# LECCIÓN 8: ESTRUCTURAS de DATOS LINEALES

## ESTRUCTURAS de DATOS LINEALES (EDL)

- Son Contenedores: Almacenan elementos de un tipo base (enteros, strings, etc)
- Son E.D. Lineales → porque contienen una secuencia de elementos $a_0, a_1, \ldots, a_n$ dispuestos en una dimensión.

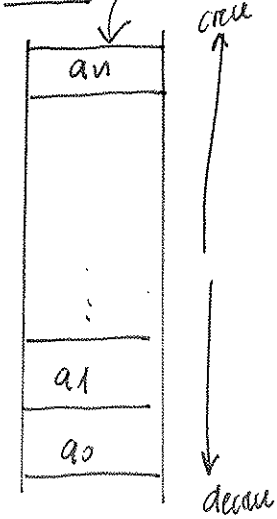
ED. LINEALES
- . PILA — Política LIFO (last input first output)
- . COLA — Política FIFO (first input first output)
- . COLA CON Prioridad
- . Listas

↕ asociaremos

ABSTRACCIÓN POR ITERACIÓN (Iteradores)

## PILAS — POLITICA LIFO

Especificación — Contienen una secuencia de valores de $a_0, a_1, \ldots, a_n$ y especialmente diseñadas para realizar inserciones y borrados por uno de sus extremos denominado TOPE.




OPERACIONES

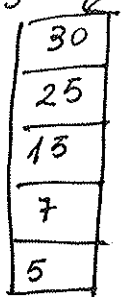
- . Tope (top): Consulta el elemento en el TOPE
- . Vacía (empty): devuelve true si la pila está vacía o false en otro caso
- . Quitar (pop): elimina el elemento en el tope de la pila
- . Poner (push): Inserta un nuevo elemento por el TOPE de la pila.

Ver transparencia 1.
ejemplo de uso de una Pila.

Ejemplo:
En la entrada estándar aparece los datos

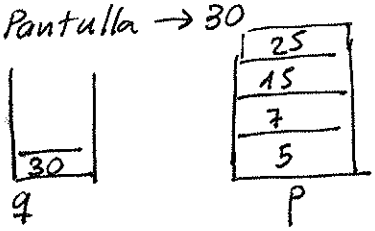
5 7 15 25 30

El código genera una pila p con la siguiente información (tope.)

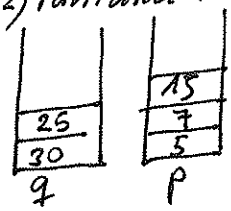



Sentencia 5-8

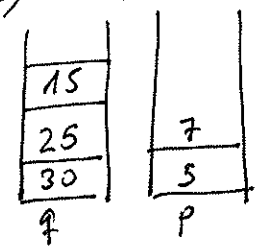
1) Pantalla → 30



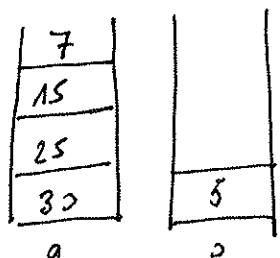

2) Pantalla → 30 25



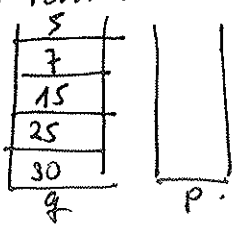

3) Pantalla 30 25 15




4) Pantalla 30 25 15 7




5) Pantalla 30 25 15 7



TRANSPARENCIA 1.



# Pilas

## *Esquema de pila*

Una posible *clase Pila* para almacenar datos de tipo *char* puede tener la siguiente sintaxis.

```
#ifndef __PILA_H__
# define __PILA_H__

  class Pila{
      ...                    // La implementación que deseemos
    public:
      Pila();
      Pila(const Pila& p);
      ~Pila();
      Pila& operator= (const Pila& p);

      bool vacia() const;
      void poner (char c);
      void quitar();
      char tope() const;
  };

#endif
```

## *Uso de una pila*

*Con esta sintaxis y la semántica comentada*

```
#include <iostream>
#include <pila.h>
using namespace std;

int main()
{
  Pila p,q;
  char dato;

1 cout << "Escriba una frase" << endl;
2 while ((dato=cin.get())! ='\n')
3   p.poner(dato);

4 cout << "Los escribimos al revés" << endl;
5 while (!p.vacia()) {
6   cout << p.tope();
7   q.poner(p.tope());
8   p.quitar();
  }
9 cout << endl << "Los originales eran" << endl;
10 while (!q.vacia()) {
11   cout << q.tope();
12   q.quitar();
  }
13 cout << endl;
  return 0;
}
```

LECCION 8.

(2)

VECTOR es una estructura de dato lineal. Ya que contiene una secuencia de valores $\{a_0, a_1, \ldots, a_{n-1}\}$ especialmente diseñada para realizar accesos en tiempo constante.

STL → Standard Template LIBRARY → Un vector se define dentro de la clase vector en la librería vector

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector <int> first
    vector <int> dos (4,100); // 100 100 100 100
    vector <int> tres (dos);

    // leemos los datos de first
    for (int i=0; i<10; i++){
        int aux;
        cin >>aux;
        first.push_back(aux); //NO first[i]=aux
                              //TENEMOS.
    }
    //accedemos a un elemento i para modificarlo
    first[i] = tres[0] + dos[1];

    //eliminamos el ultimo elemento de first
    first.pop_back();

    //consultamos el nº de elementos.
    cout << first.size();
    // copiamos con la asignacion.
    clase = first;
```

# LECCION 8

## IMPLEMENTACION DEL T.D.A PILA USANDO VECTOR de la STL

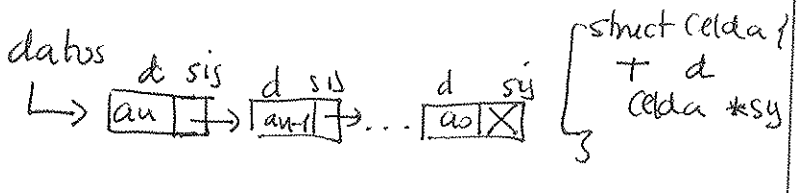
```
#ifndef _PILA_H
#define _PILA_H
#include<vector>
using namespace std
template <class T>
class Pila {
  private
    vector<T> datos;
  public:
    T top() const {
        return datos[datos.size()-1];
    }
    void pop() {
        datos.pop_back( );
    }
    void push(const T &v) {
        datos.push_back(T);
    }
    int size() const {
        return datos.size();
    }
    bool empty() const {
        return datos.size()==0;
    }
};
```

## EJEMPLO de USO

```
#include "Pila.h"
int main() {
  Pila<char> mipila;
  cout<<"Dame una frase:";
  char c;
  while (cin.get(c)) {
      mipila.push(c);
  }
  Pila<char> otrapila;
  while (!pila.empty()) {
      c = mipila.top();
      mipila.pop();
      otrapila.push(c);
  }
  cout<<"Frase al revés"
  while (!otrapila.empty()){
      c = otrapila.top();
      otrapila.pop();
      cout<<c;
  }
}
```

# LECCION 8

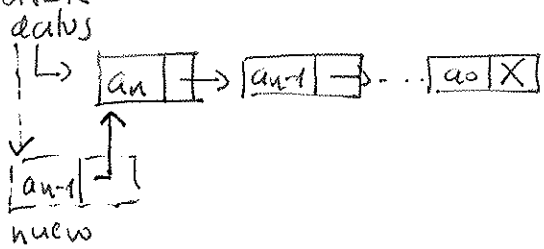
## PILAS: IMPLEMENTACIÓN BASADA en CELDAS ENLAZADAS



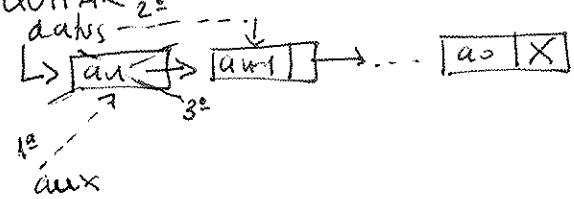

### OPERACIONES

Tope = Consulta datos→d

PONER




QUITAR




```
#ifndef __PILA_H
#define __PILA_H
template <class T>
struct Celda {
   T d;
   Celda *sig;
};
template <class T>
class Pila {
  private:
    Celda<T> *primera;
    void Copiar(const Pila<T> &P);
    void Borrar();
  public:
    Pila();
    Pila(const Pila<T> &P);
    Pila<T> operator=(const Pila<T> &P);
    ~Pila();
    T Tope() const;
    void Quitar();
    void Poner(const T &v);
    int size() const;
    bool Vacia() const;
};
#include "Pila.cpp"
#endif
```

```
// Pila.cpp
template <class T>
void Pila<T>::Copiar(const Pila<T> &P){
   if (P.primera==0) //es vacía
       primera=0
   }
   else{
     primera = new Celda<T>;
     primera->d = P.primera->d;
     Celda<T>* p = primera;
     Celda<T>* q = P.primera->sig;
     while (q!=0) {
        p->sig = new Celda;
        p = p->sig;
        p->d = q->d;
        q = q->sig;
     }
     p->sig=0;
   }
}
template <class T>
void Pila<T>::Borrar() {
    while (primera!=0){
       Celda<T>* aux=primera;
       primera = primera->sig;
       delete aux;
    }
}
```