



**Grado en Ingeniería Información**

**Estructura de Datos y Algoritmos**

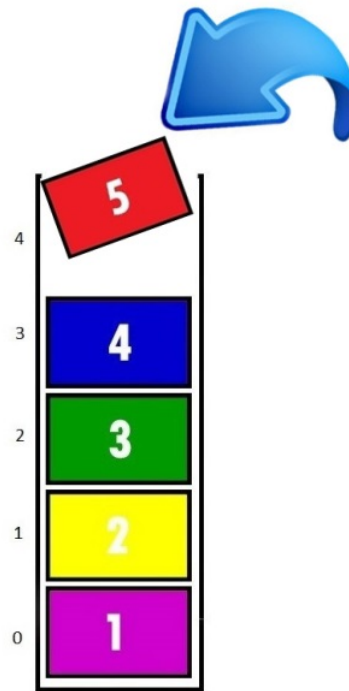
**Sesión 7**

**Curso 2023-2024**

Marta N. Gómez

## T3. Tipos Abstractos de Datos (TAD)

- **Concepto.**
- Tipos de datos lineales:
  - Pilas
  - Colas
  - Listas



Un ***Tipo Abstracto de Datos (TAD)*** es un conjunto de **datos** y de **operaciones** definidos mediante una especificación.

Los **datos** y las **operaciones** básicas definidas se estudian sin considerar la **implementación**:

- Se piensa en ***qué*** se puede hacer con los datos
- No se piensa ***cómo*** hacerlo

Los **tipos de datos predefinidos** tienen este mismo enfoque, indicar el **conjunto de datos** y especificar una **serie de operaciones básicas y válidas**:

*int*

*bool*

*float*

*char*

Lo mismo con los **tipos de datos y contenedores** incluidos en la **librería estándar (std)**:

*std::string*

*std::vector<int>*

*std::array<int>*

...

- **Proceso de Abstracción**

El análisis permite obtener una especificación concreta que no puede cambiarse.

- **Proceso de Implementación**

Siempre se puede mejorar.

```
class Persona {  
private:  
    string nombre;  
    vector<shared_ptr<Mascota>> las_mascotas;  
  
public:  
    Persona();  
    ~Persona();  
  
    void nueva_mascota(shared_ptr<Mascota> una_mascota);  
  
    string getNombre() const;  
  
    vector<shared_ptr<Mascota>> nombreMascotas() const;  
    vector<shared_ptr<Mascota>> pesoMascotas() const;  
};
```

```
class Mascota {  
private:  
    string nombre;  
    float peso;  
    shared_ptr<Persona> propietario;  
  
public:  
    Mascota (const string &nom, shared_ptr<Persona> prop);  
    ~Mascota();  
  
    shared_ptr<Persona> get_propietario() const;  
  
    float get_peso() const;  
    void set_peso(float p);  
  
    void mostrar() const;  
};
```



```
Persona::Persona() {}
```

```
Persona::~~Persona() {}
```

```
void Persona::nueva_mascota(shared_ptr<Mascota> una_mascota) {  
    las_mascotas.push_back(una_mascota);  
}
```

```
vector<shared_ptr<Mascota>> Persona::nombreMascotas() const {  
    vector<shared_ptr<Mascota>> ret = las_mascotas;  
    // quicksort(ret, 0, ret.size());  
    return ret;  
}
```

```
vector<shared_ptr<Mascota>> Persona::pesoMascotas() const {  
    vector<std::shared_ptr<Mascota>> ret = las_mascotas;  
    // bubble_sort(ret);  
    return ret;  
}
```

```
Mascota::Mascota(const string &nom, shared_ptr<Persona> prop) :  
    nombre(nom), propietario(prop) {  
    peso = -1;  // -1 cuando no se conoce  
}
```

```
Mascota::~~Mascota () {}
```

```
shared_ptr<Persona> Mascota::get_propietario() const {  
    return propietario;  
}
```

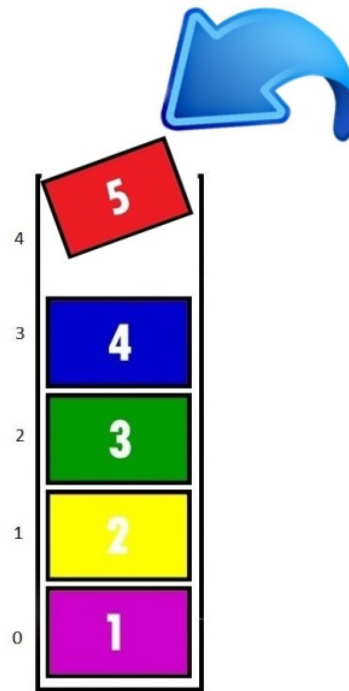
```
void Mascota::set_peso(float p) {  
    peso = p;  
}
```

```
float Mascota::get_peso() const {  
    return peso;  
}
```

```
void Mascota::mostrar() const {  
    cout << "Soy una mascota y mi propietario es ";  
    propietario->getNombre();  
    cout << " y mi nombre es " << nombre << endl;  
}
```

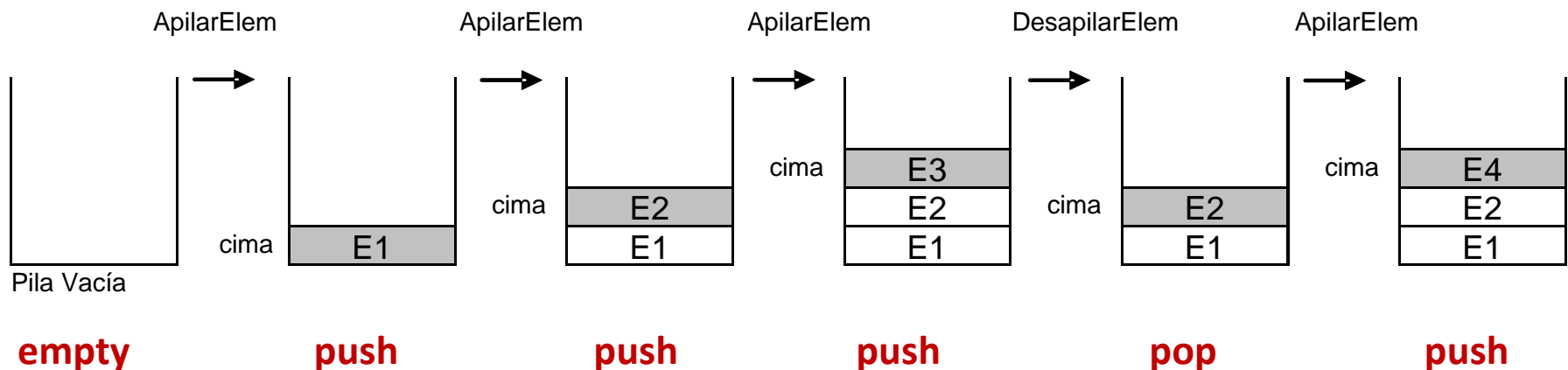
## T3. Tipos Abstractos de Datos (TAD)

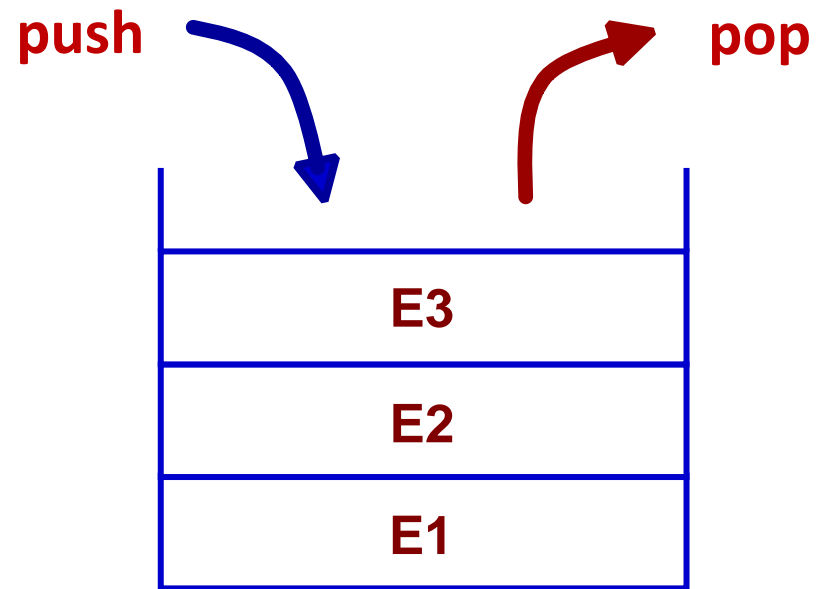
- Concepto.
- **Tipos de datos lineales:**
  - Pilas
  - Colas
  - Listas



Una **PILA (STACK)** es una **estructura ordenada** y de **elementos homogéneos**, en la que podemos **apilar** o **desapilar** elementos en una **única posición** llamada **CIMA**, y siguiendo una política **LIFO** (**Last In, First Out**).

Es una estructura de datos **Lineal**.





## Operaciones Básicas de Pila

**empty:** Determina si la pila está vacía o no.

**Precondición:** Ninguna.

**Postcondición:** Decide si la pila  $p$  tiene elementos o no.

Por tanto, la pila  $p$  no se modifica.

**top:** Devuelve el elemento de la cima de la pila.

**Precondición:** La pila  $p$  no puede estar vacía.

**Postcondición:** Obtiene el elemento que ocupa la cima de la pila  $p$ .

Por tanto, la pila  $p$  no se modifica.

## Operaciones Básicas de Pila

**push:** Inserta un elemento en la pila y se obtiene la pila con un elemento más.

**Precondición:** Ninguna.

**Postcondición:** Almacena en la pila  $p$  el elemento  $e$ .

Por tanto, la pila  $p$  se modifica.

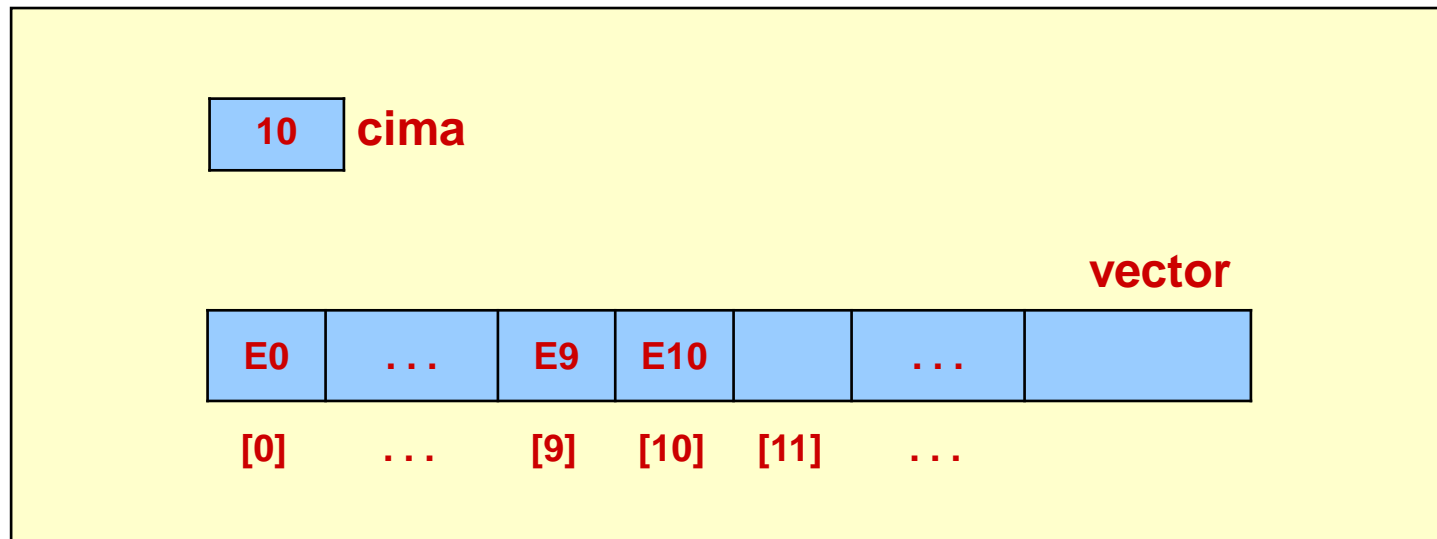
**pop:** Elimina el elemento de la cima de la pila y devuelve la pila resultante.

**Precondición:** La pila  $p$  no puede estar vacía

**Postcondición:** Elimina de la pila  $p$  el elemento que ocupa la cima.

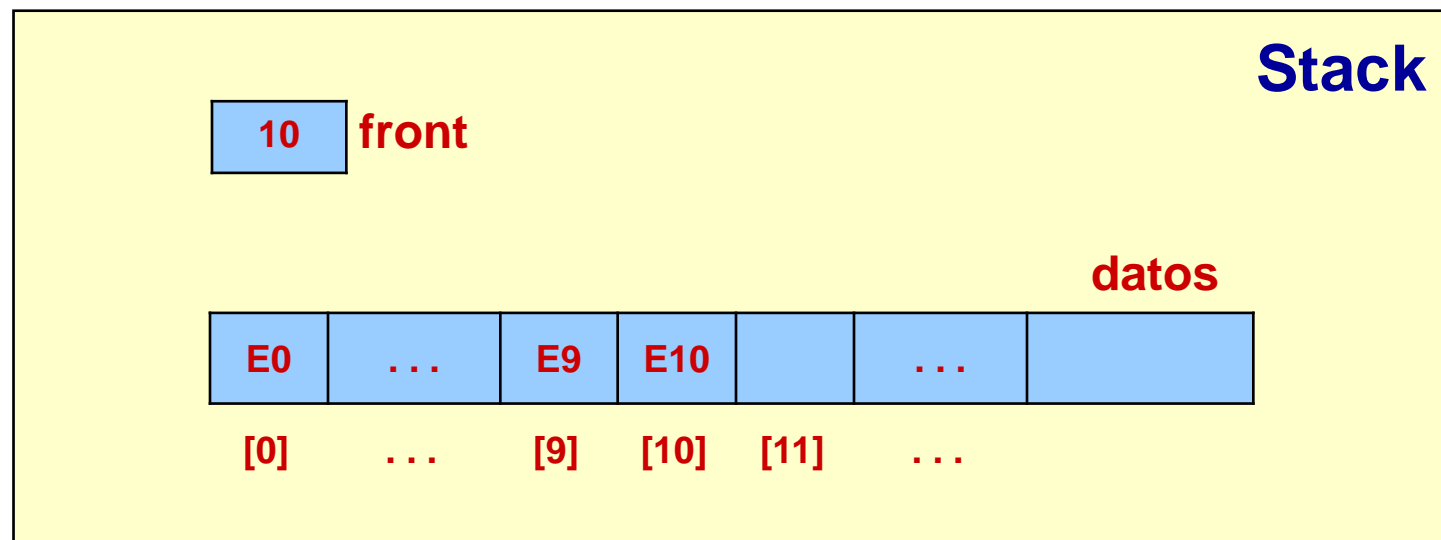
Por tanto, la pila  $p$  se modifica.

## Pila





```
class Stack {  
    public:  
        Stack():front(-1){}  
  
        bool empty() const;  
  
        void push(const TipoDato &dato);  
        void pop();  
        TipoDato top() const;  
  
    private:  
        vector<TipoDato> datos;  
        int front;  
};
```



```
class Stack {  
    public:  
        Stack():front(-1){}  
  
        bool empty() const;  
        void push(int elem);  
        void pop();  
        int top() const;  
  
    private:  
        int front;  
        vector<int> pila;  
};
```

**Stack**

10
----

**front****pila**

23	...	58	30		...	
[0]	...	[9]	[10]	[11]	...	

```
class Stack {  
    public:  
        Stack():front(-1){}  
  
        bool empty() const;  
        void push(int elem);  
        void pop();  
        int top() const;  
  
    private:  
        int front;  
        vector<int> pila;  
};
```

```
// Determina si la pila está vacía o no
```

```
class Stack {  
    public:  
        Stack():front(-1){}  
  
        bool empty() const;  
        void push(int elem);  
        void pop();  
        int top() const;  
  
    private:  
        int front;  
        vector<int> pila;  
};
```

```
// Determina si la pila está vacía o no
```

```
bool Stack::empty() const {  
    return (front == -1);  
}
```

```
class Stack {  
    public:  
        Stack():front(-1){}  
  
        bool empty() const;  
        void push(int elem);  
        void pop();  
        int top() const;  
  
    private:  
        int front;  
        vector<int> pila;  
};
```

```
// Devuelve el elemento cima de la pila
```

```
class Stack {  
    public:  
        Stack():front(-1){}  
  
        bool empty() const;  
        void push(int elem);  
        void pop();  
        int top() const;  
  
    private:  
        int front;  
        vector<int> pila;  
};
```

// Devuelve el elemento cima de la pila

```
int Stack::top() const {  
    return (pila.at(front));  
}
```

```
class Stack {  
    public:  
        Stack():front(-1){}  
  
        bool empty() const;  
        void push(int elem);  
        void pop();  
        int top() const;  
  
    private:  
        int front;  
        vector<int> pila;  
};
```

// Inserta un elemento en la cima de la pila

```
class Stack {  
    public:  
        Stack():front(-1){}  
  
        bool empty() const;  
        void push(int elem);  
        void pop();  
        int top() const;  
  
    private:  
        int front;  
        vector<int> pila;  
};
```

// Inserta un elemento en la cima de la pila

```
void Stack::push(int elem) {  
    front++;  
    pila.push_back(elem);  
}
```



```
class Stack {  
    public:  
        Stack():front(-1){}  
  
        bool empty() const;  
        void push(int elem);  
        void pop();  
        int top() const;  
  
    private:  
        int front;  
        vector<int> pila;  
};
```

// Elimina un elemento en la cima de la pila

```
class Stack {  
    public:  
        Stack():front(-1){}  
  
        bool empty() const;  
        void push(int elem);  
        void pop();  
        int top() const;  
  
    private:  
        int front;  
        vector<int> pila;  
};
```

// Elimina un elemento en la cima de la pila

```
void Stack::pop() {  
    front--;  
}
```

```
class Nodo {  
    private:  
        int dato;  
        shared_ptr<Nodo> next;  
    public:  
        Nodo():next(nullptr){}  
        Nodo(int const &d, shared_ptr<Nodo> ptr):dato(d),next(ptr){}  
  
        int getDato() const;  
        void setDato(const int &newDato);  
        shared_ptr<Nodo> getNext() const;  
        void setNext(const shared_ptr<Nodo> &newNext);  
};
```

```
class Stack {  
    public:  
        Stack():front(nullptr){}  
  
        int size() const;  
        bool empty() const;  
  
        void push(const int &dato);  
        void pop();  
        int top() const;  
  
    private:  
        shared_ptr<Nodo> front;  
};
```

```
// Determina si la pila está vacía o no  
bool Stack::empty() const {  
    return (front == nullptr);  
}
```

```
// Devuelve el elemento cima de la pila  
int Stack::top() const {  
    return (front->getDato());  
}
```

```
// Inserta un elemento en la cima de la pila
void Stack::push(const int& dato) {
    shared_ptr<Nodo> elem = make_shared<Nodo>(Nodo(dato, nullptr));

    if (front != nullptr) {
        elem->setNext(front);
    }
    front = elem; // La pila debe señalar siempre al elemento incluido
}
```

```
// Elimina un elemento en la cima de la pila  
void Stack::pop() {  
    front = front->getNext();  
}
```

```
struct TipoDato {
    string elem;
    int aa;
};

class Nodo {
private:
    TipoDato dato;
    shared_ptr<Nodo> next;
public:
    Nodo():next(nullptr){}
    Nodo(TipoDato const &d, shared_ptr<Nodo> ptr):dato(d),next(ptr){}

    TipoDato getData() const;
    void setData(const TipoDato &newDato);
    shared_ptr<Nodo> getNext() const;
    void setNext(const shared_ptr<Nodo> &newNext);
};

class Stack {
public:
    Stack():front(nullptr){}

    bool empty() const;
    void push(const TipoDato &dato);
    void pop();
    TipoDato top() const;

private:
    shared_ptr<Nodo> front;
};
```



```
// Determina si la pila está vacía o no  
bool Stack::empty() const {  
    return (front == nullptr);  
}
```

```
// Devuelve el elemento cima de la pila  
int Stack::top() const {  
    return (front->getDato());  
}
```

```
// Inserta un elemento en la cima de la pila
void Stack::push(const int& dato) {
    shared_ptr<Nodo> elem = make_shared<Nodo>(Nodo(dato, nullptr));

    if (front != nullptr) {
        elem->setNext(front);
    }
    front = elem; // La pila debe señalar siempre al elemento incluido
}
```

```
// Elimina un elemento en la cima de la pila  
void Stack::pop() {  
    front = front->getNext();  
}
```