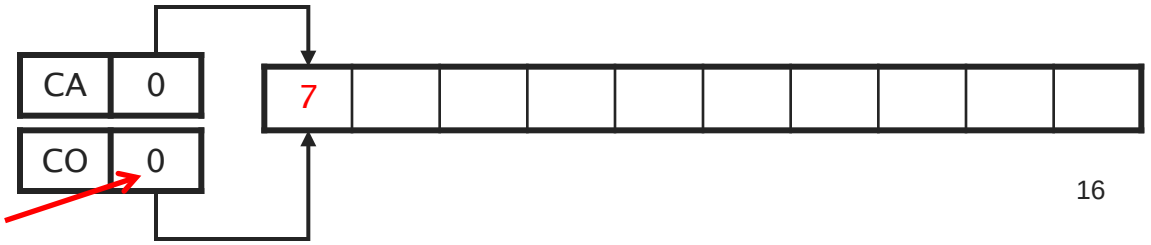


Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```

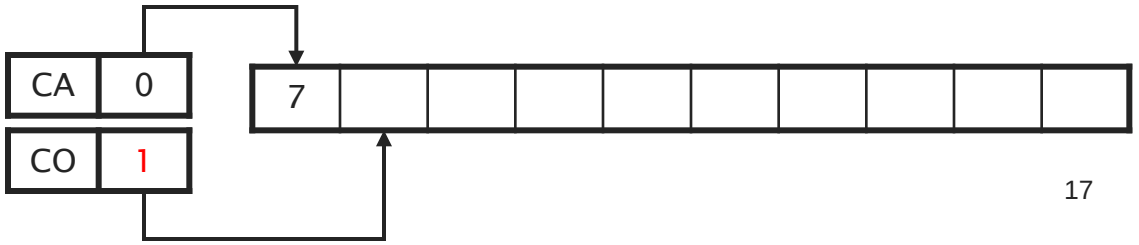


Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```

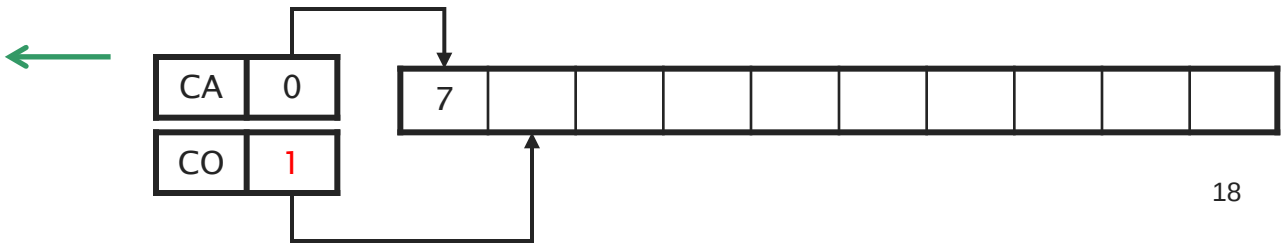


Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```



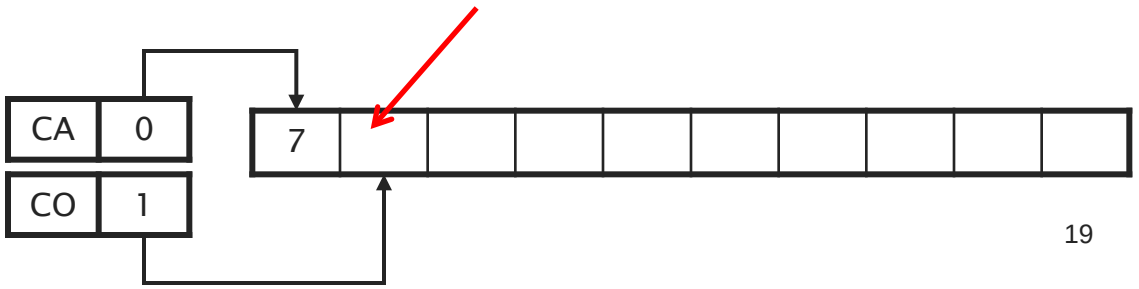
Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

¿2 == 0?

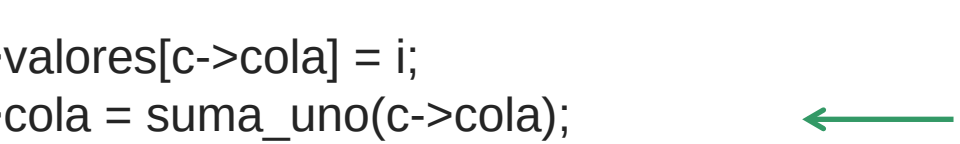
```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```



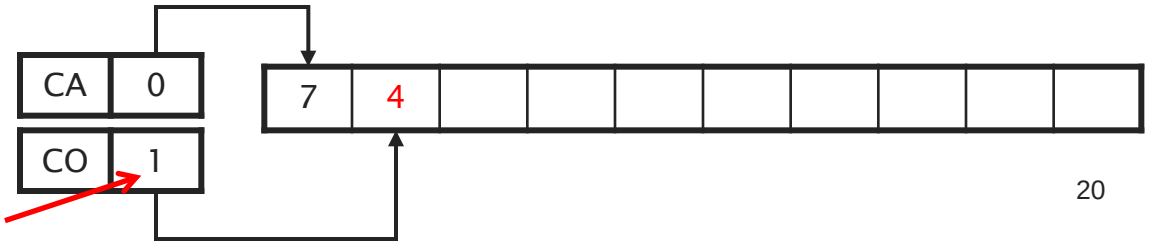
Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```



```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```

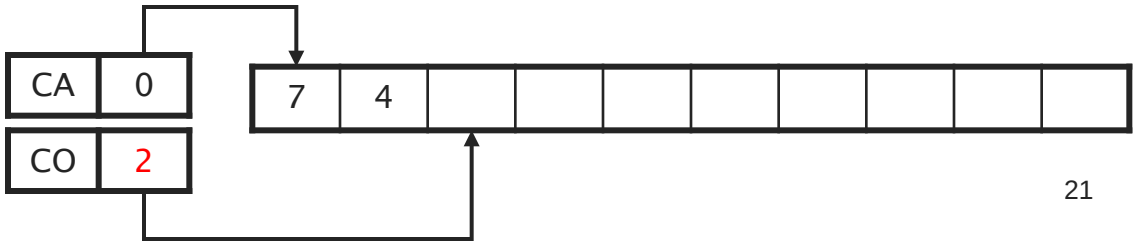


Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```

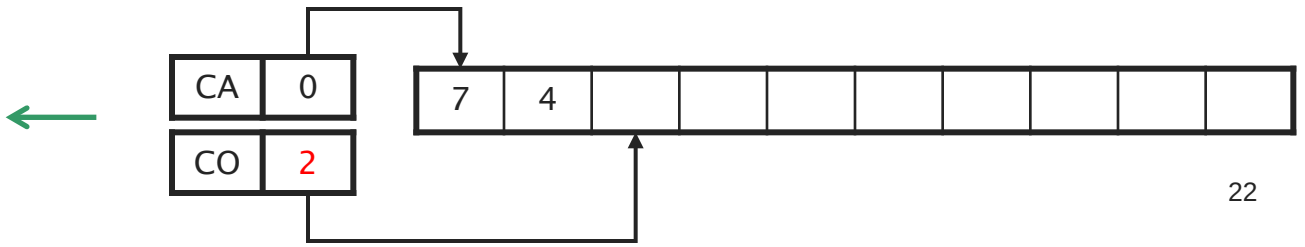


Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```



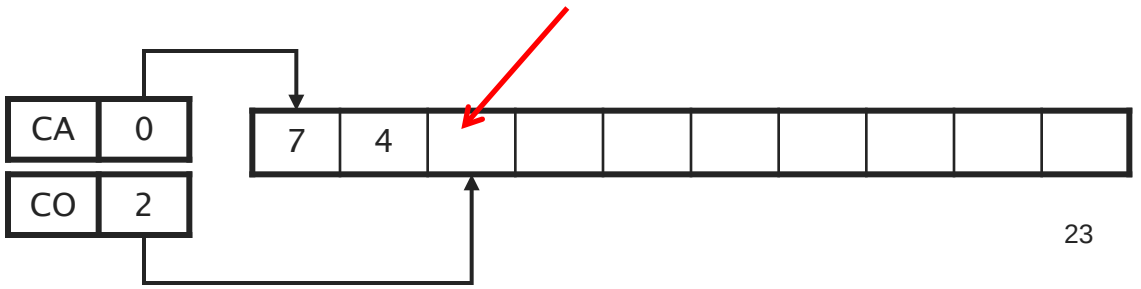
Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

¿3 == 0?

```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```

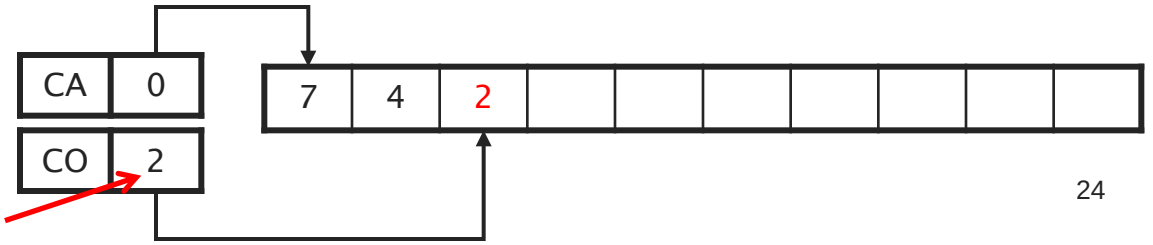


Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```

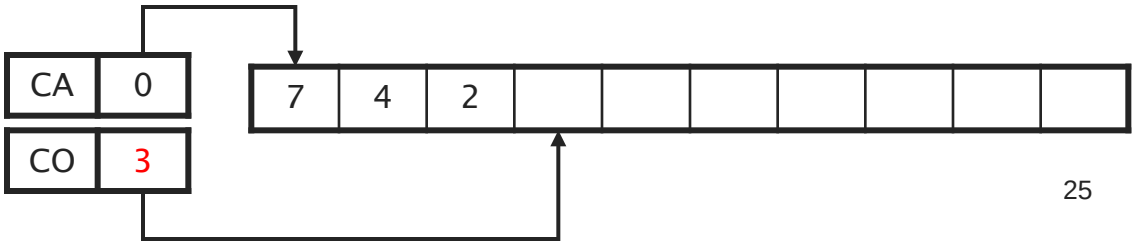


Cola: versión estática

- Función push (apilar)

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

```
apilar(&cola,7);
apilar(&cola,4);
apilar(&cola,2);
```

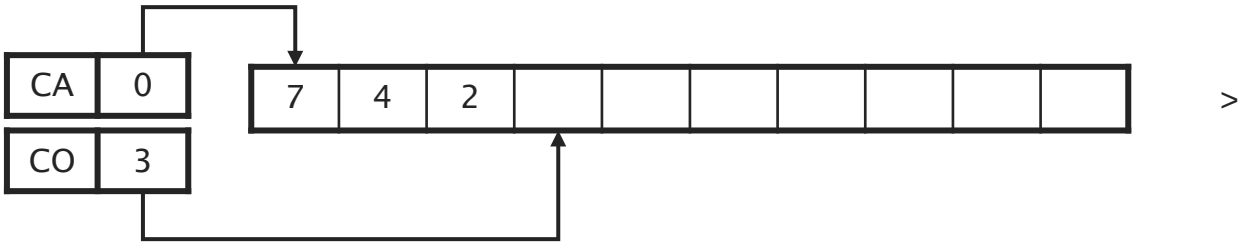


Cola: versión estática

Función pop (desapilar)

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```

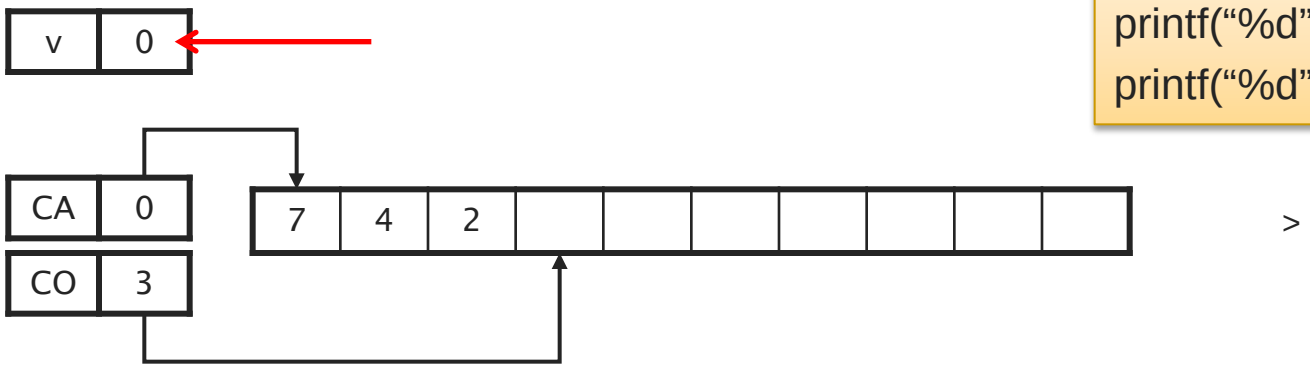
```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```



Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```

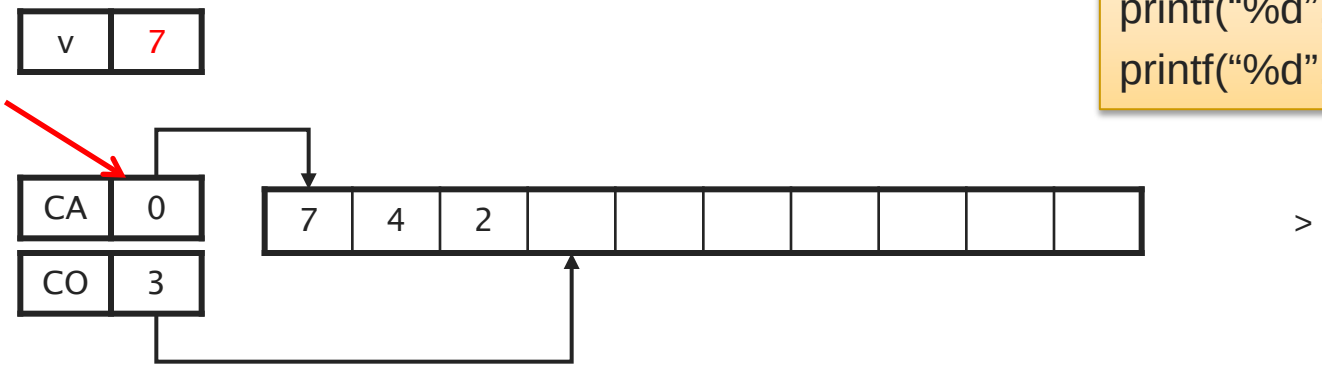
```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```



Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```

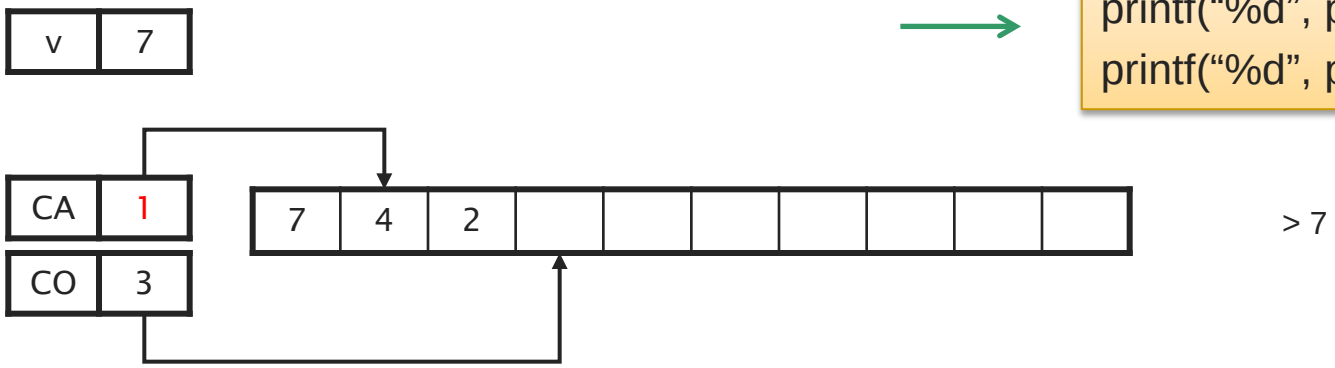
```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```



Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```

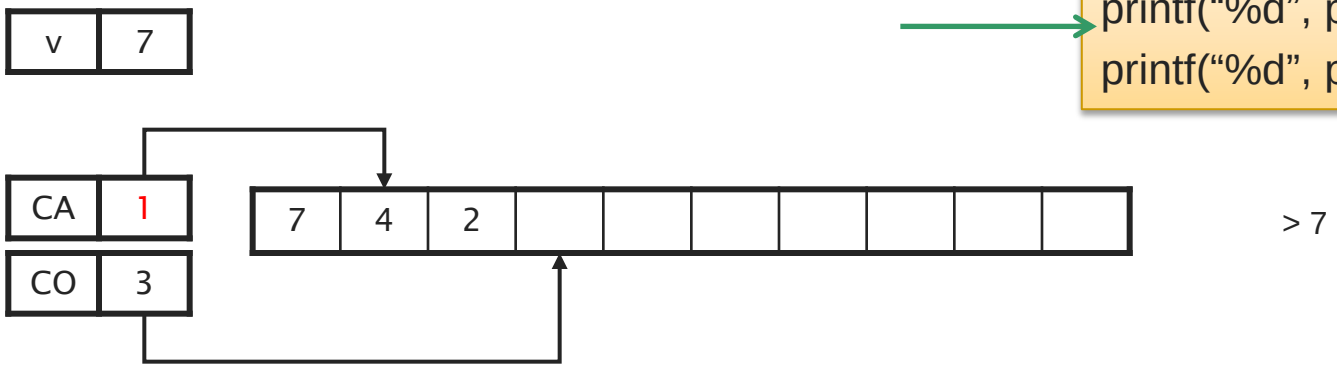
```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```



Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```

```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```

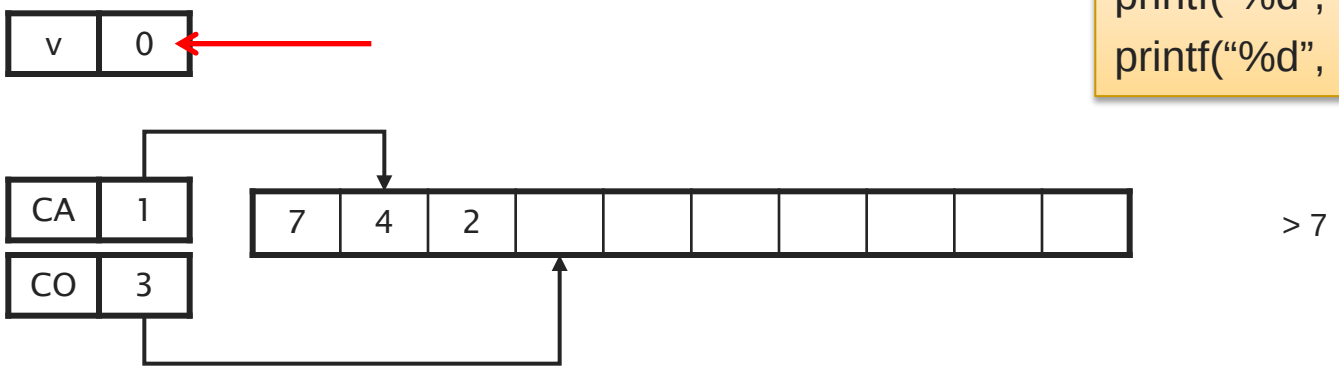


> 7

Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```

```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```

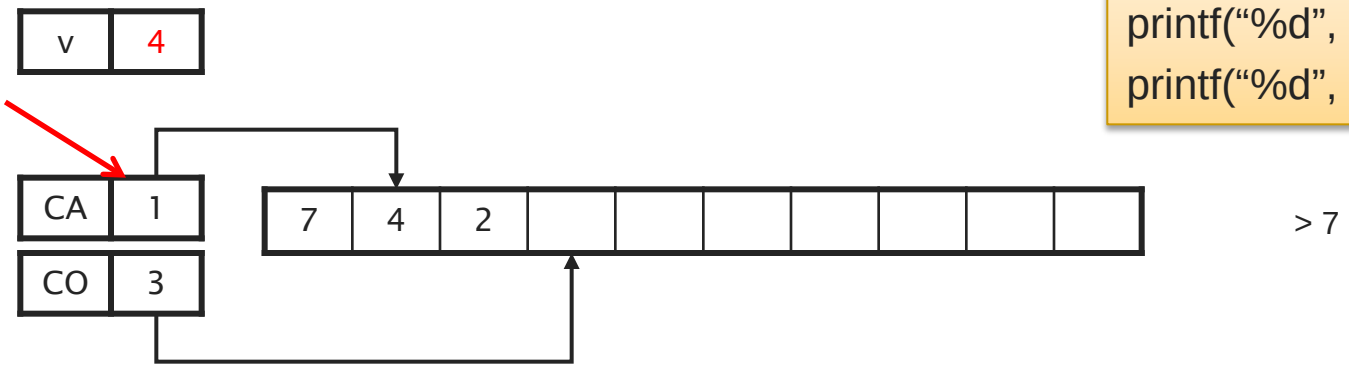


> 7

Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```

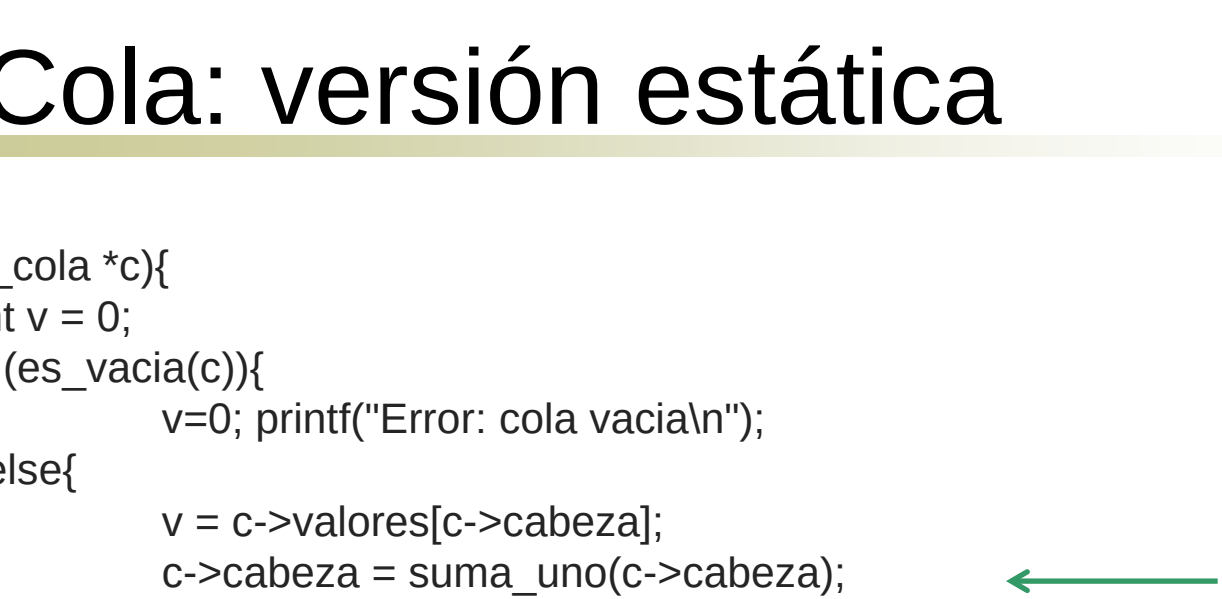
```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```



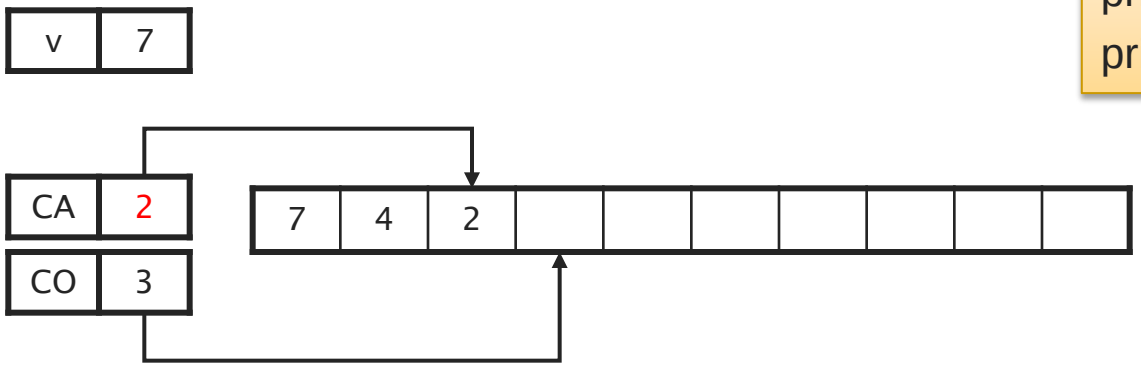
> 7

Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```



```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```

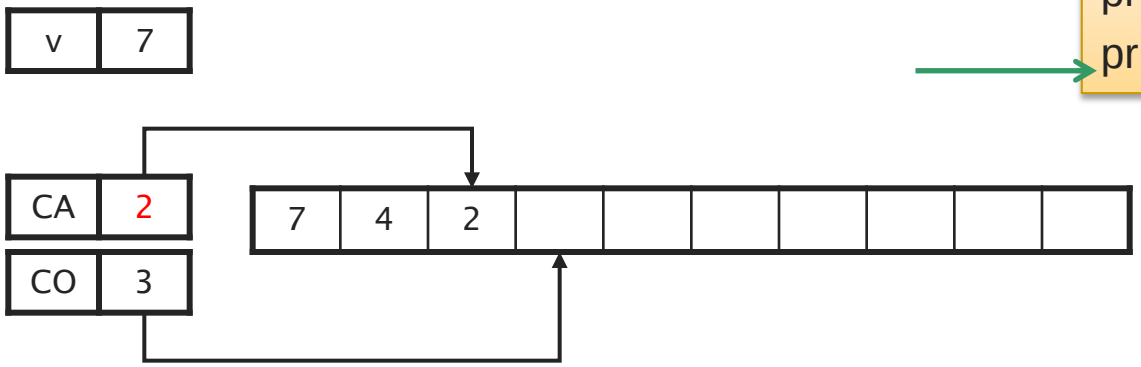


> 7 4

Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```

```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```

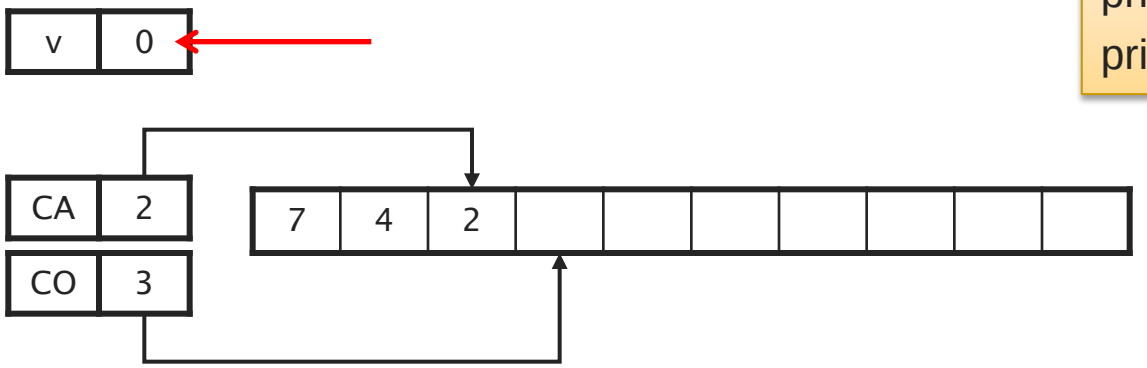


> 7 4

Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```

```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```

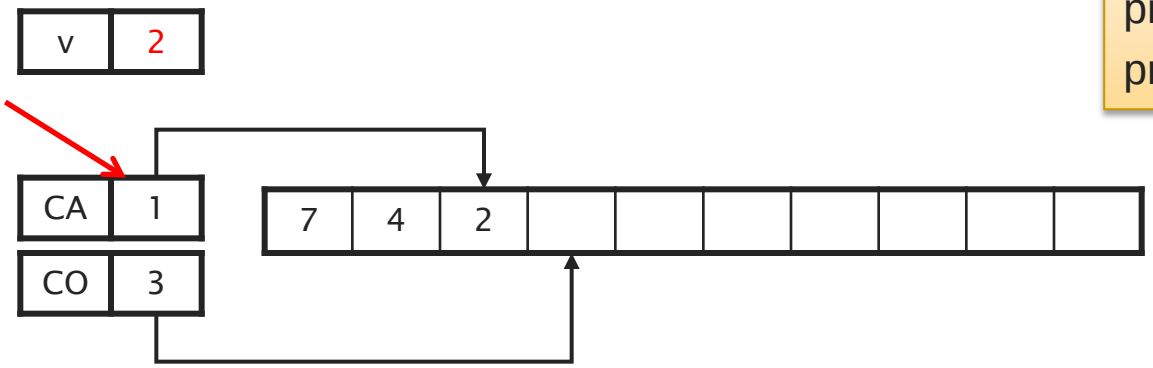


> 7 4

Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```

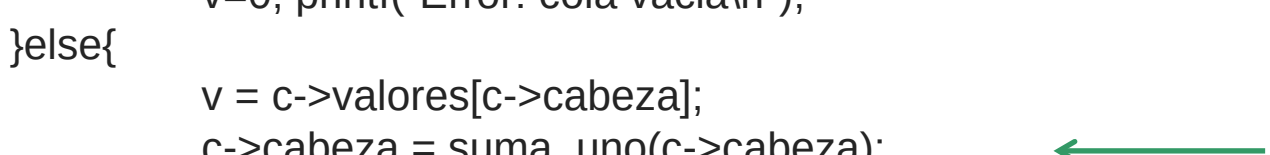
```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```



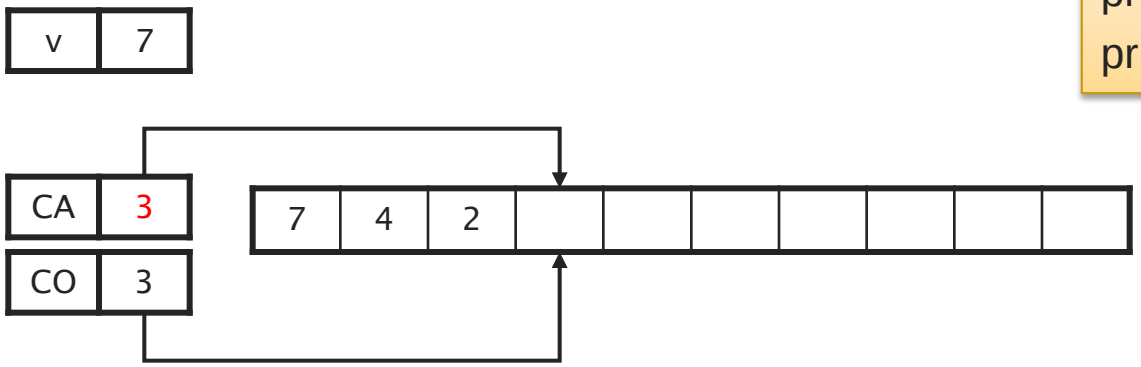
> 7 4

Cola: versión estática

```
int pop(tipo_cola *c){
        int v = 0;
        if (es_vacia(c)){
                v=0; printf("Error: cola vacia\n");
        }else{
                v = c->valores[c->cabeza];
                c->cabeza = suma_uno(c->cabeza);
        }
        return v;
}
```



```
printf("%d", pop(&cola));
printf("%d", pop(&cola) );
printf("%d", pop(&cola) );
```

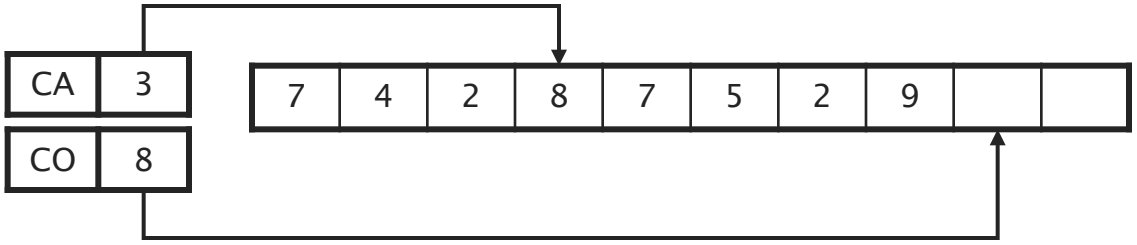


> 7 4 2

Cola: versión estática

- **¿Y si llegamos al final del array?**

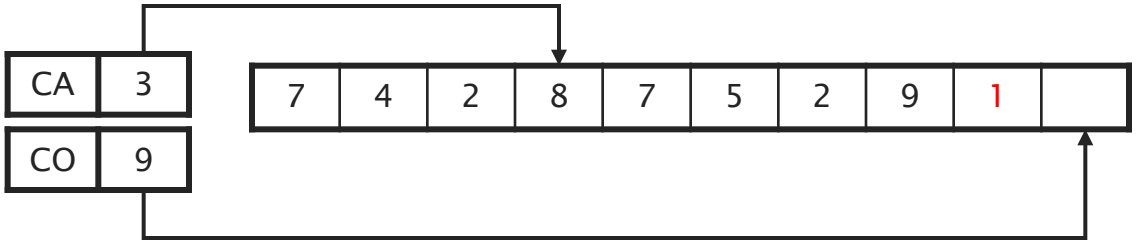
apilar(cola,1);
apilar(cola,8);
apilar(cola,6);
apilar(cola,5);
apilar(cola,1);



Cola: versión estática

- ¿Y si llegamos al final del array?

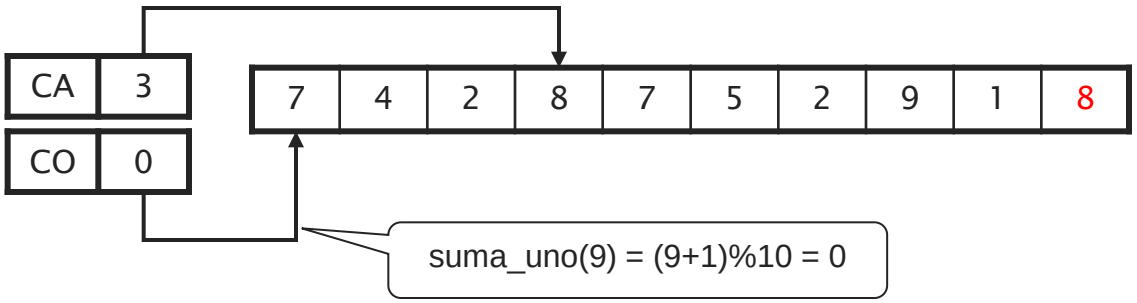
```
apilar(cola,1);
apilar(cola,8);
apilar(cola,6);
apilar(cola,5);
apilar(cola,1);
```



Cola: versión estática

- ¿Y si llegamos al final del array?

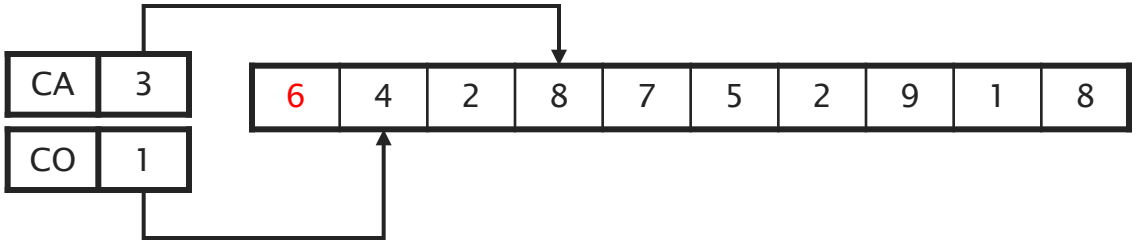
```
apilar(cola,1);
apilar(cola,8);
apilar(cola,6);
apilar(cola,5);
apilar(cola,1);
```



Cola: versión estática

- ¿Y si llegamos al final del array?

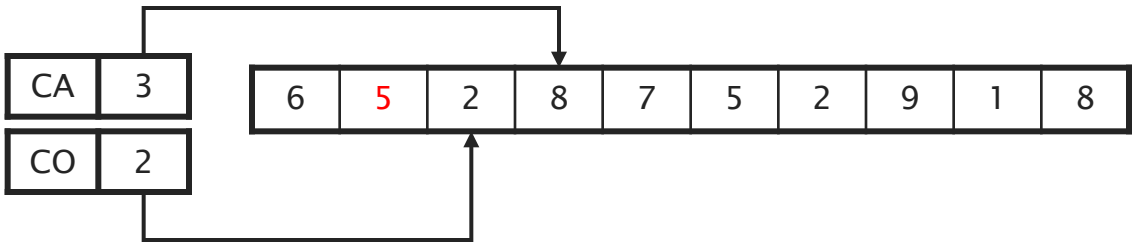
```
apilar(cola,1);
apilar(cola,8);
apilar(cola,6);
apilar(cola,5);
apilar(cola,1);
```



Cola: versión estática

- ¿Y si llegamos al final del array?

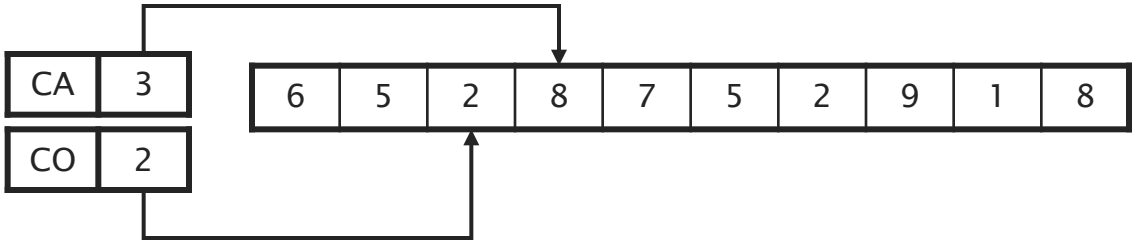
```
apilar(cola,1);
apilar(cola,8);
apilar(cola,6);
apilar(cola,5);
apilar(cola,1);
```



Cola: versión estática

- ¿Y si llegamos al final del array?

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```



Cola: versión estática

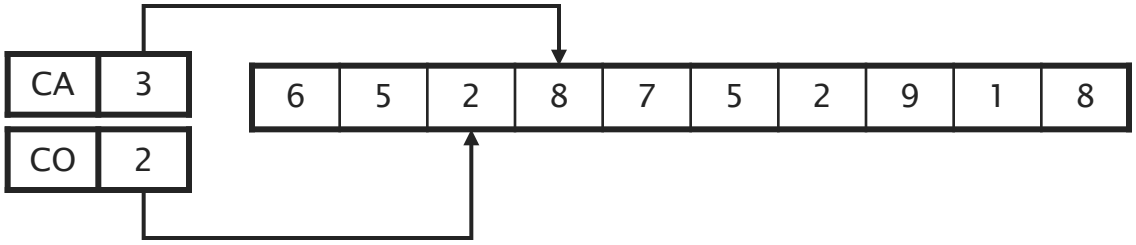
- ¿Y si llegamos al final del array?

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

suma_uno(2)

(2+1)%10

¿3 == 3?

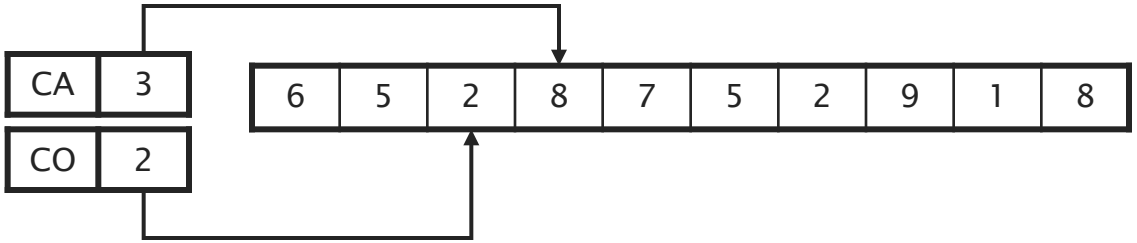


Cola: versión estática

- ¿Y si llegamos al final del array?

```
void apilar(tipo_cola *c, int i){
        if (suma_uno(c->cola) == c->cabeza){
                printf("Error: cola llena \n");
        }else{
                c->valores[c->cola] = i;
                c->cola = suma_uno(c->cola);
    }
}
```

¡¡No se puede realizar la inserción!!



Contenidos

- Introducción
- Funcionamiento de una cola estática
- **Funcionamiento de una cola dinámica**
- Modalidades de colas

Cola: versión dinámica

- Se implementa mediante una estructura autorreferenciada con encadenamiento simple.
- Permite disponer de toda la memoria.
- Los datos se almacenan en un campo.

Cola: versión dinámica

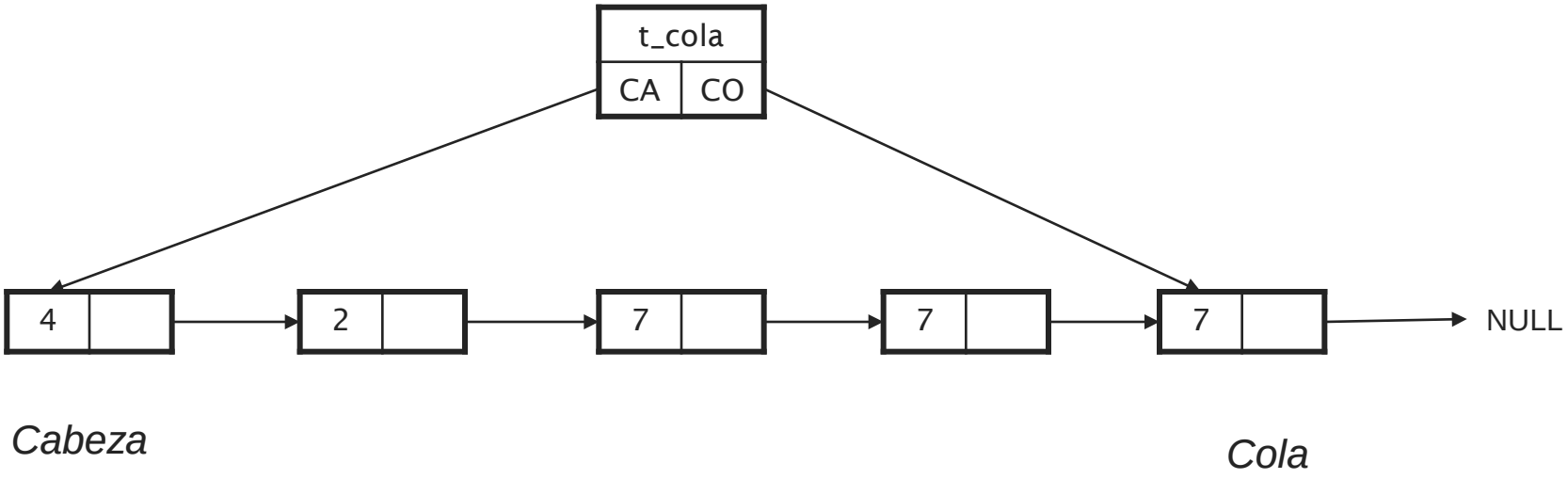
- Tendremos dos registros:
  - **struct n_cola**: almacena los datos y un puntero al siguiente elemento.

```
typedef struct n_cola{
        int valor;
        struct n_cola *sig;
}NODO_COLA, *P_NODO_COLA;
```

  - **struct t_cola**: almacena dos punteros al registro anterior (cola y cabeza).

```
typedef struct t_cola{
        P_NODO_COLA cabeza;
        P_NODO_COLA cola;
}TIPO_COLA;
```

Cola: versión dinámica

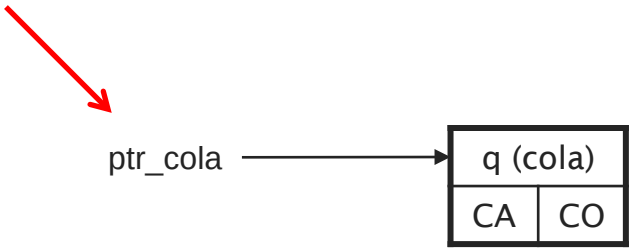


Cola: versión dinámica

Función iniciar

```
void iniciar(TIPO_COLA *q){
        q->cabeza = NULL;
        q->cola = NULL;
}
```

```
TIPO_COLA cola, *ptr_cola;
ptr_cola = &cola;
ptr_cola = iniciar(ptr_cola);
```

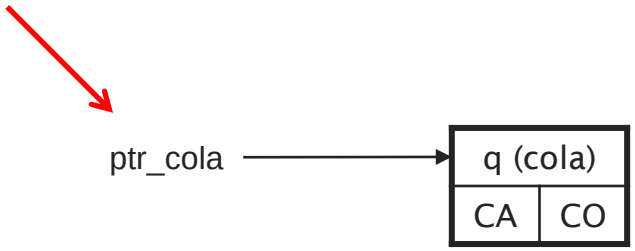


Cola: versión dinámica

Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

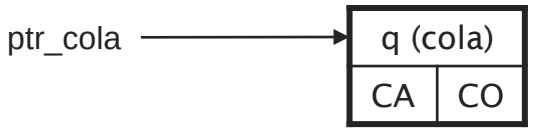


Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

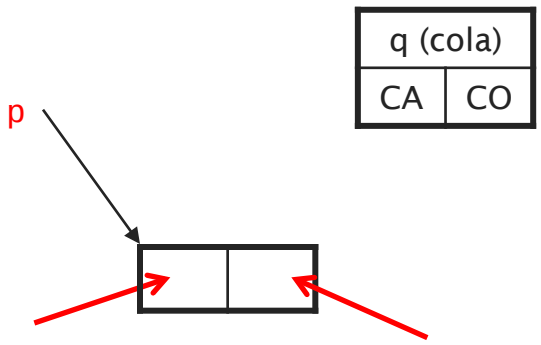
```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```



Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```



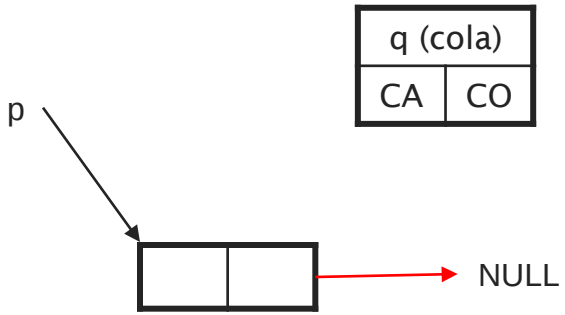

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

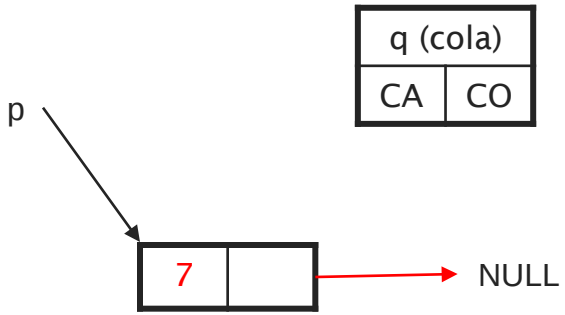


Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

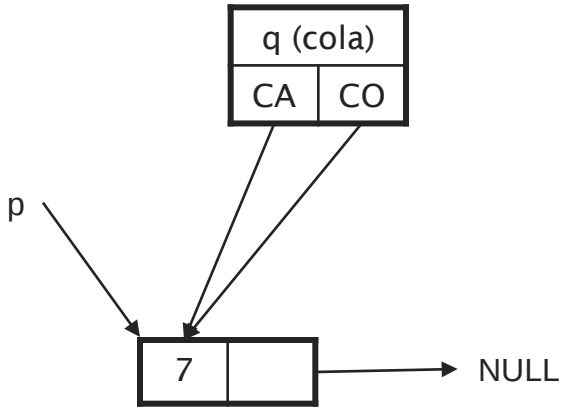
```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```



Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```



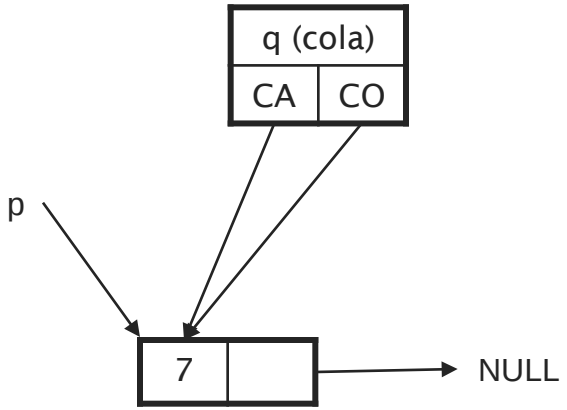

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

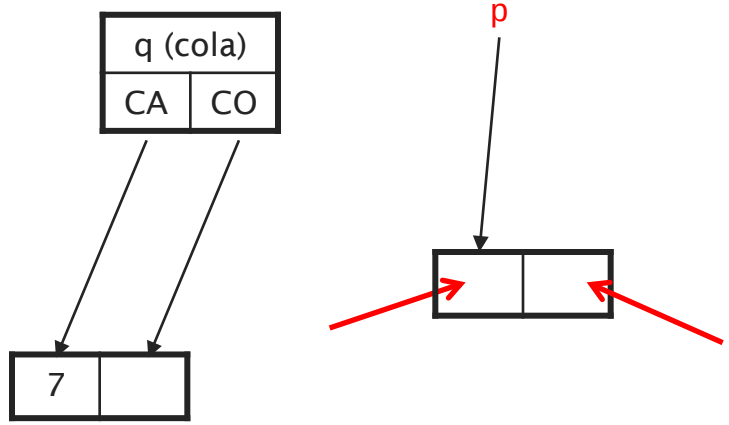


Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

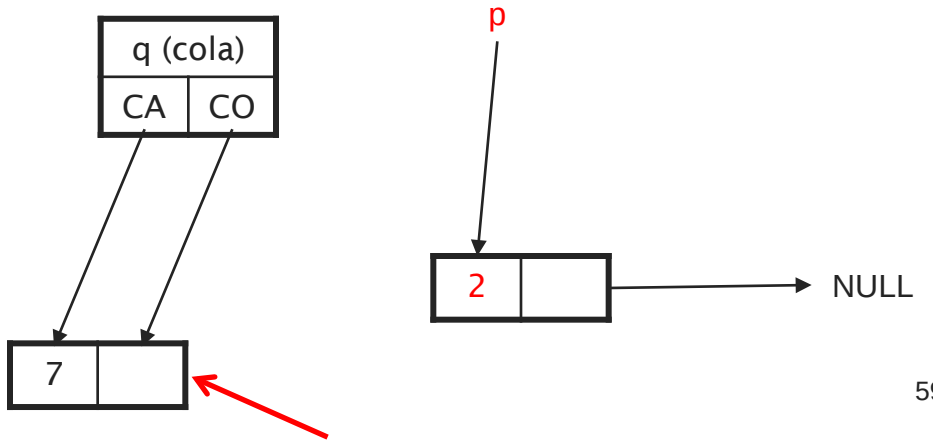


Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

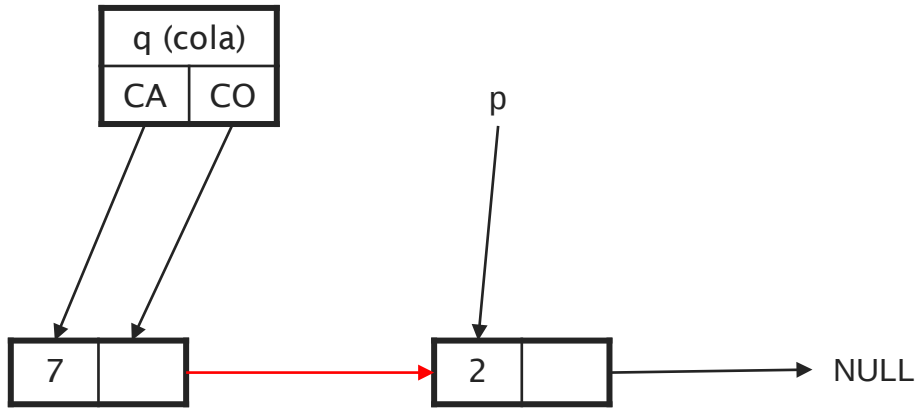


Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

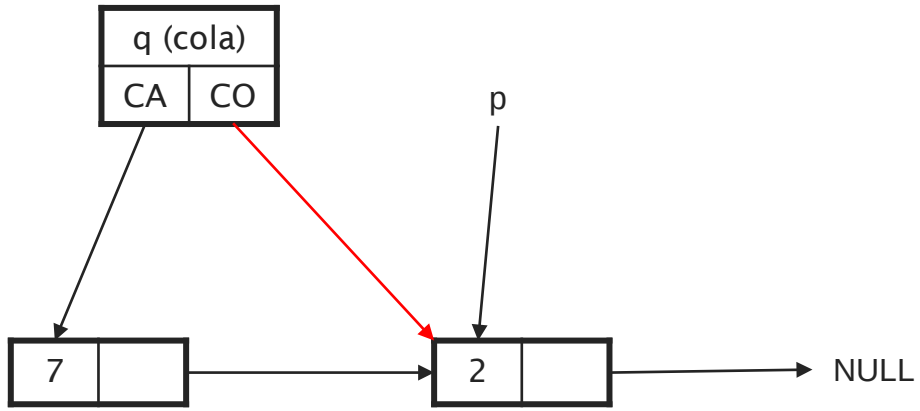


Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

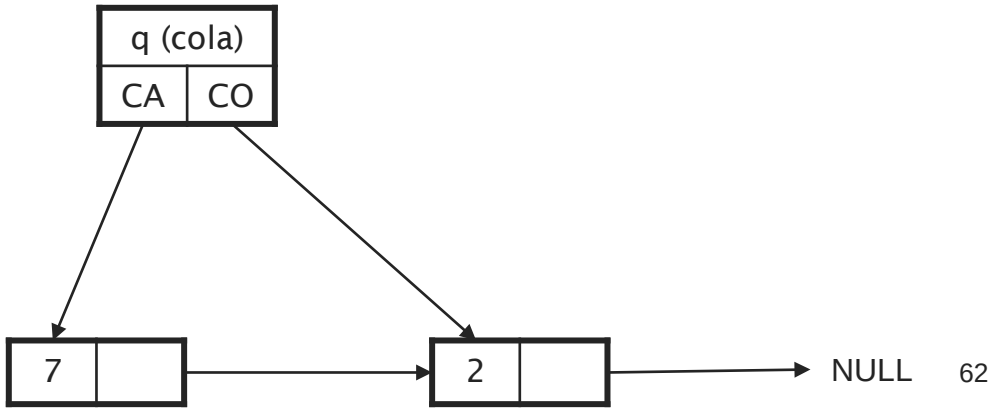


Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

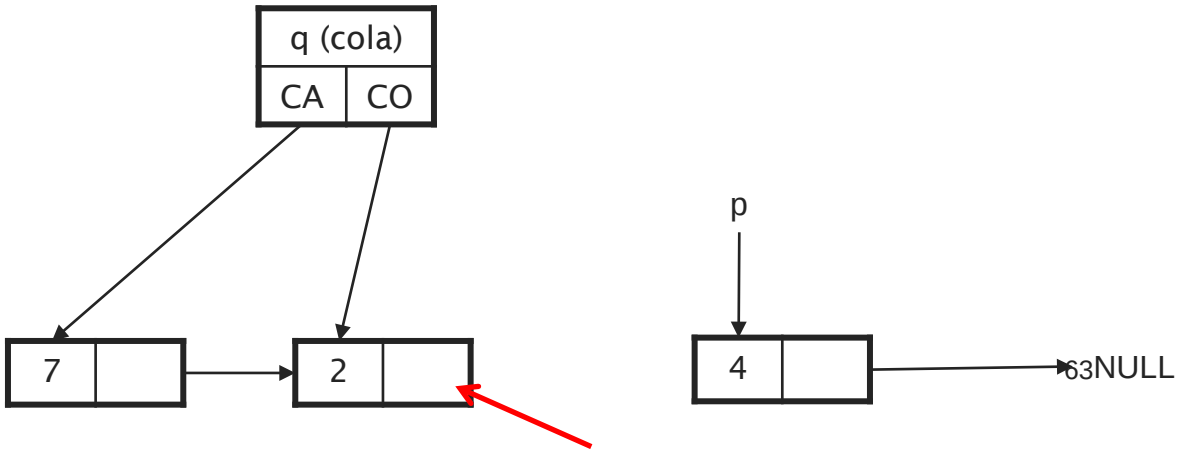


Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

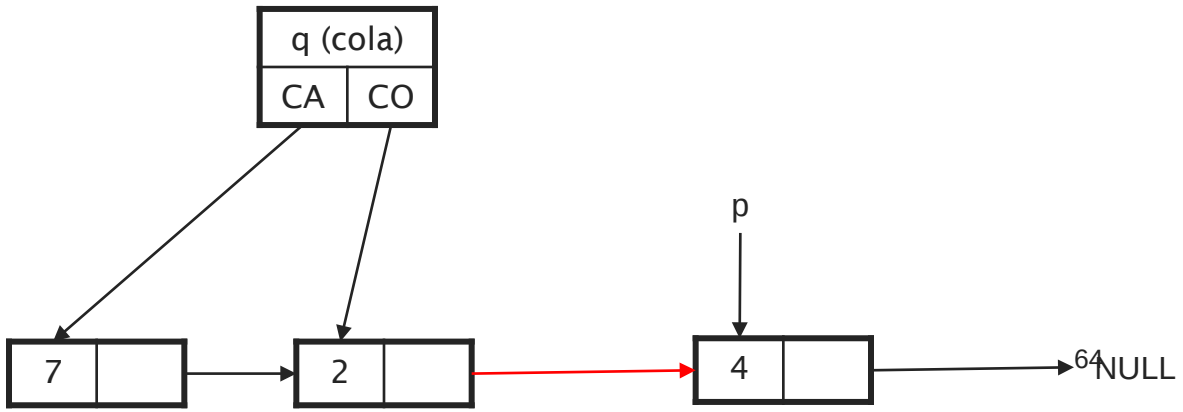


Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

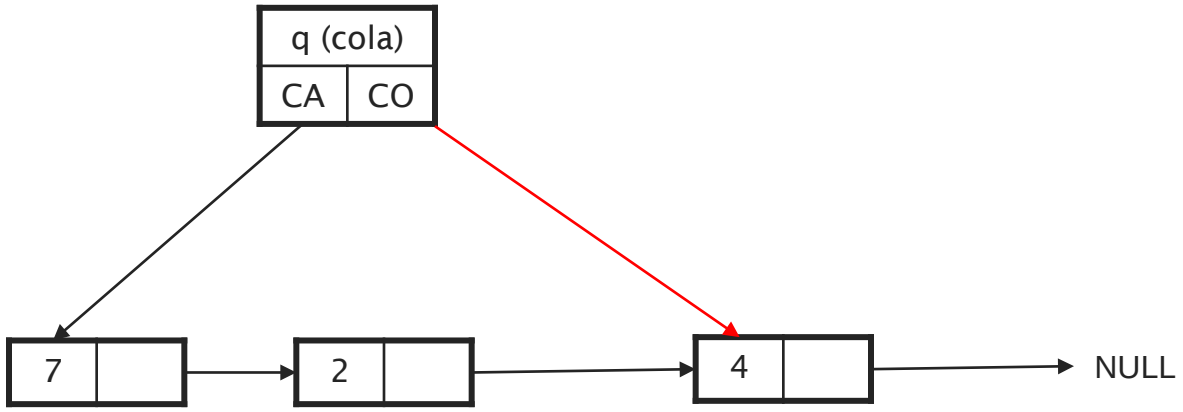


Cola: versión dinámica

- Función push (introducir)

```
void push(TIPO_COLA *q, int i){
        P_NODO_COLA p;
    p = (P_NODO_COLA)malloc(sizeof(NODO_COLA));
        p->sig = NULL;
        p->valor = i;
        if(es_vacia(q)) q->cola = q->cabeza = p;
        else{
                q->cola->sig= p;
                q->cola = p;
        }
}
```

```
push(ptr_cola ,7);
push(ptr_cola ,2);
push(ptr_cola ,4);
```

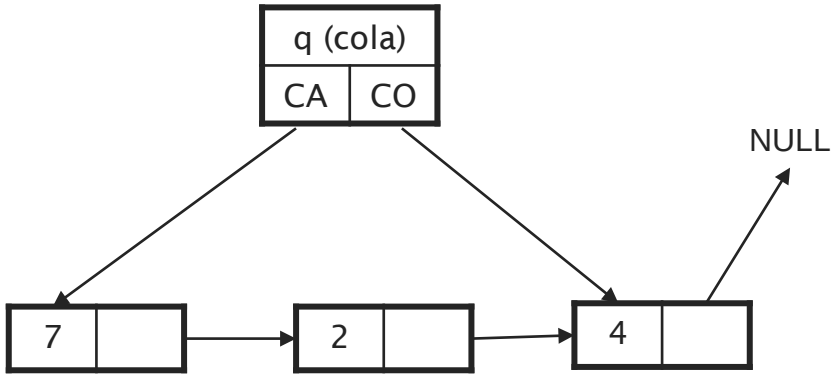


Cola: versión dinámica

Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

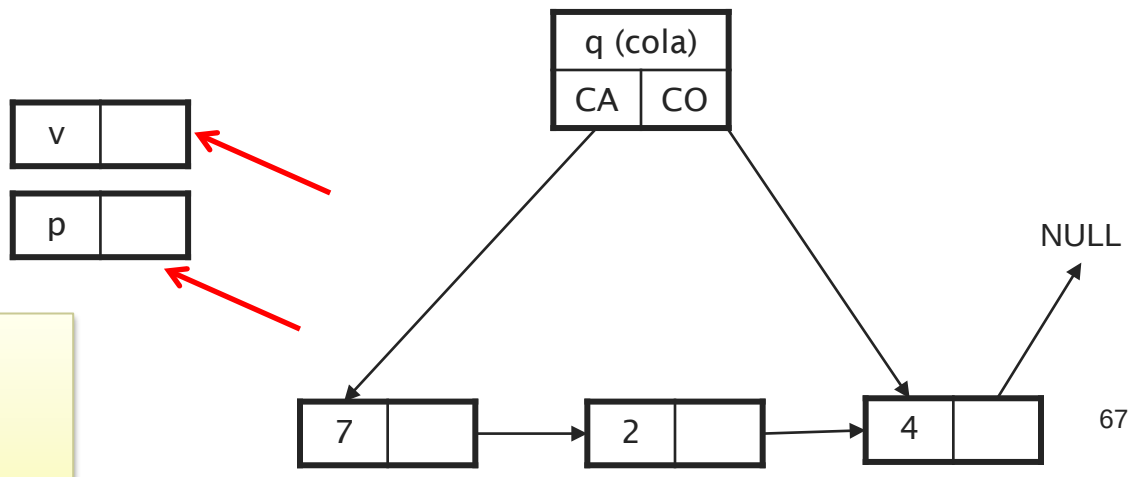
```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```



Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

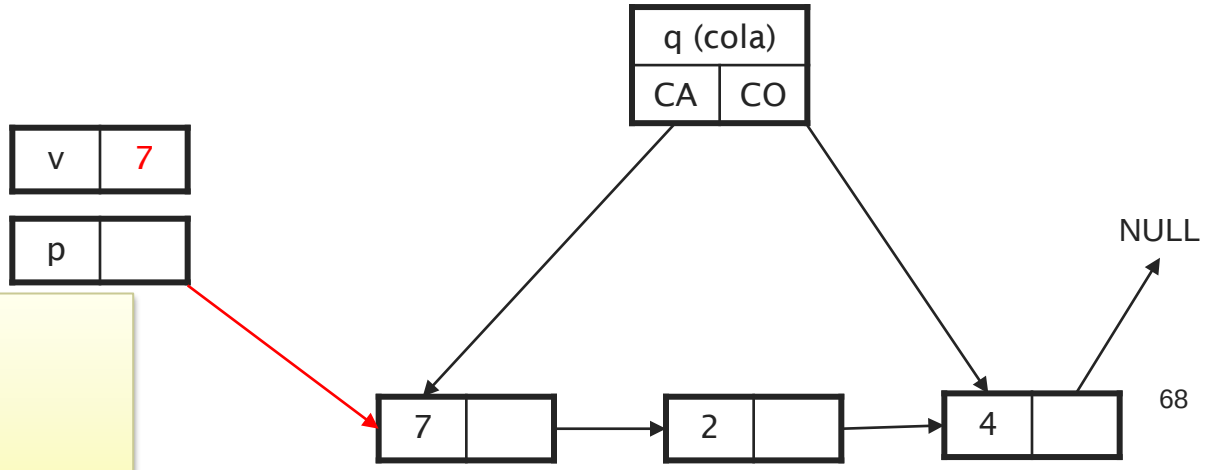



```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

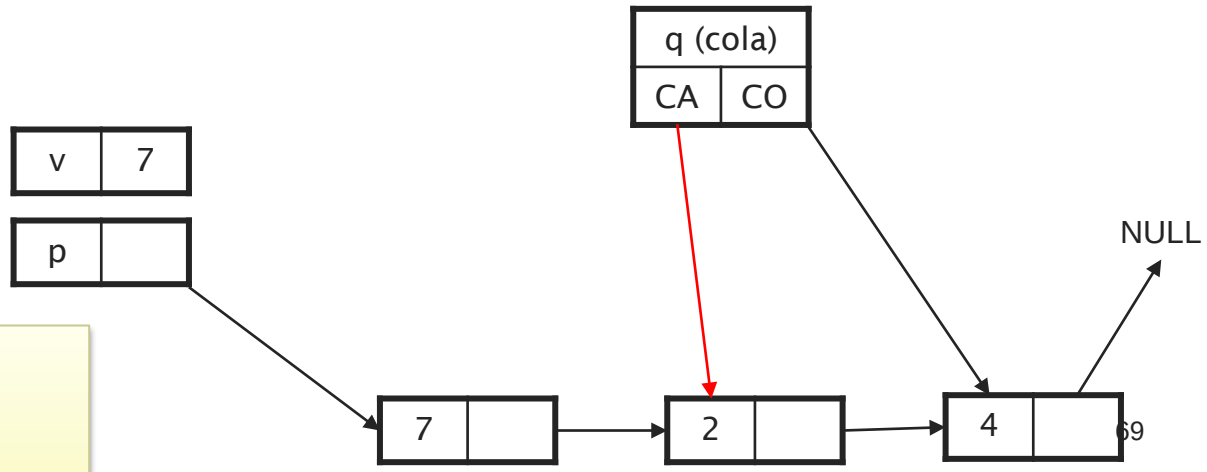



```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

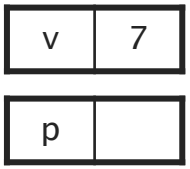


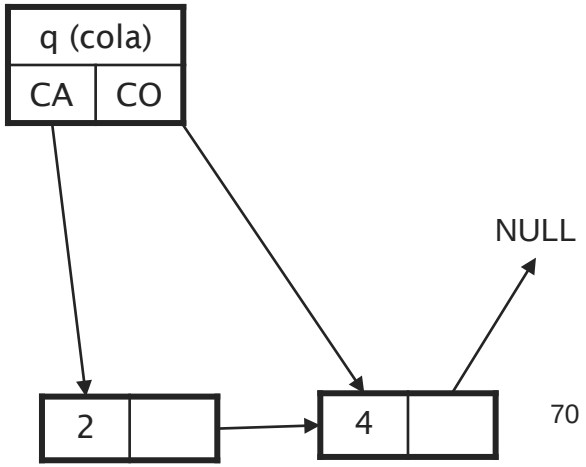

```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```





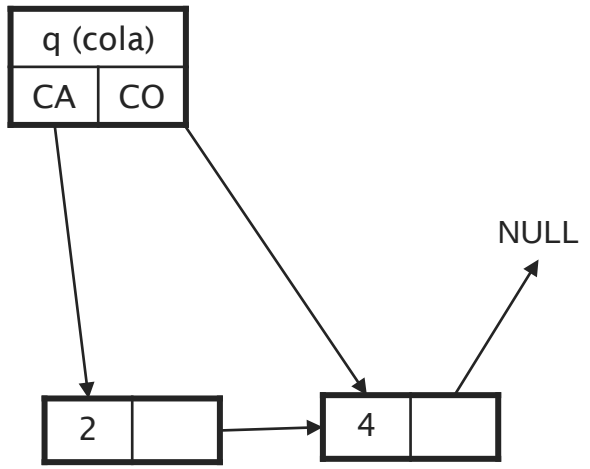

```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

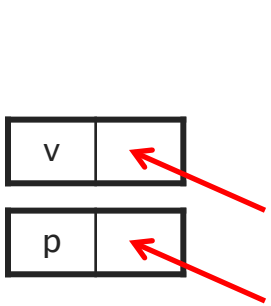
```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

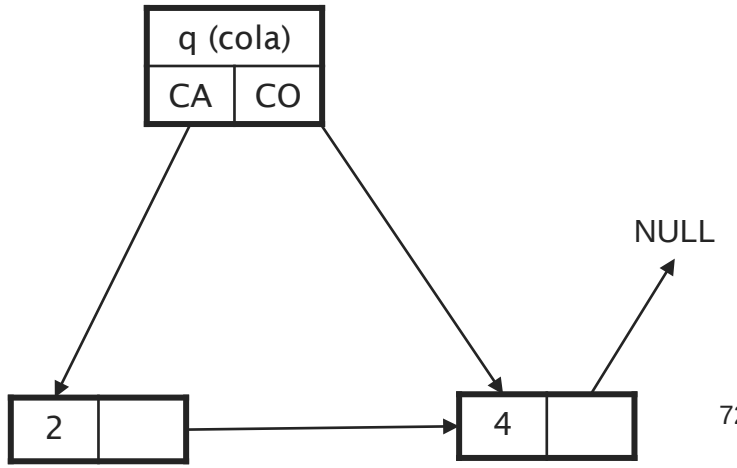


Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```



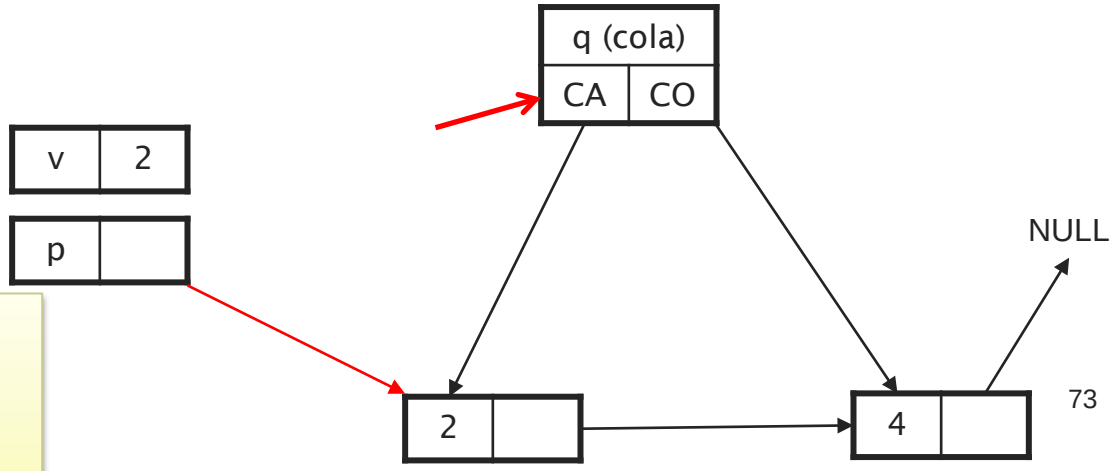



```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

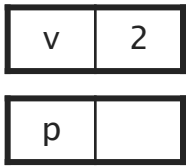


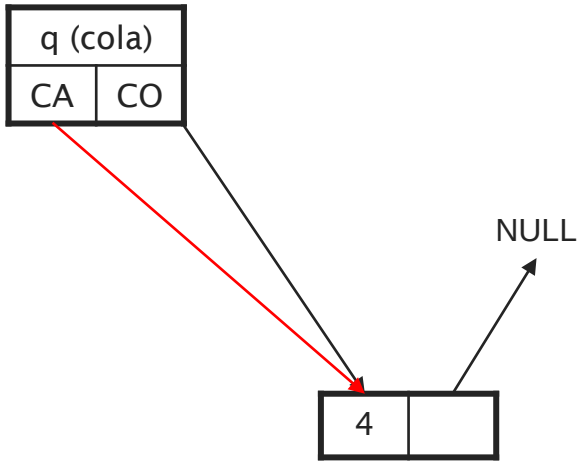

```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```





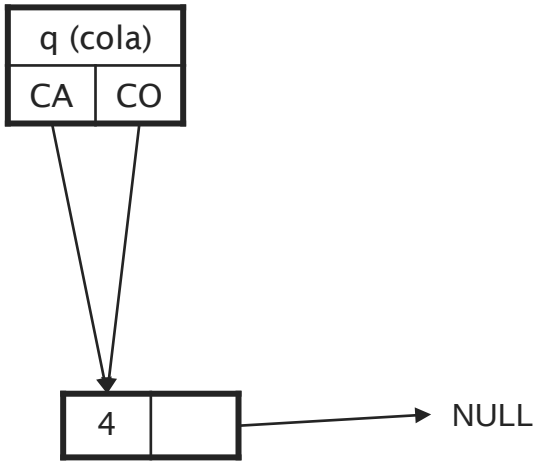

```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

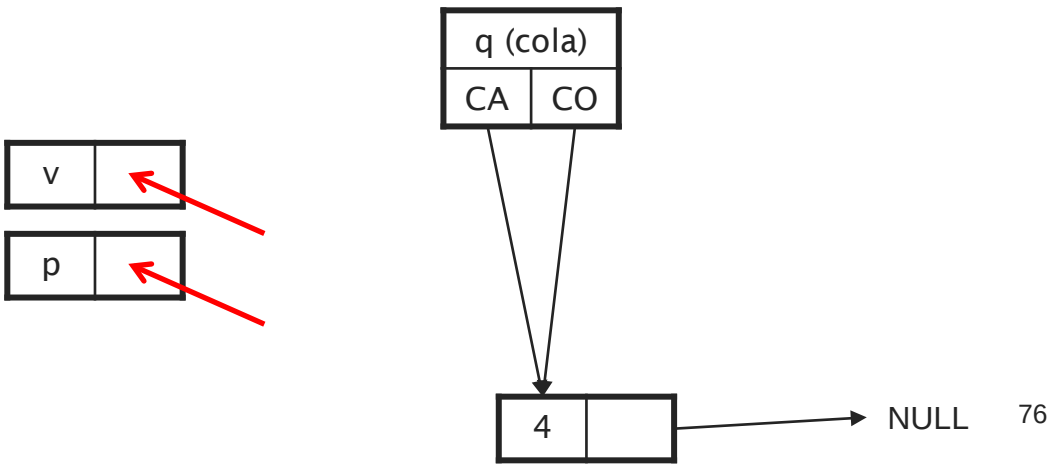
```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```



Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

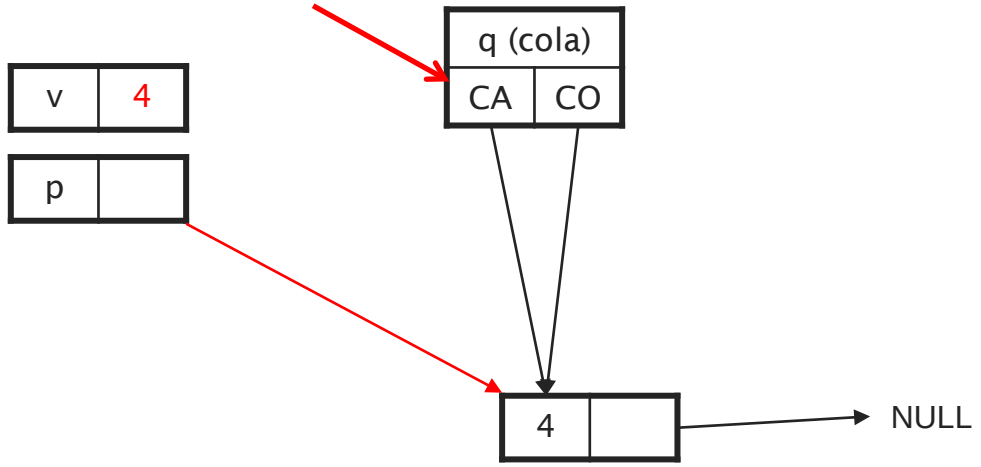



```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

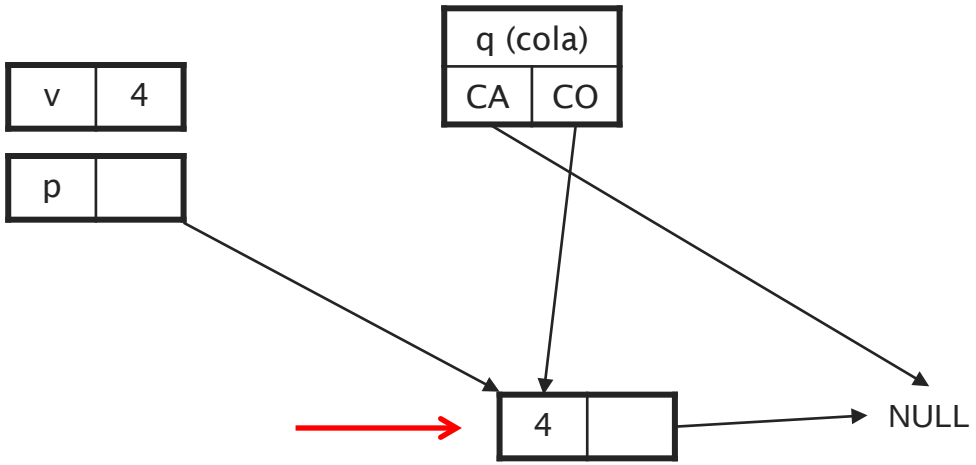



```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

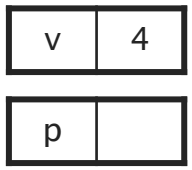


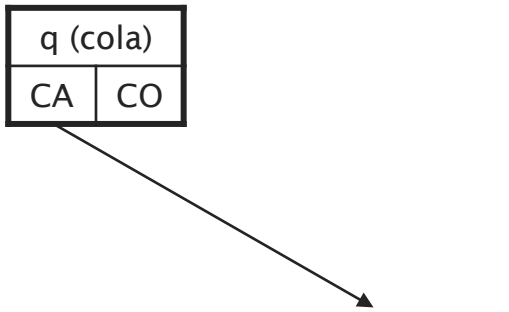

```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```





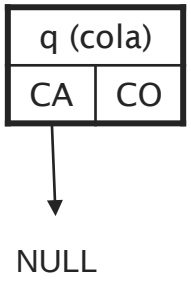

```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```

Cola: versión dinámica

- Función pop (extraer)

```
int pop(TIPO_COLA *q){
        P_NODO_COLA p; int v;
        if(es_vacia(q)){
                v = 0; printf("Error: cola vacia\n");
        }else{
                v = q->cabeza->valor; p = q->cabeza;
                q->cabeza = q->cabeza->sig;
                free(p);
        }
        return v;
}
```

```
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
printf("%d\n", pop(ptr_cola);
```



Contenidos

- Introducción
- Funcionamiento de una cola estática
- Funcionamiento de una cola dinámica
- **Modalidades de colas**
  - **Dicolas**
  - **Colas de prioridad**

Dicolas

- Son un tipo de colas donde las inserciones y eliminaciones tiene lugar por ambos extremos.
- Operaciones sobre una dicola:
  - Insertar elemento en la cabeza
  - Insertar elemento en la cola
  - Eliminar elemento de la cabeza
  - Eliminar elemento de la cola
- Ejercicio:
  - Ampliar el código de la estructura de datos *Cola* visto hasta ahora (versión estática y dinámica) para conseguir el comportamiento de una dicola.

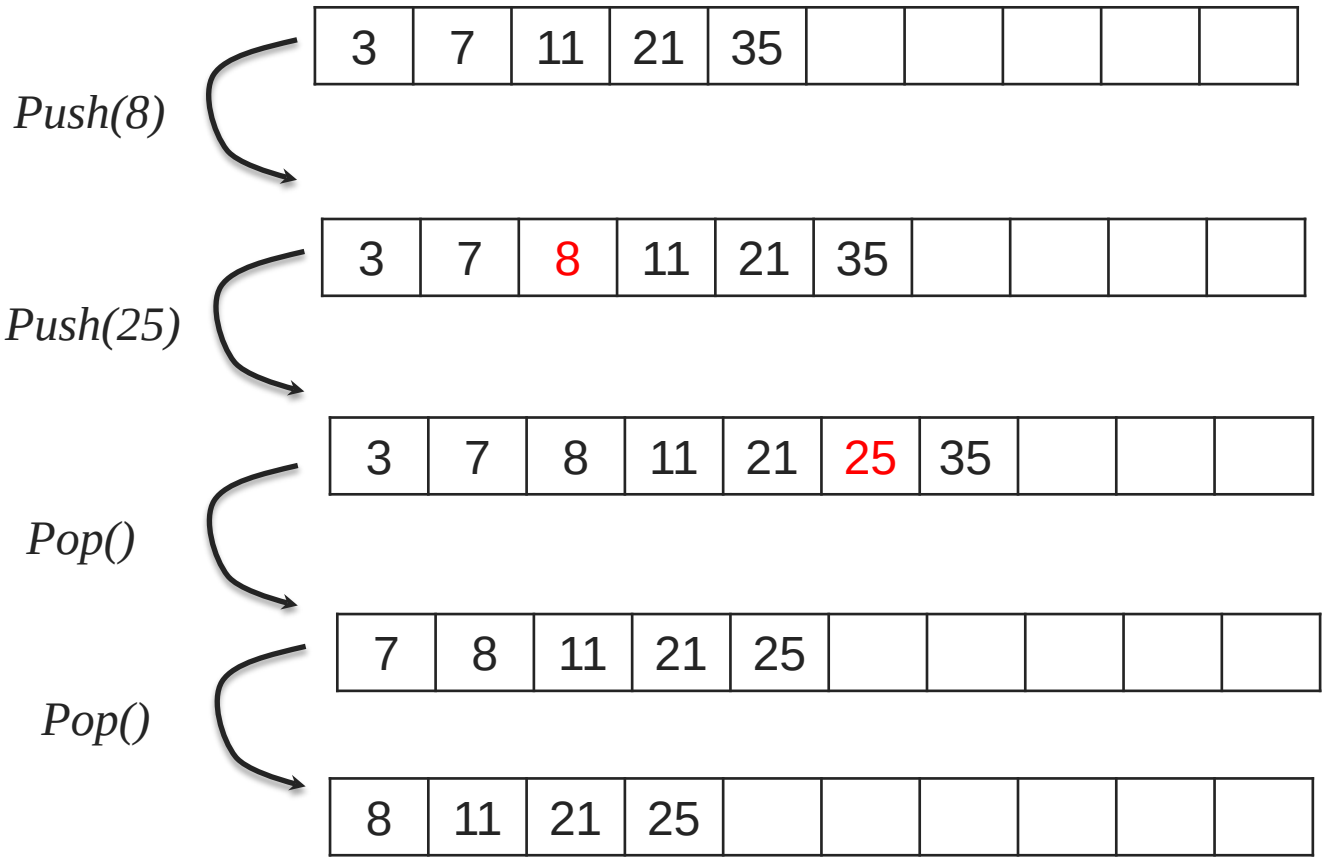
Colas de prioridad

- Las colas de prioridad son un tipo de colas donde los elementos tienen asignada una prioridad
  - El elemento con mayor prioridad es procesado primero
  - El elemento con menor prioridad es procesado el último
  - Dos elementos con la misma prioridad son procesados según el orden en que fueron introducidos en la cola.
- Método básico de representación:
  A. Tener la cola siempre ordenada
     - Los elementos se insertan en el lugar correspondiente según su prioridad
     - Se saca el elemento de la cabeza, es decir, el de mayor prioridad
  B. No tener la cola ordenada
     - Los elementos se insertan al final de la cola.
     - Cuando se va a sacar un elemento, se debe buscar el de mayor prioridad.

Colas de prioridad

(A) Tener la cola siempre ordenada

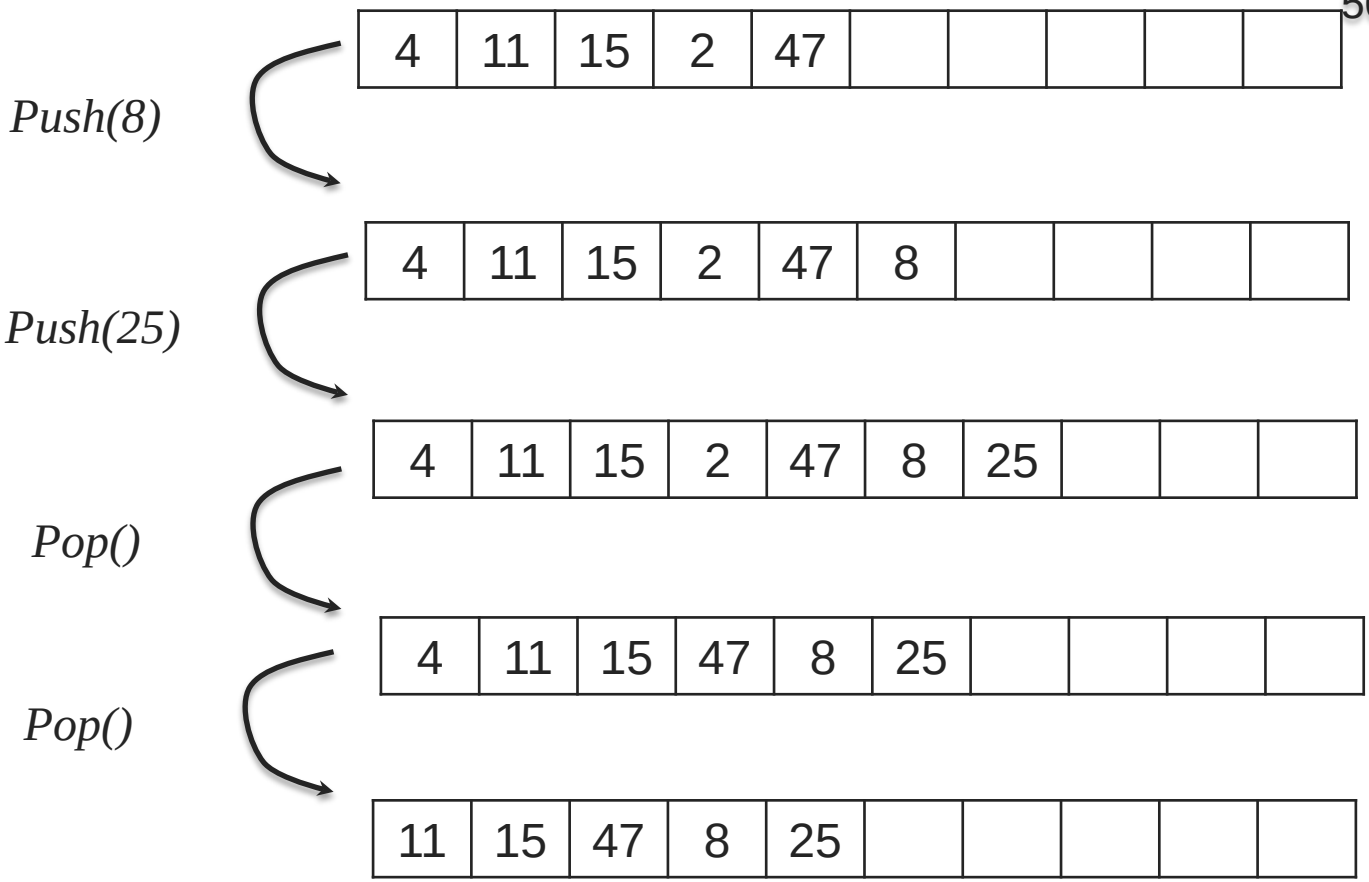
Mayor prioridad: 0
Menor prioridad: 50



Colas de prioridad

(B) No tener la cola ordenada

Mayor prioridad: 0
Menor prioridad: 50



Bibliografía

- King, K.N. **C Programming. A modern approach.** 2ªed. Ed. W.W. Norton & Company. Inc. 2008.
- Khamtane Ashok**. Programming in C**. Ed. Pearson. 2012.
- Ferraris Llanos, R. D. **Fundamentos de la Informática y Programación en C**. Ed. Paraninfo. 2010.