Laboratorio de Multimedia e Internet

PILAS Y COLAS

Instructor: Cruz Matías Yuridia Elizabeth

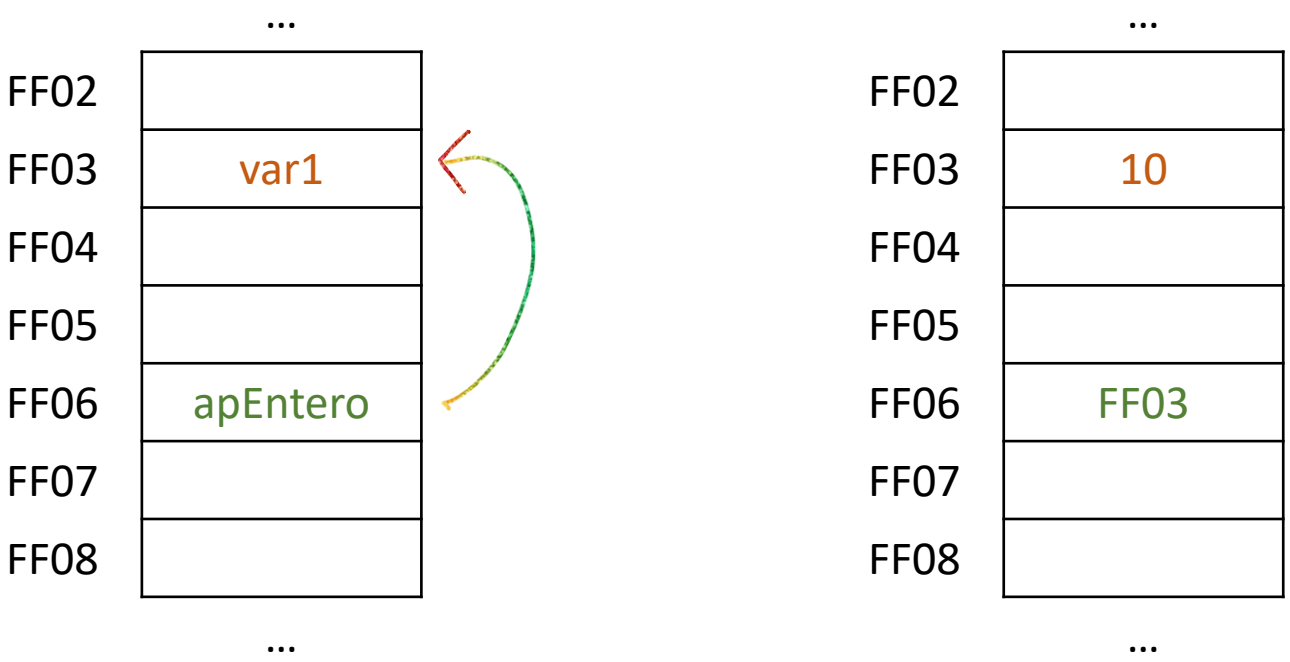
Estructuras de Datos

- ESTÁTICAS
  - SIMPLES O PRIMITIVAS → BOOLEAN, CHAR, INT, ETC...
  - COMPUESTAS → ARREGLOS, CONJUNTOS, STRINGS, ETC...
- DINÁMICAS
  - LINEALES → PILAS, COLAS, LISTAS
  - NO LINEALES → ARBOLES Y GRAFOS

Conceptos Básicos

APUNTADOR

Variable que permite almacenar direcciones de memoria de otras variables



Sintaxis

Tipo * nombreVariable

Tipo nombreVariable *

* Operador indirección que hace referencia al contenido de dirección que almacena la variable

& Operador que permite obtener la dirección de memoria asignada a esa variable

Ejemplo 1

Crear un programa con una variable, un apuntador a la variable. posteriormente modificar el valor de la variable usando el apuntador

```c
#include <stdio.h>

int main(){
    int var1 = 10;
    int *apEntero = &var1;

    printf ("Valor de var1 : %d", var1);

    //*apEntero = 30;

    //printf ("Valor de var1 : %d", var1);

    printf ("\n\n");

    return 0;

}
```

```c
#include <stdio.h>

int main(){
    int var1 = 10;
    int *apEntero = &var1;

    printf ("Valor de var1 : %d", var1);

    *apEntero = 30;

    printf ("Valor de var1 : %d", var1);

    printf ("\n\n");

    return 0;

}
```

ESTRUCTURAS

Conjunto de datos de diferente tipo que se almacenan de forma consecutiva en memoria

Sintaxis

```
Struct nombreEstructura{
        tipo campo1;
        tipo campo2;
        …
        tipo campo3;
}; listaDeVariables;
```

Ejemplo 2

Crear un programa que sume 2 números complejos usando una estructura.

```c
#include <stdio.h>

struct complejo{
    double real;
    double ima;
}n1, n2, res;

int main(){
    //struct complejo n1, n2, res;

    puts ("introduce numero complejo 1");
    scanf("%lf, %lf", &n1.real, &n1.ima);

    puts ("introduce numero complejo 2");
    scanf("%lf, %lf", &n2.real, &n2.ima);

    res.real=n1.real + n2.real;
    res.ima=n2.ima + n2.ima;
    printf("El resultado es: %lf i + %lf j",res.real,res.ima);
  return 0;
}

```

TYPEDEF

El lenguaje C ofrece al programador la posibilidad de definir nuevos tipos de datos. Los nuevos tipos definidos pueden utilizarse de la misma forma que los tipos de datos predefinidos por el lenguaje.

Sintaxis

```
typedef tipoDeDatos nombreNuevoTipo;
```

Ejemplo 3

Realizar la sima de dos números complejos

```c
#include <stdio.h>

//typedef
struct complej{
    double real;
    double ima;
}; //complejo;

typedef struct complej complejo;

int main(){
    complejo n1, n2, res;

    puts ("introduce numero complejo 1");
    scanf("%lf, %lf", &n1.real, &n1.ima);

    puts ("introduce numero complejo 2");
    scanf("%lf, %lf", &n2.real, &n2.ima);

    res.real=n1.real + n2.real;
    res.ima=n1.ima + n2.ima;
    printf("El resultado es: %lf i + %lf j", res.real, res.ima);
  return 0;
}
```

FUNCIONES

Es una porción de programa, identificable mediante un nombre, que realiza determinadas tareas bien definidas por un grupo de sentencias sobre un conjunto de datos

Sintaxis

```
tipo nombreFunción(tipo1 param1, ..., tipoN paramN);

tipo nombreFunción(tipo1 param1, ..., tipoN paramN)
{
cuerpo
}
```

Ejemplo 4

Realizar la distancia de dos puntos en el plano cartesianos recordando la fórmula

$$d=\sqrt{(x1-x2)^2+(y1+y2)^2+(z1+z2)^2}$$

```c
#include <stdio.h>
#include <math.h>

struct coordenadas
{
    float x;
    float y;
    float z;
};

float distancia(struct coordenadas a, struct coordenadas b);

int main()
{
    struct coordenadas punto_a = { 3.5, 2.5, 1.5 };
    struct coordenadas punto_b = { 5.3, 3.1, 6.3 };
    float d;

    d = distancia(punto_a, punto_b);

    printf("%f\n", d);

    return 0;
}

float distancia(struct coordenadas a, struct coordenadas b)
{
    return sqrtf(pow(a.x - b.x, 2.0) +  pow(a.y - b.y, 2.0) + pow(a.z - b.z, 2.0));
}
```

MANEJO DE MEMORIA EN TIEMPO REAL

STACK

PILA

HEAP

MONTICULO

MEMORIA ESTÁTICA

Es el espacio en memoria que se crea al **declarar variables** de cualquier tipo de dato (primitivas [int,char...] o derivados [struct,matrices,punteros...]). La memoria que estas variables ocupan no puede cambiarse durante la ejecución y tampoco puede ser liberada manualmente.

MEMORIA DINÁMICA

Es memoria que se reserva en **tiempo de ejecución**. El programador es encargado de liberar esta memoria cuando no la utilice más.

HEAP

8

MEMORIA
DINÁMICA

STACK

2 FF02

MEMORIA
ESTÁTICA

FUNCIONES PARA EL MANEJO DE MEMORIA

malloc()

calloc()

realloc()

free()

Librería

stdlib.h

MALLOC

Reserva un bloque de memoria de tamaño size bytes y retorna la dirección del bloque de memoria.

Sintaxis

```
void *malloc(size_t size);
```

CALLOC

Funciona de modo similar a **malloc**, pero además de reservar memoria, inicializa a 0 la memoria reservada.

Sintaxis

```
void *calloc(size_t n, size_t size);
```

REALLOC

Redimensiona el espacio asignado de forma dinámica anteriormente a un puntero.

Sintaxis

```
void *realloc(void *ptr, size_t size);
```

FREE

Sirve para liberar memoria que se asignó dinámicamente.

Sintaxis

```
void free(void *puntero);
```

Ejemplo 5

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a,*b;
    a=2;
    b=(int*)malloc(sizeof(int));
    *b=8;

    printf("%i \n",*b);
    printf("%i \n",b);
    printf("%i \n",a);
    free(b);
    b=(int*)malloc(sizeof(int));
    printf("%i \n",*b);
    free(b);
    return 0;
}

```

LISTAS

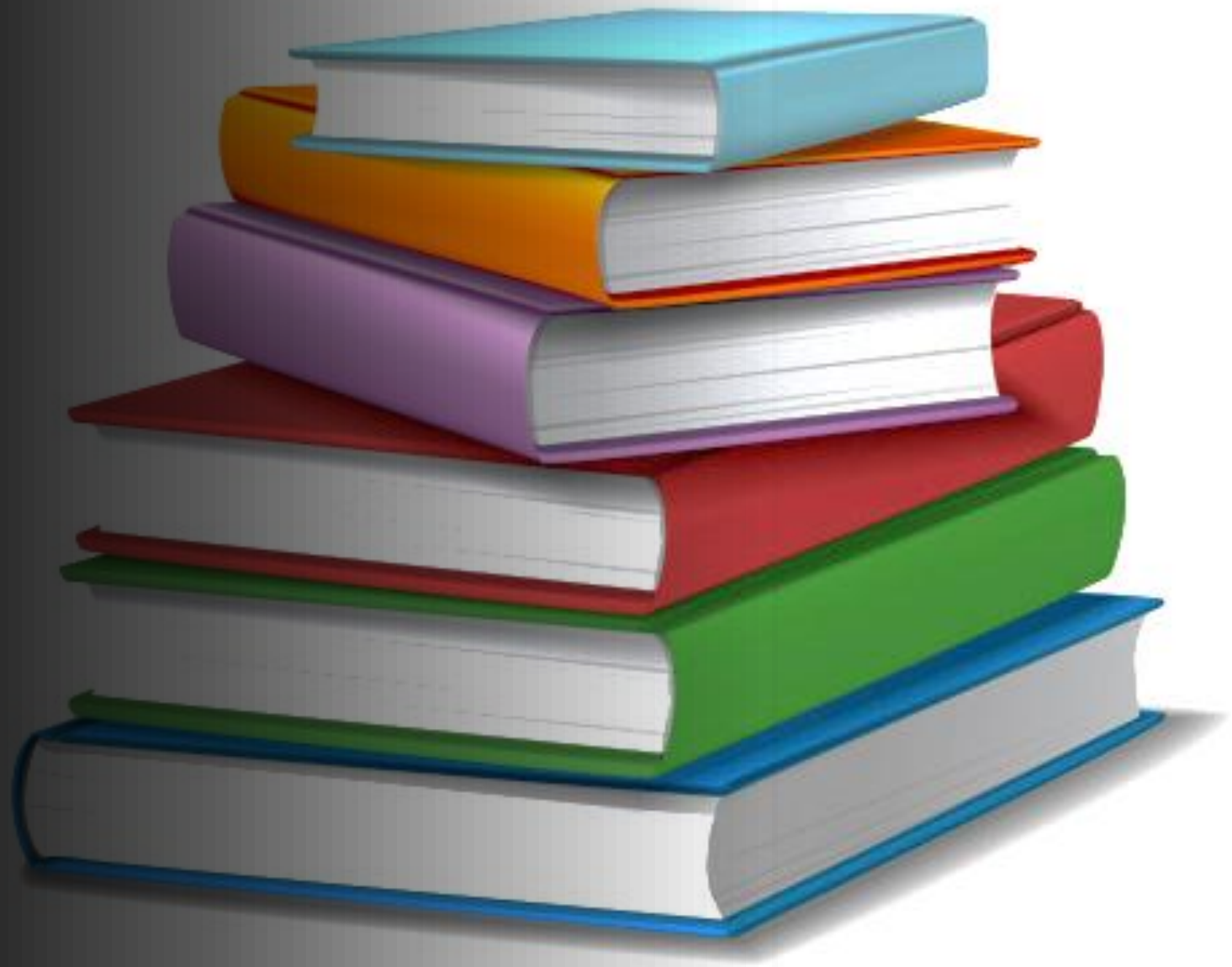
Definición

Son secuencias de elementos de un tipo determinado, se puede representar como una sucesión de elementos.

$a_0, a_1, a_2, a_3, \ldots, a_{n-1}$ $\quad n \geq 0$

n= es el número de elementos de la lista

PILA



Definición

Tipo especial de lista donde las inserciones y supresiones tienen lugar en un extremo llamado tope o cima.

También llamado Lista LIFO

Last In First Out

Último en llegar primero en salir

Implementación de la Pila

Representación de pilas se realizará
con arreglos unidimensionales

Importante definir:

- Número Máximo de Elementos
- Una variable auxiliar TOPE para saber cual es el último elemento de la pila.

Se debe reservar el espacio en memoria con anticipación.

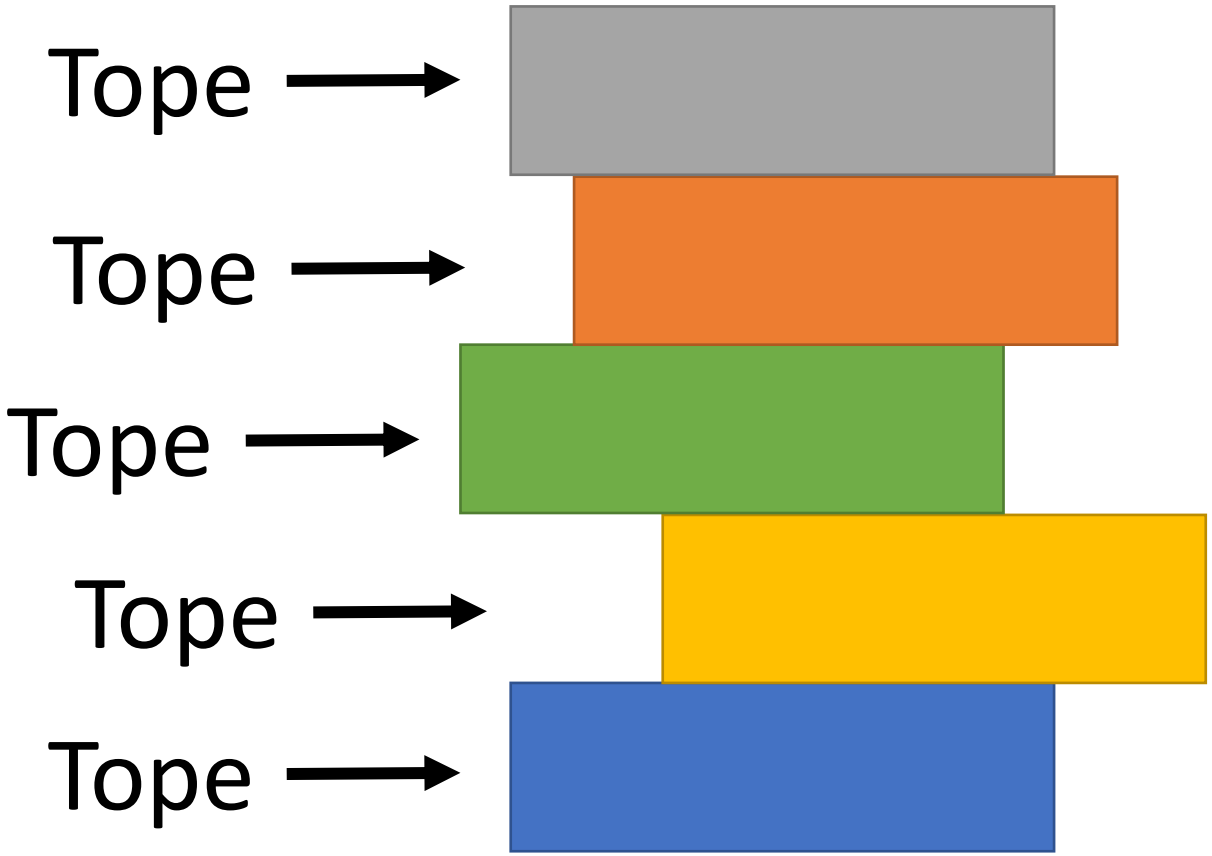
```
struct Pila{
    int tope;
    int dato[valor];
};
```

Operaciones básicas

Pop (Sacar)

Suprime y devuelve el elemento superior de la pila.

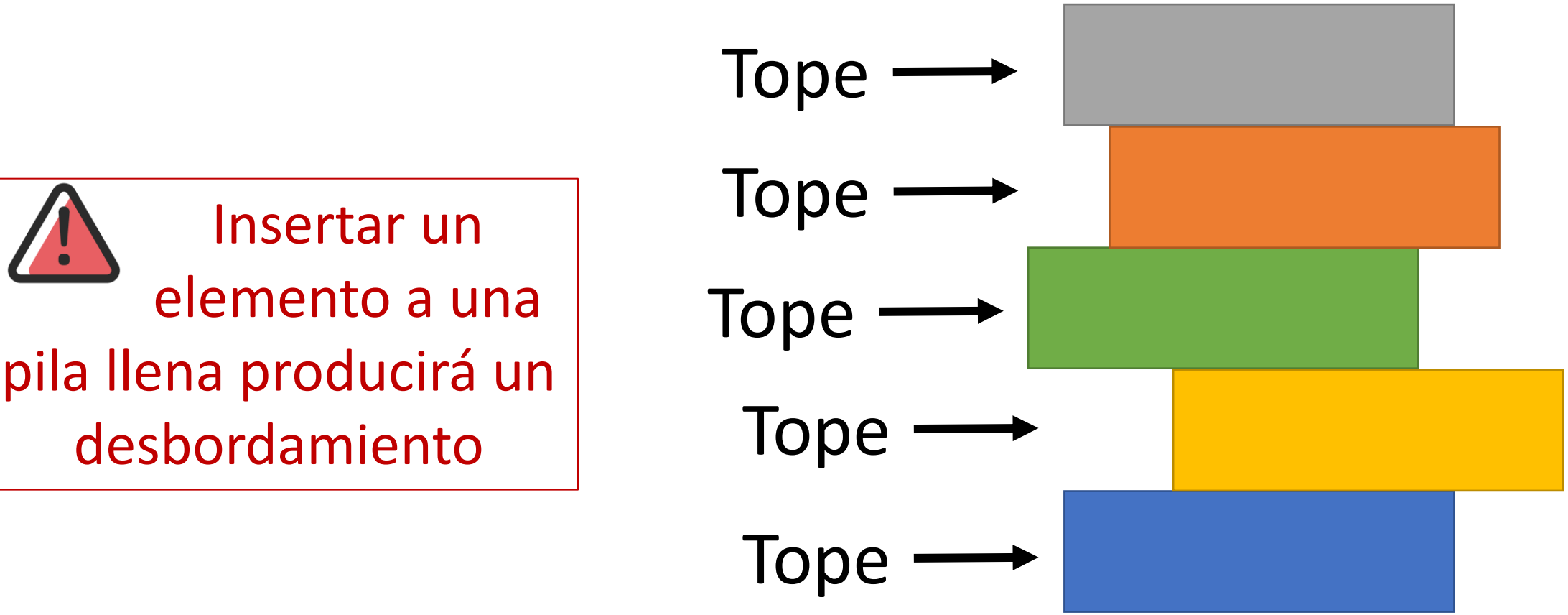
Eliminar un elemento de una pila vacía produce un desbordamiento



```
int pop(struct Pila*pila) {
    pila->tope--;
    return (pila->dato[Pila->tope]);
}
```

Push (Meter)

Inserta un elemento en la parte superior de la pila.



Insertar un elemento a una pila llena producirá un desbordamiento

```
void push (struct Pila*pila, int x){
    pila->dato[pila->tope]=x;
    pila->tope++;
}
```

Otras Operaciones

crearPila Inicializa la pila en 0

```
void creaPila(struct Pila* pila){
      pila->tope=0;
}
```

vacia( )

Función que regresa verdadero si la pila es vacía

```
int vacia(struct Pila* pila){
        if(pila->tope==0)
                return 1;
        else
                return 0;
}
```

llena( ) Regresa verdadero si la pila esta llena

```
int llena(struct Pila* pila, int tam){
        if(pila->tope==tam)
                return 1;
        else
                return 0;
}
```

tope( ) Regresa el tope de la pila

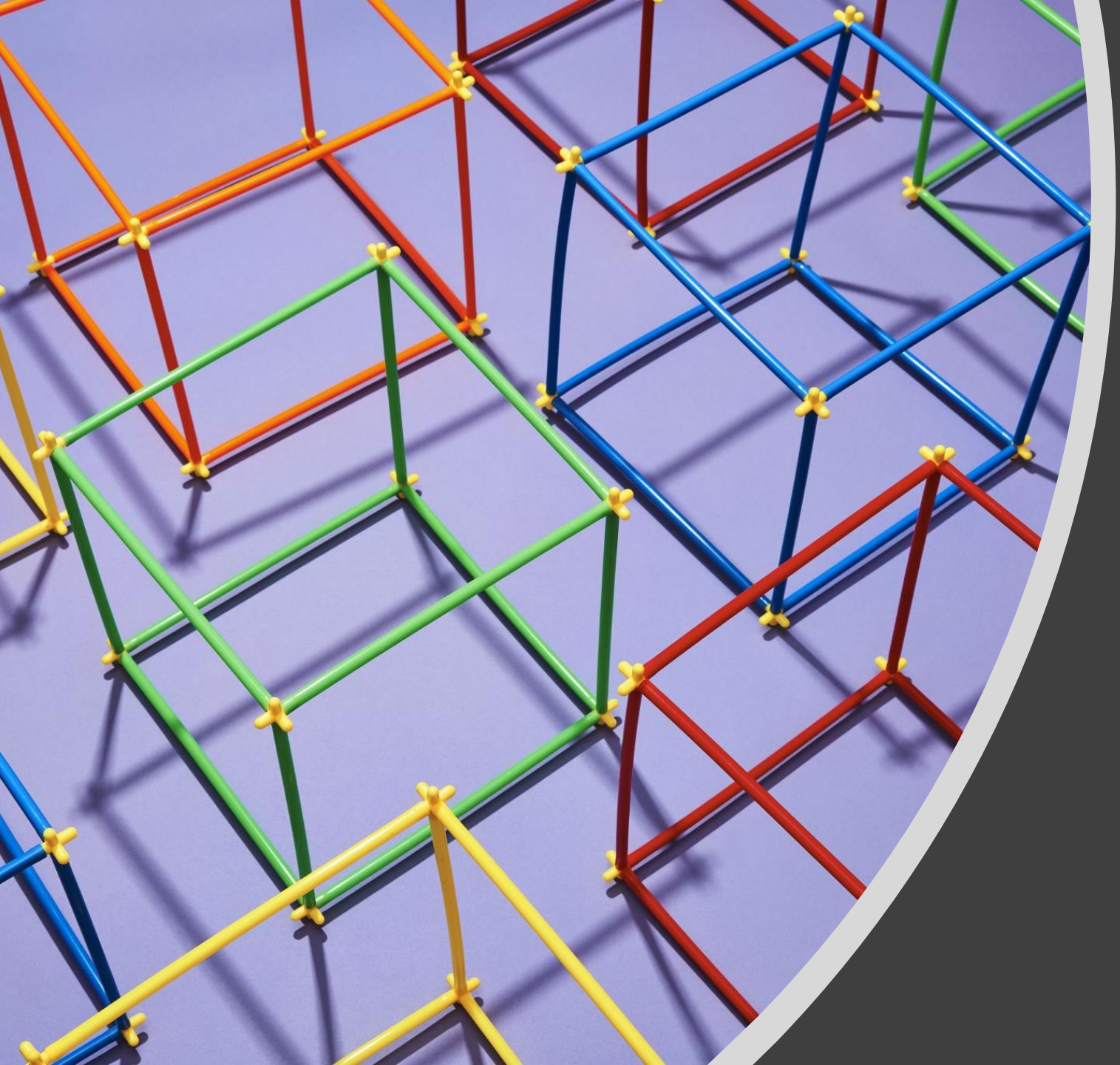
```
int topeP(t_Pila * pila){
    return(pila->tope);
}
```

vaciar( ) Elimina todos los elementos de las pila

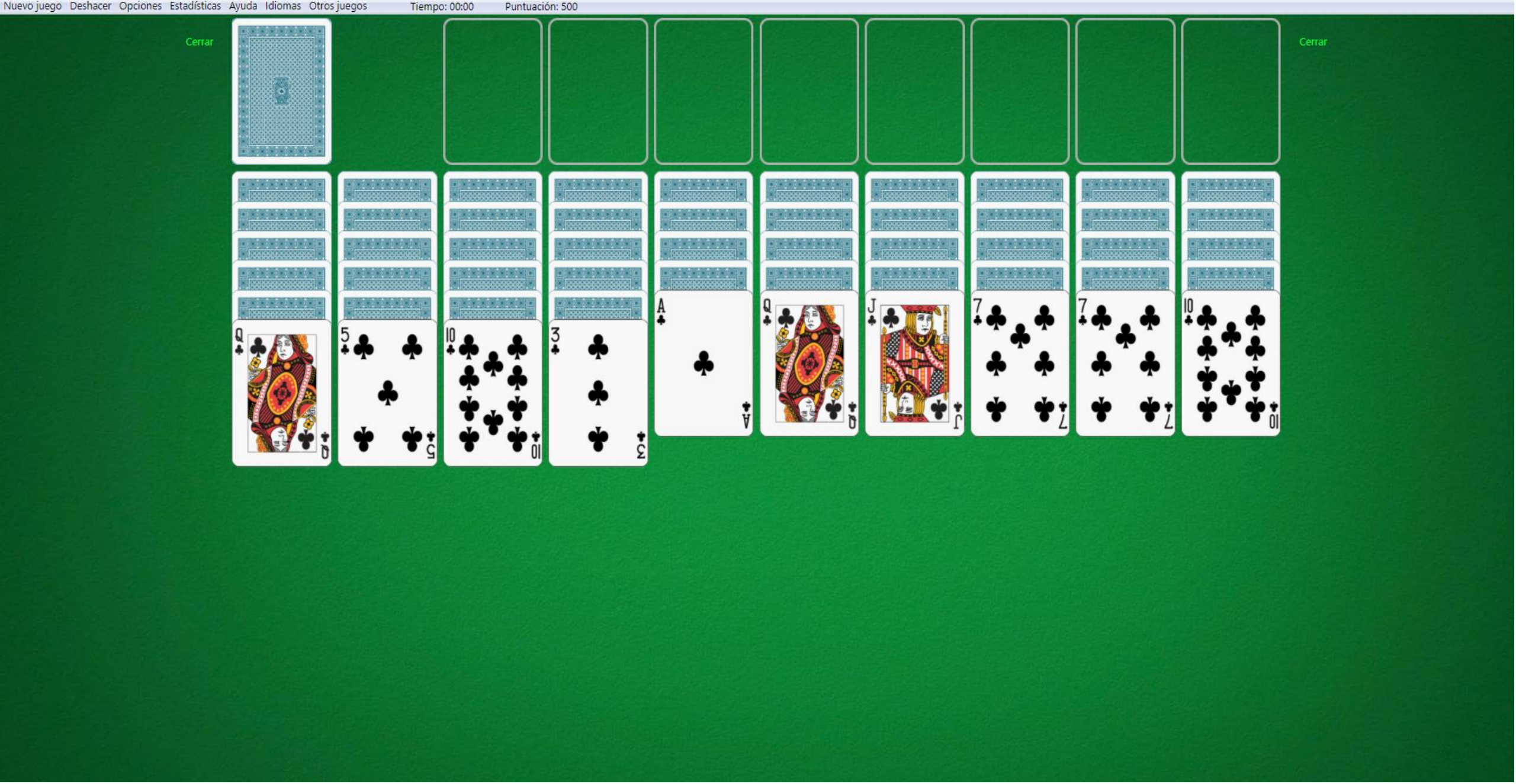
```
void vaciar(t_Pila * pila,int tam){
      int i;
      for(i=0;i<tam;i++){
            pila->tope--;
      }
}
```

mostrar( ) Imprime en pantalla toda la pila

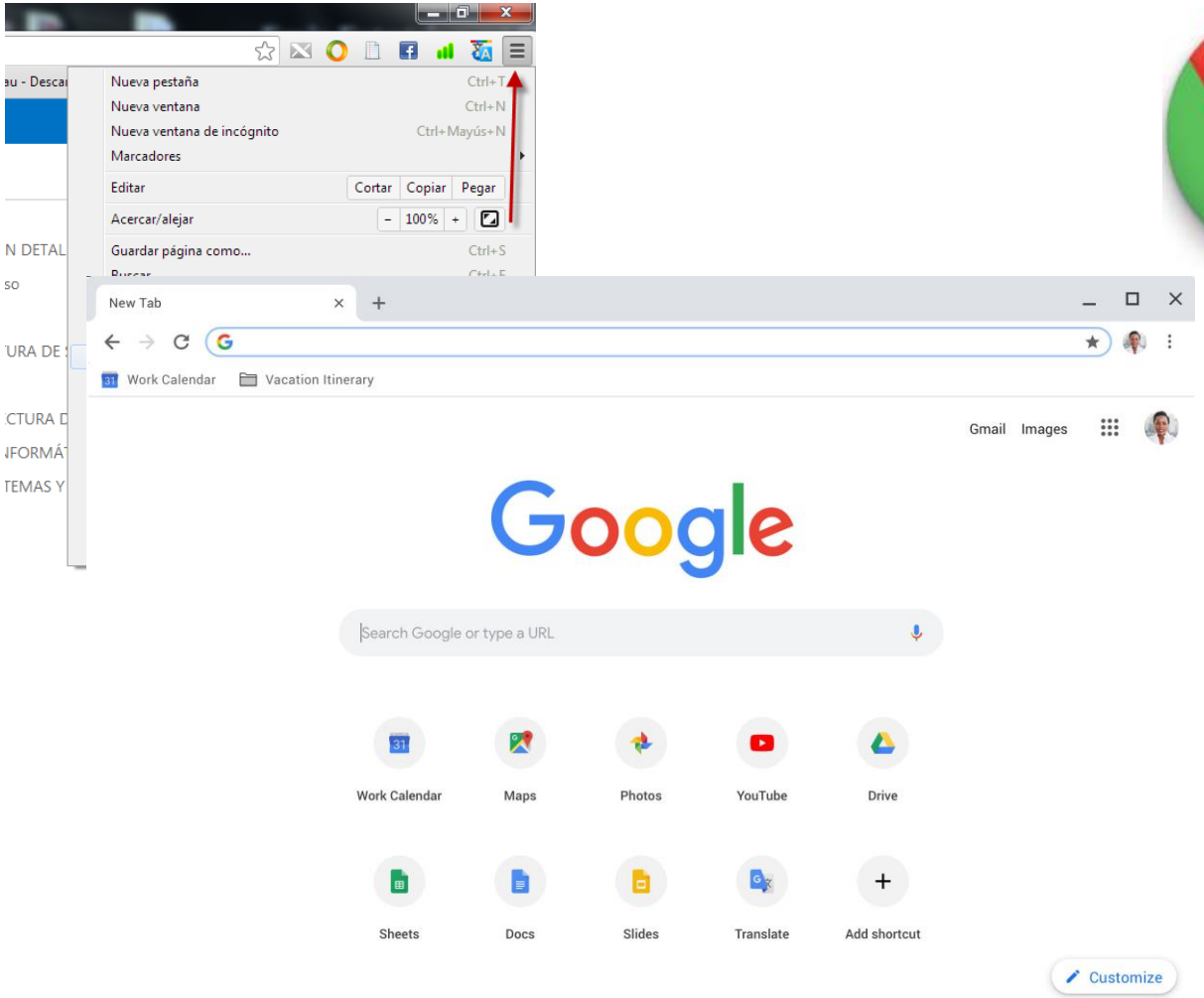
```
void mostrar(t_Pila * pila){
        int i;
        for(i=0;i<pila->tope;i++){
                printf("\%d\n”,pila->dato[i]);
        }
}
```



Aplicaciones de una pila



https://solitarios-online.com/spider/



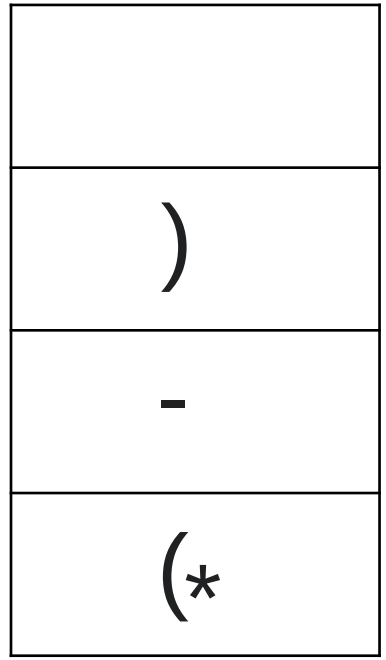
Se almacenan los sitios previamente visitados

Cuando el usuario quiere regresar (presiona el botón de retroceso), simplemente se extrae la última dirección (pop) de la pila de sitios visitados.

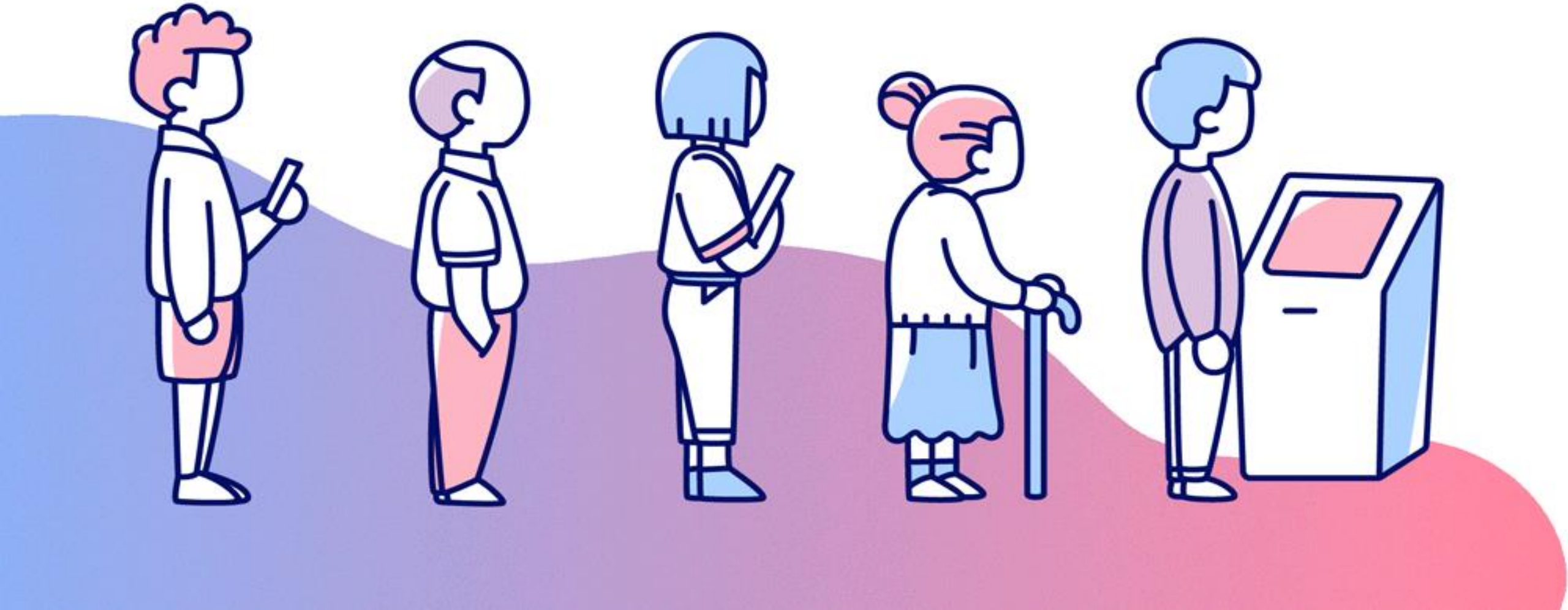
El algoritmo de notación polaca inversa, que utiliza pilas, es el principio de operación de ciertas para calculadoras

(5 - 6) * 7

5 6 - 7 *



Colas



Definición

Tipo especial de lista en el cual los elementos se insertan en un extremo y se suprime en el otro.

También llamado Lista FIFO

First In First Out

Primero en llegar primero en salir

Implementación de la Pila

Los elementos se almacenan en memoria de forma continua.

Importante definir:

- Numero Máximo de Elementos.
- Una variable que indique el inicio y fin de la cola.

Se debe reservar el espacio en memoria con anticipación.

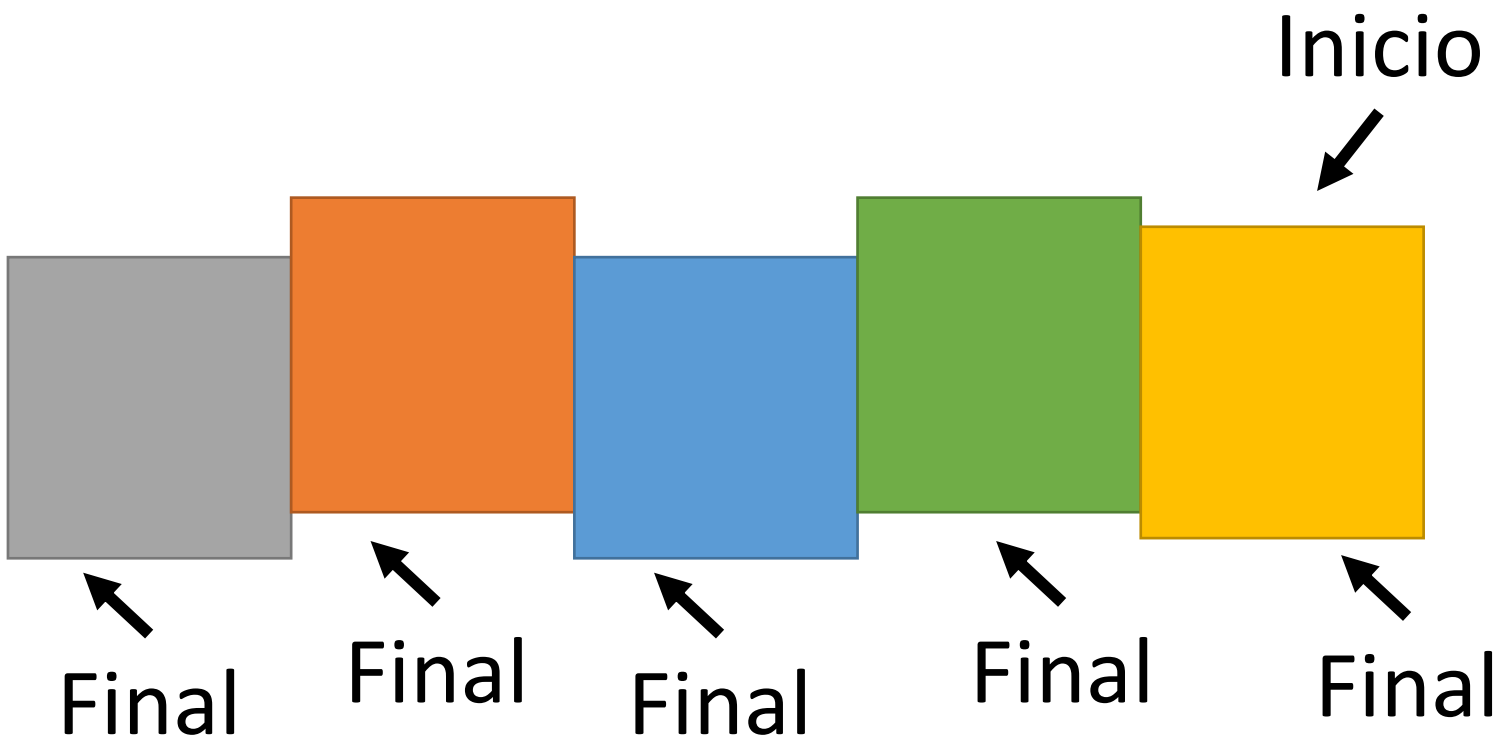
```
struct Cola{
    int inicio;
    int final;
    int dato[max];
};
```

Operaciones básicas

enCola (insertar)

Agrega elementos al final de la cola

Insertar un elementos a una cola llena producirá un desbordamiento

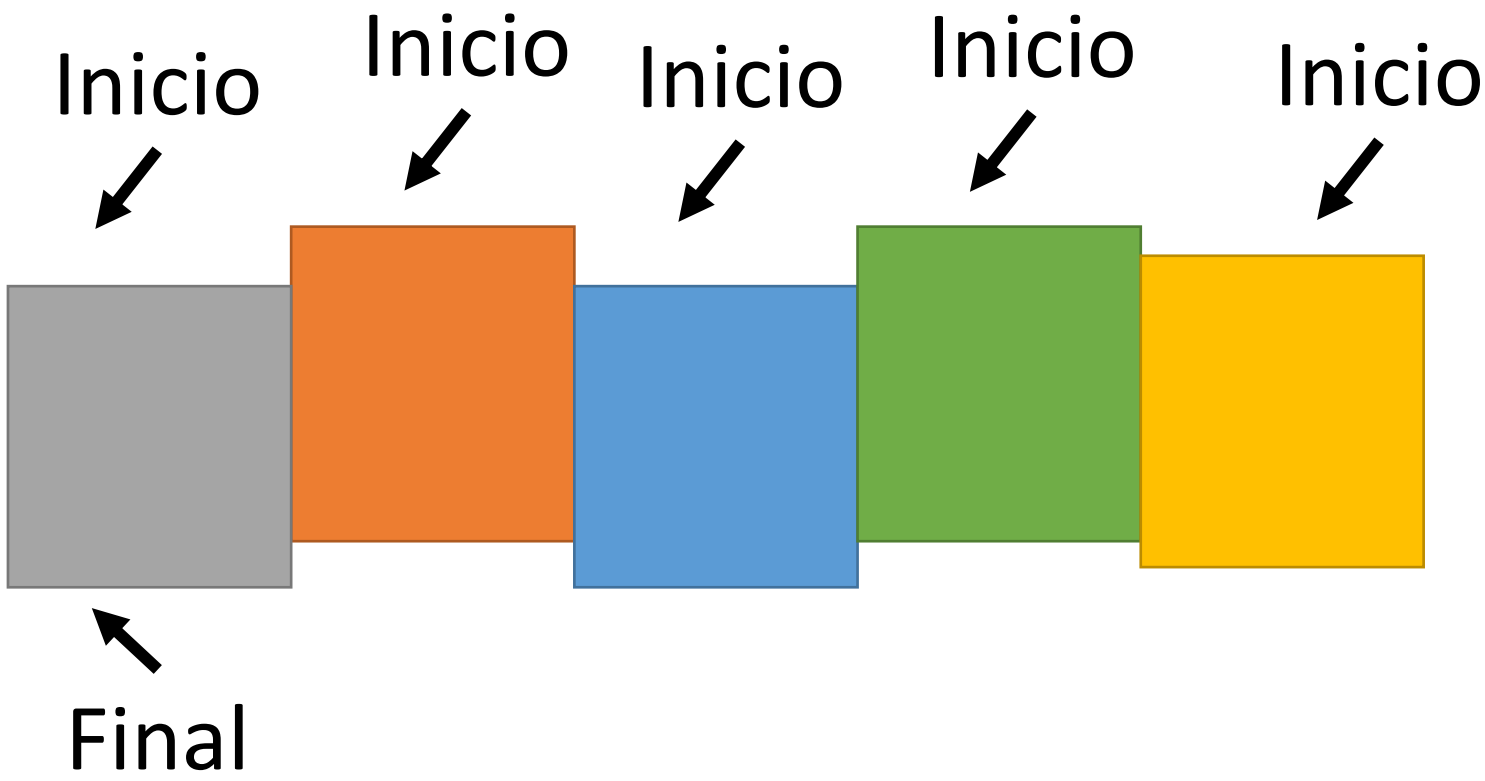


```
void enCola(struct Cola* cola, int valor){
	cola->dato[cola->final]=valor;
	cola->final ++;
}
```

desencolar (quitar)

Remueve el primer elemento de la cola

Eliminar un elemento de una cola vacía produce un desbordamiento



```
int remover(t_Cola * cola){
        int x, i;
        x=cola->dato[cola->inicio];
        for(i=0;i<cola->final-1;i++){
                cola->dato[i]=cola->dato[i+1];
        }
        cola->final--;
        return x;
}
```

Otras Operaciones

crearCola Inicializa la cola con inicio y final en cero.

```
void crearCola(struct Cola* cola){
    cola->inicio=0;
    cola->final=0;
}
```

vacia( )

Función que regresa verdadero si la cola es vacía.

```
int vacia(t_Cola * cola){
        if(cola->final==cola->inicio)
                return 1;
        else
                return 0;
}
```

llena( ) Regresa falso si la cola esta llena.

```
int llena(t_Cola * cola,int tam){
      if (cola->final==tam)
            return 0;
      else
            return 1;
}
```

finalC( ) Regresa el final de la cola.

```
int finalC(t_Cola * cola){
        return(cola->final);
}
```

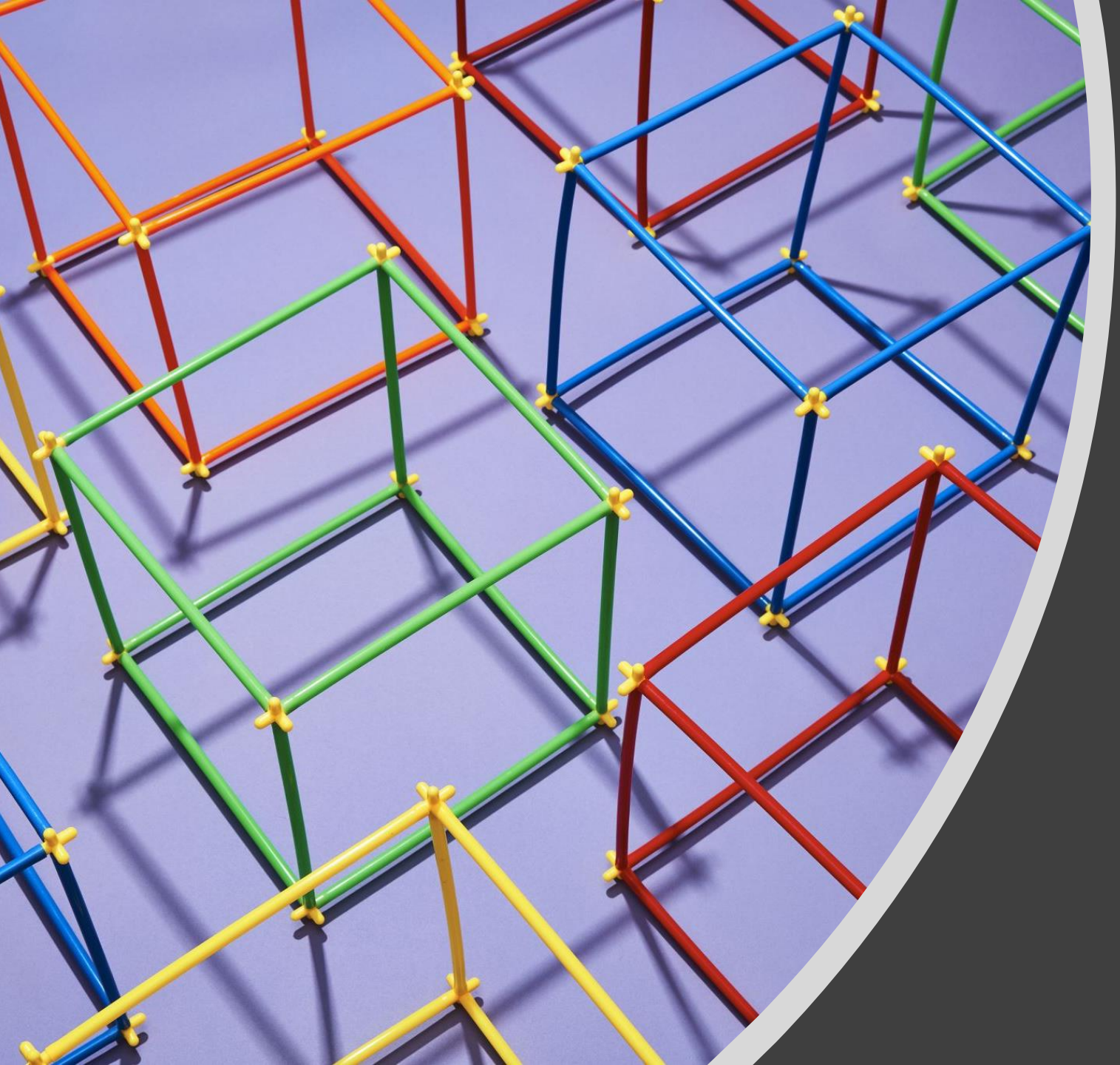
vaciar( )

Elimina todos los elementos de la pila.

```
void vaciar(t_Cola * cola){
        int x, i;
        while(cola->inicio!=cola->final){
                x=cola->dato[cola->inicio];
            for(i=0;i<cola->final-1;i++){
                cola->dato[i]=cola->dato[i+1];
            }
            cola->final--;
        }
}
```

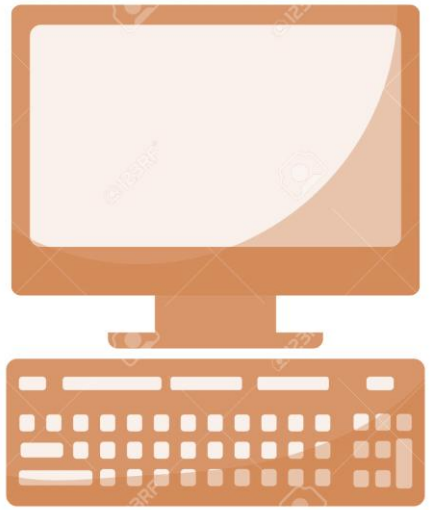
mostrar( ) Imprime en pantalla toda la cola.

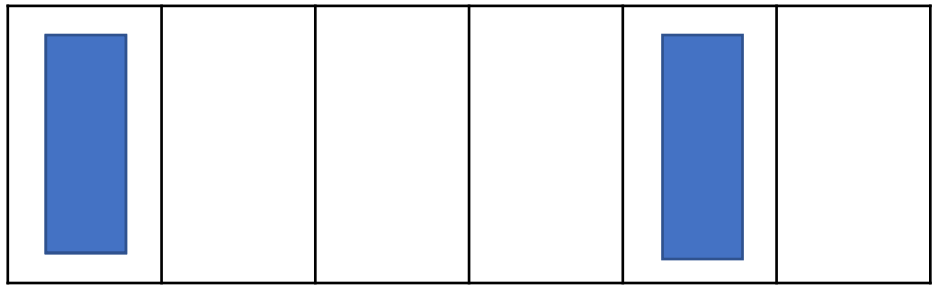
```
void mostrar(t_Cola * cola){
        int i;
        for(i=0;i<cola->final;i++){
                printf("\n%d\n", cola->dato[i]);
        }
}
```

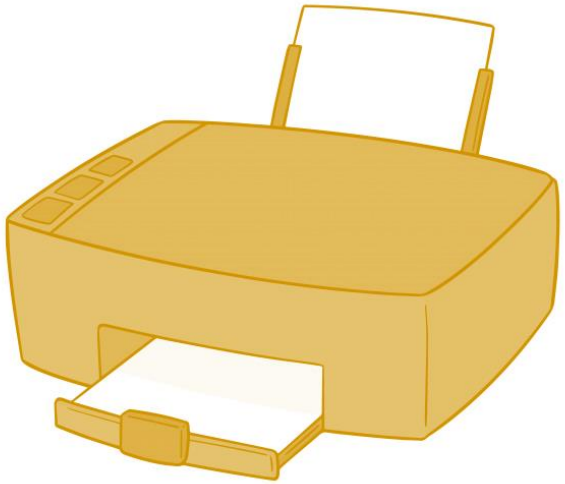


Aplicaciones de una cola

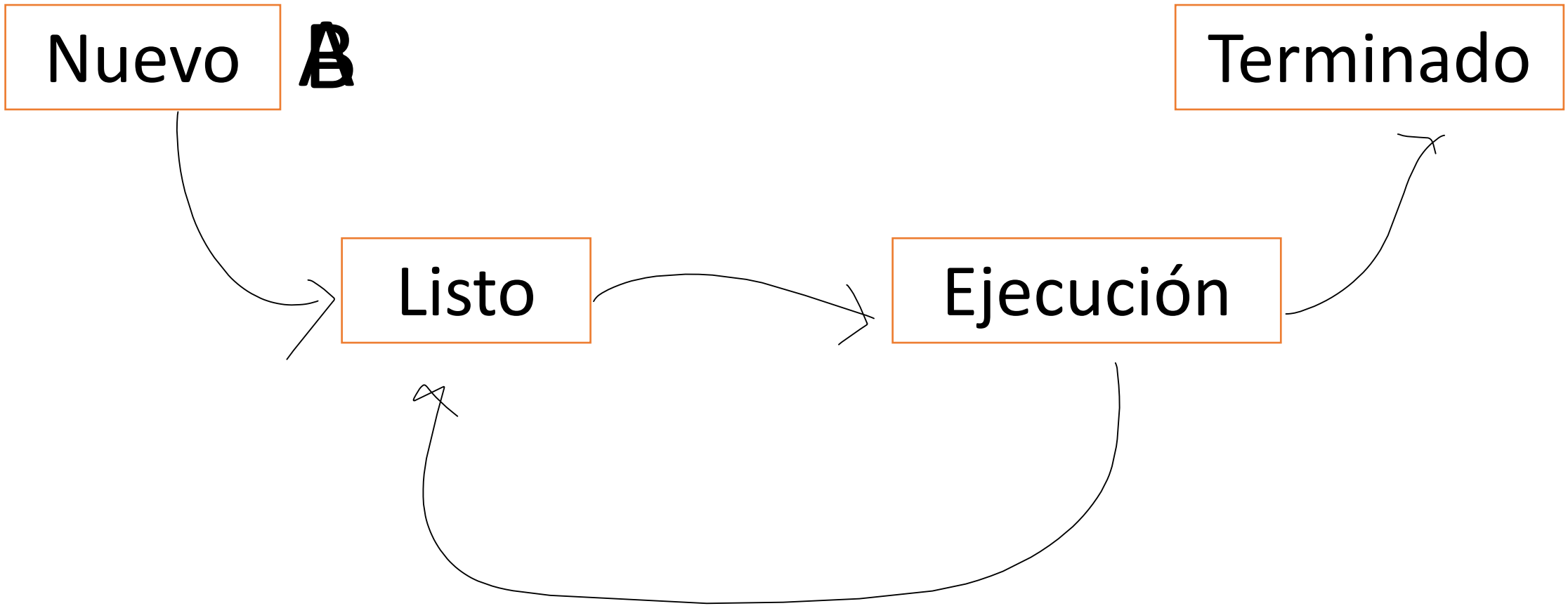
Operaciones en redes de computadoras







Transiciones de procesos o ciclos de vida de un proceso



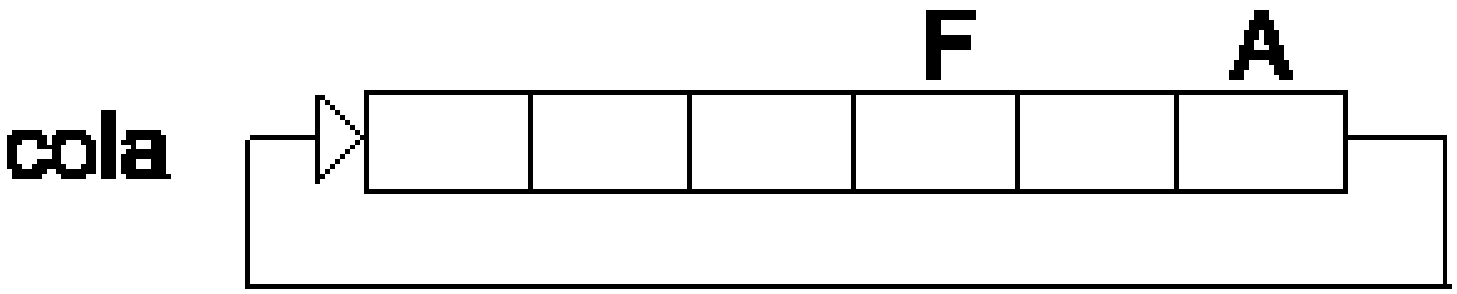
COLA CIRCULAR

Definición

Es una mejora de la cola simple.

Utiliza de manera más eficiente la memoria.

El siguiente elemento del **último** es el **primero.**



```
struct ColaC{
    int inicio;
    int final;
    int dato[5];
};
```

Operaciones básicas

Consideración

Casos Extremos

- Estructura vacía
- Estructura llena

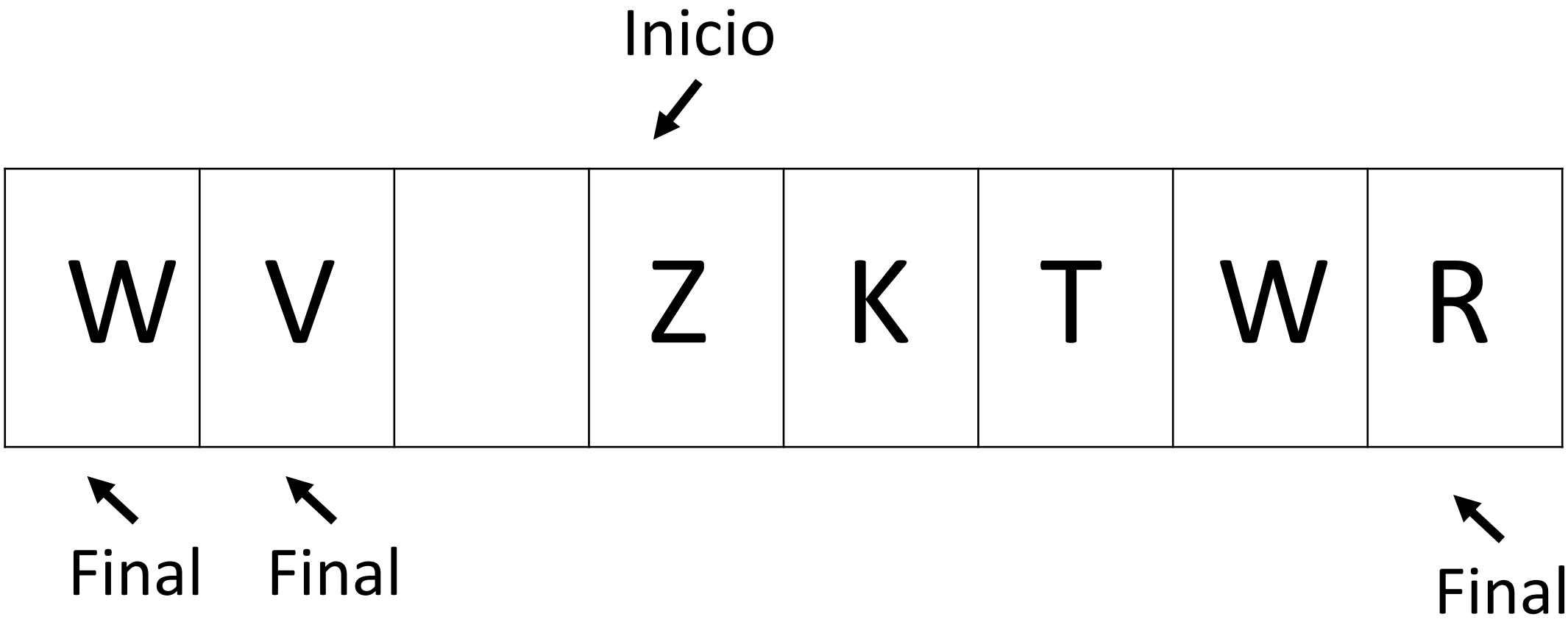
Caso base

- Estructura con elementos

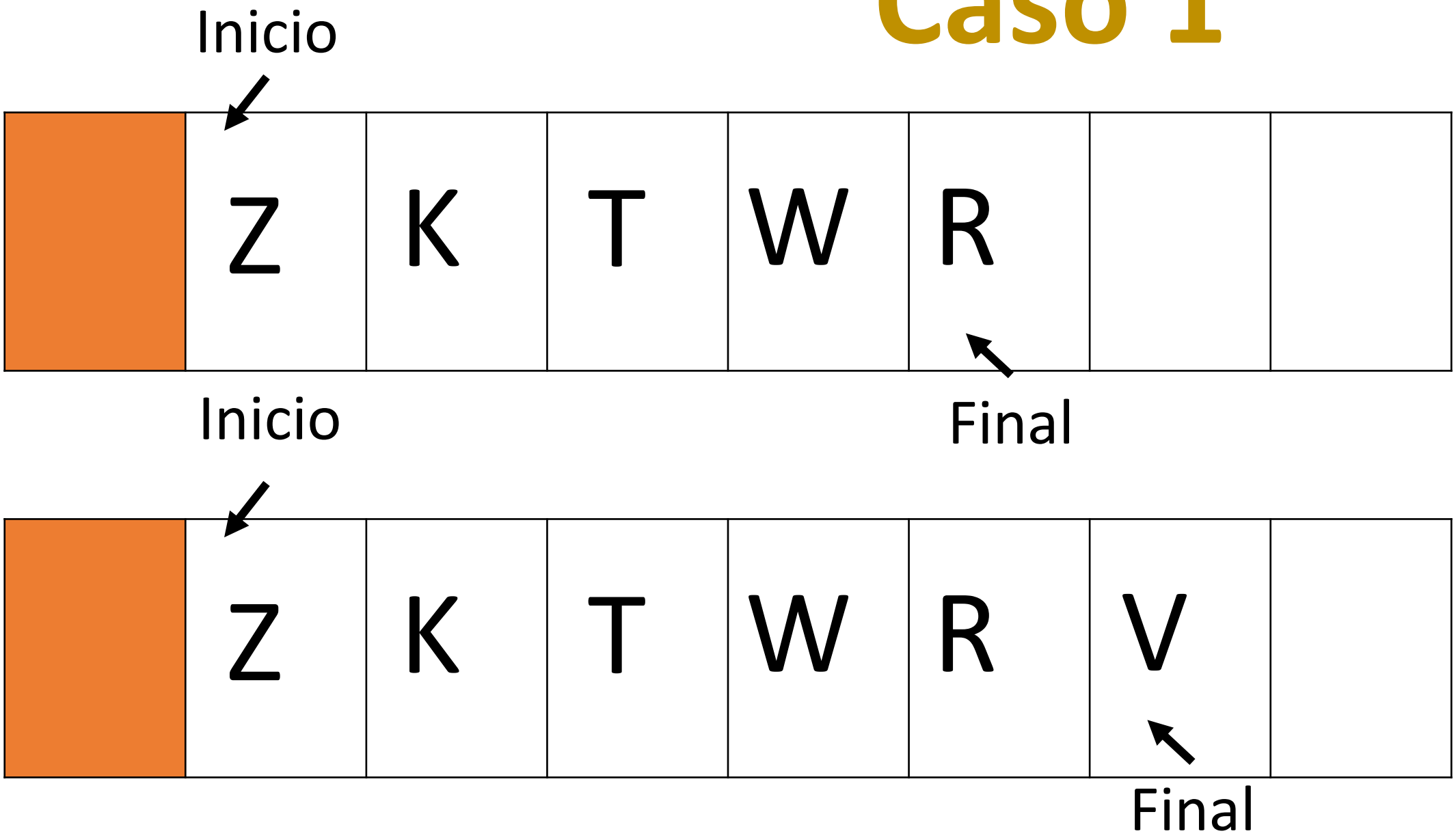
*En algoritmo de una cola circular para los casos extremos (cuando la estructura está vacía y cuando la estructura está llena) es el mismo con respecto a la cola simple

enCola (insertar)

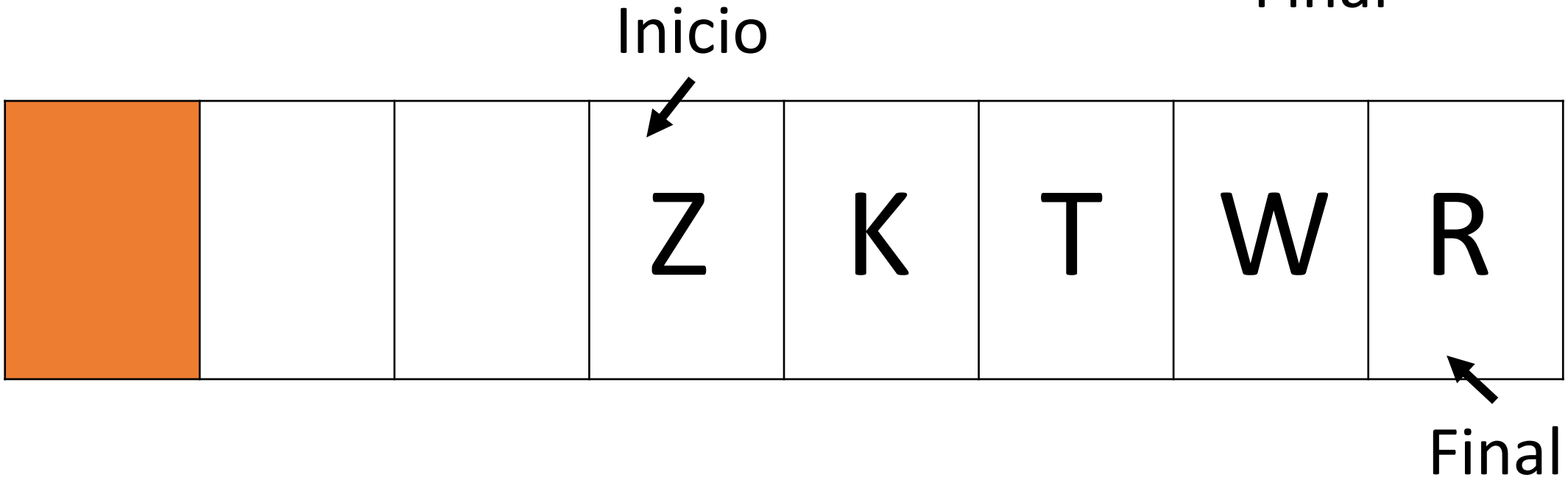
Agrega elementos al final de la cola.



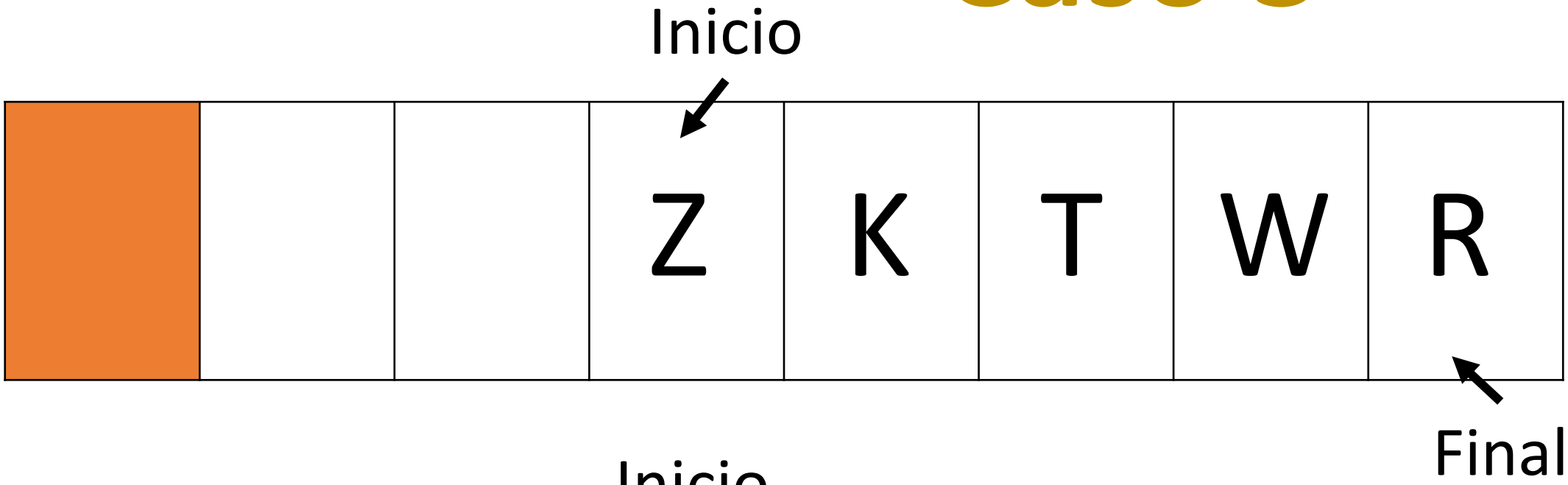
Caso 1

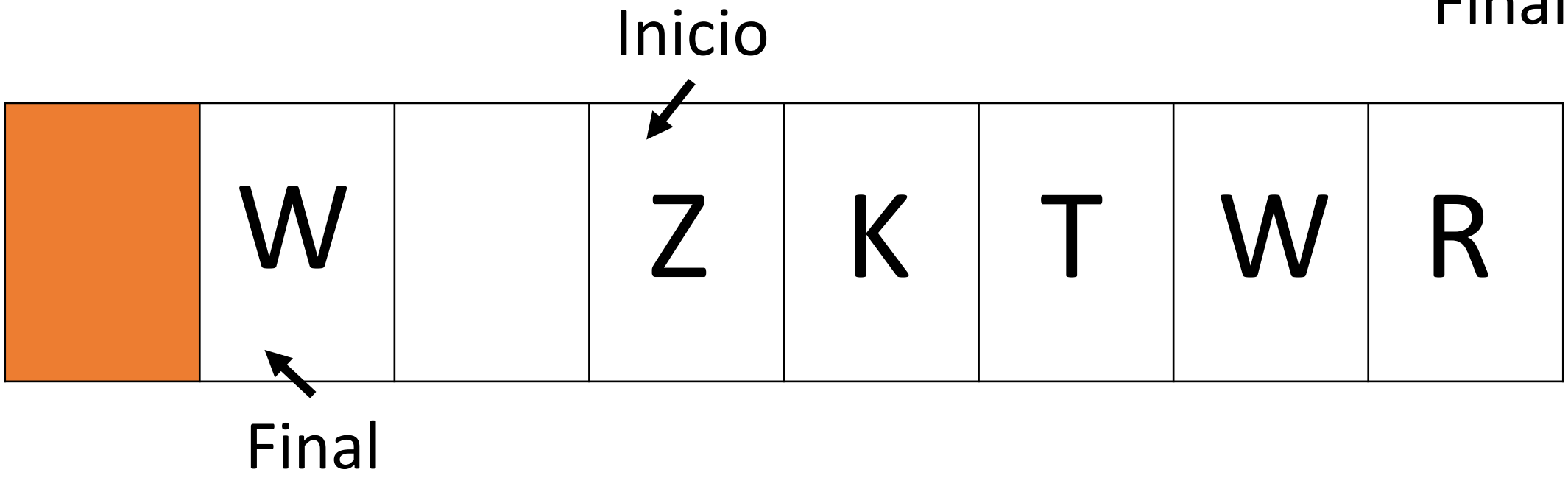


Caso 2



Caso 3

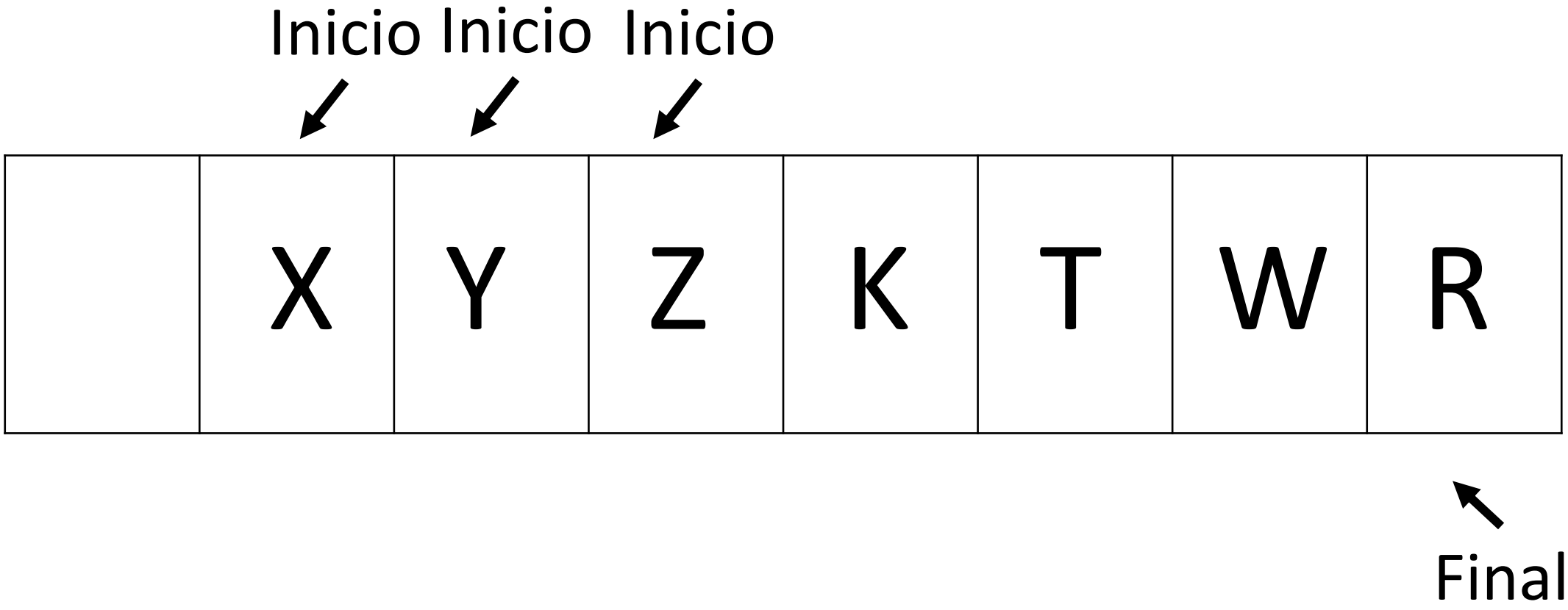




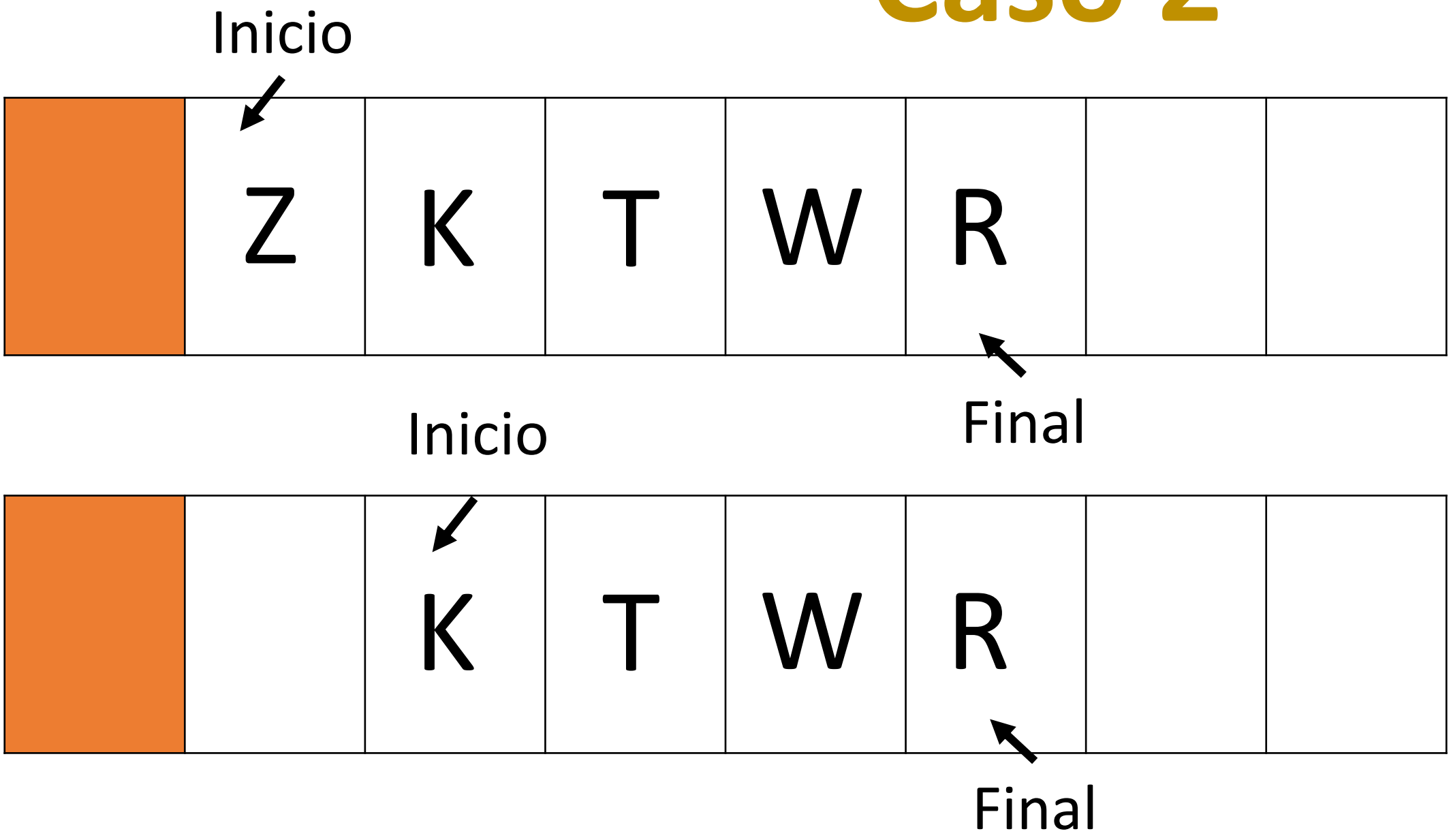
```
void insertar(tColaC * colaC, int valor){
        colaC->dato[colaC->final]=valor;
        if(colaC->final==5){
                colaC->final=1;
        }
        else{
                colaC->final++;
        }
        if(colaC->inicio==0){
                colaC->inicio=1;
        }
}
```

desencolar (quitar)

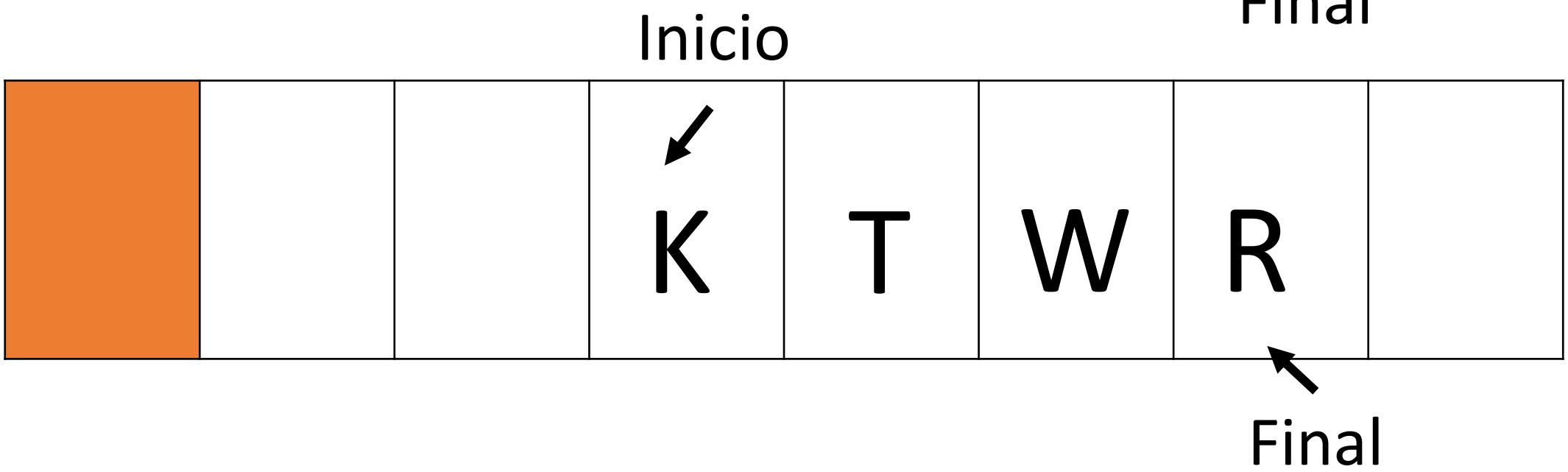
Remueve el primer elemento de la cola.



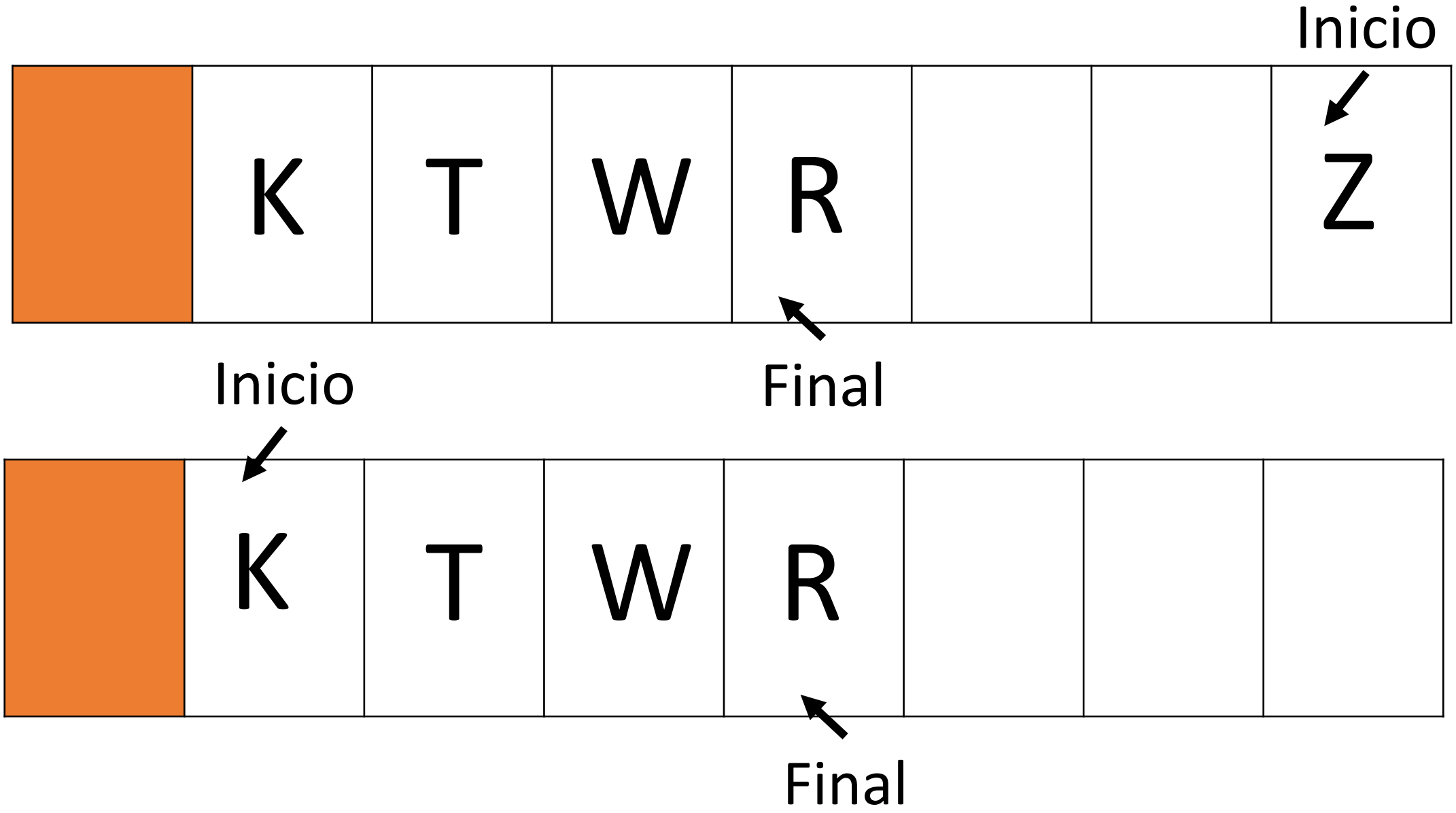
Caso 2



Caso 2



Caso 3



```
int remover(tColaC * colaC){
        int x;
        x=colaC->dato[colaC->inicio];
        if(colaC->inicio==5){
                colaC->inicio=1;
        }
        else{
                colaC->inicio++;
        }
        if(colaC->inicio==colaC->final){
                colaC->inicio=0;
                colaC->final=0;
        }
        return x;
}
```

Otras Operaciones

crearCola Inicializa la cola con inicio y final en cero.

```
void crearCola(tColaC * colaC){
    colaC->inicio=0;
    colaC->final=0;
}
```

vacia( )

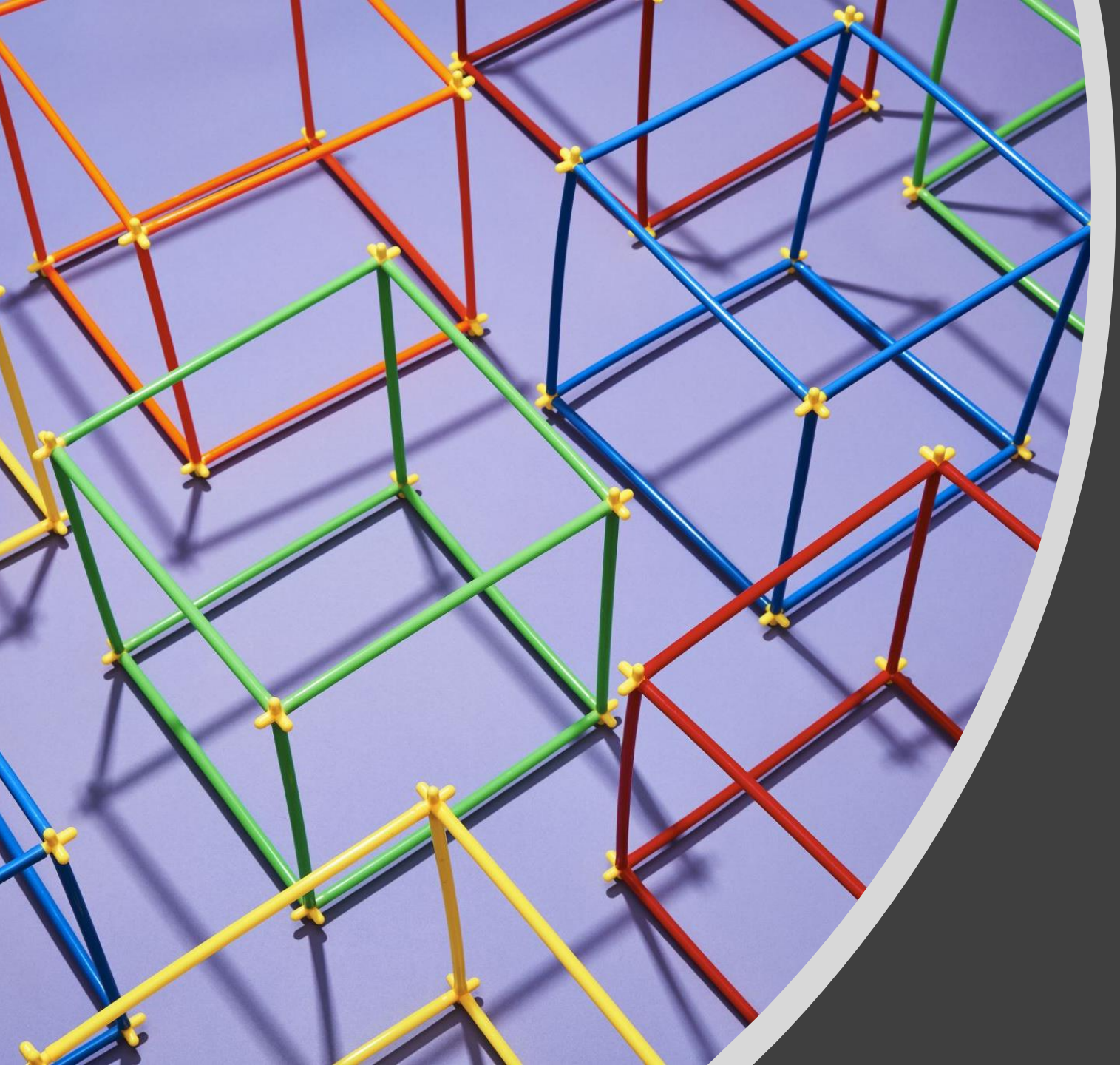
Función que regresa verdadero si la cola es vacía.

```
int vacia(tColaC * colaC){
      if(colaC->inicio==0 && colaC->final==0)
            return 1;
      else
            return 0;
}
```

llena( )

Regresa falso si la cola esta llena.

```
int llena(tColaC * colaC){
      if (colaC->final==5 && colaC->inicio==1)
            return 0;
      else
            return 1;
}
```



Aplicaciones de cola circular

Respondia - YouTube | Netflix

netflix.com/browse/genre/11

NETFLIX Inicio Programas Películas Más recientes Mi lista NIÑOS

Dramas militares

ORIGINALES DE NETFLIX >

Películas asiáticas


48
19
90
7
18


FIFA WORLD CUP
Brasil
SELECCIÓN UNIFORME
18
¡LISTO!
ARGENTINA
LOC. - BRASIL 2014
A SELECCIONAR
B ATRÁS

BICOLA (COLA DOBLE)

Definición

Es una mejora de la cola simple

Utiliza de manera más eficiente la memoria.

Los elementos se pueden **insertar** o **eliminar** de cualquiera de los dos extremos.

Doble cola (bicola).

Existen 2 variantes

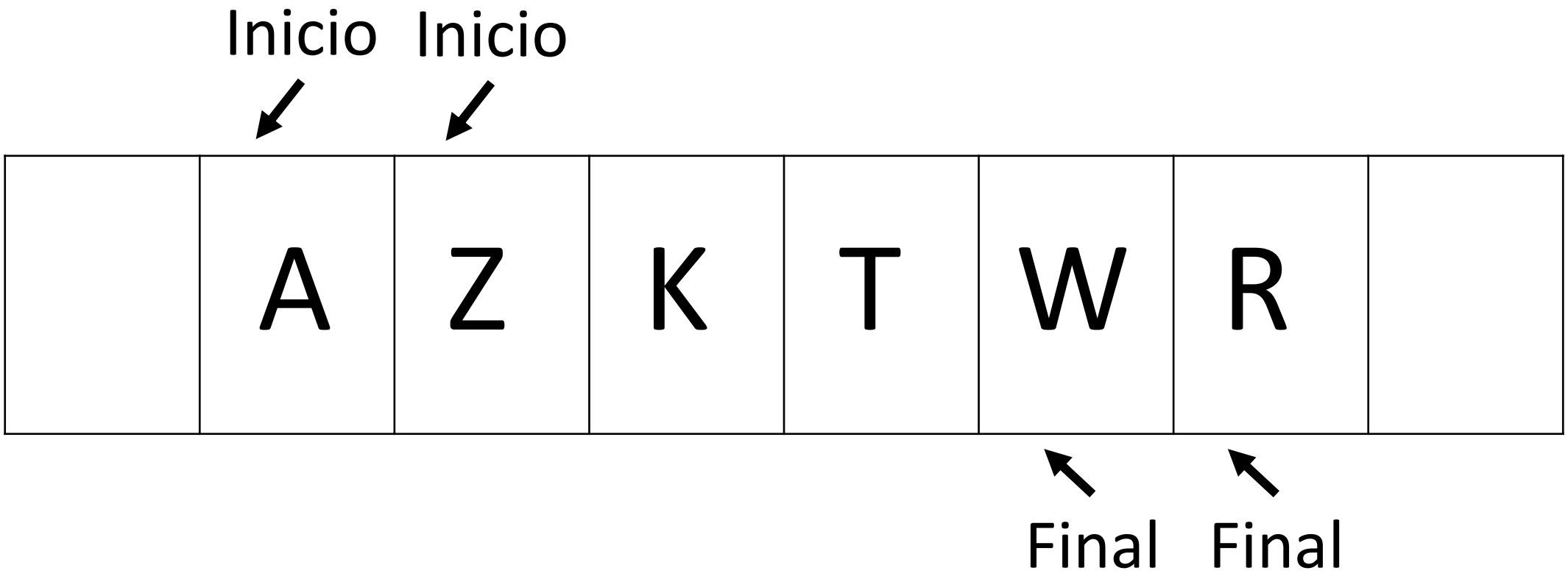
Cola con **entrada** restringida

- ✓ Eliminación por ambos extremos.
- ✓ Inserción solo por el final.

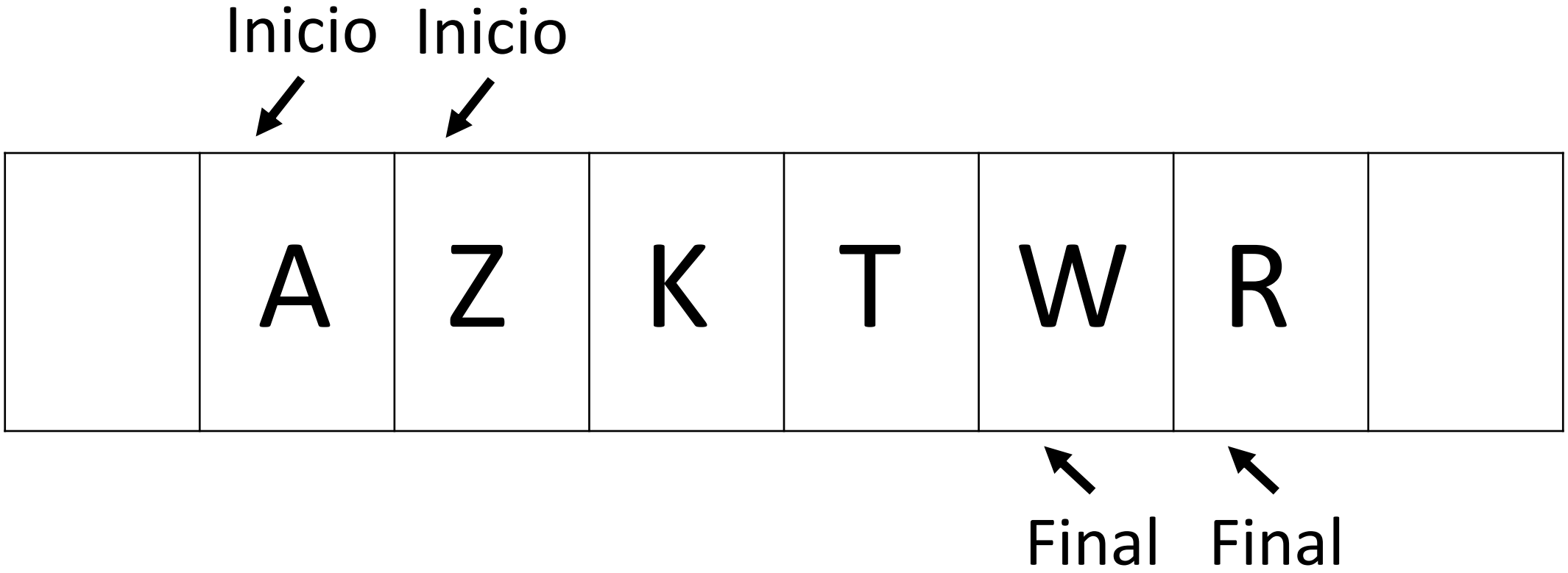
Cola con **salida** restringida

- ✓ Eliminación solo por inicio.
- ✓ Inserción por ambos extremos.

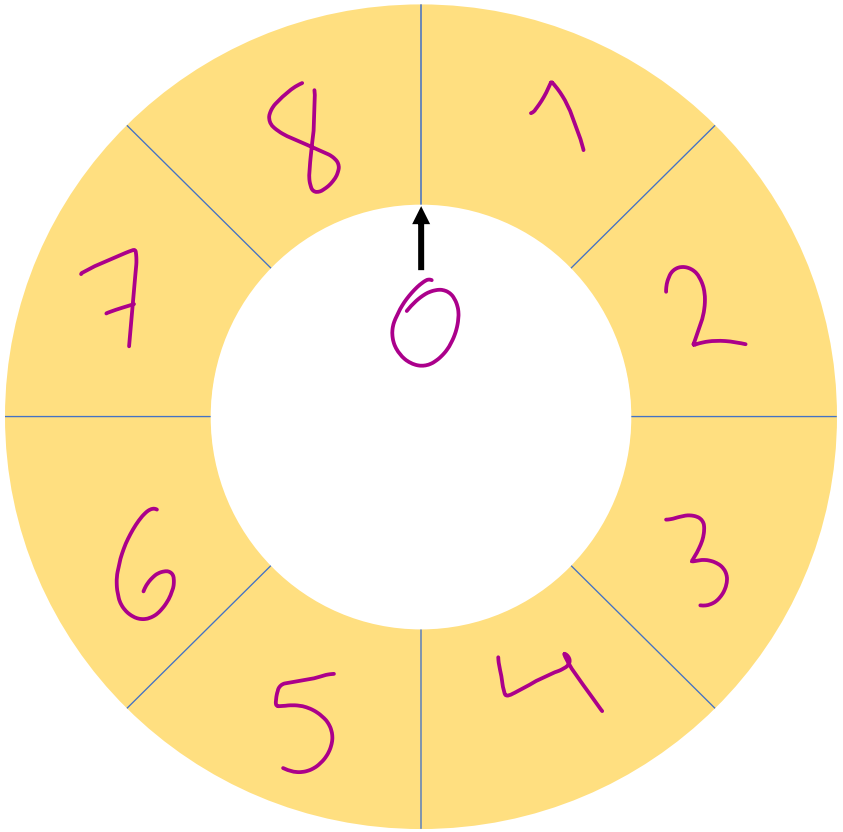
insertar



quitar



Cola Doble Circular

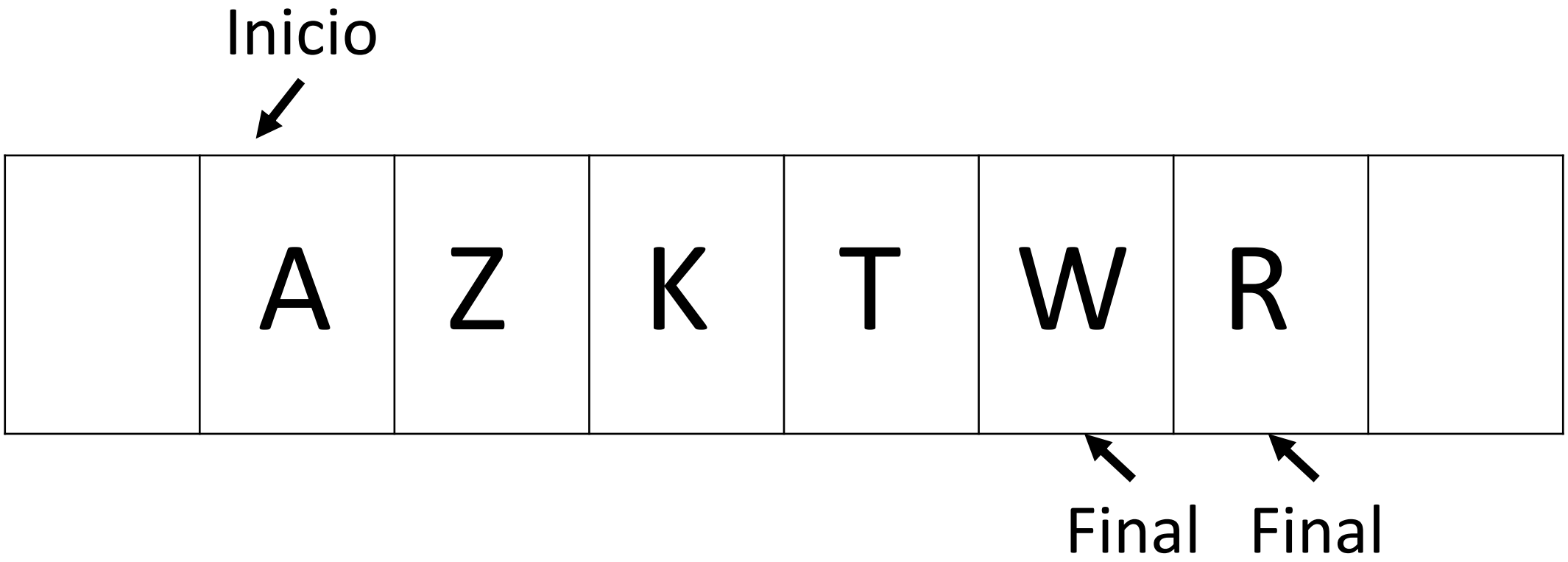


```
struct ColaCD{
    int inicio;
    int final;
    int dato[5];
};
```

Operaciones básicas

encolarFin (insertar)

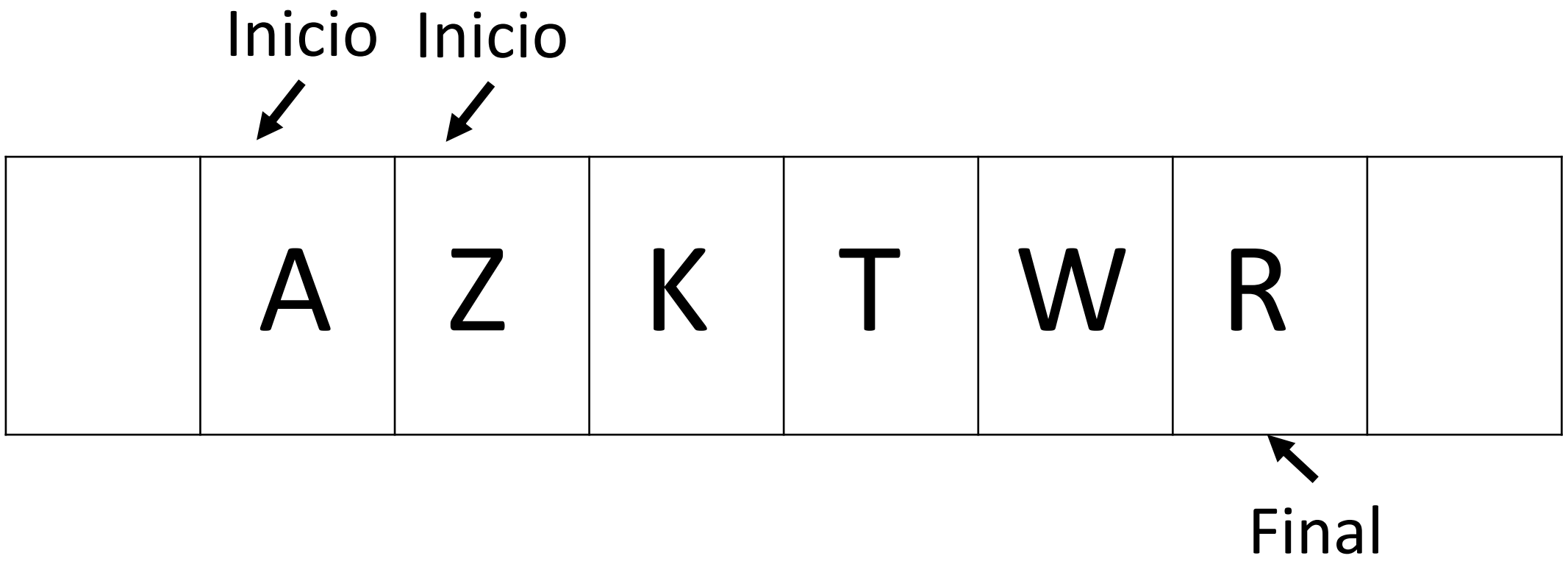
Agrega elementos al final de la cola.



```
void insertarFin(tColaCD * colaCD, int valor){
        colaCD->dato[colaCD->final]=valor;
        if(colaCD->final==5){
                colaCD->final=1;
        }
        else{
                colaCD->final++;
        }
        if(colaCD->inicio==0){
                colaCD->inicio=1;
        }
}
```

desencolarIni (quitar)

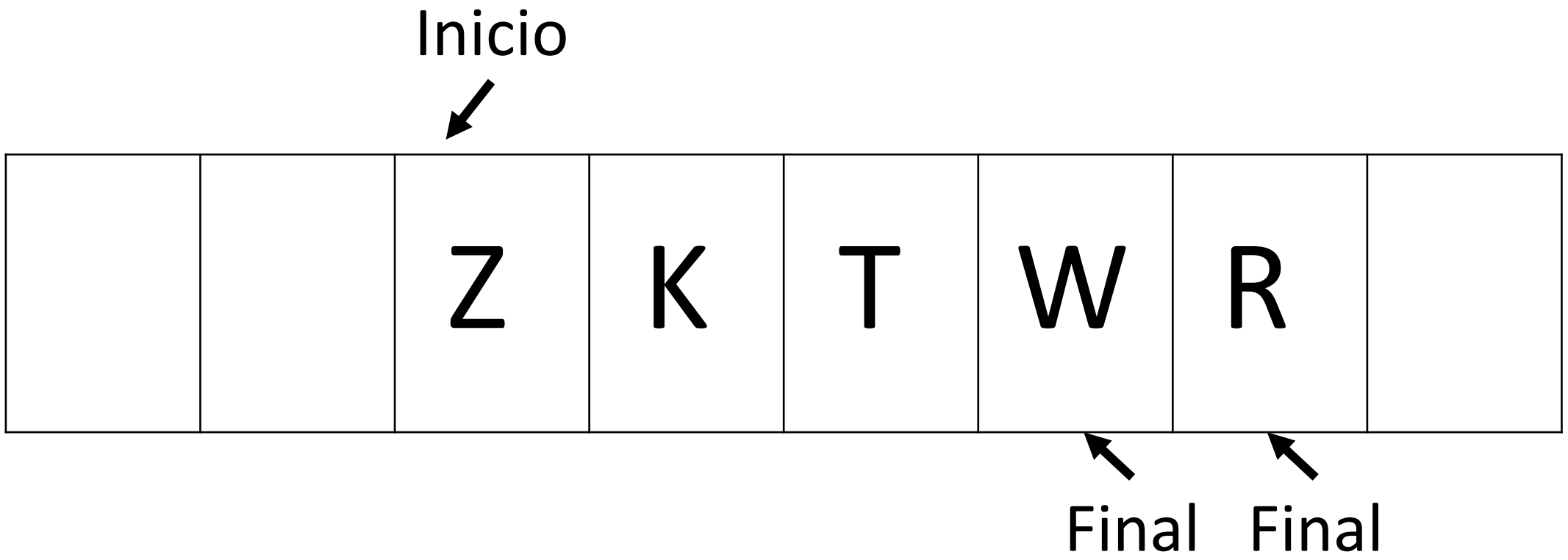
Elimina elementos al inicio de la cola.



```
int removerIni(tColaCD * colaCD){
        int x;
        x=colaCD->dato[colaCD->inicio];
        if(colaCD->inicio==5){
                colaCD->inicio=1;
        }
        else{
                colaCD->inicio++;
        }
        if(colaCD->inicio==colaCD->final){
                colaCD->inicio=0;
                colaCD->final=0;
        }
        return x;
}
```

desencolarFin (quitar)

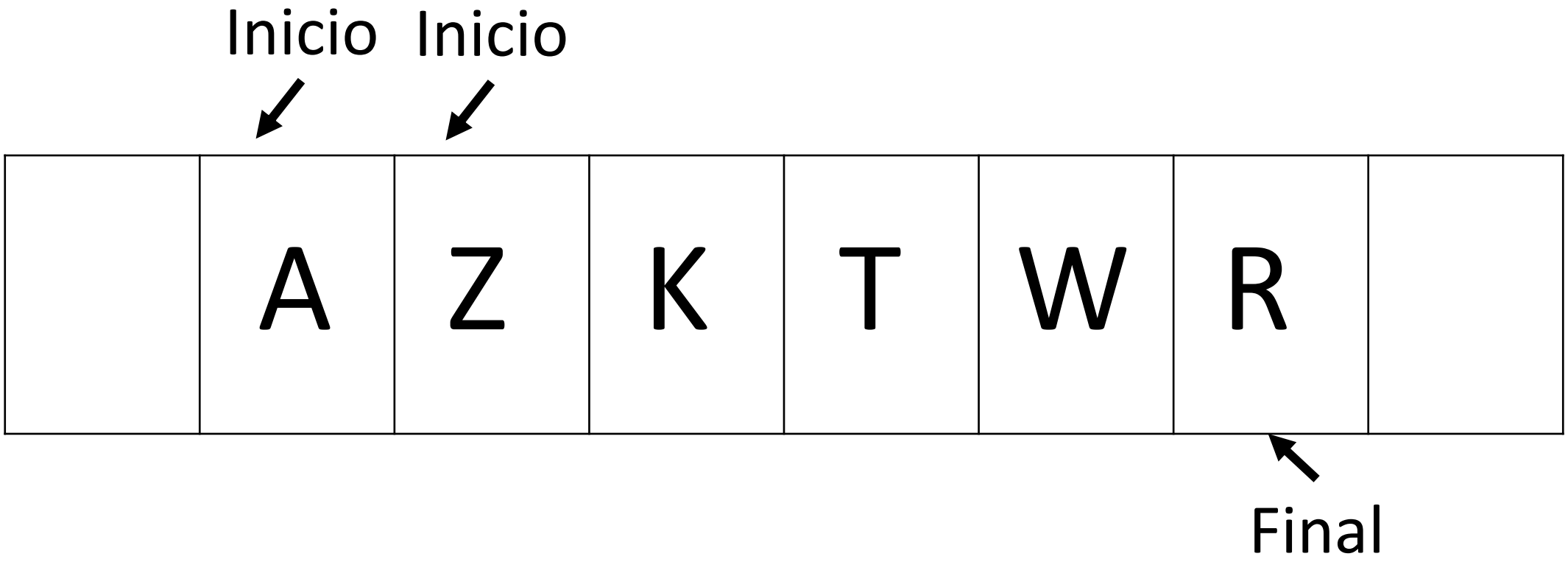
Elimina elementos al final de la cola



```
int removerFin(tColaCD * colaCD){
        int x;
        x=colaCD->dato[colaCD->final];
        if(colaCD->final==5){
                colaCD->final=1;
        }
        else{
                colaCD->final++;
        }
        if(colaCD->final==colaCD->inicio){
                colaCD->inicio=0;
                colaCD->final=0;
        }
        return x;
}
```

encolarIni (insertar)

Agrega elementos al inicio de la cola.



```
void insertarIni(tColaCD * colaCD, int valor){
        colaCD->dato[colaCD->inicio]=valor;
        if(colaCD->inicio==5){
                colaCD->inicio=1;
        }
        else{
                colaCD->inicio++;
        }
        if(colaCD->final==0){
                colaCD->final=1;
        }
}
```

Consideración

Casos Extremos

- Estructura vacía
- Estructura llena

Caso base

- Estructura con elementos

Otras Operaciones

crearCola Crea una cola vacía.

```
void crearCD(tColaCD * colaCD){
        colaCD->inicio=0;
        colaCD->final=0;
}
```

vacia( )

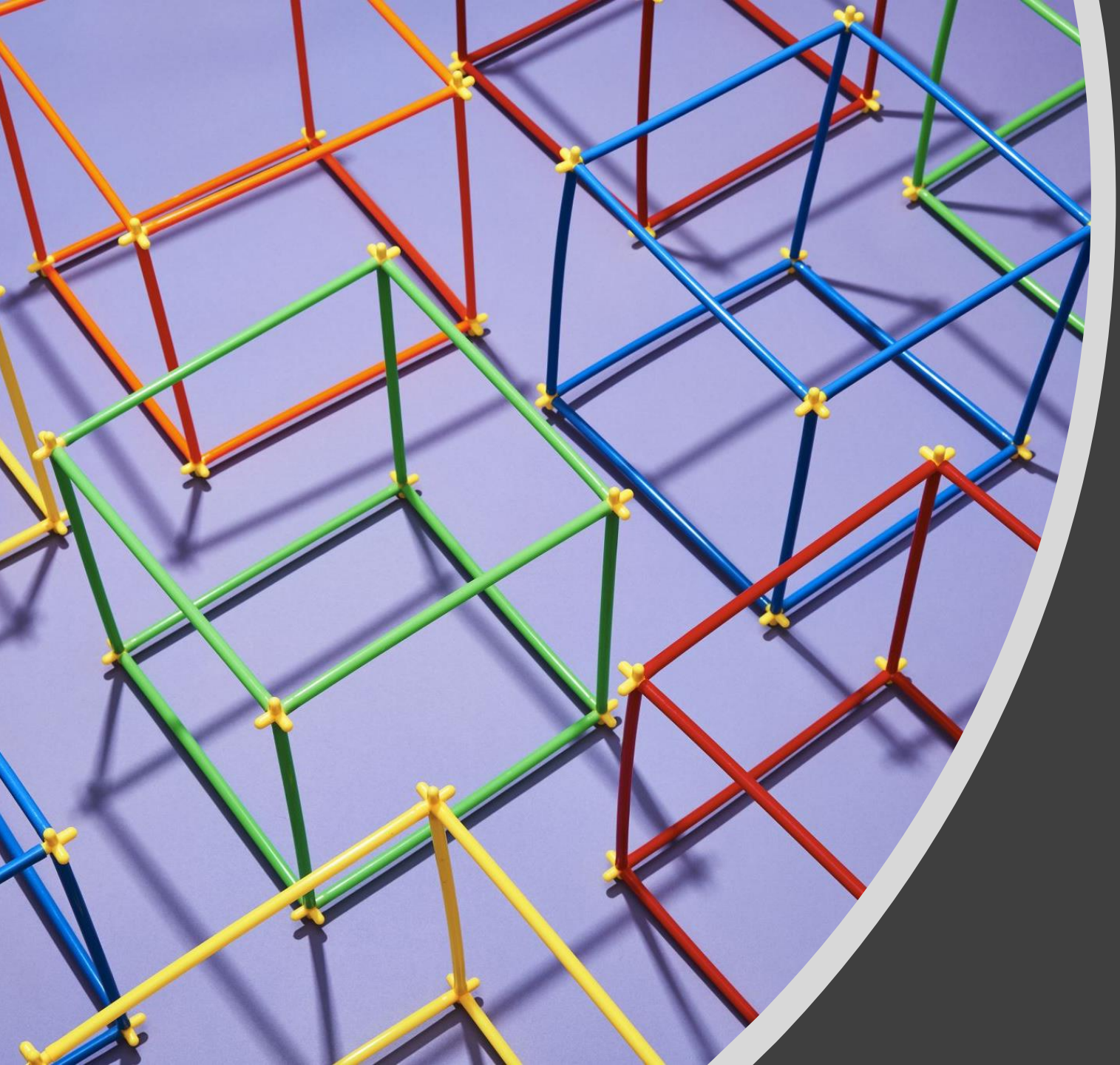
Función que regresa verdadero si la cola es vacía.

```
int vacia(tColaCD * colaCD){
    if(colaCD->inicio==0 && colaCD->final==0)
        return 1;
    else
        return 0;
}
```

llena( )

Regresa falso si la cola esta llena.

```
int llena(tColaCD * colaCD){
      if (colaCD->final==5 && colaCD->inicio==1)
            return 0;
      else
            return 1;
}
```



Aplicación de una cola doble

Dentro del **sistema operativo** no todas las aplicaciones tienen la misma exigencia en cuanto a tiempo y recursos, existen procesos que se **tienen que ejecutar de manera inmediata** ante algún suceso que se presente en el sistema, mientras que otros solo tengan que procesar información y puedan (y deban) **esperar** a que el sistema se recupere.

¡GRACIAS!