Tema 5. Estructura de datos Pila

`http://aulavirtual.uji.es`

José M. Badía, Begoña Martínez, Antonio Morales y José M. Sanchiz

{badia, bmartine, morales, sanchiz}@icc.uji.es

Estructuras de datos y de la información

Universitat Jaume I

Índice

Bibliografía

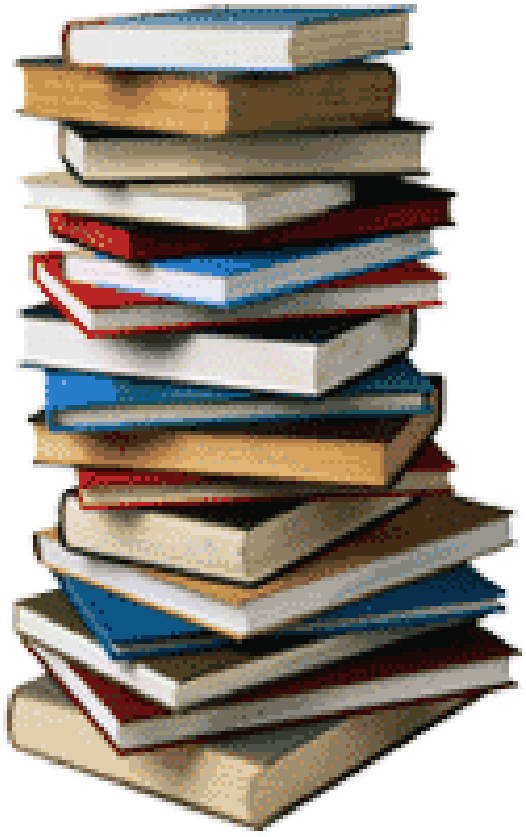
- (Nyhoff'06), Capítulo 7
- (Main y Savitch'01), Capítulo 7
- (Drozdek'01), Capítulo 4

Objetivos

- Conocer el concepto, funcionamiento y utilidad del tipo Pila.
- Conocer el TAD Pila y sus operaciones asociadas.
- Saber utilizar el tipo Pila para resolver problemas.
- Saber implementar el tipo Pila mediante el uso de vectores.

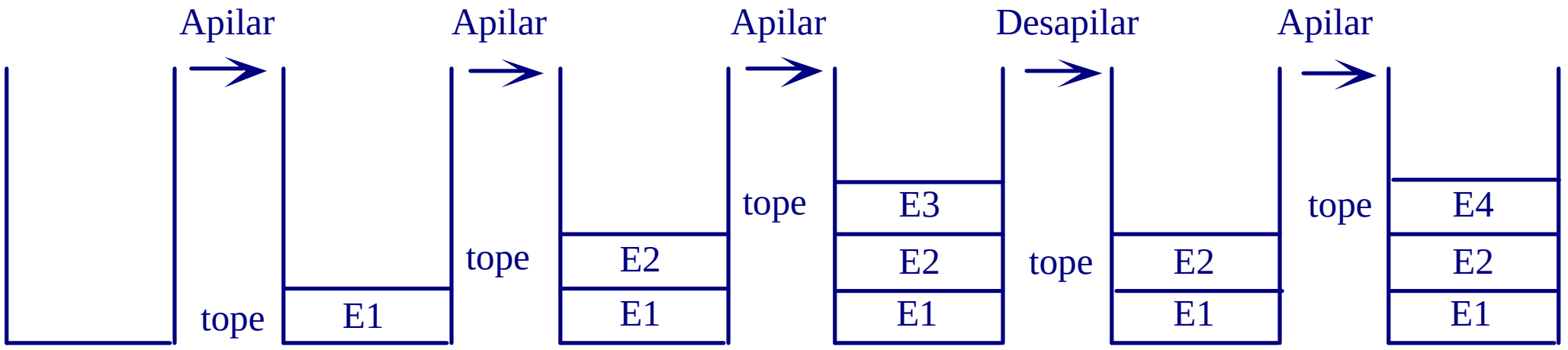
1 Definición y aplicaciones

- Pila=Estructura lineal manejada siguiendo una política LIFO: Last In First Out.
- Tope: Posición en la que se insertan y eliminan elementos de la pila.
- Concepto manejado de modo cotidiano: pila de platos, bandejas, ...



1 Definición y aplicaciones (II)

- Operaciones de manipulación:
  - Apilar: Añade un elemento en la posición siguiente al tope actual.
  - Desapilar: Elimina el elemento situado en el tope.



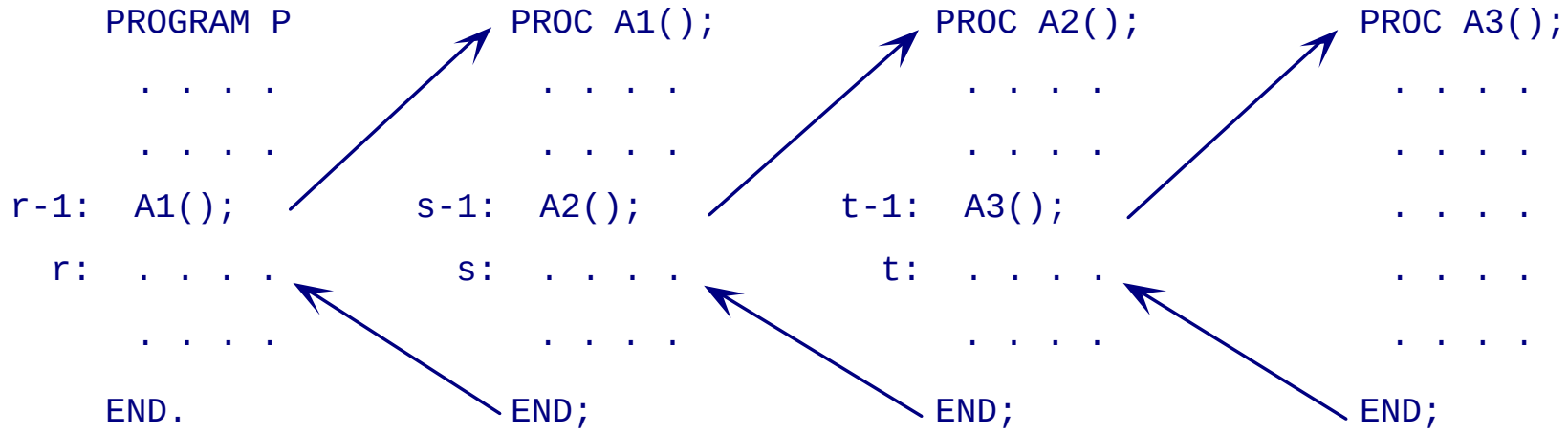
1 Definición y aplicaciones (III)

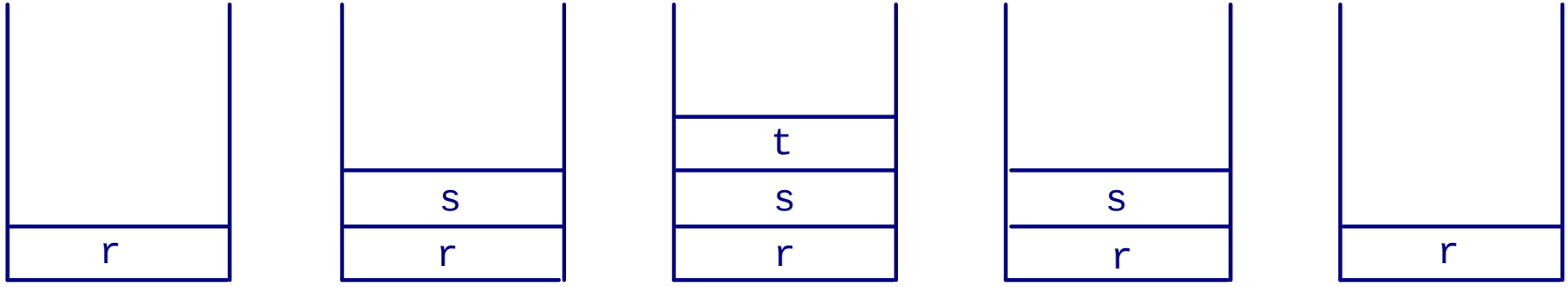
- La Pila es una estructura de datos dinámica.
  - Su tamaño no queda definido en tiempo de compilación.
  - Las operaciones del TAD permiten modificar su tamaño.

¿ Restricciones de tamaño?

Aplicación. Llamadas a procedimiento

- Problemas en los que los datos se manejan siguiendo una política LIFO.



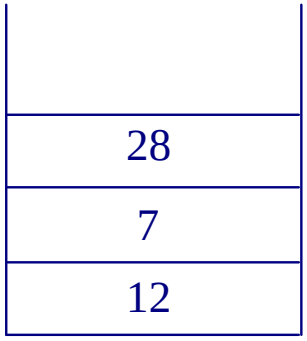


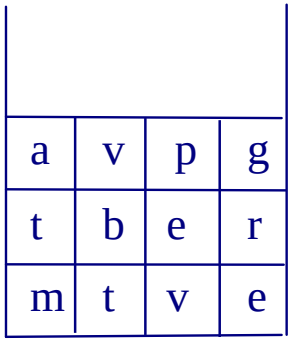
Tras llamar a A1 | Tras llamar a A2 | Tras llamar a A3 | Al volver de A3 | Al volver de A2

1 Definición y aplicaciones (V)

Una Pila es una estructura ordenada, lineal y homogénea, en la que podemos insertar o extraer elementos en una única posición llamada Tope*, siguiendo una política LIFO.*

- **Ordenada:** Los elementos *se sitúan* en un cierto orden. Sus valores NO tienen porque estar ordenados.
- **Homogénea:** Todos los elementos son del mismo tipo. Pueden ser de cualquier tipo: simple o compuesto.





2 Tipo Abstracto de Datos Pila

TAD Pila

Usa logico, tb

Operaciones:

CrearPila: → Pila

PilaVacia: Pila → logico

Tope: Pila → tb

Apilar: Pila x tb → Pila

Desapilar: Pila → Pila

Axiomas: ∀ S∈Pila, ∀ e∈tb,

1) PilaVacia(CrearPila) = Verdadero

2) PilaVacia(Apilar(S,e)) = Falso

3) Desapilar(CrearPila) = error

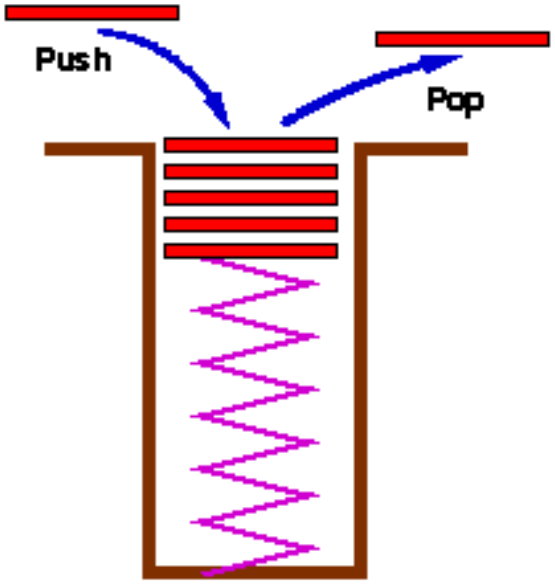
4) Desapilar(Apilar(S,e)) = S

5) Tope(CrearPila) = error

6) Tope(Apilar(S,e)) = e

3 La clase *stack*

```
class stack {
  public:
    stack();
    bool empty() const;
    const int & top() const;
    void push(const int & dato);
    void pop();
  private:
    ...
};
```



Diferencias entre vector y Pila

Vector

- Puede accederse a cualquiera de sus elementos.
- Tiene un número limitado de elementos.

Pila

- Sólo puede accederse al elemento situado en el tope.
- No tiene restricciones teóricas de tamaño.

Cuestión de implementación

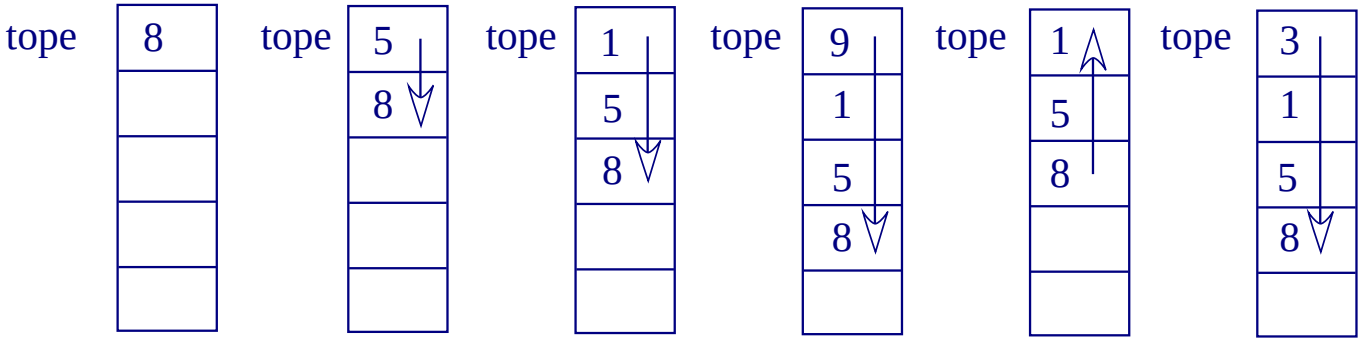
- Los vectores tienen un tamaño limitado $\Rightarrow$ Pueden llenarse.

¿PilaLlena?

Primera opción

Pila = vector

- La posición del tope se mantiene fija en el elemento 0 del vector.
- Los elementos se desplazan al apilar y desapilar.



Problema

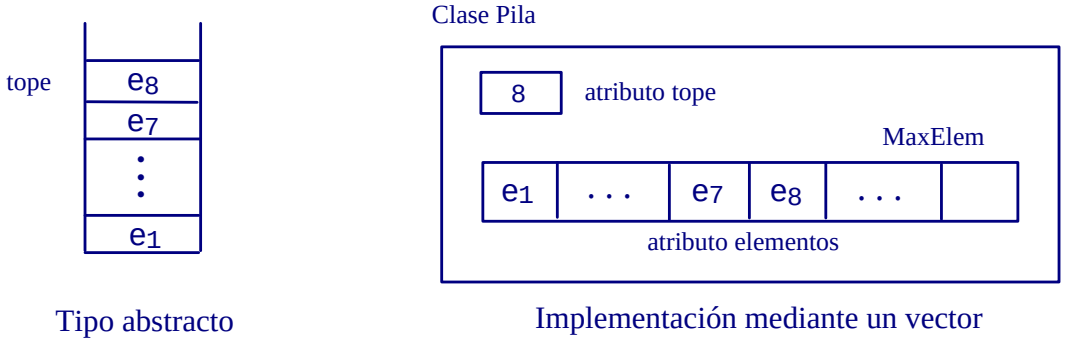
- Coste elevado de las operaciones apilar y desapilar.

4 Implementación estática (III)

Segunda opción

Pila = vector + entero

- El vector guarda los elementos de la pila.
- El entero indica la posición "móvil" del tope.



Tipo abstracto

Implementación mediante un vector

Ventaja

- Coste constante de las operaciones.

4 Implementación estática (IV)

Definición del tipo - fichero *stack.h*

```
#ifndef STACK
#define STACK
const int MaxElem = 128;
class stack {
  public:
    // Definición de las operaciones públicas
  private:
   int elementos[MaxElem];
   int tope;
};
#endif
```

4 Implementación estática (V)

- Implementación de las operaciones - fichero *stack.cpp*

```
/* Crea una pila vacía */
stack::stack() {
    tope = -1;
}


/* Comprueba si la pila esta vacía */
bool stack::empty() const {
  return (tope == -1);
}
```

4 Implementación estática (VI)

```
/* Devuelve el elemento situado en el tope */
const int & stack::top() const {
  if (!empty())
    return elementos[tope];
  else
    cerr  << "Error: Pila vacia" << endl;
}
```

- Devuelve el elemento, NO lo desapila

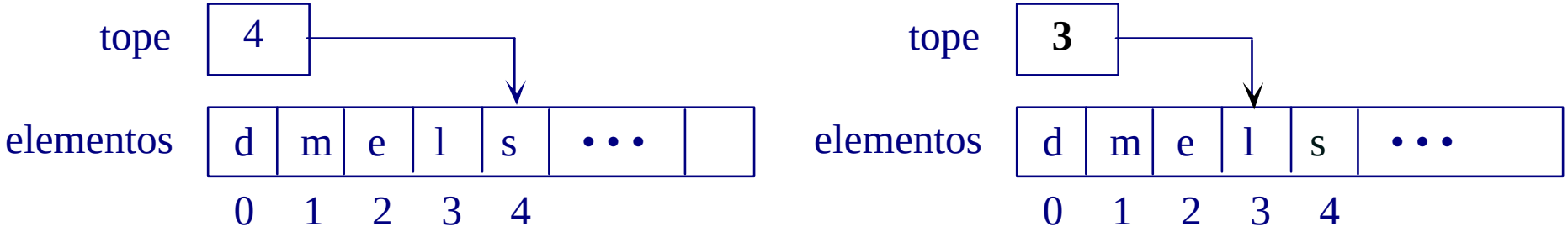
4 Implementación estática (VII)

```
/* Apila un elemento "por encima" del tope */
void stack::push(const int & dato) {
  if (tope < MaxElem - 1) { // ¿Final del vector?
    ++tope;
    elementos[tope] = dato;
  }
  else
    cerr << "Error: Pila llena" << endl;
}
```

4 Implementación estática (VIII)

```
/* Desapila el elemento situado en el tope */
void stack::pop() {
  if (!empty())
    tope--;
  else
    cerr  << "Error: Pila vacia" << endl;
}
```

- El elemento no se "borra"del vector



4 Implementación estática (IX)

- **Problema**

  Array: estructura estática

  Pila: estructura dinámica

- **Consecuencias**
  - Reserva de un espacio fijo en tiempo de compilación.
  - Posibilidad de llenado del vector $\Rightarrow$ Llenado de la pila.
  - Espacio reservado máximo $\Rightarrow$ Desaprovechamiento de memoria.
- **Solución**
  - Uso de memoria dinámica.