



## Práctica 3 - Pilas, Colas y Heaps

### 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() : inicializa los datos de la pila (valor de 'último').
- b) int pila\_es\_vacia(Pila): determina si la pila está vacía.
- c) int pila\_top(Pila): toma una pila y devuelve el elemento en la cima.
- d) void pila\_push(Pila, int): toma una pila y un elemento y agrega el elemento a la pila.
- e) void pila\_pop(Pila): toma una pila y elimina el elemento de la cima.
- f) void pila\_imprimir(Pila): toma una pila y la imprime en orden.

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\_top(Pila);
- d) Pila pila\_push(Pila, int);
- e) Pila pila\_pop(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\_inicializar(): inicializa los datos de la cola (valor de 'primero' y 'último').
- 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.

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 como:

```
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);

## Heaps Binarios

7. Implemente heaps binarios utilizando arreglos para representar árboles binarios completos parcialmente ordenados. Utilice la siguiente estructura:

```
typedef struct _BHeap {
    int datos[MAX_HEAP];
    int nelems;
} *BHeap;
```

Procure que la interfaz provea las siguientes operaciones:

- a) BHeap bheap\_crear() : crea un heap vacío.
- b) int bheap\_es\_vacio(BHeap) : determina si el heap está vacío.
- c) int bheap\_minimo(BHeap): toma un heap y devuelve el menor elemento.
- d) void bheap\_eliminar\_minimo(BHeap): toma un heap y borra su menor elemento.
- e) void bheap\_insertar(BHeap, int): toma un heap y agrega un elemento.
- f) void bheap\_imprimir(BHeap): toma un heap e imprime sus elementos utilizando el orden 'Primero por Extensión'.
- g) void bheap\_destruir(BHeap) : Destruye un heap.

8. Llamamos “cola de prioridad” a una estructura con las siguientes acciones asociadas:

- a) int cola\_prioridad\_es\_vacia(PCola) : determina si la cola está vacía.
- b) int cola\_prioridad\_obtener\_minimo(PCola) : obtiene el elemento prioritario.
- c) void cola\_prioridad\_eliminar\_minimo(PCola) : quita el elemento prioritario.
- d) void cola\_prioridad\_insertar(PCola, int) : inserta un elemento con determinada prioridad.

Implementela:

- 1) Utilizando un arreglo circular ordenado.
- 2) Utilizando una lista enlazada ordenada.
- 3) Utilizando un heap.

¿Cuáles son las ventajas y desventajas de cada implementación, en relación a la eficiencia de cada acción?

9. Implemente `heapify(int arr[], size_t tamaño)` que transforme un arreglo dado en un heap.

10. Implemente `BHeap bheap_merge(BHeap heap1, BHeap heap2)` que una el segundo heap al primero, de manera eficiente.