



# **ALGORITMOS Y ESTRUCTURAS DE DATOS II**

**Serie Práctica 2: Listas, Pilas y Colas implementadas  
con arreglos.**

**Docentes: Burghardt – Zacarías**

**2017**

# ESTRUCTURAS DE DATOS DINÁMICAS

- Las estructuras de datos dinámicas son una colección de elementos, que normalmente son registros, con la particularidad que crecen a medida que se ejecuta un programa.
- La estructura de datos dinámica se amplía y se contrae a medida que se ejecuta el programa.
- Se pueden dividir en dos grandes grupos:

Lineales	No Lineales
Pilas Colas Listas	Árboles Grafos



# ESTRUCTURAS DE DATOS DINÁMICAS

- A cada elemento de la estructura lo denominamos **NODO**.
- Una **Estructura de Datos Dinámica** puede modificar su estructura mediante el programa, puede **modificar su tamaño** añadiendo o eliminando Nodos mientras esta en ejecución el programa.



# LISTAS

- Es un conjunto ordenados de elementos de un tipo.
- Representación:

Lista Lineal

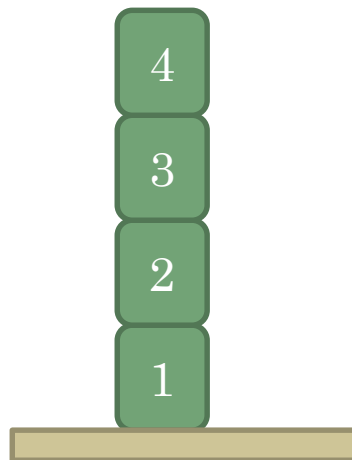


- Operaciones:
  - Recorrido de la lista.
  - Inserción de un elemento.
  - Eliminación de un elemento.
  - Búsqueda de un elemento.



# PILAS

- Un **PILA** es una **LISTA** de elementos a la cual se puede insertar o eliminar alguno de ellos solo por uno de los extremos.
- Los elementos de la pila serán eliminados en orden inverso al que se insertaron. **LIFO**: ultimo en entrar, primero en salir.



# REPRESENTACIÓN DE PILAS

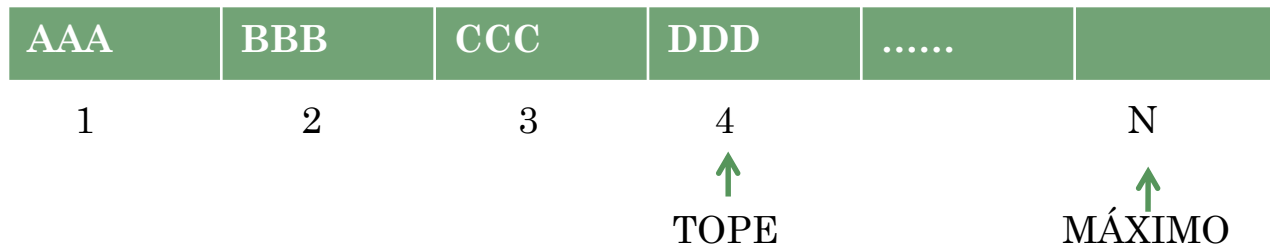
- Se pueden representar mediante el uso de arreglos o listas enlazadas.

```
#define MAX 10;
```

```
int v_pila[MAX];
```

```
int tope = 0;
```

- Al utilizar arreglos debemos definir:
  - Tamaño máximo de la pila.
  - Variable auxiliar Tope o Cima.
- Cuando el Tope = Máximo → Pila Llena



# OPERACIONES CON PILA

## ○ Poner un elemento (PUSH)

### PONER

**Si**  $TOPE < MAX$  (verifica que haya espacio libre)

**entonces**

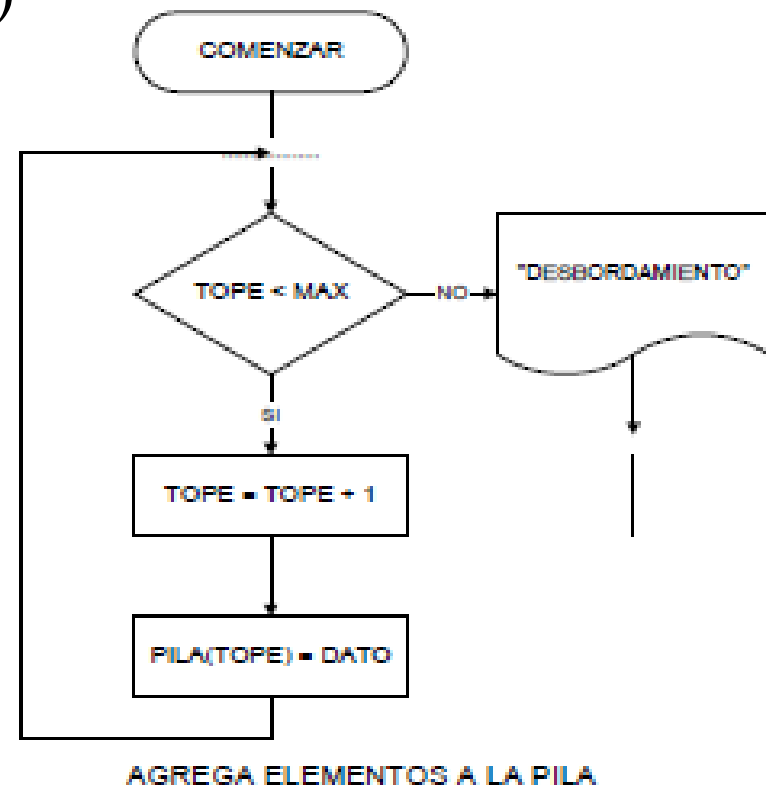
$TOPE = TOPE + 1$  (actualiza  $TOPE$ )

$PILA(TOPE) = DATO$

**sino**

escribir desbordamiento

**Fin\_si**



# OPERACIONES CON PILA

## ○ Quitar un elemento (Pop)

**QUITA**

**Si**  $\text{TOPE} > 0$  (verifica que la pila no este vacía)

**entonces**

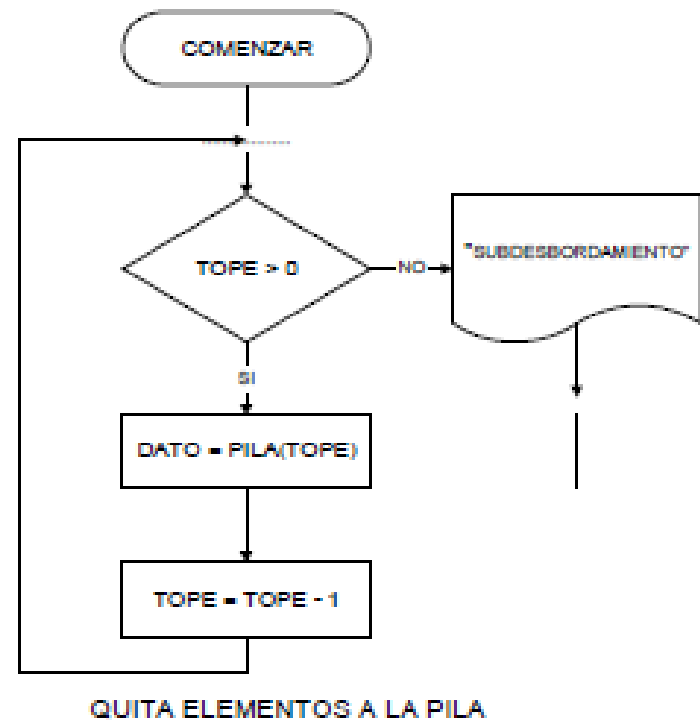
$\text{DATO} = \text{PILA}(\text{TOPE})$

$\text{TOPE} = \text{TOPE} - 1;$

**sino**

escribir “*Subdesbordamiento*”

**Fin\_si**





EJEMPLO: CREAR UNA PILA DE NÚMEROS ENTEROS, DESARROLLANDO LAS FUNCIONES BÁSICAS PARA MANEJO DE PILAS.

```
void crearPilaVacía();  
bool pilaVacía();  
bool pilaLlena();  
void apilar(int);  
void desapilar();  
int cima();  
void  
visualizarElementos();
```



# PILAS: OPERACIONES

```
void crearPilaVacía() {  
    tope = -1;  
}
```

```
bool pilaVacía(){  
    if (tope == -1) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

```
bool pilaLlena() {  
    if (tope == (max-1)) {  
        return true;  
    } else {  
        return false;  
    }  
}
```



# PILAS: OPERACIONES

```
void apilar(int pElemento) {  
    if (pilaLlena() != true) {  
        tope = tope + 1;  
        pila[tope] = pElemento;  
        printf("Elemento Insertado! %d\n", pila[tope]);  
    } else {  
        printf("Pila Llena!\n");  
    }  
}  
  
void desapilar() {  
    if (pilaVacía() == true) {  
        printf("Pila Vacía!!!\n");  
    } else {  
        pila[tope] = 0;  
        tope = tope - 1;  
        printf("Elemento eliminado!!!\n");  
    }  
}
```



# PILAS: OPERACIONES

```
int cima(){  
    return pila[tope];  
}
```

```
void visualizarElementos(){  
    int i;  
    printf("Elementos en pila: \n");  
    for (i = 0; i <= tope; i++) {  
        printf("%d ", pila[i]);  
    }  
    printf("\n\n");  
}
```



# COLAS

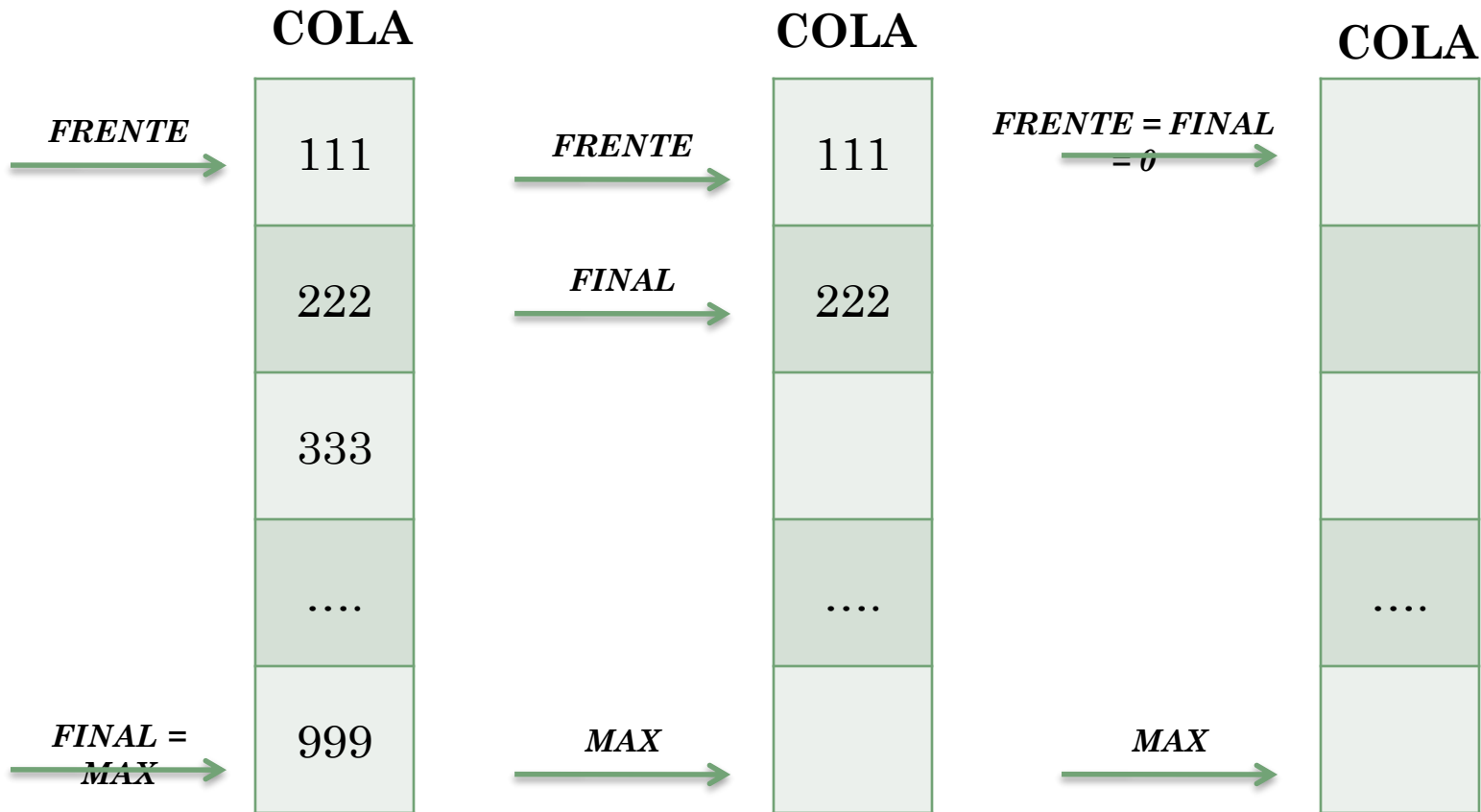


# COLAS

- Las **COLAS** son **LISTAS** de elementos en la que estos se introducen por un extremo (final) y se eliminan por otro (frente). **FIFO**
- Por esta característica este tipo de estructura se utiliza para almacenar datos que necesitan ser procesados según el orden de llegada.
- Ejemplos:
  - Cola de supermercado.
  - Cola de impresión.



# COLAS – REPRESENTACIÓN



# OPERACIONES

- Crear la cola.
- Saber si la cola esta vacía.
- Saber si la cola esta llena.
- Agregar un elemento.
- Eliminar un elemento.
- Recorrer:
  - Mostrar los elementos.
  - Buscar un elemento.
  - Etc.





# COLAS – OPERACIONES: AGREGAR ELEMENTO

## PONER

**Si**  $FINAL < MAX$  (verifica que haya espacio libre)

**entonces**

$FINAL = FINAL + 1$  (actualiza  $FINAL$ )

$COLA(FINAL) = DATO$

**Si**  $FINAL = 1$

**entonces**

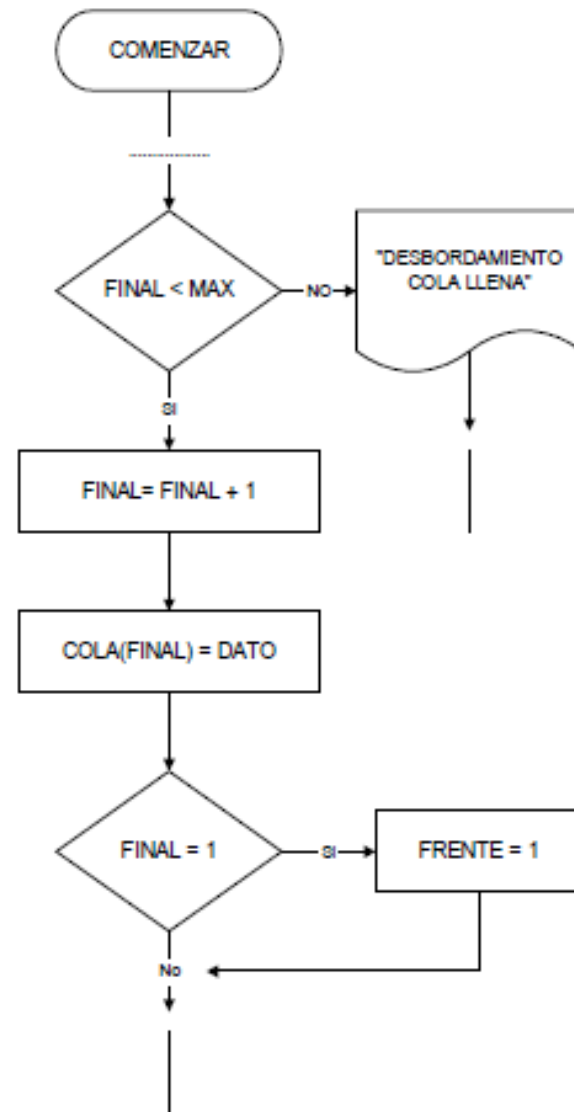
$FRENTE = 1$

**fin\_si**

**sino**

escribir “*Desbordamiento*”

**Fin\_si**



# COLAS – OPERACIONES: QUITAR ELEMENTO

## QUITAR

**Si** FRENTE  $\neq$  -1 (verifica que no esté vacía)

**entonces**

DATO = COLA(FRENTE)

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

**entonces**

FRENTE = -1

FINAL = -1

**sino**

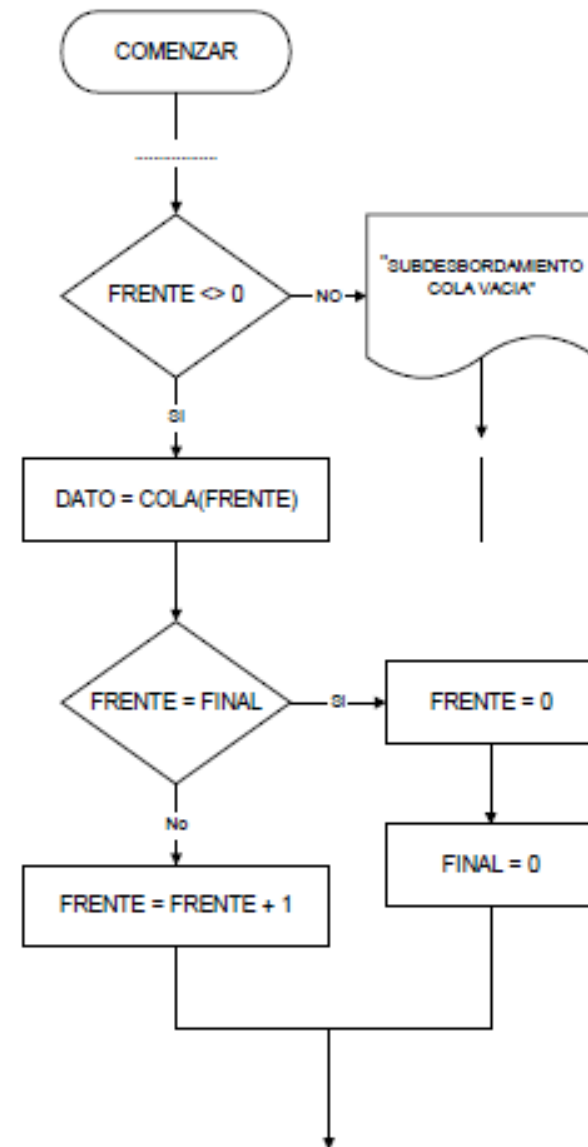
FRENTE = FRENTE + 1

**fin\_si**

**sino**

escribir “Subdesbordamiento”

**Fin\_si**

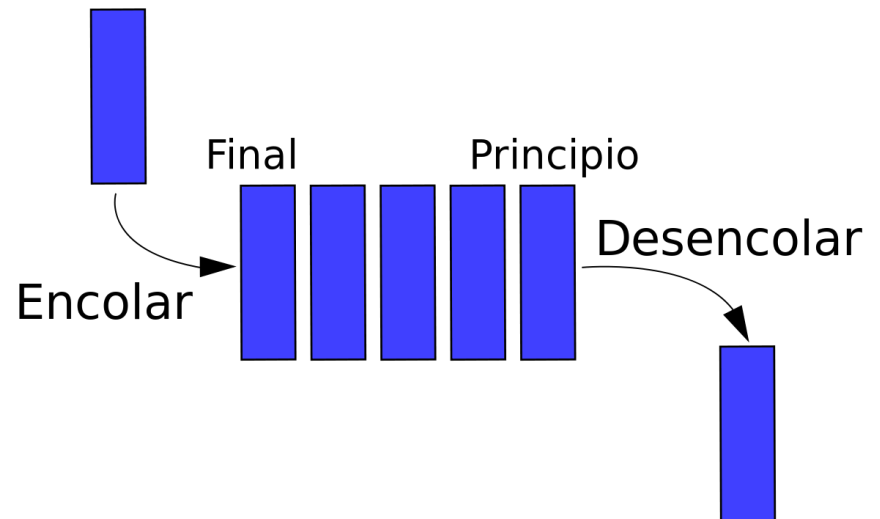


# COLAS

- Definición de la estructura:

```
typedef int tCola [5];
```

```
tCola cola;  
int frente, final;
```



# COLAS: OPERACIONES

```
void crearColaVacia() {  
    frente = -1;  
    final = -1;  
}
```

```
bool colaVacia() {  
    if ((frente == -1) && (final == -1))  
        return true;  
    else  
        return false;  
}
```

```
bool colaLlena() {  
    if (final == (max-1))  
        return true;  
    else  
        return false;  
}
```



# COLAS: OPERACIONES

```
int primerElemento() {  
    return cola[frente];  
}
```

```
void visualizarElementos() {  
    int i;  
    printf("Elementos en cola: \n");  
    for (i = frente; i <= final; i++) {  
        printf("%d ", cola[i]);  
    }  
    printf("\n\n");  
}
```




# COLAS: OPERACIONES – INSERTAR

```
void agregarElemento(int pElemento) {  
    if (colaLlena() != true) {  
        final = final + 1;  
        cola[final] = pElemento;  
        printf("Elemento Insertado! %d\n", cola[final]);  
        //significa que es el primer elemento  
        if (final == 0 ) {  
            frente = 0;  
        }  
    } else {  
        printf("No hay lugar!\n");  
    }  
}
```




# COLAS: OPERACIONES – ELIMINAR (FRENTE MÓVIL )

```
void eliminarElementoFrenteMovil() {  
    if (colaVacia() == true) {  
        printf("No hay elementos por eliminar!!!\n");  
    } else {  
        //poner en cero o vacío el elemento del frente  
        cola[frente] = 0;  
  
        printf("Elemento eliminado!!!\n");  
  
        //si frente y final son iguales significa que elimine el ultimo  
  
        if (frente == final) {  
            frente = -1;  
            final = -1;  
        } else {  
            frente = frente + 1;  
        }  
    }  
}
```



# COLAS: OPERACIONES – ELIMINAR (FRENTE FIJO )

```
void eliminarElementoFrenteFijo() {  
    if (colaVacia() == true) {  
        printf("No hay elementos por eliminar!!!\n");  
    } else {  
        //poner en cero o vacio el elemento del frente  
        cola[frente] = 0;  
        printf("Elemento eliminado!!!\n");  
        //reacomodando...  
        for (i = frente; i <= final-1; i++) {  
            cola[i] = cola[i + 1];  
        }  
        cola[final] = 0;  
        if (frente == final) {  
            frente = -1;  
            final = -1;  
        } else {  
            final = final - 1;  
        }  
    }  
}
```





# BIBLIOGRAFÍA

- Material de teoría de la catedra Algoritmos y Estructuras de Datos II.
- Pablo A. Sznajdleder. Algoritmos a fondo, con implementaciones en C y Java. Editorial Alfaomega. 2012.

