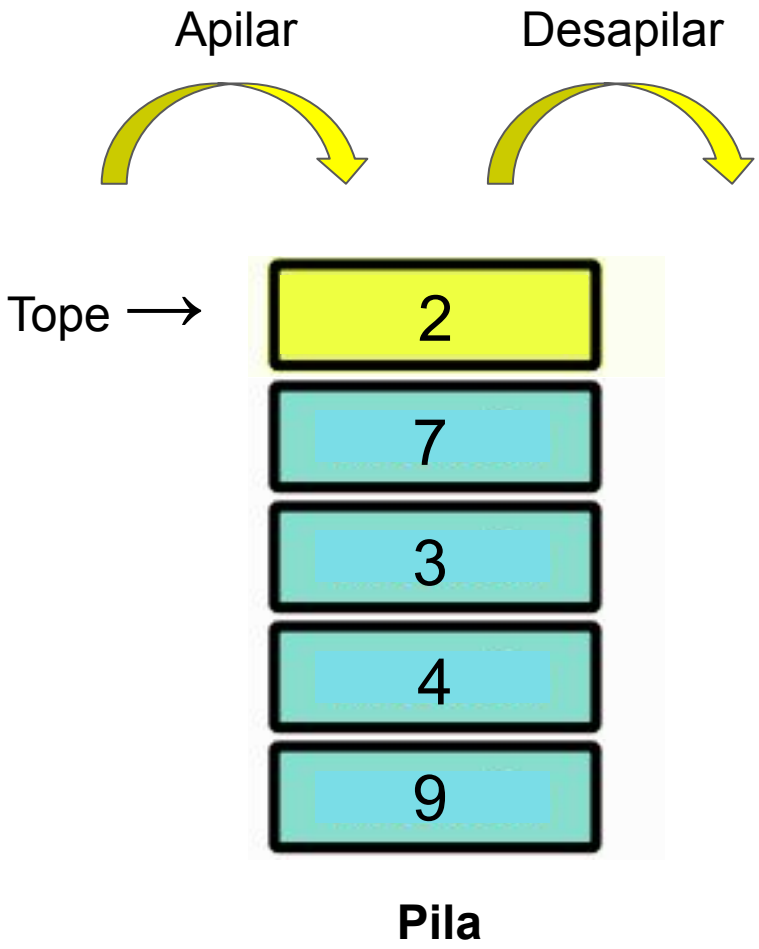
Pilas

75.41 - Algoritmos y Programación II

2° Cuatrimestre 2019

¿Qué es una pila?

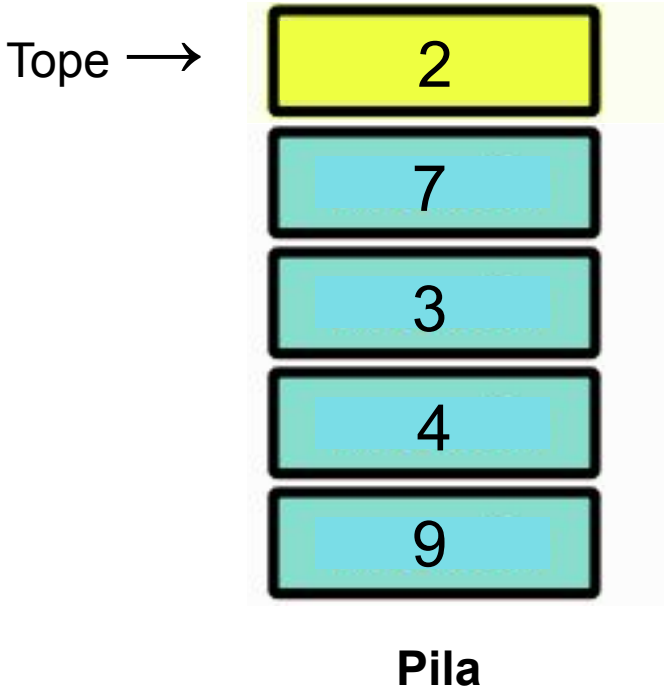
- Estructura de datos
- Agrupa elementos
- LIFO → Last In, First Out



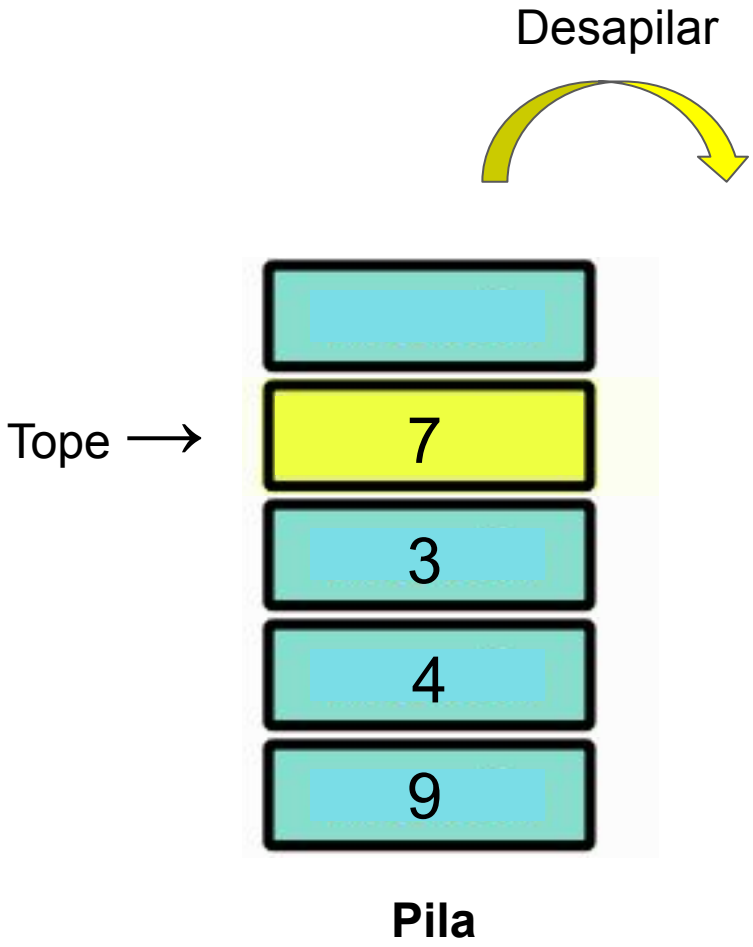
Operaciones

- Crear
- Destruir
- Apilar (*push*)
- Desapilar (*pop*)
- Tope
- Vacía

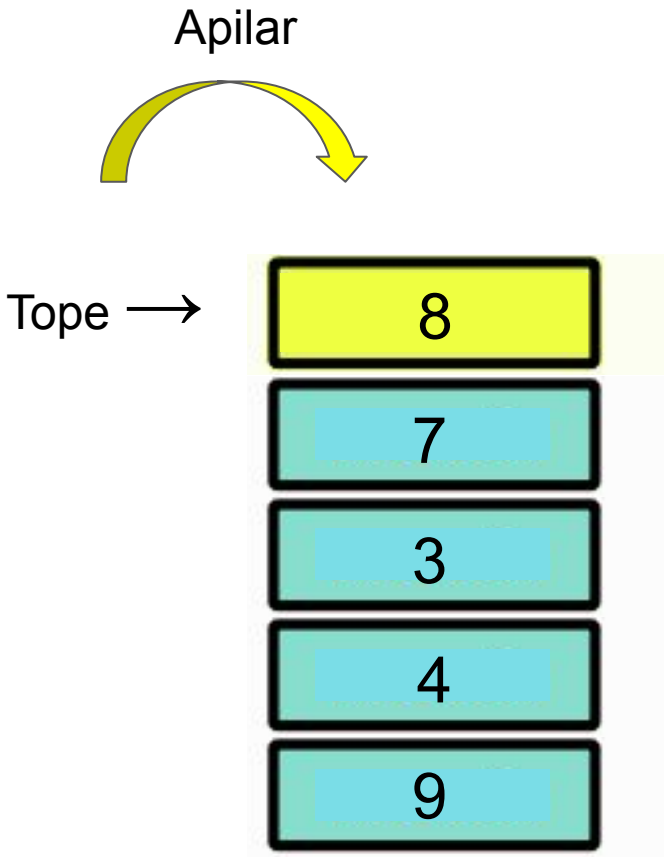
Ejemplo de pila:



Desapilar:



Apilar un 8:



Implementaciones

- Vector estático
- Vector dinámico
- Pila como lista de nodos

Vector estático

PILA:

TOPE: 0 → cantidad de elementos actualmente almacenados

CAPACIDAD: 5 → cantidad de elementos que puedo almacenar

Vector estático

PILA:

TOPE: 0 → cantidad de elementos actualmente almacenados

CAPACIDAD: 5 → cantidad de elementos que puedo almacenar

¿Puedo apilar? **SI**. TOPE != CAPACIDAD
¿Está vacía? **TRUE,** TOPE == 0
¿Puedo desapilar? **NO**. TOPE == 0

PILA:

Apilo el elemento 55

TOPE: 1

CAPACIDAD: 5

PILA:

Apilo el elemento 55

TOPE: 1

CAPACIDAD: 5

¿Puedo apilar? **SI**. TOPE != CAPACIDAD
¿Está vacía? **FALSE,** TOPE != 0
¿Puedo desapilar? **SI**. TOPE != 0

PILA:

Apilo el elemento 6

TOPE: 2

CAPACIDAD: 5

PILA:

Apilo el elemento 6

TOPE: 2

CAPACIDAD: 5

¿Puedo apilar? **SI**. TOPE != CAPACIDAD
¿Está vacía? **FALSE,** TOPE != 0
¿Puedo desapilar? **SI**. TOPE != 0

PILA:

Apilo más elementos

TOPE: 5

CAPACIDAD: 5

PILA:

Apilo más elementos

TOPE: 5

CAPACIDAD: 5

¿Puedo apilar? **NO**. TOPE == CAPACIDAD
¿Está vacía? **FALSE,** TOPE != 0
¿Puedo desapilar? **SI**. TOPE != 0

PILA:

Desapilo

TOPE: 4

CAPACIDAD: 5

PILA:

Desapilo

TOPE: 4

CAPACIDAD: 5

¿Puedo apilar? **SI**. TOPE != CAPACIDAD
¿Está vacía? **FALSE,** TOPE != 0
¿Puedo desapilar? **SI**. TOPE != 0

```c
typedef struct pila {
    int tope;  // cantidad de elementos almacenados
    void* elementos[CAPACIDAD_PILA];  // vector en donde
                                      // se almacenarán los
                                      // elementos
} pila_t;
```

Problema…

¿Qué pasa si quiero almacenar más elementos?

Solución: ¡**vector dinámico**!

PILA:

Pila llena

TOPE: 5

TAMANIO: 5

PILA:

Pila llena

TOPE: 5

TAMANIO: 5

¿Puedo apilar? **¡SI! Puedo pedir más memoria**
¿Está vacía? **FALSE,** TOPE != 0
¿Puedo desapilar? **SI**. TOPE != 0

PILA:

TOPE: 5

TAMANIO: 10

¿Puedo apilar? **SI**. TOPE != TAMANIO
¿Está vacía? **FALSE,** TOPE != 0
¿Puedo desapilar? **SI**. TOPE != 0

Redimensión

- Depende de la implementación. Por ejemplo, la pila crece:
  - Cuando se supera el 75% de la capacidad
  - Cuando se llena
  - Etc
- Cuando desapilo, también redimensiono:
  - Cuando se llega al 50% de la capacidad
  - Cuando desapilo
  - Cuando se llega al 25% de la capacidad
  - Etc

Repaso de REALLOC

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

Modifica el tamaño del bloque de memoria apuntado por *ptr* en *tamanio_nuevo* bytes

A tener en cuenta...

- Sirve siempre y cuando haya memoria **contigua** disponible

- Sino… ¡no voy a poder redimensionar!

¿Y ahora?

Solución: **lista de nodos**

- Los elementos son **nodos**
- Cada uno tiene una referencia al nodo anterior

¿Cuándo reservo / libero memoria?

- Reservo memoria para cada nodo cuando quiero apilar
- Libero memoria para cada nodo cuando quiero desapilar

Ventaja:

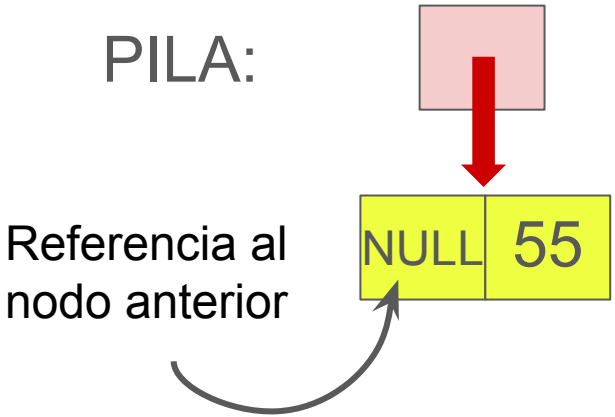
- Memoria **no debe ser contigua**

Lista de nodos

PILA: NULL

Referencia a nodo_tope es NULL

PILA:

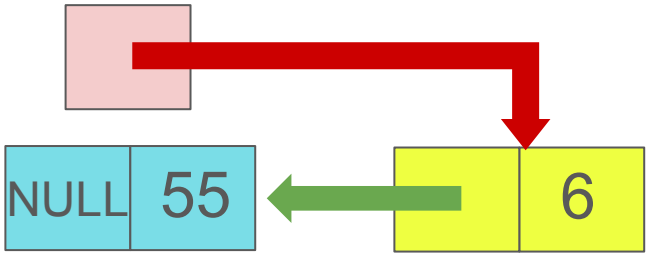


Apilo un elemento

Lista de nodos

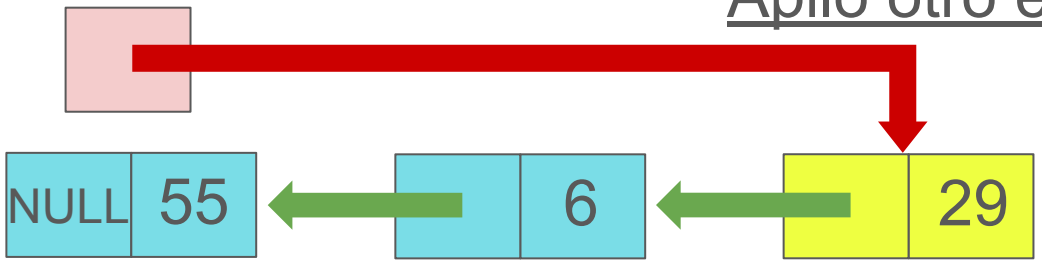
PILA:

Apilo un elemento



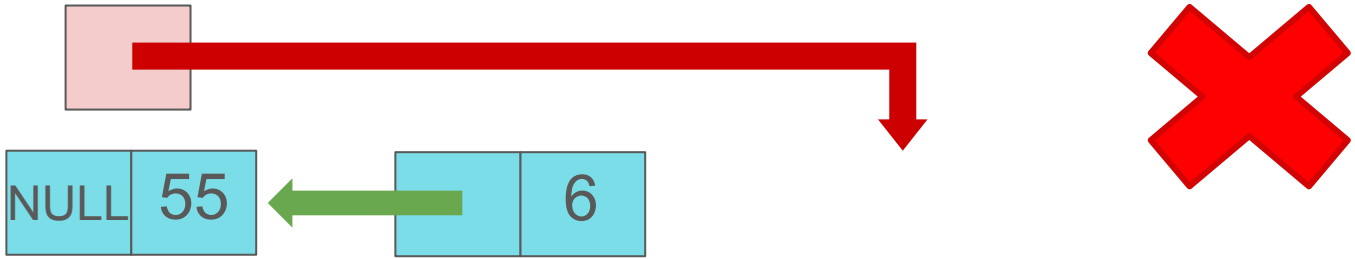
PILA:

Apilo otro elemento



¡Cuidado al desapilar!

PILA:



¡Perdí la referencia a los otros nodos!

¡Cuidado al desapilar!