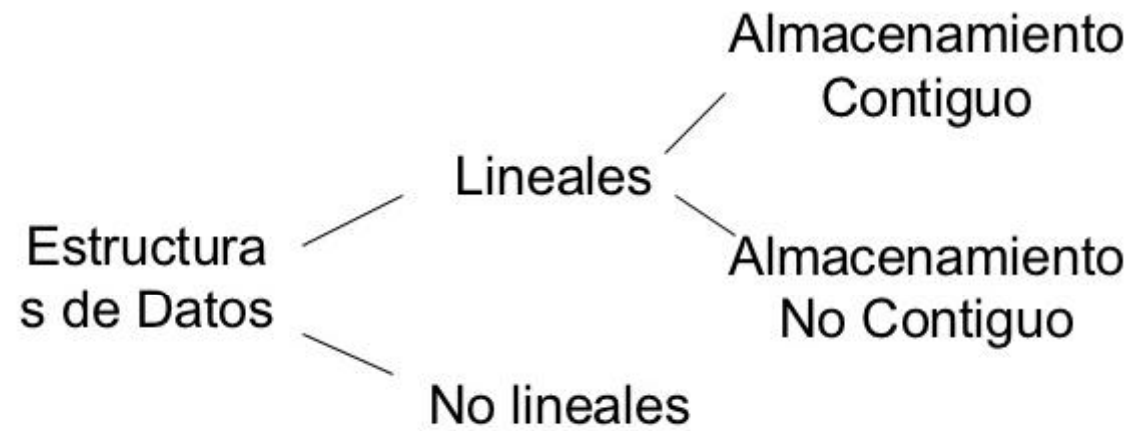


# Pila, Cola, Listas

## ***Unidad 2: Estructuras de Datos***

**Tema II. Estructuras de datos Compuestas.** Declaración de listas. Operaciones sobre listas. Declaración de pilas. Operaciones sobre pilas. Declaración de colas. Operaciones sobre colas. Cola circulares. Doble cola.

# Estructuras de Datos





Las estructuras de datos dinámicas se puede dividir en dos grandes grupos

LINEALES	NO LINEALES
Pilas Colas Listas	Árboles Grafos

Una *estructura estática de datos* es aquella donde sus diferentes valores pueden cambiar pero no su estructura dado que ella es fija. Su estructura se especifica en el momento que se escribe el programa y no puede ser modificada por el mismo.

Una *estructura dinámica de datos*, por el contrario puede modificar su estructura mediante el programa, puede modificar su tamaño añadiendo o eliminando nodos mientras está en ejecución el programa.



### Listas

Una **lista** es un conjunto ordenado de elementos de un tipo, la misma puede variar el número de elementos.

**Lista L:** L1, L2, L3, L4,....., LN

*donde LN es un elemento de la lista.*

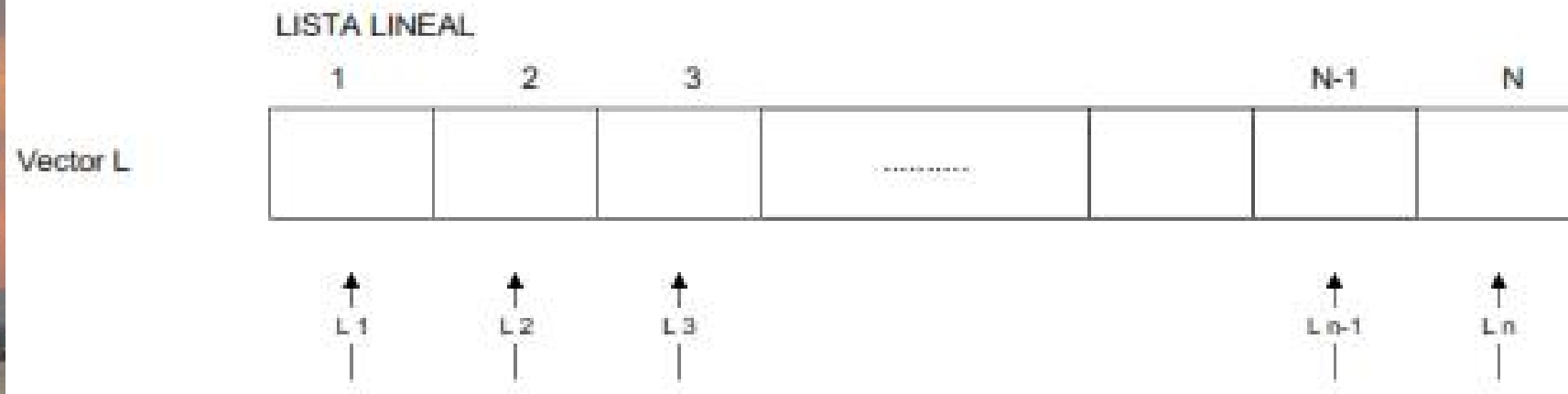
Cada elemento de la lista – a excepción del primero – tiene un único predecesor y a su vez cada elemento – a excepción del último – tiene un único sucesor.

Los elementos de una lista lineal normalmente se almacenan uno detrás de otro en posiciones consecutivas de memoria, por ejemplo las entradas en una guía telefónica. Cuando una lista se almacena en la memoria principal de una computadora lo está haciendo en posiciones sucesivas de memoria; cuando se almacena en cinta magnética, los elementos sucesivos se presentan en sucesión en la cinta.

Esta asignación de memoria toma el nombre de *almacenamiento secuencial*, en apartados posteriores veremos el tipo de *almacenamiento encadenado o enlazado*.

## Representación de listas

Uno de los medios mas clásicos de realizar una lista lineal es mediante un vector, donde los elementos se sitúan en posiciones físicamente contiguas. Su representación





## Operaciones

Las operaciones que pueden llevarse a cabo en una lista son, entre otras:

- *Recorrido de la lista*: Consiste en visitar todos o varios de los elementos que formen la lista.
- *Inserción de un elemento*: Consiste en añadir un nuevo elemento al arreglo. Esta operación tiene dos modalidades: a) añadir al final, b) añadir en el interior del arreglo, lo que provoca desplazamiento de algunos elementos.
- *Eliminación de un elemento*: Esta operación consiste en borrar un elemento deseado y desplazar a todos los elementos siguientes al que se desea borrar.
- *Búsqueda de un elemento*: Buscar la posición de un elemento que posee un valor determinado.

Para declara una lista que contenga las temperaturas y la asignación de las mismas ...  
Y además el programa calcula de la temperatura media...

```
#include <stdio.h>

int main()
{
    int temp[24]; /*Con esto ya tenemos declaradas las 24 variables */
    float media;
    int hora;

    /* Ahora tenemos que dar el valor de cada una */
    for( hora=0; hora<24; hora++ )
    {
        printf( "Temperatura de las %i: ", hora );
        scanf( "%i", &temp[hora] );
        media += temp[hora];
    }
    media = media / 24;
    printf( "\nLa temperatura media es %f\n", media );
}
```



Cual es la diferencia con el anterior ...

```
#include <stdio.h>
#define ELEMENTOS      24
int main()
{
    int temp[ELEMENTOS]; /*Con esto ya tenemos declaradas las 24
    variables */
    float media;
    int hora;
    /* Ahora tenemos que dar el valor de cada una */
    for( hora=0; hora<ELEMENTOS; hora++ )
    {
        printf( "Temperatura de las %i: ", hora );
        scanf( "%i", &temp[hora] );
        media += temp[hora];
    }
    media = media / ELEMENTOS;
    printf( "\nLa temperatura media es %f\n", media );
}
```

## Inicializar un array

Pues con arrays se puede hacer:

```
int temperaturas[24] = { 15, 18, 20, 23, 22, 24, 22, 25, 26, 25, 24, 22, 21, 20, 18, 17, 16, 17, 15, 14, 14, 14, 13, 12 };
```

Ahora el elemento 0 (que será el primero), es decir temperaturas[0] valdrá 15. El elemento 1 (el segundo) valdrá 18 y así con todos. Vamos a ver un ejemplo:

```
#include <stdio.h>
int main()
{
    int hora;
    int temperaturas[24] = { 15, 18, 20, 23, 22, 24, 22, 25, 26, 25, 24, 22, 21, 20, 18, 17, 16, 17, 15, 14, 14, 14, 13, 12 };

    for (hora=0 ; hora<24 ; hora++ )
    {
        printf( "La temperatura a las %i era de %i grados.\n", hora, temperaturas[hora] );
    }
}
```



## Recorrer un array

```
#include <stdio.h>

int main()
{
    int hora;
    /* Faltan los tres últimos elementos */
    int temperaturas[24] = { 15, 18, 20, 23, 22, 24, 22, 25, 26, 25,
        24, 22, 21, 20, 18, 17, 16, 17, 15, 14, 14, 13, 13, 12 };
    for (hora=0 ; hora<28 ; hora++ )
    {
        printf( "La temperatura a las %i era de %i grados.\n",
            hora, temperaturas[hora] );
    }
}
```

## Que pasa con este código cuando se ejecuta ?

```
#include <stdio.h>
int main()
{
    int temp[24];
    float media;
    int hora;
    for( hora=0; hora<28; hora++ )
    {
        printf( "Temperatura de las %i: ", hora );
        scanf( "%i", &temp[hora] );
        media += temp[hora];
    }
    media = media / 24;
    printf( "\nLa temperatura media es %f\n", media );
}
```



## Declaración de cadenas

Las cadenas se declaran como vectores de caracteres, así que debes proporcionar el número máximo de caracteres que es capaz de almacenar: su *capacidad*. Esta cadena, por ejemplo, se declara con capacidad para almacenar 10 caracteres:

```
char a[10];
```

Puedes inicializar la cadena con un valor en el momento de su declaración:

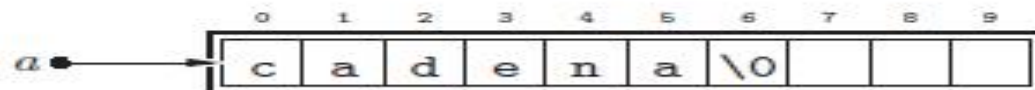
```
char a[10] = "cadena";
```

## Representación de las cadenas en memoria

Al declarar e inicializar una cadena así:

```
char a[10] = "cadena";
```

la memoria queda de este modo:



Es decir, es como si hubiésemos inicializado la cadena de este otro modo equivalente:

```
char a[10] = { 'c', 'a', 'd', 'e', 'n', 'a', '\0' };
```

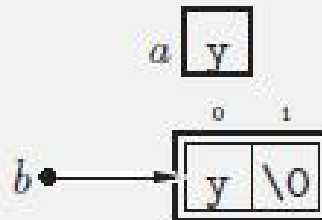
Recuerda, pues, que hay dos valores relacionados con el tamaño de una cadena:

- su *capacidad*, que es la talla del vector de caracteres;
- su *longitud*, que es el número de caracteres que contiene, sin contar el terminador de la cadena. La longitud de la cadena debe ser siempre *estrictamente menor* que la capacidad del vector para no desbordar la memoria reservada.

Cuántos bytes ocupa cada una de las siguientes definiciones ?

```
1 char a = 'y' ;  
2 char b[2] = "y" ;
```

He aquí una representación gráfica de las variables y su contenido:



Recuerda:

- Las comillas simples definen un carácter y un carácter ocupa un solo byte.
- Las comillas dobles definen una cadena. Toda cadena incluye un carácter nulo invisible al final.



Como almacenar 80 caracteres (además del nulo) sin crear una constante con respecto al número de caracteres que caben en ellas?

```
1 #include <stdio.h>
2
3 #define MAXLON 80
4
5 int main(void)
6 {
7     char cadena[MAXLON+1]; /* Reservamos 81 caracteres: 80 caracteres más el terminador */
8
9     return 0;
10 }
```

## Entrada/salida de cadenas

Las cadenas se muestran con `printf` y la adecuada marca de formato sin que se presenten dificultades especiales.

Lo que sí resulta problemático es leer cadenas.

La función `scanf` presenta una seria limitación: sólo puede leer ((palabras)), no ((frases)). Ello nos obligará a presentar una nueva función (`gets`).



## Salida con printf

Empecemos por considerar la función printf , que muestra cadenas con la marca de formato %s.

```
salida_cadena.c salida_cadena.c
1 #include <stdio.h>
2
3 #define MAXLON 80
4
5 int main(void)
6 {
7     char cadena[MAXLON+1] = "una_cadena";
8
9     printf("El_valor_de_cadena_es_%s.\n", cadena);
10
11     return 0;
12 }
```

Al ejecutar el programa obtienes en pantalla esto:

```
El valor de cadena es una cadena.
```

## Lectura con gets

Hay un problema práctico con scanf : sólo lee una ((palabra)), es decir, una secuencia de caracteres no blancos.

¿Cómo leer, pues, una frase completa? N

La función gets lee todos los caracteres que hay hasta encontrar un salto de línea. Dichos caracteres, excepto el salto de línea, se almacenan a partir de la dirección de memoria que se indique como argumento y se añade un terminador.

```
1 #include <stdio.h>
2
3 #define MAXLON 11
4
5 int main(void)
6 {
7     char a[MAXLON+1], b[MAXLON+1];
8
9     printf("Introduce una cadena: "); gets(a);
10    printf("Introduce otra cadena: "); gets(b);
11    printf("La primera es %s y la segunda es %s\n", a, b);
12
13    return 0;
14 }
```



## Asignación y copia de cadena

El siguiente código es correcto ...???

```
#define MAXLON 10
int main(void) {
    char original[MAXLON+1] = "cadena";
    char copia[MAXLON+1];
    copia = original;
    return 0;
}
```

```
1 #define MAXLON 10
2
3 int main(void)
4 {
5     char original[MAXLON+1] = "cadena";
6     char copia[MAXLON+1];
7     int i;
8
9     for (i = 0; i <= MAXLON; i++)
10         copia[i] = original[i];
11
12     return 0;
13 }
```

`strcpy(copia, original);` // Copia el contenido de original en copia.

Ten cuidado: `strcpy` (abreviatura de ((string copy))) no comprueba si el destino de la copia tiene capacidad suficiente para la cadena, así que puede provocar un esbordamiento. La función `strcpy` se limita a copiar carácter a carácter hasta llegar a un carácter nulo.

## Longitud de una cadena

El convenio de terminar una cadena con el carácter nulo permite conocer fácilmente la longitud de una cadena:

```
1 #include <stdio.h>
2
3 #define MAXLON 80
4
5 int main(void)
6 {
7     char a[MAXLON+1];
8     int i;
9
10    printf("Introduce una cadena (máx. %d cars.): ", MAXLON);
11    gets(a);
12    i = 0;
13    while (a[i] != '\0')
14        i++;
15    printf("Longitud de la cadena: %d\n", i);
16
17    return 0;
18 }
```

Calcular la longitud de una cadena es una operación frecuentemente utilizada, así que está predefinida en la biblioteca de tratamiento de cadenas (string.h): “strlen”

```
char a[MAXLON+1];
int l;
```

```
printf("Introduce una cadena (m'ax. %d cars.): ", MAXLON);
gets(a);
l = strlen(a);
```



## Concatenación

En C no puedes usar + para concatenar cadenas.

Una posibilidad es que las concatenes a mano, con bucles.

Este programa, por ejemplo, pide dos cadenas y concatena la segunda a la primera:

```
1 #include <stdio.h>
2
3 #define MAXLON 80
4
5 int main(void)
6 {
7     char a[MAXLON+1], b[MAXLON+1];
8     int longa, longb;
9     int i;
10
11     printf("Introduce un texto (máx. %d cars.): ", MAXLON); gets(a);
12     printf("Introduce otro texto (máx. %d cars.): ", MAXLON); gets(b);
13
14     longa = strlen(a);
15     longb = strlen(b);
16     for (i=0; i<longb; i++)
17         a[longa+i] = b[i];
18     a[longa+longb] = '\0';
19     printf("Concatenación de ambos: %s", a);
20
21     return 0;
22 }
```

Pero es mejor usar la función de librería strcat :

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAXLON 80
5
6 int main(void)
7 {
8     char a[MAXLON+1], b[MAXLON+1];
9
10    printf("Introduce_un_texto_(máx._%d_cars.):_", MAXLON);
11    gets(a);
12    printf("Introduce_otro_texto_(máx._%d_cars.):_", MAXLON);
13    gets(b);
14    strcat(a, b); // Equivale a la asignación Python  $a = a + b$ 
15    printf("Concatenación_de_ambos:_%s", a);
16
17    return 0;
18 }
```



Si quieres dejar el resultado de la concatenación en una variable distinta, que debo hacer...

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAXLON 80
5
6 int main(void)
7 {
8     char a[MAXLON+1], b[MAXLON+1], c[MAXLON+1];
9
10    printf("Introduce_un_texto_(máx._%d_cars.):_", MAXLON);
11    gets(a);
12    printf("Introduce_otro_texto_(máx._%d_cars.):_", MAXLON);
13    gets(b);
14    strcpy(c, a); // Ésta seguida de...
15    strcat(c, b); // ... ésta equivale a la sentencia Python c = a + b
16    printf("Concatenación_de_ambos:_%s", c);
17
18    return 0;
19 }
```

```
char linea[10] = "cadena";  
char character = 's';
```

Son validas las siguientes instrucciones?

```
streat(linea, character);
```

```
strcpy(linea, 'x');
```

### Un carácter no es una cadena

Un error frecuente es intentar añadir un carácter a una cadena con *streat* o asignárselo como único carácter con *strcpy*:

Recuerda: los dos datos de *streat* y *strcpy* han de ser *cadena*s y no es aceptable que uno de ellos sea un *carácter*.



## Comparación de cadenas

Tampoco los operadores de comparación (`==`, `!=`, `<`, `<=`, `>`, `>=`) funcionan con cadenas. La función de `<string.h>` que permite paliar esta carencia de C: `strcmp`

La función `strcmp` recibe dos cadenas, `a` y `b`, y devuelve un entero. El entero que resulta de efectuar la llamada `strcmp(a, b)` codifica el resultado de la comparación:

es menor que cero si la cadena `a` es menor que `b`,  
es 0 si la cadena `a` es igual que `b`, y  
es mayor que cero si la cadena `a` es mayor que `b`.

Naturalmente, menor significa que va delante en orden alfabético, y mayor que va detrás.

## Funciones útiles para manejar caracteres

string.h y ctype.h encontrarás unas funciones que permiten hacer preguntas acerca de los caracteres.

- *isalnum(carácter)*: devuelve cierto (un entero cualquiera distinto de cero) si *carácter* es una letra o dígito, y falso (el valor entero 0) en caso contrario,
- *isalpha(carácter)*: devuelve cierto si *carácter* es una letra, y falso en caso contrario,
- *isblank(carácter)*: devuelve cierto si *carácter* es un espacio en blanco o un tabulador,
- *isdigit(carácter)* devuelve cierto si *carácter* es un dígito, y falso en caso contrario,
- *isspace(carácter)*: devuelve cierto si *carácter* es un espacio en blanco, un salto de línea, un retorno de carro, un tabulador, etc., y falso en caso contrario,
- *islower(carácter)*: devuelve cierto si *carácter* es una letra minúscula, y falso en caso contrario,
- *isupper(carácter)*: devuelve cierto si *carácter* es una letra mayúscula, y falso en caso contrario.
- *toupper(carácter)*: devuelve la mayúscula asociada a *carácter*, si la tiene; si no, devuelve el mismo carácter,
- *tolower(carácter)*: devuelve la minúscula asociada a *carácter*, si la tiene; si no, devuelve el mismo carácter.



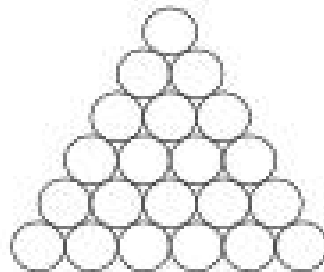
### Pilas

*Una pila es una lista de elementos a la cual se puede insertar o eliminar alguno solo por uno de los extremos, por lo tanto, los elementos de una pila serán eliminados en orden inverso al que se insertaron. Es decir, el último elemento que se introduce en la pila es el primero que se saca.*

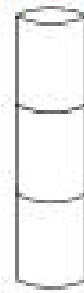
Con analogía con la vida real, por ejemplo puede verse una estructura de pila de platos, es de suponer que el cocinero, si necesita un plato limpio, tomará el que está encima de todos, que es el último que se colocó en la pila. Otros ejemplos de pilas son las latas en un supermercado dispuestas de distintas maneras



a) Pila de Platos



b) Pila de latas



c) Pila de latas

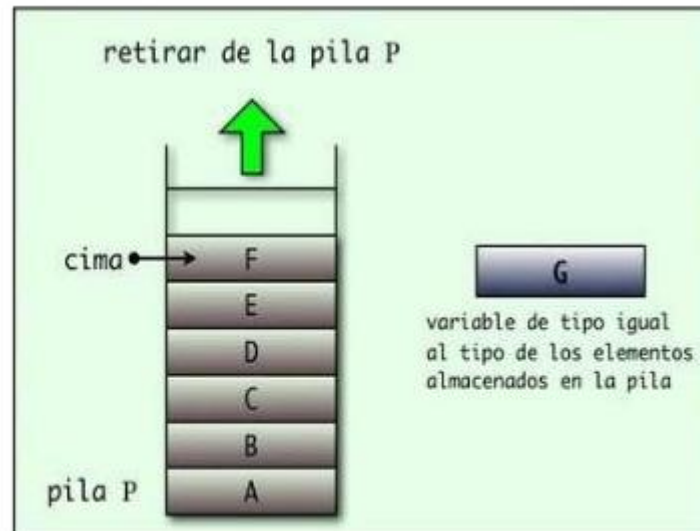
Debido al orden en el cual se insertan o eliminan elementos de una pila, a esta estructura también se la conoce como estructura

**LIFO** (*last - In, First - Out: ultimo en entrar, primero en salir*)

Las pilas pertenecen al grupo de estructuras de datos lineales, ya que los componentes ocupan lugares sucesivos en la estructura.

# Pilas

Una pila es una método de estructuración datos usando la forma LIFO (último en entrar, primero en salir), que permite almacenar y recuperar datos.



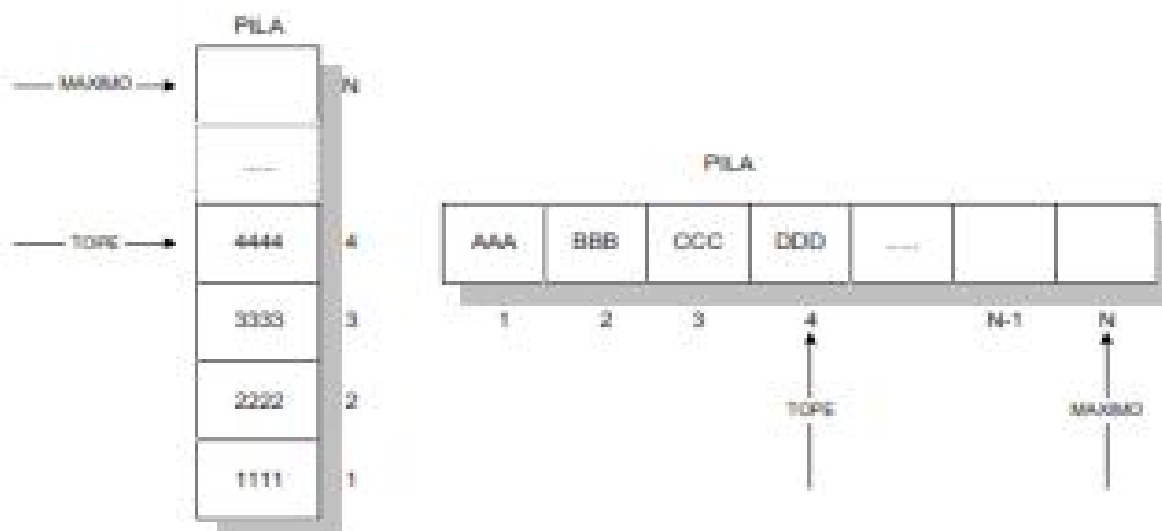


## Representación de pilas

Las pilas no son estructuras fundamentales de datos, es decir, no están definidos como tales en los lenguajes de programación, como lo están, por ejemplo los arreglos. Se pueden representar mediante el uso de:

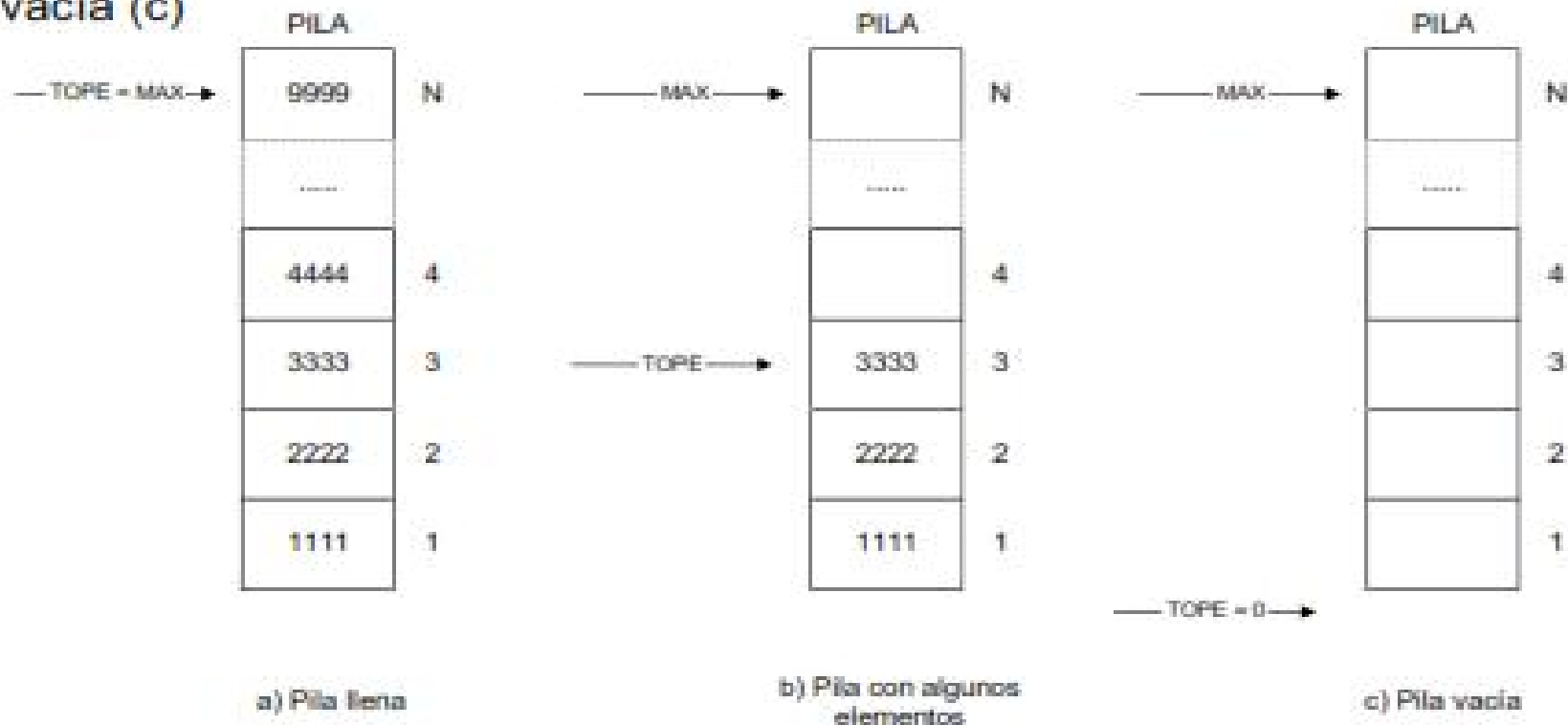
- Arreglos
- Listas Enlazadas

Aquí se utilizarán ARREGLOS, por lo que se deberá definir el tamaño máximo de la pila y además una variable auxiliar a la que se denominará **TOPE** o **CIMA**, que será un apuntador al último elemento insertado en la pila.



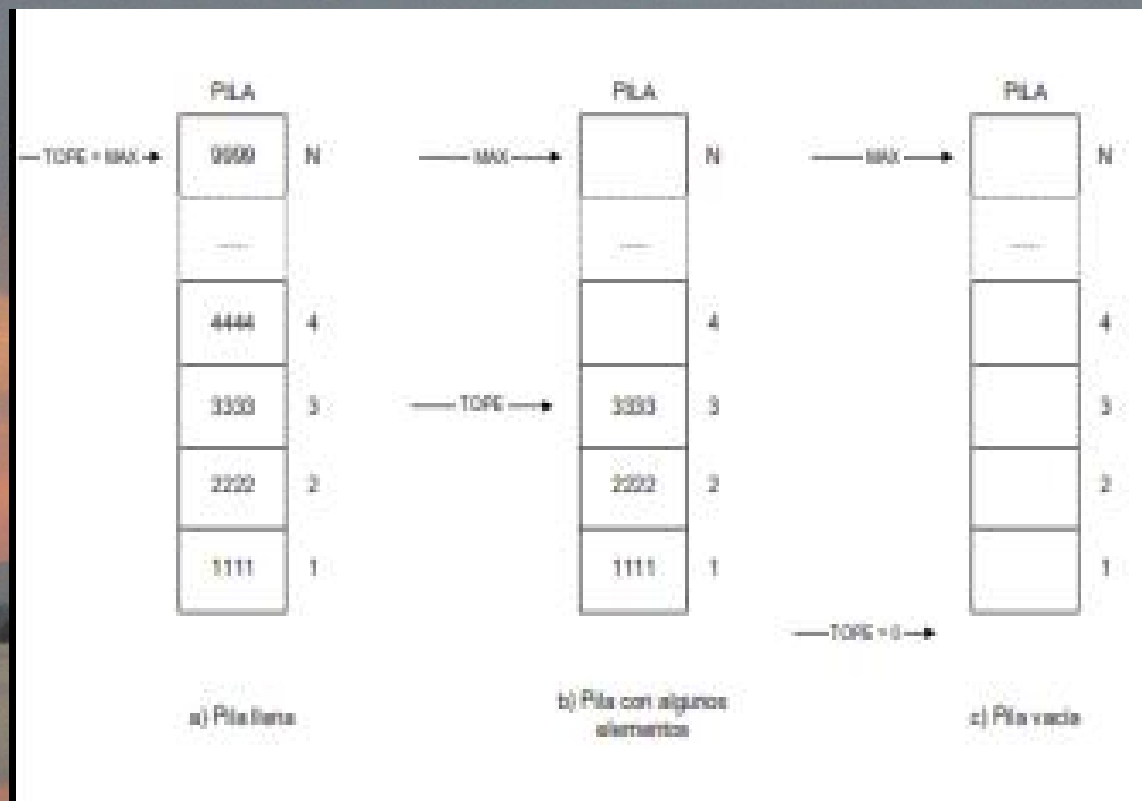
## Representación de pilas

se representan ejemplos de pila llena (a), pila con algunos elementos (b) y pila vacía (c)





## Representación de pilas



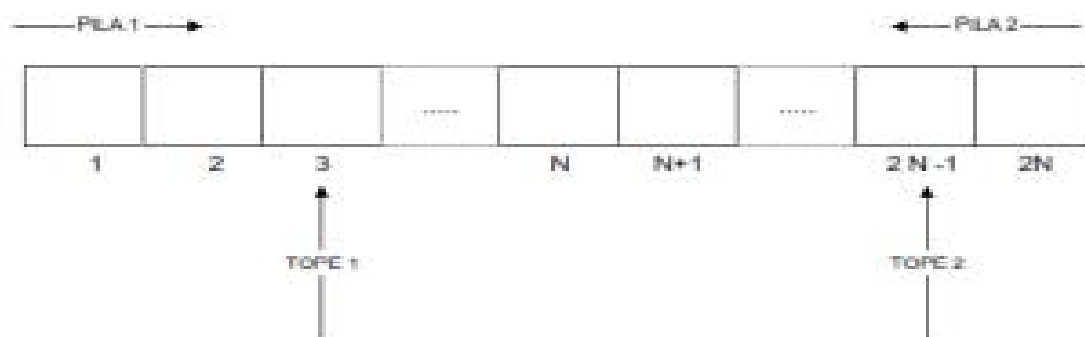
Al utilizar esta estructura con arreglos, se debe tener en cuenta el tamaño máximo de la pila; cuando la variable auxiliar TOPE llegue al máximo definido, no se podrá insertar mas elementos. Una vez que la pila esta llena ( $TOPE = MAX$ ), y si se intenta insertar un nuevo valor, ocurrirá un error conocido con el nombre de *desbordamiento K* (OVERFLOW).

Por ejemplo en la figura donde  $TOPE = MAX$  si se quisiera insertar otro elemento, se tendria un error de desbordamiento. La pila está llena y el espacio reservado de memoria es fijo, no puede en este caso ni expandirse ni contraerse.

Como una primera solución a este tipo de errores se puede definir pilas de gran tamaño, pero esto resulta costoso e ineficiente uso de la memoria si solo se ocupan algunos elementos de la misma.

## Representación de pilas

Otra alternativa que existe como solución es definir dos pilas de  $N$  elementos en un solo arreglo de  $2 \cdot N$  elementos, esto se conoce con el nombre de **espacios compartidos** de memoria. Supongamos que se necesitan dos pilas de  $N$  elementos máximos cada una, se definirá un solo arreglo de  $2 \cdot N$  elementos en lugar de dos arreglos de  $N$  elementos cada uno.



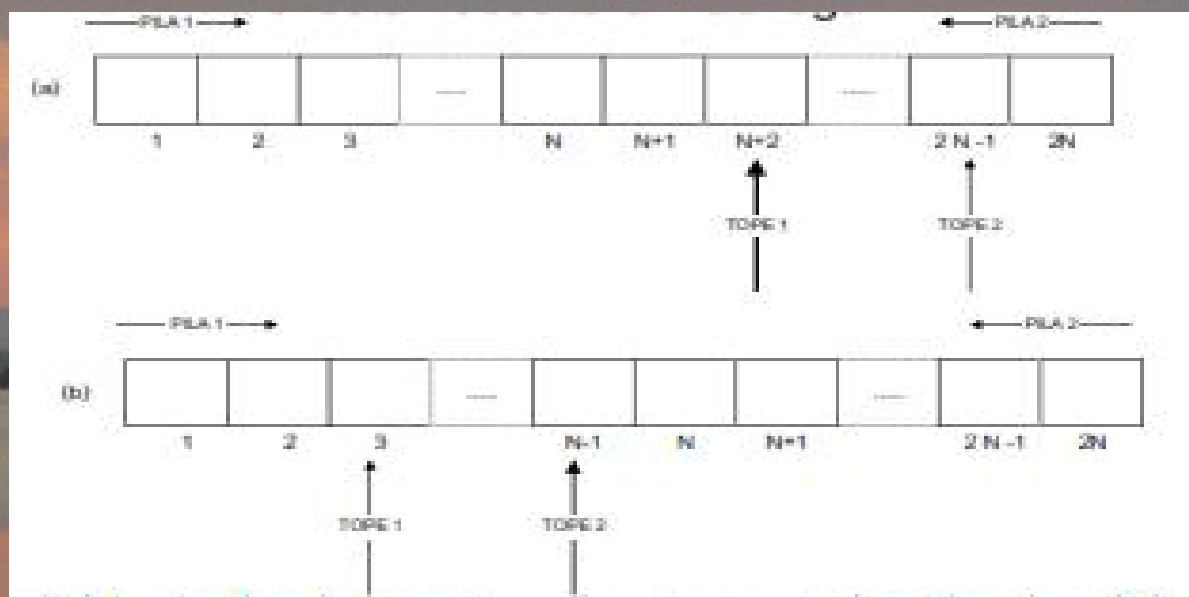
Representación de pilas en espacios compartidos

En la figura la PILA 1 ocupará desde la posición 1 en adelante, mientras que la PILA 2 ocupará desde el espacio  $2N$  hacia atrás ( $2N-1$ ,  $2N-2$ , ....).



## Representación de pilas

Puede ocurrir que en algún punto del proceso la PILA 1 necesitara más de  $N$  espacios y en ese momento la PILA 2 no tuvieran sus  $N$  lugares ocupados, en consecuencia se podría seguir agregando elementos a la PILA 1 sin caer en el error de desbordamiento (ver figura). El mismo tratamiento se haría en la PILA 2 si esta necesitara más lugar.



En la figura (a) La PILA 1 tiene más de  $N$  elementos y la PILA 2 tiene menos de  $N$  elementos.

En la figura (b) ocurre lo contrario, la PILA 2 tiene más de  $N$  elementos y la PILA 1 tiene menos de  $N$  elementos.

Existe otro tipo de error que se lo conoce con el nombre de **subdesbordamiento** (*underflow*) es cuando se intenta eliminar un elemento de una pila vacía.

## OPERACIONES BASICAS CON PILAS

- PUSH (insertar).**- Agrega un elementos a la pila en el extremo llamado **tope**.
- POP (remover).**- Remueve el elemento de la pila que se encuentra en el extremo llamado **tope**.
- VACIA.**- Indica si la pila contiene o no contiene elementos.
- LLENA.**- Indica si es posible o no agregar nuevos elementos a la pila.





## Operaciones

Las operaciones más elementales que pueden realizarse con una estructura de pila son:

- a) Poner un elemento (*Push*)
- b) Quitar un elemento (*Pop*).

La realización de cualquiera de las dos operaciones (inserción o extracción) se realizará siempre por la parte superior.

Los algoritmos genéricos para agregar o quitar elementos en una pila son los siguientes:

```
PONE (PILA, MAX, TOPE, DATO)
Si TOPE < MAX (Verifica que haya espacio libre)
entonces
    TOPE = TOPE + 1 (actualiza TOPE)
    PILA (TOPE) = DATO
sino
    escribir desbordamiento
Fin_si
```

```
QUITA (PILA, TOPE, DATO)
Si TOPE > 0 (Verifica que la pila no está vacía)
Entonces
    DATO = PILA (TOPE)
    TOPE = TOPE - 1
Sino
    Imprimir "Subdesbordamiento"
Fin_si
```

## Estructuras de datos lineales

### Pilas

#### Operaciones

*PONE (PILA, MAX, TOPE, DATO)*

Si  $TOPE < MAX$  (Verifica que haya espacio libre)  
entonces

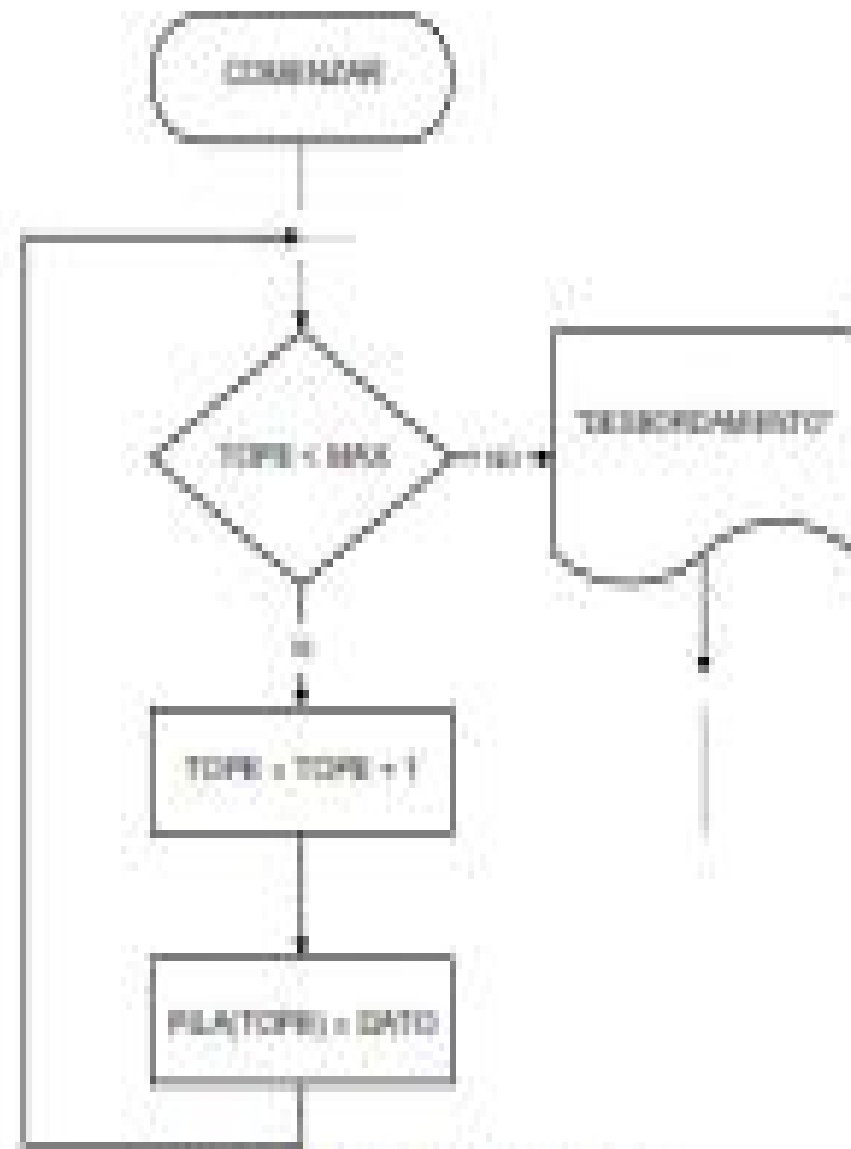
$TOPE = TOPE + 1$  (actualiza TOPE)

$PILA(TOPE) = DATO$

sino

escribir desbordamiento

Fin\_si



AGREGA ELEMENTOS A LA PILA



## Estructuras de datos lineales

### Pilas

#### Operaciones

QUITA (PILA, TOPE, DATO)

Si  $TOPE > 0$  (Verifica que la pila no está vacía)

Entonces

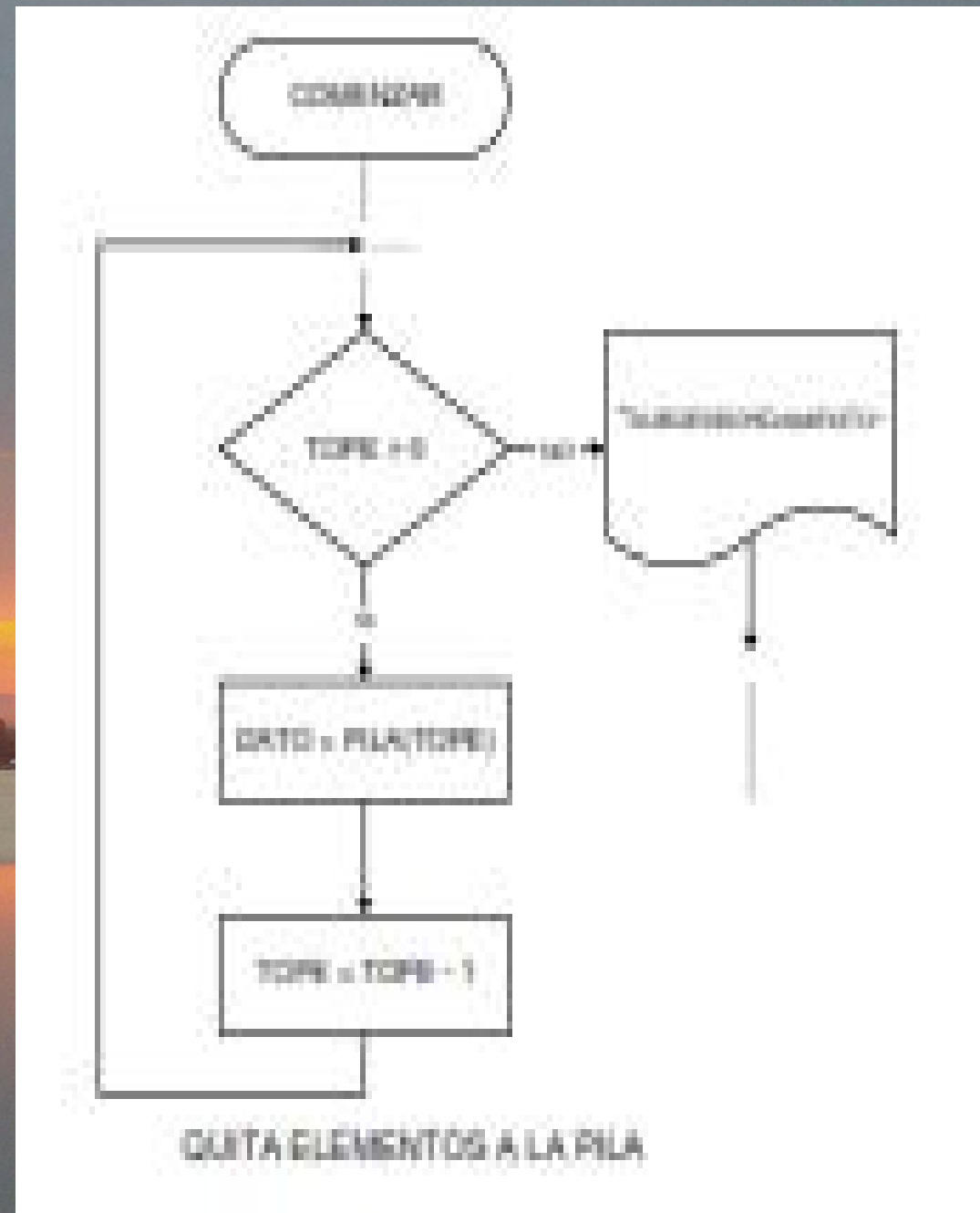
$DATO = PILA(TOPE)$

$TOPE = TOPE - 1$

Sino

Imprimir "Subdesbordamiento"

Fin\_si

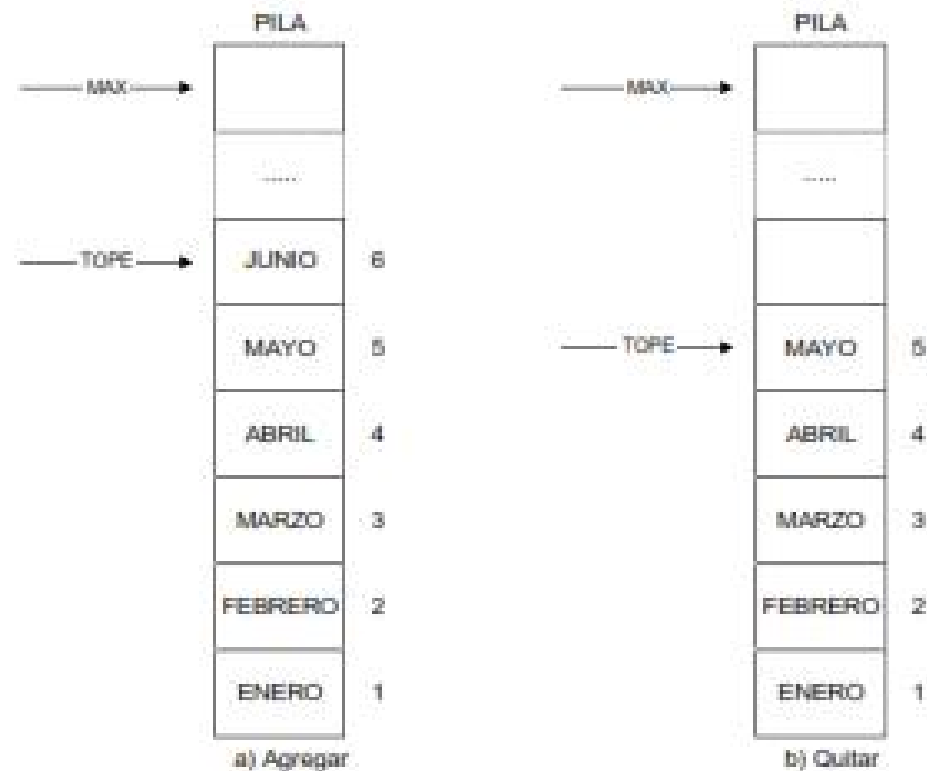


## Estructuras de datos lineales

### Pilas

### Ejemplo

Si se coloca (push) los 6 primeros meses del año en una pila la estructura quedaría como muestra la figura a). Su TOPE quedaría en 6 (Junio). En cambio si se quitara el mes junio, el TOPE apuntaría al mes Mayo (figura b)).

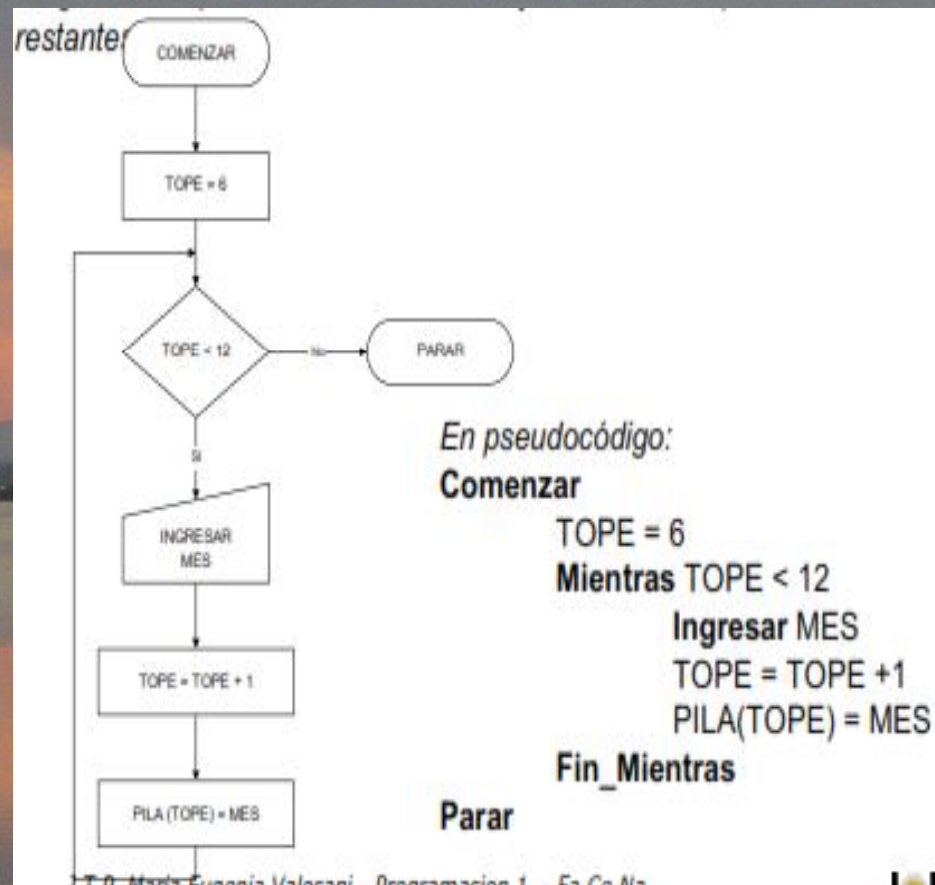




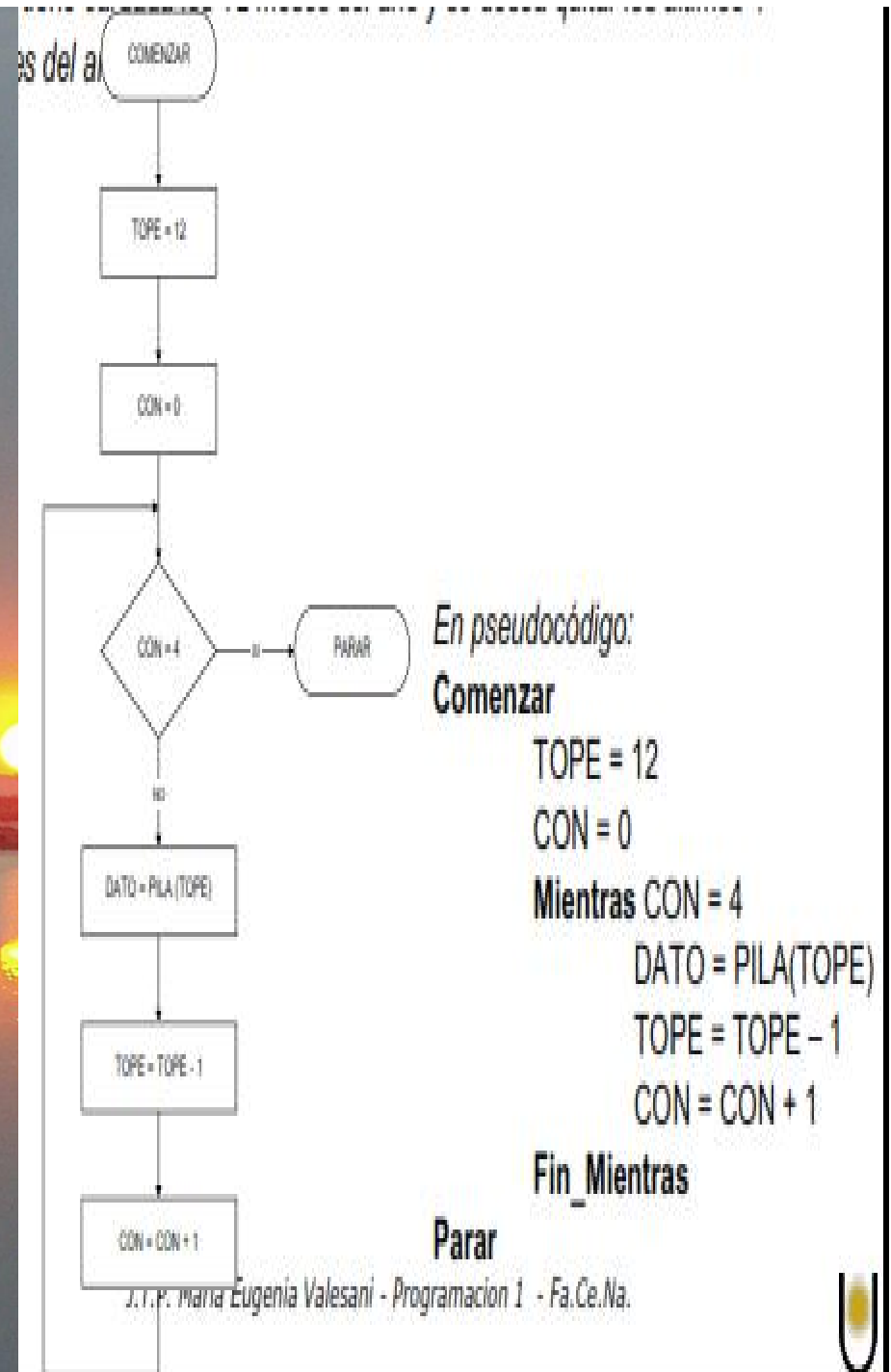
## Estructuras de datos lineales

### Pilas

### Ejemplo



Magter. Oscar Adolfo Vallejos  
FaCENA - UNNE



# Mostrar el funcionamiento de una PILA ESTATICA

...

## Pila implementada sobre un Arreglo

Vacia() : lógico

Inicio

si (tope == Nulo) entonces  
retornar verdadero

sino

retornar falso

Fin

Llena(): lógico

Inicio

retornar tope  $\leftarrow$  MAX\_ELEMENTO-1

Fin

Tope():Tipo

Inicio

si ( no Vacia() )  
retornar arreglo[tope-1]

sino

Error("LA PILA ESTA VACIA")

Fin

Apilar(x: Tipo )

Inicio

si( no Llena( ) ) entonces

inicio

elementos [tope]  $\leftarrow$  x

tope  $\leftarrow$  tope+1

fin

sino

Error("LA PILA ESTA LLENA")

Fin

Desapilar(): Tipo

Inicio

si( no Vacia( ) )

inicio

tope  $\leftarrow$  tope - 1

retornar arreglo[tope]

fin

sino

Error("LA PILA ESTA VACIA")

Fin



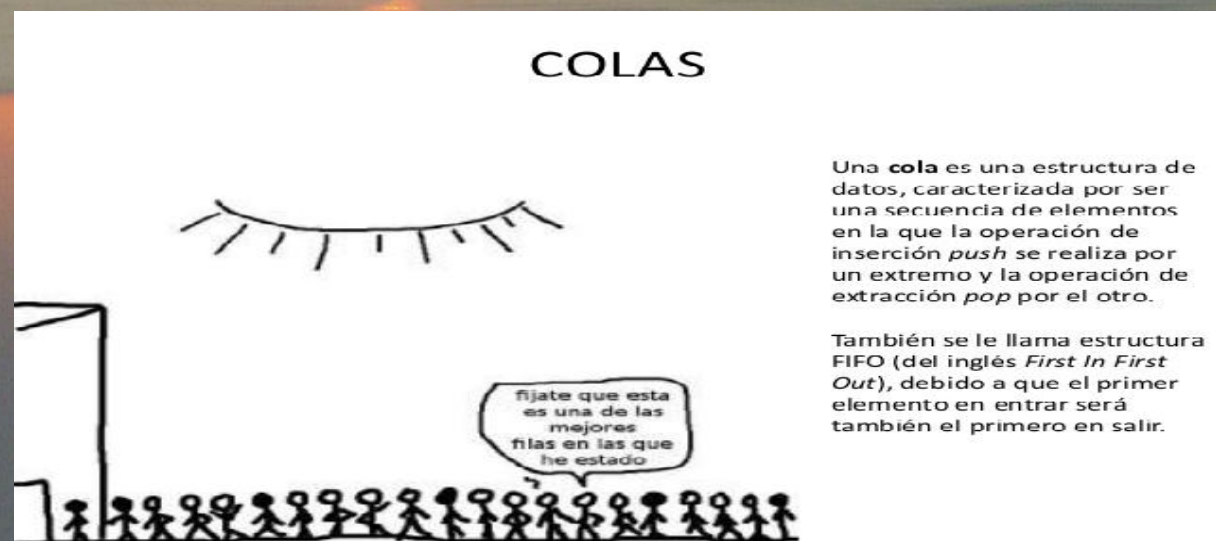
Las *colas* son otro tipo de estructura de datos, similar a las pilas, pero se diferencian de ellas en la forma de insertar/eliminar elementos.

Una **cola** es entonces una lista de elementos en la que estos se introducen por un extremo(final) y se eliminan por otro (frente). La eliminación se realiza en el mismo orden en que fueron insertados, por lo tanto el primer elemento introducido será el primero en ser eliminado.

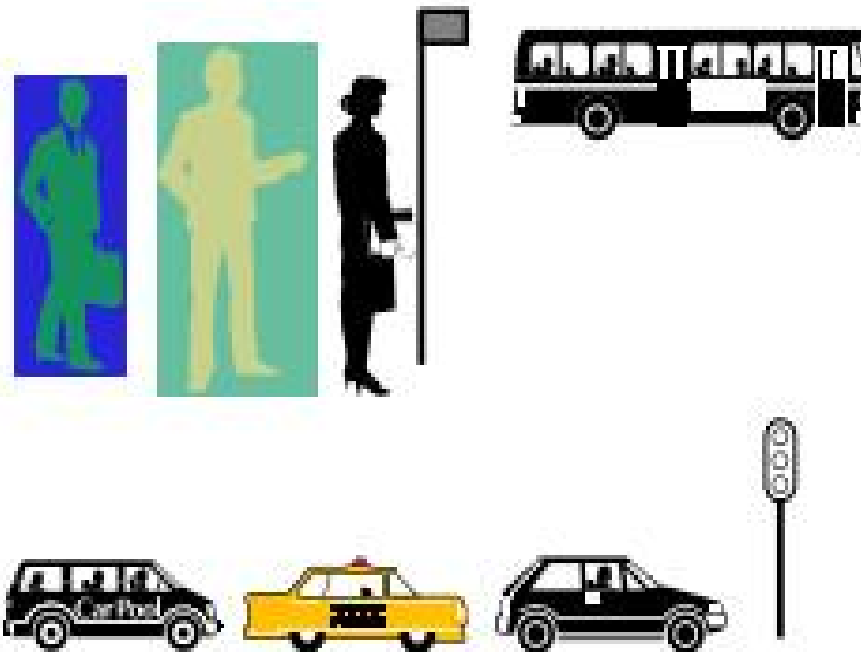
Por esta característica las colas reciben el nombre de

**FIFO (*Fist-in, First-Out: Primero en entrar, primero en salir*)**

Por esta característica este tipo de estructura se utiliza para almacenar datos que necesitan ser procesados según el orden de llegada.



En la vida real encontramos numerosos casos de colas: personas esperando un autobus, personas que esperan ser atendidas en un banco, autos que esperan señal de semáforo, entre otros. En todos estos casos (personas, autos) el primero que llega será el primero en salir.



# Tipos de Colas

- Cola Simple:  
Estructura lineal donde los elementos salen en el mismo orden en que llegan.
- Cola circular :  
Representación lógica de una cola simple en un arreglo.
- Cola de Prioridades:  
Estructura lineal en la cual los elementos se insertan en cualquier posición de la cola y se remueven solamente por el frente.
- Cola Doble (Bicola) :  
Estructura lineal en la que los elementos se pueden añadir o quitar por cualquier extremo de la cola (Cola bidireccional).

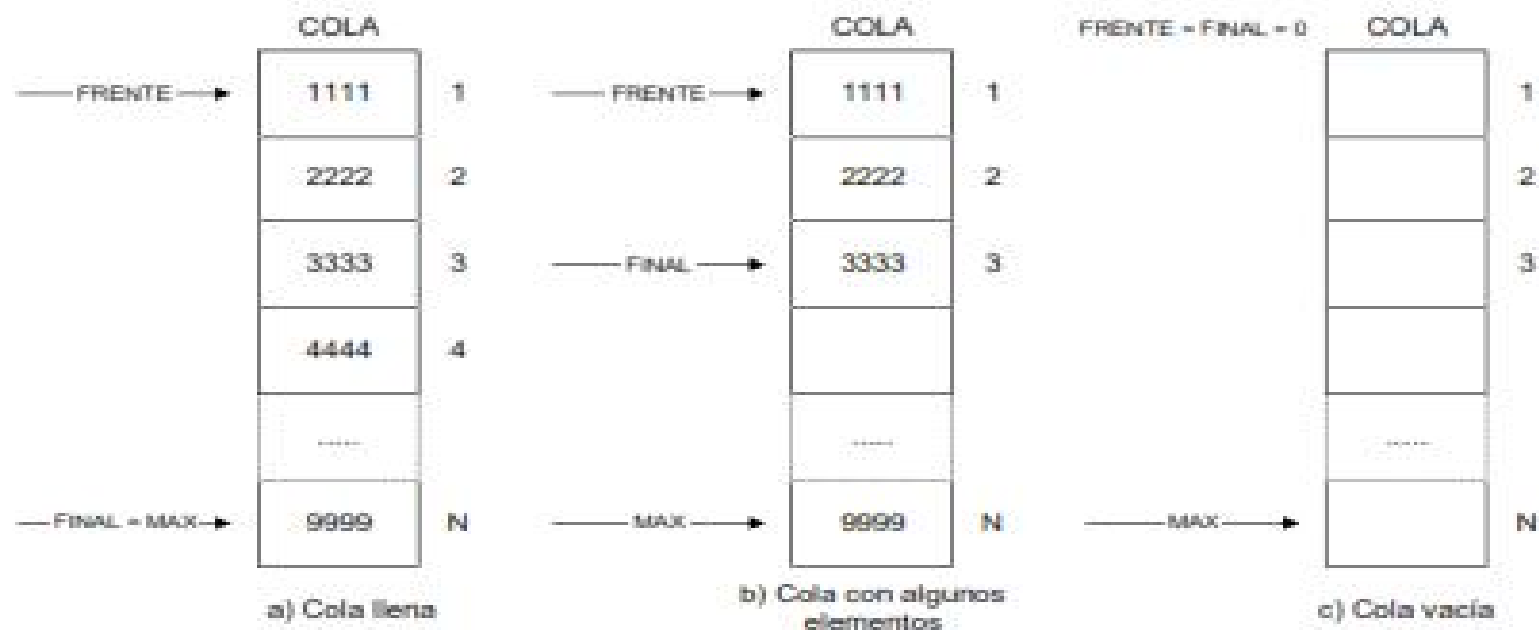


## Estructuras de datos lineales

### Colas

#### Representación

Existen casos de colas llenas, vacías o con algunos elementos, estos se ven representados en la figura



## Operaciones básicas en Colas Simples

**Insertar.**- Almacena al final de la cola el elemento que se recibe como parámetro.

**Eliminar.**- Saca de la cola el elemento que se encuentra al frente.

**Vacía.**- Regresa un valor booleano indicando si la cola tiene o no elementos (true – si la cola esta vacia, false – si la cola tiene al menos un elemento).

**Llena.**- Regresa un valor booleano indicando si la cola tiene espacio disponible para insertar nuevos elementos ( **true** – si esta llena y **false** si existen espacios disponibles).

### Operaciones:

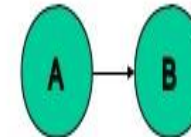
### Estado de la cola:

Inicio: Cola Vacía

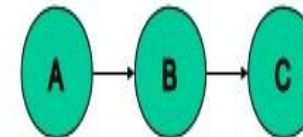
1.- Insertar A



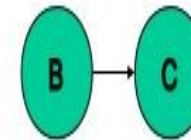
2.- Insertar B



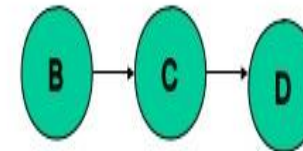
3.- Insertar C



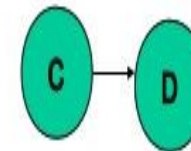
4.- Remove Elemento



5.- Insertar D



6.- Remove Elemento



# Operaciones básicas en Colas Simples

- **Insertar.**- Almacena al final de la cola el elemento que se recibe como parámetro.
- **Eliminar.**- Saca de la cola el elemento que se encuentra al frente.
- **Vacía.**- Regresa un valor booleano indicando si la cola tiene o no elementos (true – si la cola esta vacía, false – si la cola tiene al menos un elemento).
- **Llena.**- Regresa un valor booleano indicando si la cola tiene espacio disponible para insertar nuevos elementos ( true – si esta llena y false si existen espacios disponibles).



Las operaciones que podemos realizar con colas son:

- **insertar un elemento**
- **eliminar un elemento de la cola.**

Recordando la característica FIFO de las colas, la inserción se realiza por el FINAL de la cola, mientras que la eliminación se realiza por el FRENTE.

Los algoritmos genéricos para agregar o quitar elementos de una cola son los siguientes:

*PONE (COLA, MAX, FRENTE, FINAL, DATO)*

**Si** FINAL < MAX (Verifica que haya espacio libre)  
**entonces**

    FINAL = FINAL + 1 (actualiza FINAL)

    COLA (FINAL) = DATO

**Si** FINAL = 1 {se insertó el primer elemento de la pila}

**Entonces**

        FRENTE = 1

**Fin\_si**

**sino**

**Imprimir** "Subdesbordamiento"

**Fin\_si**

*QUITA (COLA, MAX, FRENTE, FINAL, DATO)*

**Si** FRENTE <> 0 (Verifica que la cola no está vacía)

**Entonces**

        DATO = COLA (FRENTE)

**Si** FRENTE = FINAL (si hay un solo elemento)

**Entonces**

                FRENTE = 0 (indica cola vacía)

                FINAL = 0

**Sino**

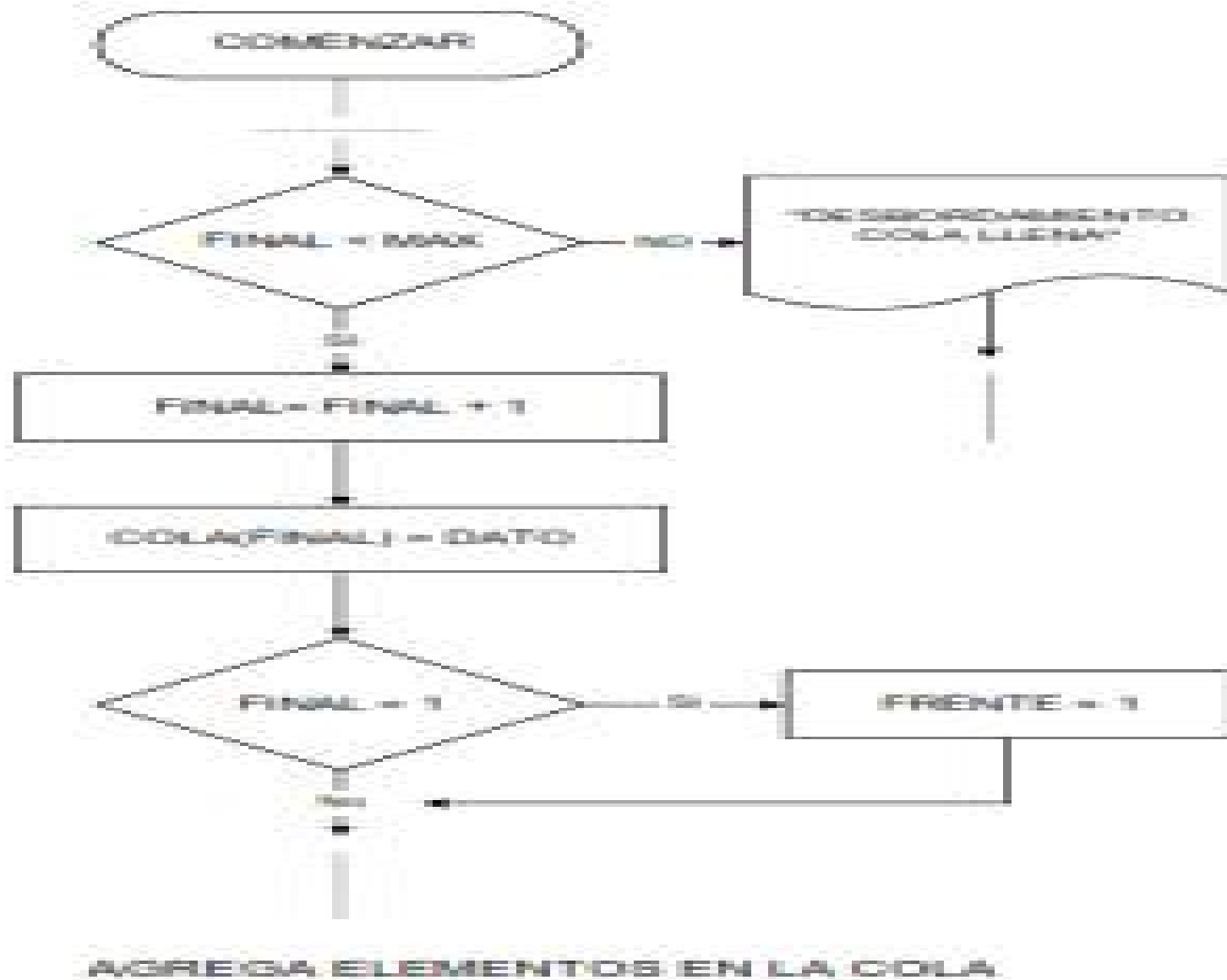
                FRENTE = FRENTE + 1

**Fin\_si**

**Sino**

**escribir** Subdesbordamiento

**Fin\_si**





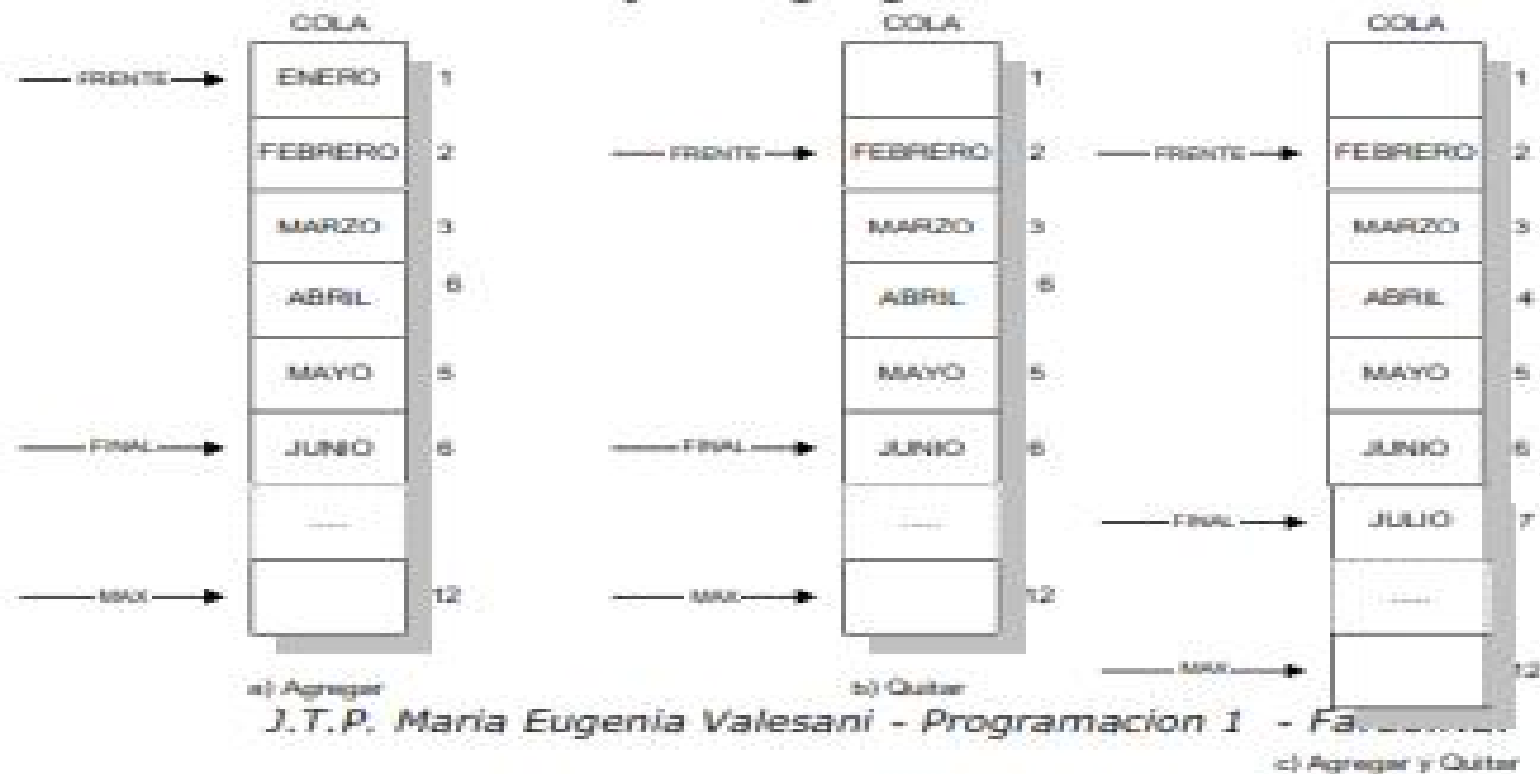


## Estructuras de datos lineales

### Colas

#### 3.2.3.2 Operaciones: Ejemplo

Si se colocan los 6 primeros meses del año en una cola, la estructura quedaría como muestra la figura a). En este caso su MAXIMO sería de 12 (Diciembre) y su FINAL en 6. Caso b): En cambio si se quitara el mes ENERO, el FINAL apuntaría al elemento 6 y el FRENTE al 2. Como tercer opción - caso c) – se realizan las dos operaciones juntas (a partir del caso a), por un lado se quita el primer mes - FRENTE = 2 – y se agrega el mes de Julio FINAL = 7.



## Estructuras de datos lineales

### Colas

Pone en la cola los 6 primeros meses del año

#### Comenzar

FRENTE = 0

FINAL = 0

MAX = 12

Ingresar COLA

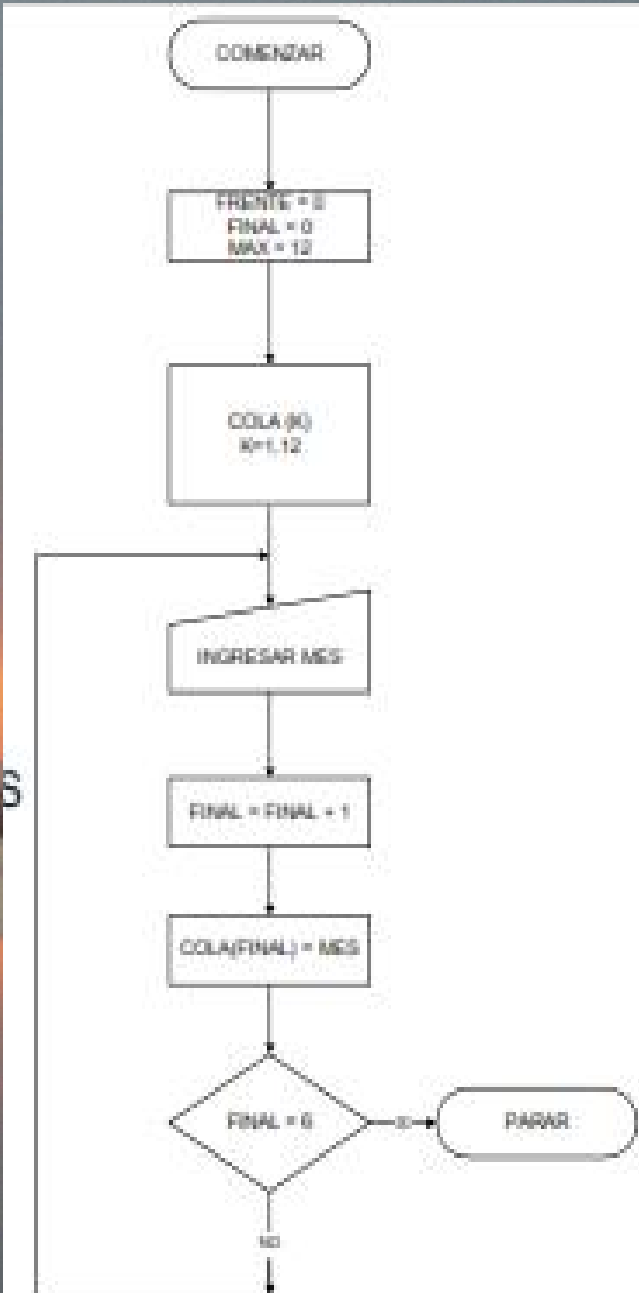
Hasta FINAL = 6

FINAL = FINAL + 1

COLA(FINAL) = MES

Fin\_hasta

#### Parar



## Estructuras de datos lineales

### Colas

Saca de la cola el primer mes

Comenzar

FRENTE = 1

FINAL = 6

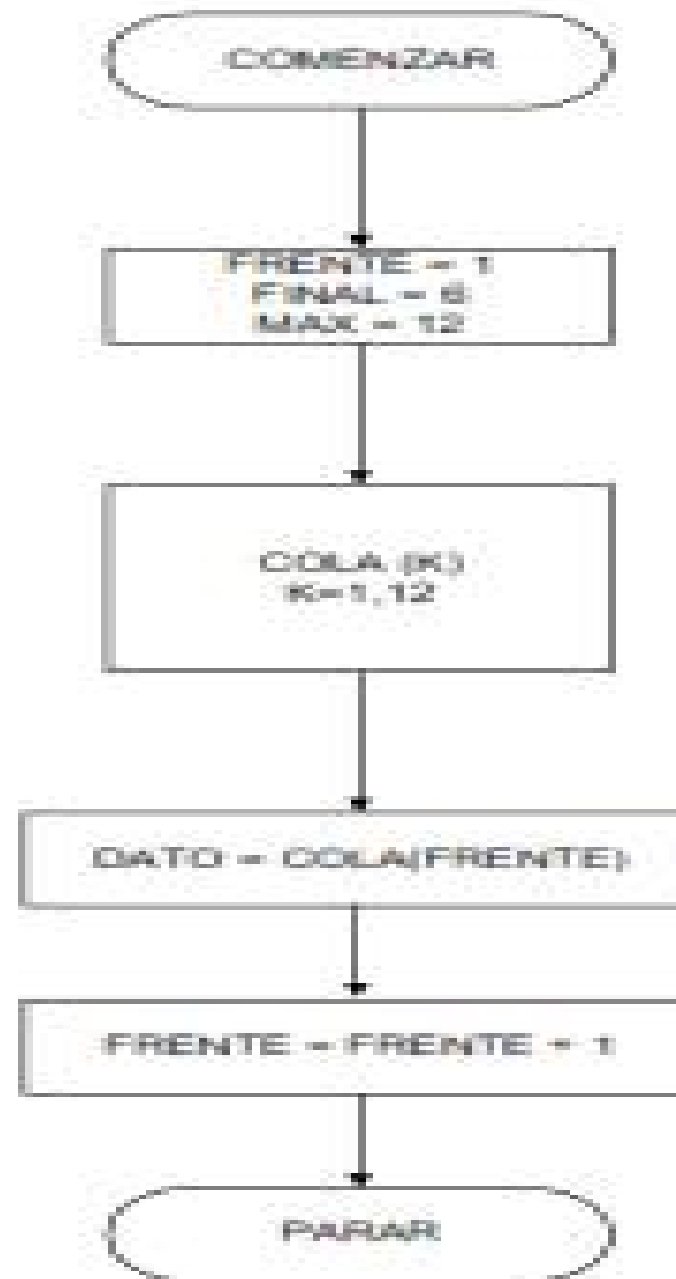
MAX = 12

Ingresar COLA

DATO = COLA(FINAL)

FRENTE = FRENTE + 1

Parar

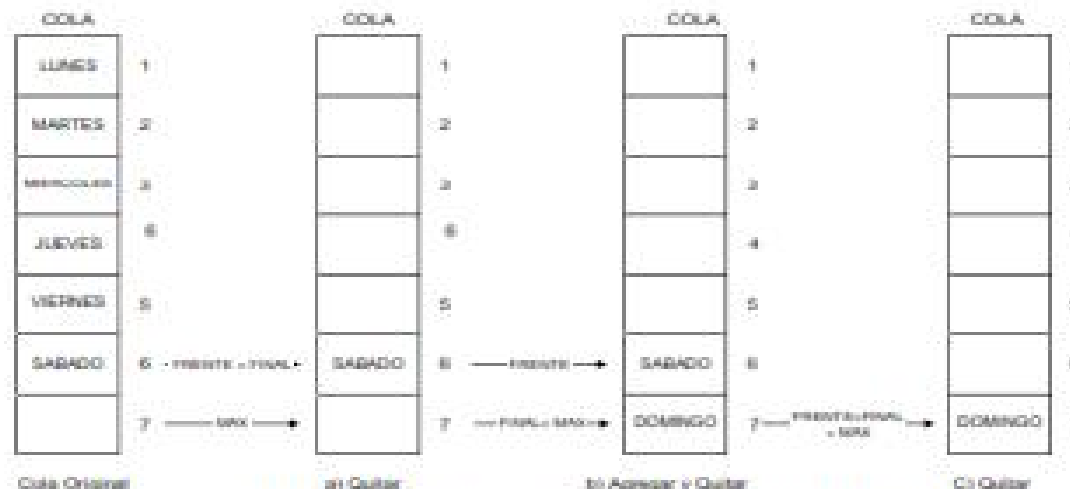




### Colas

#### 3.2.3.2 Operaciones: Ejemplo

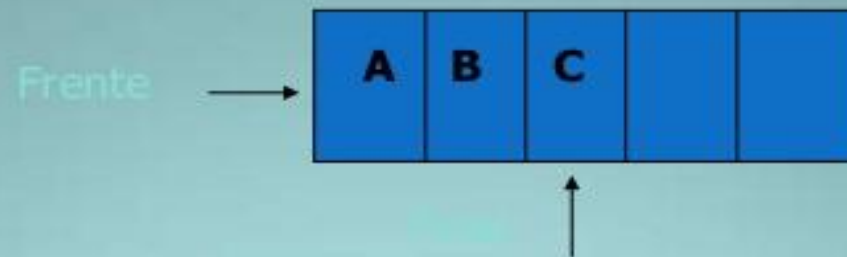
Una cola de los días de la semana cargada como muestra en la figura *Cola Original*. En el caso a) de la figura, eliminamos los elementos desde lunes al viernes. En el caso b) a continuación agregamos el día domingo. Y en el caso c) a continuación se elimina el día sábado.



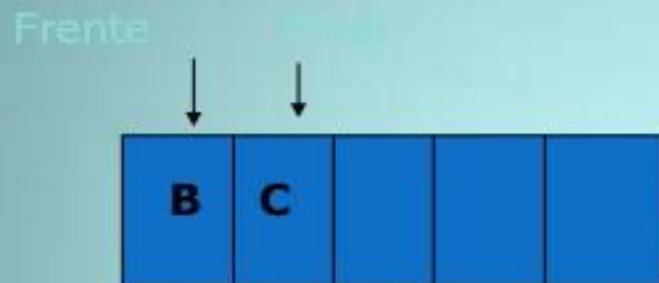
En el ejemplo representado las colas quedan de la siguiente manera:  
Luego de eliminar los días lunes, martes, miércoles, jueves y viernes.  
Luego de agregar el día domingo.  
Luego de quitar el día sábado.

Ya no se podrá agregar un nuevo elemento, dado que  $FINAL = MAX$  a pesar de que – como muestra el caso c) – hay lugar disponible en la cola. En este caso estamos en presencia de un conflicto: ¿Cómo puede ser que habiendo espacio no se pueda agregar elementos? Evidentemente esta situación nos muestra el mal uso de la memoria disponible este tipo de estructura; pero este inconveniente es superado manejando *colas circulares*.

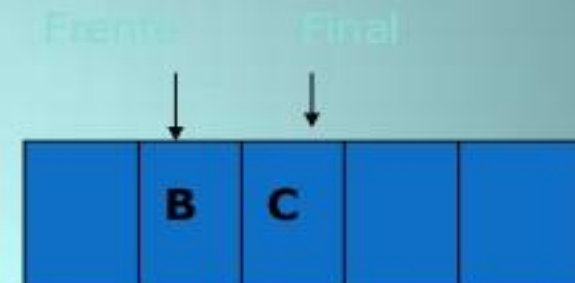
Suponer que usamos un arreglo de 5 posiciones. Usando la representación de frente fijo y frente movable.



Al remover un elemento:

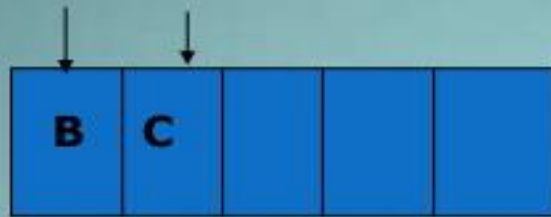


Frente fijo

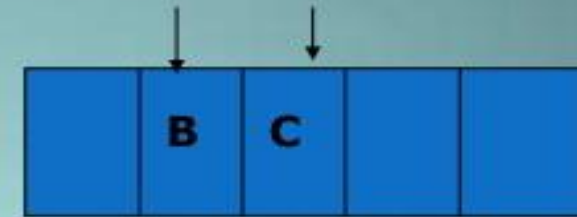


Frente movable

Frente Final

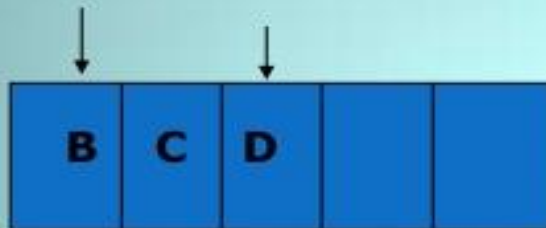


Frente Final

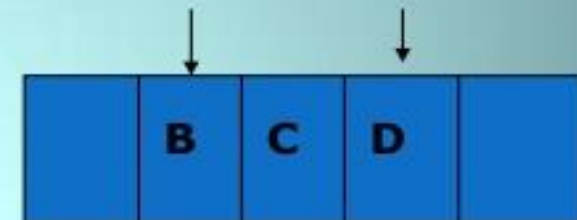


Insertar elemento D

Frente Final



Frente Final





## Insertar elemento E

Frente

Final



Frente

Final



## Insertar elemento F

Frente

Final



Frente

Final



## Insertar elemento G

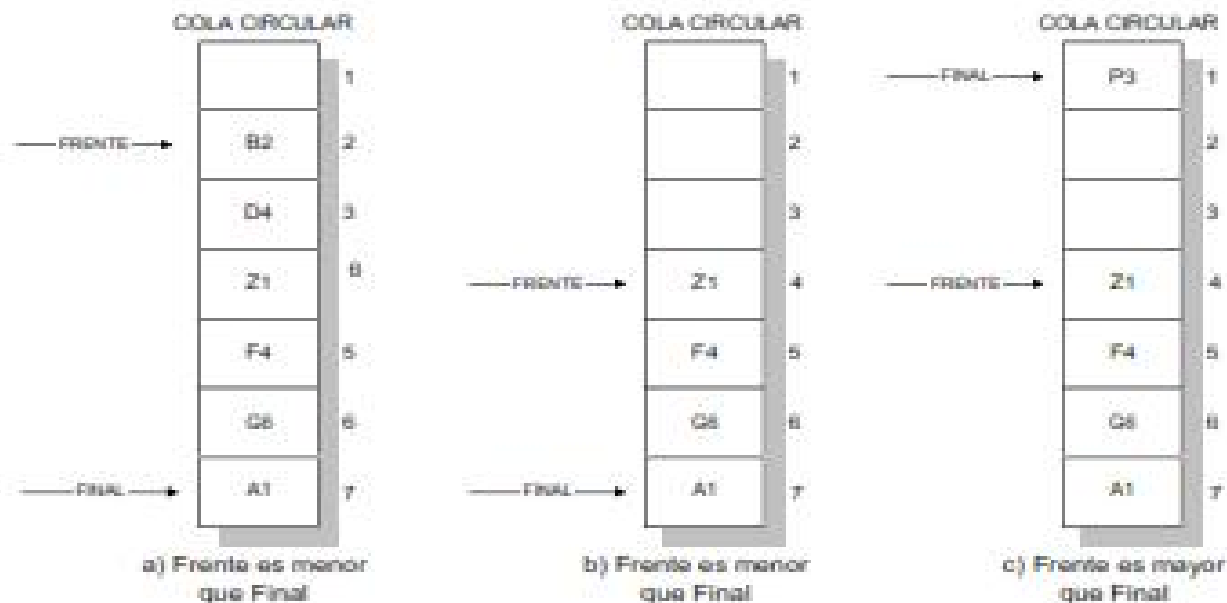
**Cola llena!!!!**

## Colas circulares

### 3.2.3.3 Colas circulares

Una *cola circular* es aquella en la que el elemento primero sigue al elemento último, es decir, el elemento anterior al primero es el último. Este tipo de estructura obliga a dejar siempre una posición libre para separar el principio del final, pero evidentemente siempre existirá la limitación de que pueda llenarse completamente la cola.

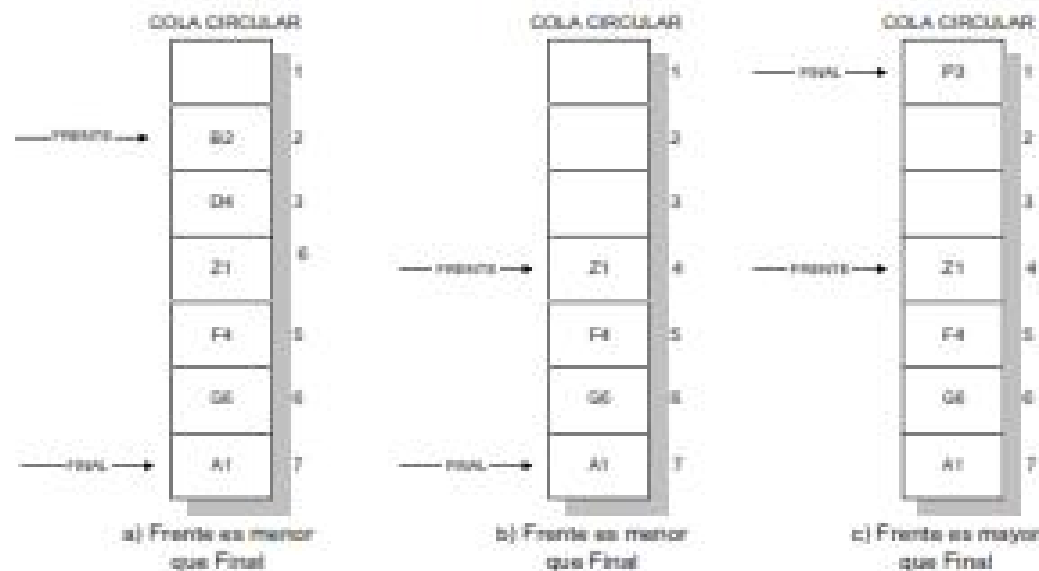
La figura muestra distintas situaciones de una cola circular.



## Colas circulares

### 3.2.3.3 Colas circulares

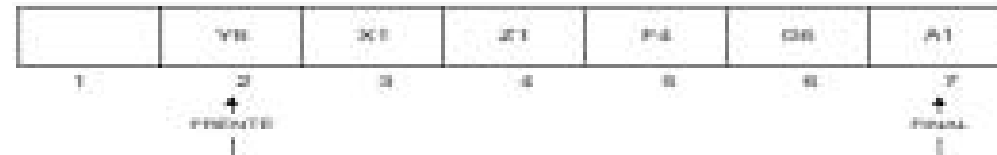
Esta figura ilustra como se ubican los punteros FRENTE y FINAL durante las operaciones de inserción y eliminación. En el caso a) la cola tiene algunos elementos y en este caso el Frente < Final. En el caso b) Se han eliminado dos elementos de la cola y el puntero frente quedó en 4. En el caso c) Se insertaron elementos (P3) que como en este caso FINAL=MÁXIMO se llevó el primer apuntador a la posición que estaba vacía (FINAL = 1). De esta manera se hace un uso mas eficiente de la memoria disponible, ya que al eliminar un elemento esa casilla de la cola queda disponible para futuras inserciones.



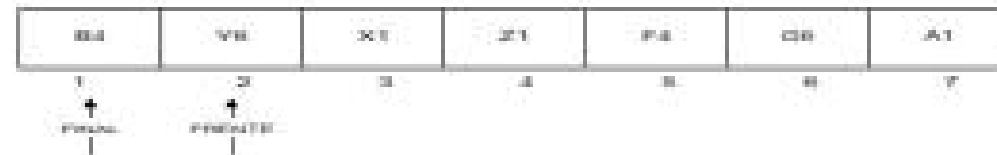


### 3.2.3.3 Colas circulares: se ilustran distintos casos que pueden ocurrir en las colas circulares

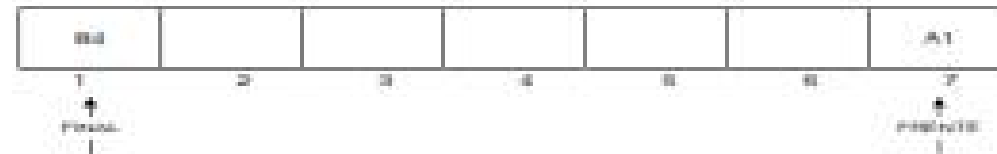
a) COLA CIRCULAR - Estado Inicial



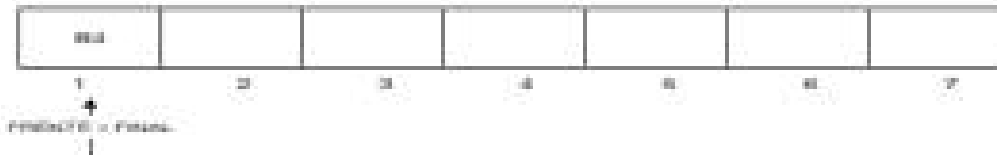
b) COLA CIRCULAR: Luego de insertar un elemento (Cola llena)



c) COLA CIRCULAR: Luego de eliminar elementos



d) COLA CIRCULAR: Luego de eliminar el primer elemento (frontera), se actualizan los punteros y  $\text{FRONTE} = \text{FINAL}$ .



e) COLA CIRCULAR: Cola vacía luego de eliminar el último elemento



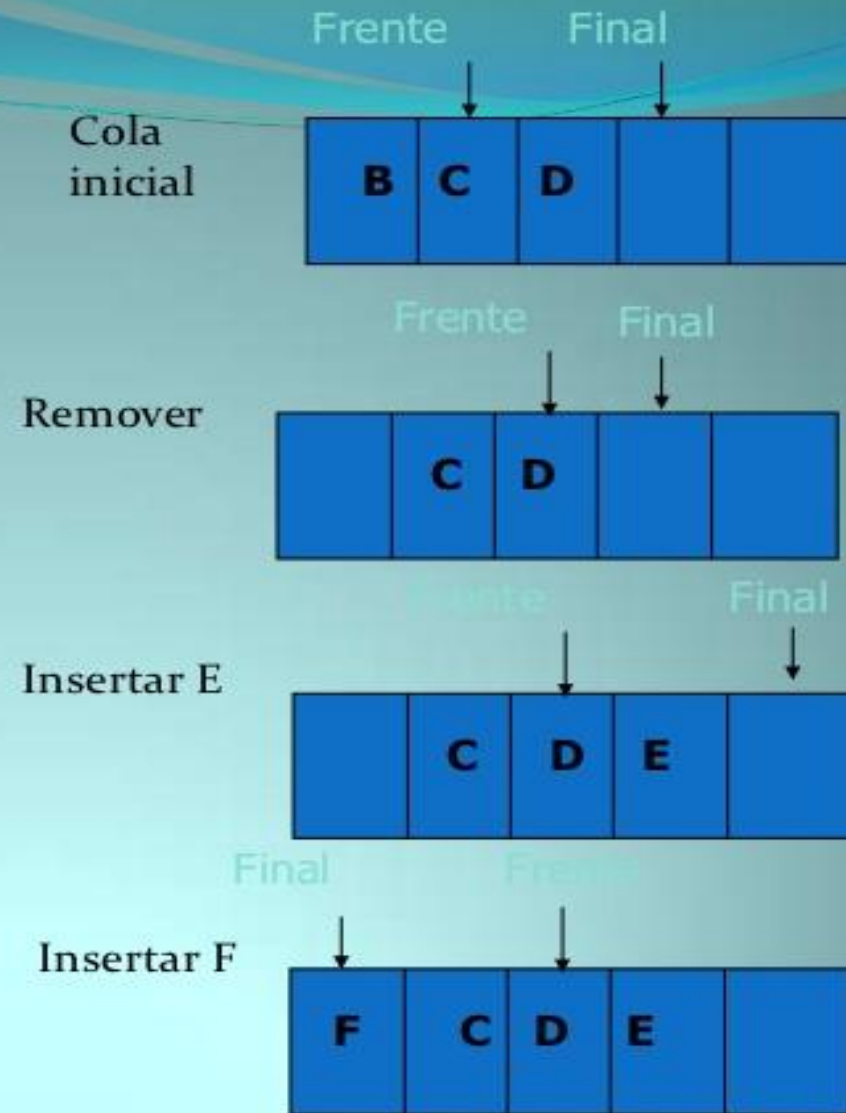
FRONTE = FINAL = 0

UTN - María Eugenia Holwerth - Documento 1 - FaCEN

# Cola Circular

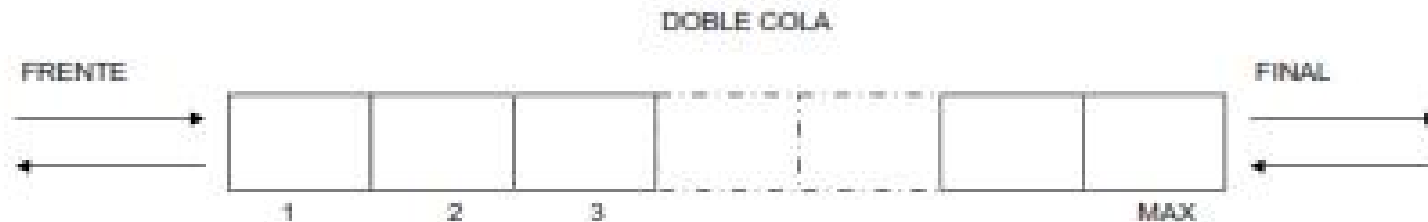
Es una representación lógica de la cola en un arreglo.

El **frente** y **final** son movibles. Cuando el **frente** o **final** llegan al extremo se regresan a la primera posición del arreglo.



## Doble cola

Existe una variante de la cola simple desarrollada anteriormente y es la *doble cola* o *bicola*. Es una cola bidireccional ya que sus elementos pueden ser insertados o eliminados por cualquiera de los extremos: FRENTE o FINAL. La doble cola se representa como se muestra en la figura



Las dos flechas de cada extremo indican que se pueden realizar las operaciones de inserción y eliminación.



## Doble cola

Las variantes que podemos encontrar son :

*Doble cola de entrada restringida:* son las que aceptan inserciones solo al final de la cola permitiendo que sus eliminaciones se realicen por cualquiera de los dos extremos.

*Doble cola de salida restringida:* acepta eliminaciones solo al frente de la cola mientras que las inserciones se pueden hacer por cualquiera de los dos extremos.



## Operaciones Básicas en Estructuras Lineales

1. **Recorrido:** Procesa c/elemento de la estructura.
2. **Búsqueda:** Recupera la posición de un elemento específico.
3. **Inserción:** Adiciona un nuevo elemento a la estructura.
4. **Borrado:** Elimina un elemento de la estructura.
5. **Ordenación:** Ordena los elementos de la estructura de acuerdo a los valores que contiene.
6. **Mezcla:** Combina 2 estructuras en una sola.



# Cola de prioridades

Una **cola de prioridades** es similar a una cola en la que los elementos tienen adicionalmente, una *prioridad* asignada.

En una cola de prioridades un elemento con mayor prioridad será desencolado antes que un elemento de menor prioridad.

Si dos elementos tienen la misma prioridad, se desencolarán siguiendo el orden de cola.

Normalmente se los trata como un TAD



# Operaciones

Una cola de prioridad ha de soportar al menos las siguientes dos operaciones:

- Añadir con prioridad: se añade un elemento a la cola, con su correspondiente prioridad.
- Eliminar elemento de mayor prioridad: se devuelve y elimina el elemento con mayor prioridad más antiguo que no haya sido desencolado de la cola.

Además suele implementarse una función *frente* (que habitualmente aquí se denomina encontrar-máximo o encontrar-mínimo), y que habitualmente se ejecuta en tiempo  $O(1)$ . Esta operación y su rendimiento en tiempo es crucial en ciertas aplicaciones de las colas de prioridades.

Ciertas implementaciones avanzadas pueden incluir operaciones más complejas para la inspección de los elementos de mayor o menor prioridad, borrar la cola o ciertos subconjuntos de la cola, realizar inserciones en masa, la fusión de dos colas en una, aumentar la prioridad de los elementos, etc.

## Características generales

Podrían verse a las colas de prioridades como colas modificadas, en las que en lugar de obtener el siguiente elemento de la cola, se obtiene elemento de mayor prioridad en la cola. De hecho, pilas y colas pueden ser vistas como tipos particulares de colas de prioridad.

Una pila podría verse como una cola de prioridades en la que los elementos son insertados en orden monótono creciente; y por tanto el último elemento insertado es siempre el primero en ser recuperado (ya que tendrá la máxima prioridad hasta el momento). En una cola, la prioridad de cada elemento insertado es monótona decreciente; y por tanto el primer elemento insertado es siempre el primero en ser recuperado (pues todos los elementos subsiguientes tendrán prioridades inferiores).







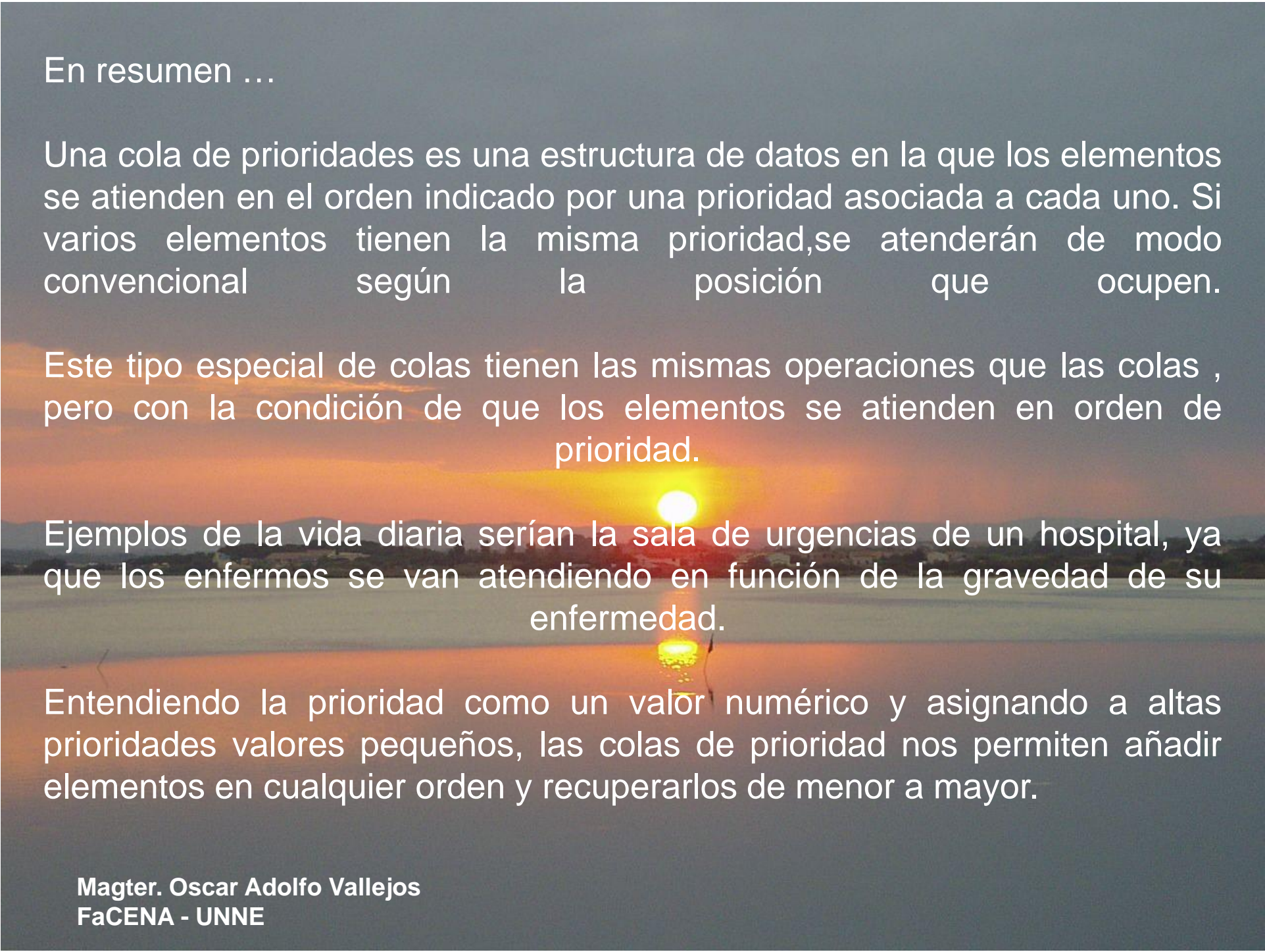
## Aplicaciones:

- La cola en hospital de urgencia.
- Las colas de prioridad se aplican cuando las solicitudes deben procesarse en el orden de prioridad y no en el orden de llegada.
- Gestión de los procesos en un sistema operativo. Los procesos no se ejecutan uno tras otro en base a su orden de llegada. Algunos procesos deben tener prioridad (por su mayor importancia, por su menor duración, etc.) sobre otros.
- Algunos algoritmos sobre grafos, como la obtención de caminos o árboles de expansión de mínimo coste, son ejemplos representativos de algoritmos voraces basados en colas de prioridad.

**Cola de prioridad utilizando arreglos...**

**Magter. Oscar Adolfo Vallejos**  
**FaCENA - UNNE**





En resumen ...

Una cola de prioridades es una estructura de datos en la que los elementos se atienden en el orden indicado por una prioridad asociada a cada uno. Si varios elementos tienen la misma prioridad, se atenderán de modo convencional según la posición que ocupen.

Este tipo especial de colas tienen las mismas operaciones que las colas, pero con la condición de que los elementos se atienden en orden de prioridad.

Ejemplos de la vida diaria serían la sala de urgencias de un hospital, ya que los enfermos se van atendiendo en función de la gravedad de su enfermedad.

Entendiendo la prioridad como un valor numérico y asignando a altas prioridades valores pequeños, las colas de prioridad nos permiten añadir elementos en cualquier orden y recuperarlos de menor a mayor.



# Bibliografía

- Plantillas de clases de Teoría de la Asignatura.
- Fundamentos de programación. Algoritmos, estructuras de datos y objetos; Luis Joyanes Aguilar; 2003; Editorial: MCGRAW-HILL. ISBN: 8448136642.
- ALGORITMOS, DATOS Y PROGRAMAS con aplicaciones en Pascal, Delphi y Visual Da Vinci. De Guisti. Armando. 2001. editorial: Prentice Hall. ISBN: 987-9460-64-2
- ESTRUCTURA DE DATOS; Luis Joyanes Aguilar, 2001, Editorial: MCGRAW-HILL. ISBN: 8448120426.
- FUNDAMENTOS DE PROGRAMACIÓN. Libro de Problemas en Pascal y Turbo Pascal; Luis Joyanes Aguilar Luis Rodríguez Baena y Matilde Fernandez Azuela; 1999; Editorial: MCGRAW-HILL. ISBN: 844110900.
- ESTRUCTURA DE DATOS; Cairó y Guardati; 2002; Editorial: MCGRAW-HILL. ISBN: 9701035348.

M.A. Weiss. Estructuras de Datos en Java. Ed. Addison Wesley, 2000. Capítulo 6, apartado 8 y capítulo 20, apartados del 1 al 5. R. Wiener, L.J. Pinson.

Fundamental of OOP and Data Structures in Java. Cambridge University Press, 2000. Capítulo 12, apartados 2 y 3.