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()` : Crea e inicializa una nueva pila vacía.

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.

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_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_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);`

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) `BHead 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 proritario.

c) `void cola_prioridad_eliminar_minimo(PCola)` : quita el elemento prioritario.

d) `void cola_prioridad_insertar(PCola, int)` : inserta un elemento con determinada prioridad.

Impleméntela:

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 tamanio)` 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.