

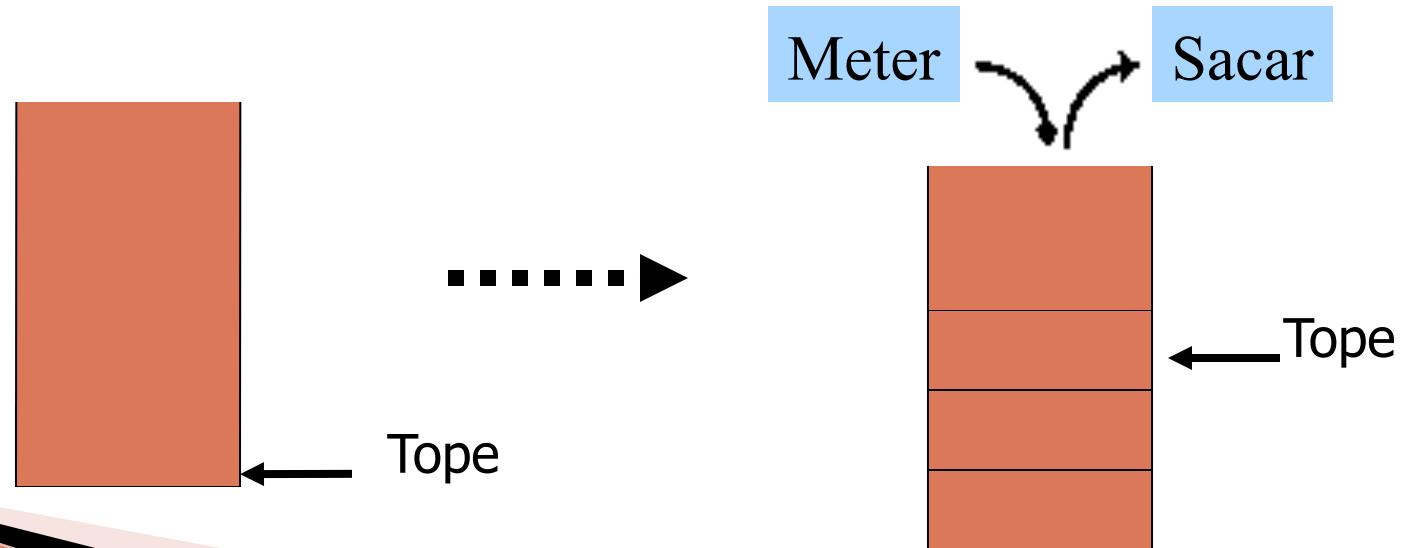
# Pilas (Stack)

Implementadas en:  
Arreglos  
Listas Encadenadas Lineales  
Listas Encadenadas Circulares

Ing. Armandina Leal Flores

# Pila

- ▶ Estructura de datos **lineal**.
- ▶ El orden de entrada sigue la filosofía:  
**LIFO** (Last Input First Output).



# Uso de una Pila

- ▶ En cualquier aplicación que requiera controlar un orden inverso al orden de entrada original.
- ▶ Ejemplos:
  - Control de llamadas a métodos.
  - Conversión y Evaluación de expresiones.

# ADT Pila

## (Nivel Lógico)

- ▶ **Elementos:** El tipo de elemento depende de la aplicación.
- ▶ **Estructura:** Lineal.

Operaciones:

crearPila  
meterPila  
sacarPila  
pilaVacía  
pilaLlena  
observaPila

0	1	2	3	4	MAX-1

↑ iTope

Pila vacía

0	1	2	3	4	MAX-1
4					

↑ iTope

Meter a la pila el 4

0	1	2	3	4	MAX-1
4	2				

↑ iTope

Meter a la pila el 2

0	1	2	3	4	MAX-1
4	2	8			

↑ iTope

Meter a la pila el 8

0	1	2	3	4	MAX-1
4	2	8			

↑ iTope

Meter a la pila el 8

0	1	2	3	4	MAX-1
4	2	8			

↑ iTope

Sacar

0	1	2	3	4	MAX-1
4	2	8			

↑ iTope

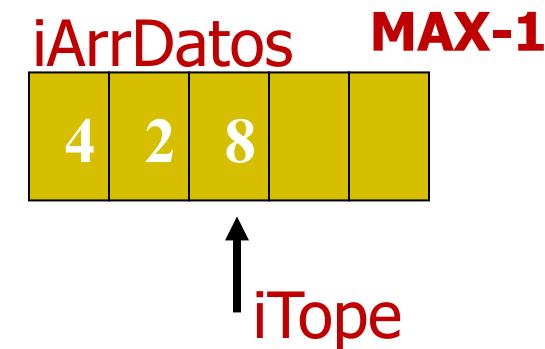
Sacar

# Implementación en Arreglos

Nivel Físico

```
const int MAX = 50;  
class PilaArreglos  
{ public:  
    PilaArreglos( );  
    void meterPila(int iValor);  
    void sacarPila();  
    int observaPila();  
    bool pilaVacia( );  
    bool pilaLlena( );  
private:  
    int iArrDatos[MAX];  
    int iTope;  
};
```

MAX es la cantidad máxima de elementos



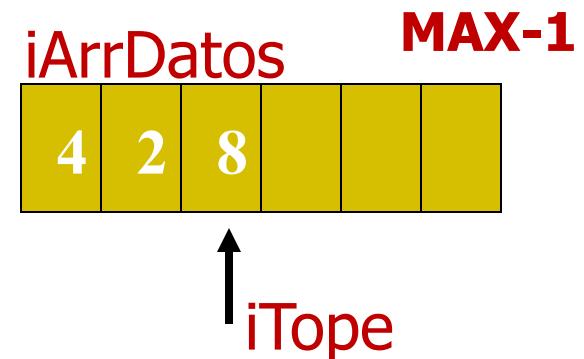
Lugar donde se guardarán los valores

iTope apunta al último valor agregado

```
void meterPila(int iValor)
{
    iTope++;
    iArrDatos[iTope] = iValor;
}
```

```
pilaArreglos()
{ iTope = -1; }
```

```
bool pilaLlena( )
{
    return (iTope == MAX-1); }
```



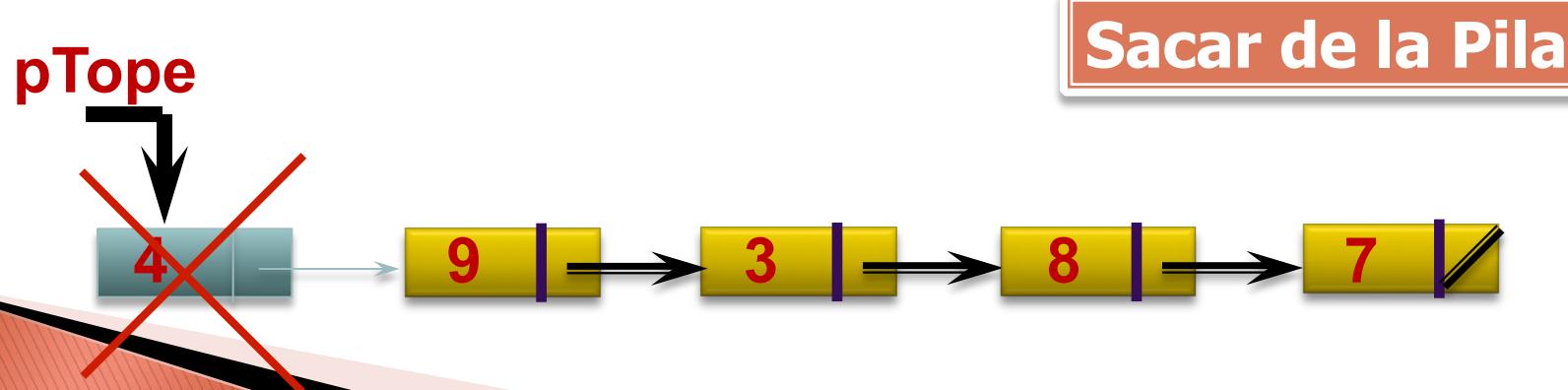
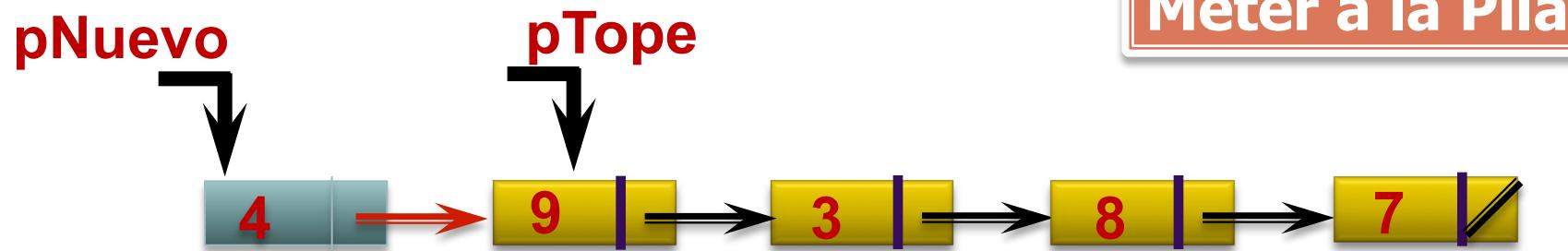
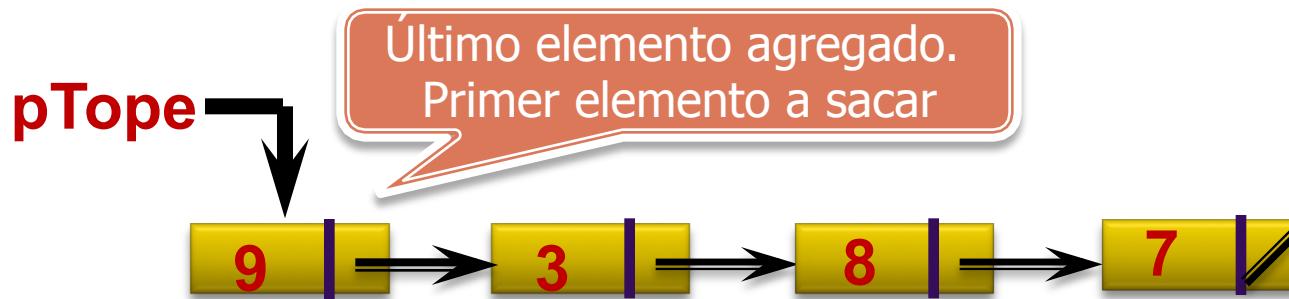
```
void sacarPila()
{
    iTope--;
}
```

```
bool PilaVacia( )
{
    return (iTope == -1); }
```

```
int observaPila()
{
    return iArrDatos[iTope];
}
```

# Implementación en una Lista Encadenada Lineal

Nivel  
Físico



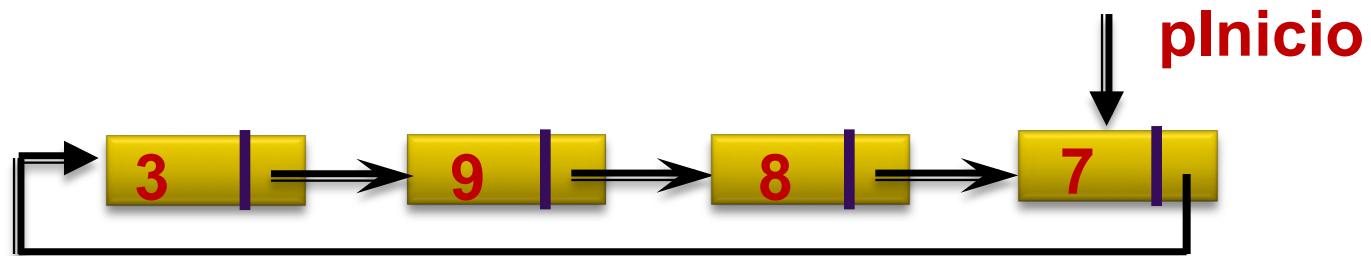
# Implementación en una Lista Encadenada Lineal

```
class Nodo
{
public:
    int iInfo;
    Nodo *pSig;
    Nodo() { pSig = NULL; }
    Nodo ( int iDato )
        { iInfo = iDato; pSig = NULL; }
    Nodo (int iDato, Nodo *pSiguiente)
        { iInfo = iDato; pSig = pSiguiente; }
};
```

```
class pilaLineal
{
private:
    Nodo *pTope;
public:
    pilaLineal() { }
    pilaLineal (const pilaLineal& p) {}
    pilaLineal& operator
        = (const pilaLineal& p) {}
    void meterPila(int iDato){}
    int observaPila() {}
    void sacarPila() {}
    bool pilaVacia() {}
    ~pilaLineal() {}
};
```

# Listas Encadenadas Circulars

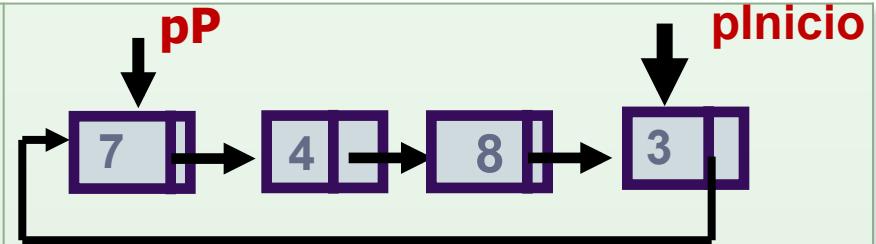
- ▶ Una lista encadenada circular es una lista lineal en la que el último nodo apunta al primero.
- ▶ Es una lista donde físicamente no se tiene un inicio y un final.
- ▶ El único caso en que existe un NULL es cuando la lista se encuentra vacía.



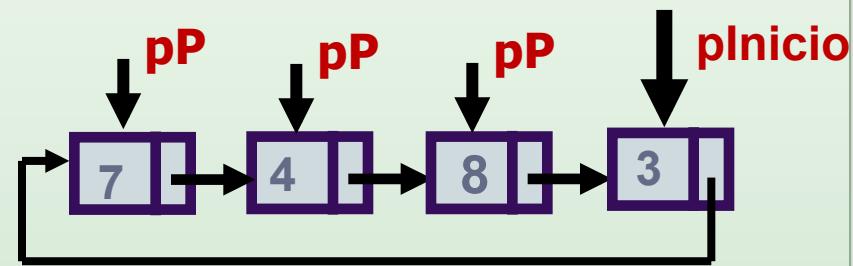
# Ejercicio....

## Desplegar todos los valores de la lista

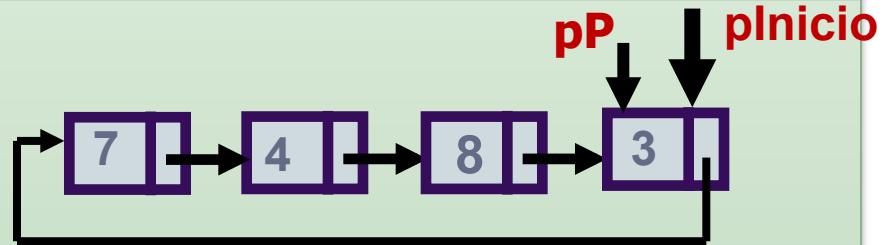
```
Nodo *pP = iInicio -> pSig;
```



```
while (pP != iInicio)
{
    cout << pP -> iInfo;
    pP = pP -> pSig;
}
```



```
cout << pP -> iInfo;
```

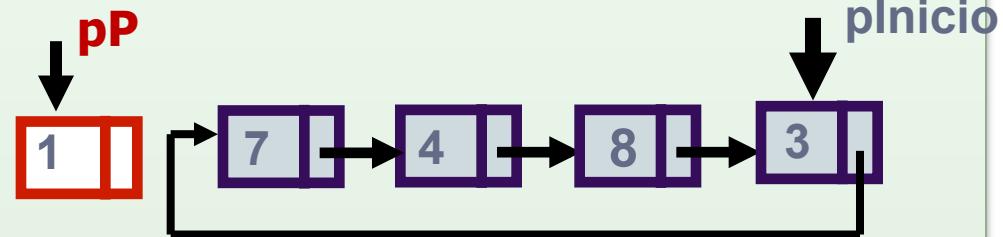


Desplegado en Pantalla

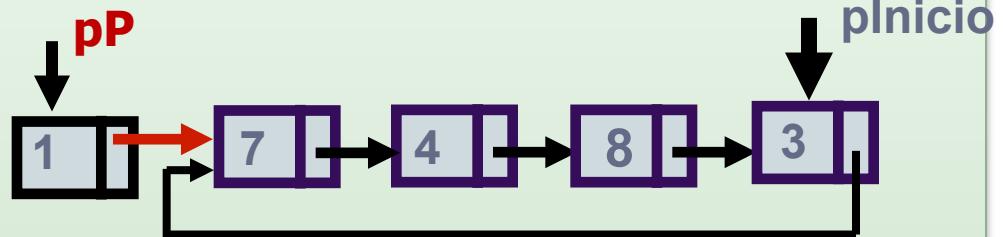
7 4 8 3

# Agregar un nodo después de inicio

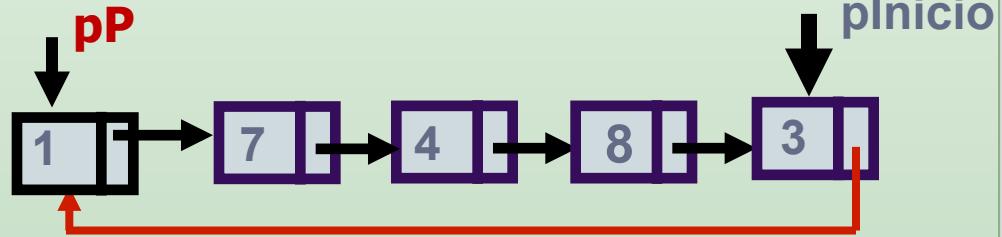
Nodo \*pP = new Nodo(iData);



pP->pSig = plnicio -> pSig;

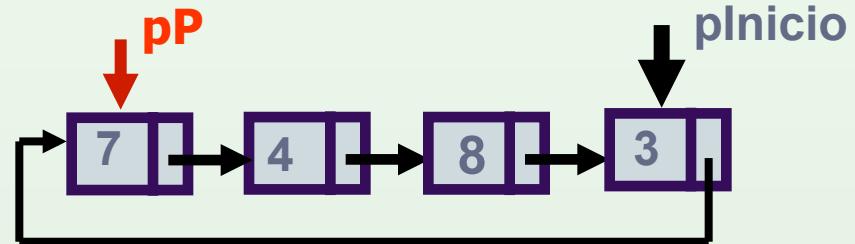


plnicio -> pSig = pP;

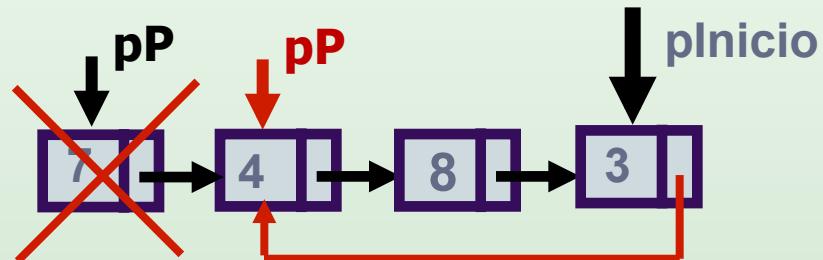


# Borrar todos los nodos de la lista

```
Nodo *pP = pInicio -> pSig;
```



```
while (pP != pInicio)
{
    pInicio -> pSig = pP -> pSig;
    delete pP;
    pP = pInicio -> pSig;
}
```

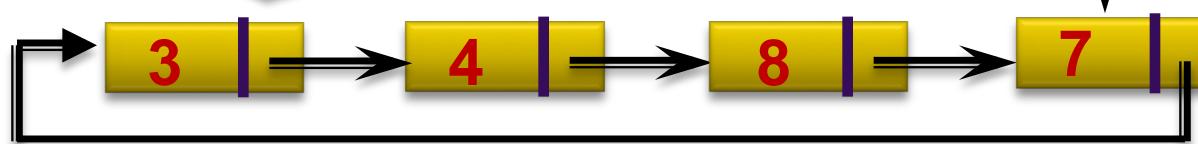


```
delete pP;
pInicio = NULL;
```

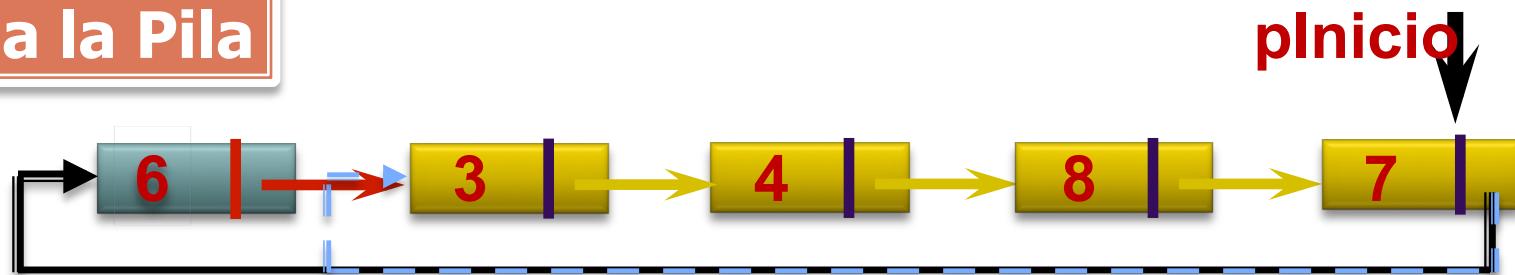


# Implementación de una Pila en una Lista Encadenada Circular

Último elemento agregado.  
Primer elemento a sacar



Meter a la Pila



Sacar de la Pila

