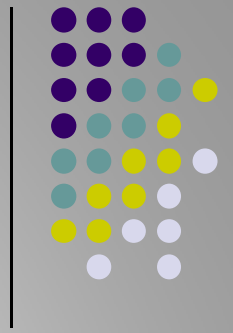
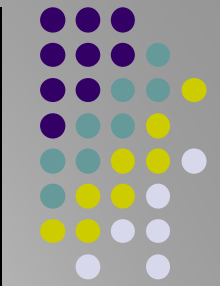
Pilas (definición)



"Una «pila» es conjunto de elementos encadenados o en secuencia donde el primer elemento se reconoce como «tope» y el resto están detrás o debajo de este. Solo es posible acceder al elemento del «tope». Para poder llegar a la base de la pila debemos des-apilar todos los elementos de la pila."

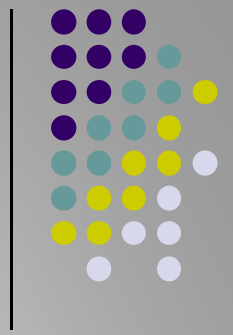
"También podemos decir que una «pila» es un caso especial de una «lista» "

Implementaciones de Pilas



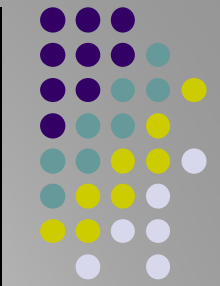
- Hay 3 formas de implementar una pila usando distintos recursos del lenguaje de programación:
  - Usar Arreglos
  - Usar Apuntadores a Memoria (Pilas enlazadas)
  - Usar Arreglos de Registros (Cursor simulando pilas enlazadas, no la vamos a implementar)

Pilas (Restricciones)



1. "Una «pila» no se puede recorrer como una lista.
2. La única manera de recorrerla es des-apilarla. Se suele guardar los elementos en una auxiliar y luego.
3. Solo esta disponible el elemento del «tope».
4. No existe la operación de «inserción».
5. A la pila se la llama «LIFO»

IMPLEMENTACION DE PILAS CON ARREGLOS



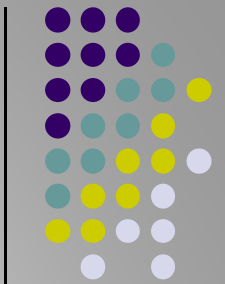
Objetivo:

1. Los datos son almacenados dentro de un array dinámico definido con una constante de MAXIMO.
2. La estructura es un registro llamado "pila" que almacena los datos (TipoElemento) y la cantidad establecida dentro de ella.
3. Cada casillero del array almacenara un registro "TipoElemento" de la librería "tipo_elemento.h".
4. Solo se puede llegar a los datos a través de las operaciones del TAD, no se debe acceder de forma directa al array de elementos.

IMPLEMENTACION DE PILAS CON ARREGLOS (cont)

```
#ifndef PILAS_H
#define PILAS_H
#include <stdbool.h>
#include "../tipo_elemento.h"

struct PilaRep;
typedef struct PilaRep *Pila;

Pila p_crear();

void p_apilar (Pila pila, TipoElemento elemento);
TipoElemento p_desapilar (Pila pila);

TipoElemento p_tope (Pila pila);
bool p_es_vacia (Pila pila);
bool p_es_llena (Pila pila);
void p_mostrar (Pila pila);

#endif // PILAS_H
```

IMPLEMENTACION DE PILAS CON ARREGLOS (cont)



Gráficamente podríamos pensar la estructura de la siguiente manera:

Array donde se guarda la pila denominado "**valores**" (es un puntero)

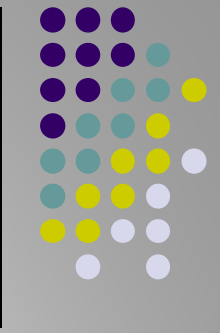
Tipo Elemento (MIN=0)	Tipo Elemento 2	Tipo Elemento 3	… … …	… … …	Tipo Elemento SIZE=N

Base de la Pila

Tope de la Pila

El registro pila además del array de elementos contiene el indicador del tope de la pila y la cantidad de elementos de la misma.

IMPLEMENTACION DE PILAS CON ARREGLOS (cont)



Descripción de las operaciones del TAD

Pila p_crear ();

Objetivo: Inicializar la pila para dejar en condiciones de ser usada como pila vacía.

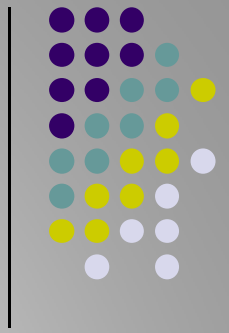
bool p_es_vacia (Pila p);

Objetivo: determinar si la pila esta vacía. Retorna "true" cuando esto sucede, caso contrario retorna "False".

bool p_es_llena (Pila p);

Objetivo: determinar si la pila esta llena. Retorna "true" cuando esto sucede, caso contrario retorna "False".

IMPLEMENTACION DE PILAS CON ARREGLOS (cont)



Descripción de las operaciones del TAD (continua)

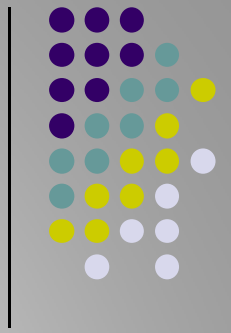
void p_apilar (Pila p, TipoElemento X);

Objetivo: Apila un elemento en el TOPE de la pila. Recibe como parámetro el "tipoelemento" a apilar. Controla que no este llena.

TipoElemento p_desapilar (Pila p);

Objetivo: retorna el elemento y elimina el tope de la pila. Controla que no este vacía.

IMPLEMENTACION DE PILAS CON ARREGLOS (cont)



Descripción de las operaciones del TAD (continua)

TipoElemento p_tope(Pila p);

Objetivo: Extraer el "tipoelemento" del tope de la pila. En este caso no desapila. En caso de error retorna un «tipoelemento» vacío.

void p_mostrar (Pila p);

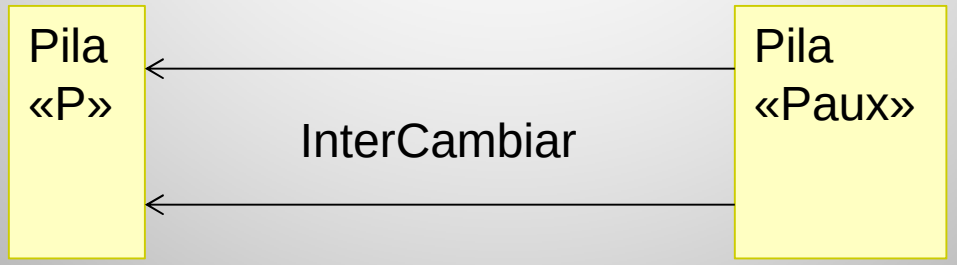
Objetivo: Muestra en pantalla el contenido de la Pila sin perderla. Solo muestra la clave.

IMPLEMENTACION DE PILAS CON ARREGLOS (cont)

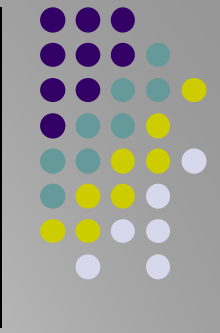
Utilidad de la Pila generica para todas las implementaciones

void p_intercambiar (Pila P, Pila Paux);

Objetivo: esta utilidad permite pasar todos los elementos de la pila «Paux» a la pila «P». Luego del proceso la pila «Paux» quedará vacía y «P» contendrá todos los elementos de forma inversa.
Recibe por referencia ambas pilas.
Esta función NO es parte del TAD.



IMPLEMENTACION DE PILAS CON APUNTADORES (Def. del Tipo)

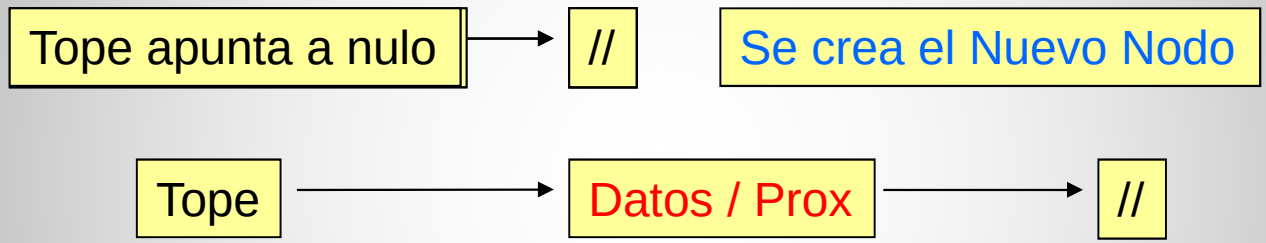


```
#include "pilas.h"
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

struct Nodo {
   TipoElemento datos;          // Concepto del Nodo
   struct Nodo *siguiente;
};

struct PilaRep {
   struct Nodo *tope;
};
```

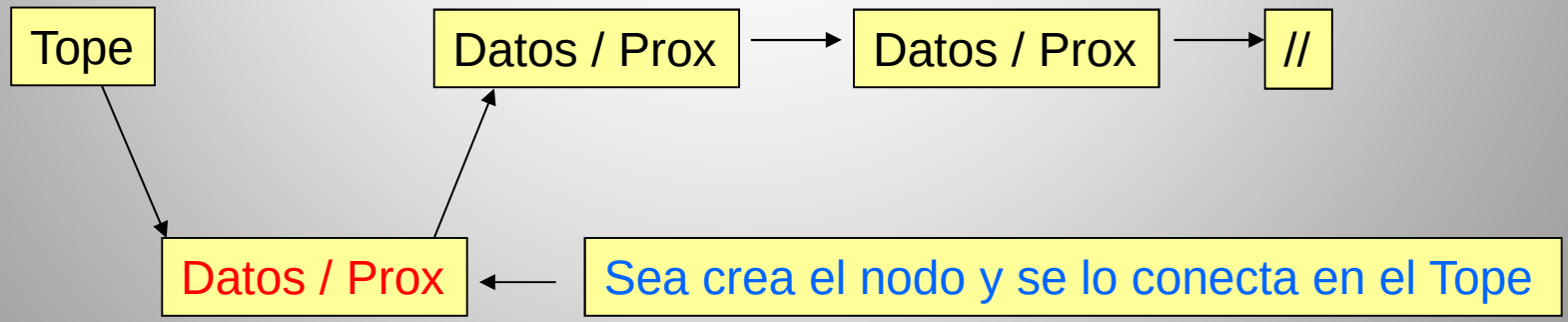
IMPLEMENTACION DE PILAS CON APUNTADORES (Operaciones Básicas)

APILAR

1 – Pila esta vacía



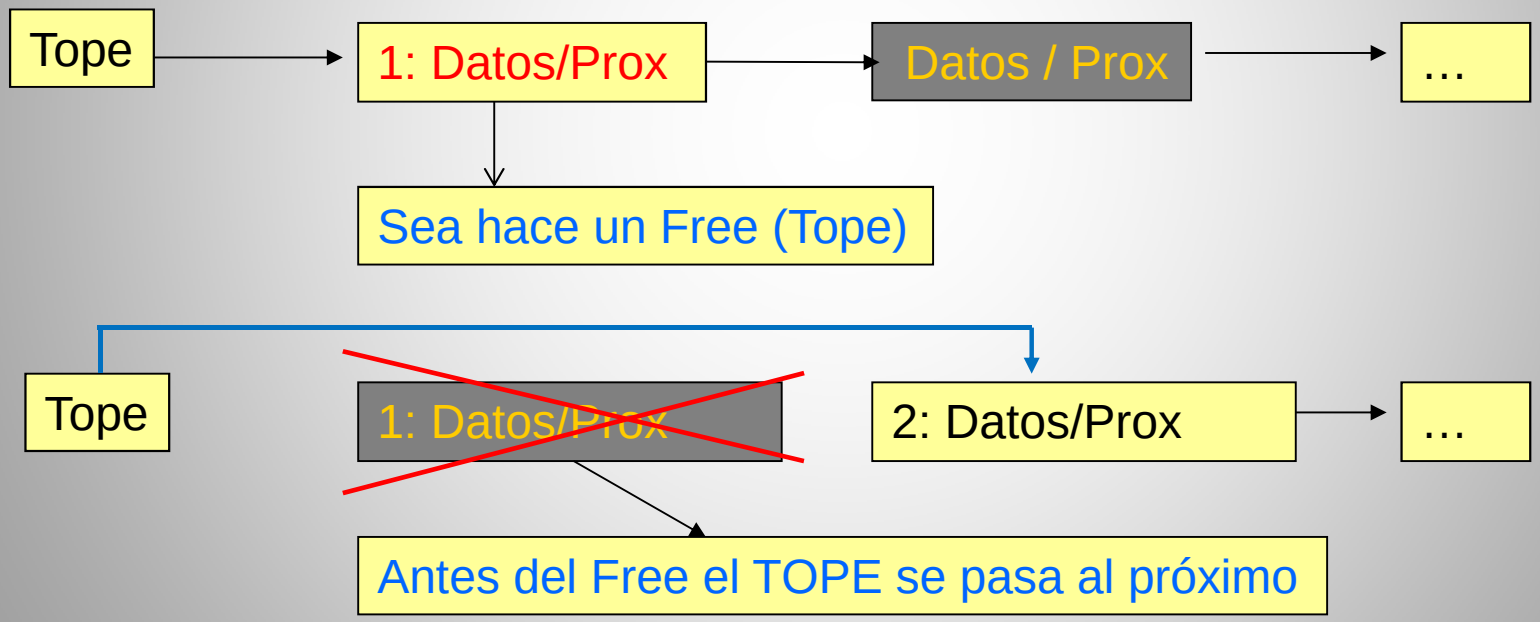
2 – Pila contiene elementos



IMPLEMENTACION DE PILAS CON APUNTADORES (Operaciones Básicas)

DESAPILAR

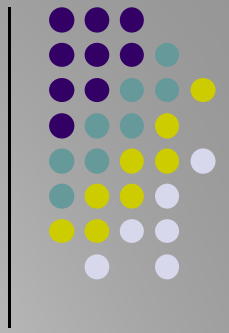
1 – Siempre se des-apila el TOPE (cambia el nodo inicial)



IMPLEMENTACION DE PILAS CON APUNTADORES (Operaciones del TAD)

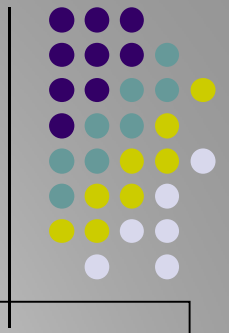
```
Pila p_crear() {
    Pila nueva_pila = (Pila) malloc(sizeof(struct PilaRep));
    nueva_pila->tope = NULL;
    return nueva_pila;
}

void p_apilar (Pila pila, TipoElemento elemento) {
    struct Nodo *nuevo_nodo = malloc(sizeof(struct Nodo));
    nuevo_nodo->datos = elemento;
    nuevo_nodo->siguiente = pila->tope;
    pila->tope = nuevo_nodo;
}

TipoElemento p_desapilar (Pila pila) {
    struct Nodo *tope_actual = pila->tope;
    TipoElemento elemento = tope_actual->datos;
    pila->tope = tope_actual->siguiente;
    free(tope_actual);
    return elemento;
}
```

IMPLEMENTACION DE PILAS CON APUNTADORES (Operaciones del TAD)



```
TipoElemento p_tope (Pila pila) {
    struct Nodo *tope_actual = pila->tope;
    return tope_actual->datos;
}


bool p_es_vacia (Pila pila) {
    return pila->tope == NULL;
}
```

IMPLEMENTACION DE PILAS CON APUNTADORES (Operaciones del TAD)



```
void p_mostrar (Pila pila) {
    Pila Paux = p_crear();
    TipoElemento X = crear_te(0);

    printf("Contenido de la pila: ");

    // Recorro la pila desopilándola y pasándola al auxiliar
    while (p_es_vacia(pila) != true) {
        X = p_desapilar(pila);
        printf("%d ", X->clave);
        p_apilar(Paux, X);
    }

    // Recorro la pila auxiliar para pasarla a la original  (o bien construyo la utilidad
       intercambiar)
    while (p_es_vacia(Paux) != true) {
        X = p_desapilar(Paux);
        p_apilar(pila, X);
    }
    printf("\n");
}
```