



Práctica 3 - Pilas y Colas

Pilas

1. Implemente pilas utilizando arreglos. Utilice la siguiente estructura:

```
typedef struct _Pila {  
    int datos[MAX_PILA];  
    int ultimo;  
} *Pila;
```

Procure que la interfaz provea las siguientes operaciones:

- a Pila pila_crear() : Crea e inicializa una nueva pila vacía.
 - b int pila_es_vacia(Pila): determina si la pila está vacía.
 - c int pila_ultimo(Pila): toma una pila y devuelve el elemento en la cima.
 - d void pila_apilar(Pila, int): toma una pila y un elemento y agrega el elemento a la pila.
 - e void pila_desapilar(Pila): toma una pila y elimina el elemento de la cima.
 - f void pila_imprimir(Pila): toma una pila y la imprime en orden.
 - g void pila_destruir(Pila) : libera la memoria requerida para la pila.
2. Modifique la estructura recién utilizada para poder almacenar cualquier cantidad de elementos (modifique las funciones necesarias para que, en caso de quedarse sin lugar, se solicite más memoria automáticamente).
3. Implemente pilas enlazadas. Para ello, reutilice la implementación de listas enlazadas para almacenar datos de tipo entero:

```
typedef SList Pila;
```

Procure que la interfaz provea las mismas operaciones que en el ejercicio anterior:

```
a Pila pila_crear();  
b int pila_es_vacia(Pila);  
c int pila_ultimo(Pila);  
d Pila pila_apilar(Pila, int);  
e Pila pila_desapilar(Pila);  
f void pila_imprimir(Pila);  
g void pila_destruir(Pila);
```

4. Considere las listas simplemente enlazadas implementadas en la práctica 2. Implemente la función `SList slist_reverso(SList)` que tome una lista simplemente enlazada y la invierta utilizando una pila.

Colas

5. Implemente colas utilizando arreglos circulares. Utilice la siguiente estructura:

```
typedef struct _Cola {
    int datos[MAX_COLA];
    int primero, ultimo;
} *Cola;
```

Procure que la interfaz provea las siguientes operaciones:

- a Cola cola_crear(): crea e inicializa una nueva cola vacía.
- b int cola_es_vacia(Cola): determina si la cola está vacía.
- c int cola_primero(Cola): toma una cola y devuelve el elemento en la primera posición.
- d void cola_encolar(Cola, int): toma una cola y un elemento y agrega el elemento al fin de la cola.
- e void cola_desencolar(Cola): toma una cola y elimina su primer elemento.
- f void cola_imprimir(Cola): toma una cola y la imprime en orden.
- g void cola_destruir(Cola) : libera la memoria requerida para la cola.

6. Implemente colas enlazadas. Para ello, utilice el siguiente tipo Cola para guardar referencias al primer y último elemento de una lista enlazada.

```
typedef struct _Cola {
    SNodo* primero;
    SNodo* ultimo;
} *Cola;
```

Alternativamente, puede usar listas circulares doblemente enlazadas, donde podemos acceder fácilmente al último elemento, definida cómo:

```
typedef CDList Cola;
```

Procure que la interfaz provea las mismas operaciones que en el ejercicio anterior:

- a Cola cola_crear();
- b int cola_es_vacia(Cola);
- c int cola_primero(Cola);
- d void cola_encolar(Cola, int);
- e void cola_desencolar(Cola);
- f void cola_imprimir(Cola);
- g void cola_destruir(Cola);