

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo			Parcial 17/10/2011
Curso	Página 1	Ejercicio	Puntaje
Apellido y Nombre		1	
Padrón		2	
		3	
Nota Final	Corrigió	4	

Práctica

Entregar la resolución de la Teoría y la Práctica en hojas separadas

Ejercicios 1 y 2) APLICACIÓN - En estos ejercicios debe utilizar en forma abstracta, mediante sus primitivas, los TDAs vistos en clase en los lugares donde los considere apropiados (por ejemplo, el TCola dado en la estructura de Cancionero).

Problema Marco:

Un nuevo dispositivo personal de reproducción de música, el podl, lanzará al mercado su versión Shuffle. Como parte de la implementación de su sistema operativo, cuenta con el TDA Cancionero, descripto a continuación.

```
typedef struct {
#include "cola/cola.h"

/*
 * Todas las primitivas devuelven OK si su ejecución ha sido exitosa
 * o ERROR si no lo ha sido (y dejan inalterado el TDA) */
#define OK 0
#define ERROR -1

typedef struct {
    int nroDeOrdenOriginal;
    char autor[128];
    char rutaArchivo[256];
} Cancion;

typedef struct {
    Tcola canciones;
    int cantcanciones;
    int proximoNroDeOrden;
} Cancionero;

/*
 * PRE: ninguna (Cancionero no creado)
 * POST: El Cancionero ha sido creado */
int Cancionero_Crear(Cancionero*);

/*
 * PRE: Cancionero creado
 * POST: El Cancionero ha sido destruido */
int Cancionero_Destruir(Cancionero*);

/*
 * PRE: Cancionero creado, Cancion contiene una canción válida
 * POST: Se ha agregado la canción dada como última canción a reproducir
 *
 * Nota de implementación:
 * El numeroDeOrdenOriginal no es más que un número de secuencia (del que
 * no se reaprovechan los números utilizados cuando se toma la próxima
 * canción y se quita del cancionero; se supone que se puede incrementar
 * proximoNroDeOrden y siempre alcanza el rango). */
int Cancionero_AgregarAlFinal(Cancionero*, Cancion);

/*
 * PRE: Cancionero creado, Canción apunta a un sector de memoria con suficiente
 * memoria reservada para contener una Cancion
 * POST: Cancion contiene la próxima canción a reproducir */
```

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo		Parcial 17/10/2011
Padrón	Apellido y Nombre	Curso

```

int Cancionero_ObtenerProxima(Cancionero*, Cancion*);

/*
 * Mezcla las canciones
 *
 * PRE: Cancionero creado, n>1
 * POST: Las canciones contenidas en el cancionero han sido mezcladas
 *
 * Nota de implementación:
 * Reordena internamente las canciones según el siguiente algoritmo:
 * Toma las próximas n canciones y las coloca en orden inverso, luego las
 * siguientes n y las coloca en orden inverso a continuación, y así
 * siguiendo hasta que se agoten las canciones (la última pasada será de
 * n canciones o menos).
 * Ejemplo (mostrando nroDeOrdenOriginal-autor):
 * 1-Autor1; 2-Autor2; 3-Autor3; 4-Autor4; 5-Autor5
 * con int n=2, hace que las canciones queden en el siguiente orden:
 * 2-Autor2;1-Autor1;4-Autor4;3-Autor3;5-Autor5
 * De esta forma ahora una llamada a Cancionero_ObtenerProxima devolvería
 * la canción 2, luego la 1, luego la 4, etc.
 * en cambio, con int n=3 hace que las canciones queden de la forma:
 * 3-Autor3;2-Autor2;1-Autor1;5-Autor5;4-Autor4
 *
 * Si no hay canciones suficientes en el cancionero el resultado es exitoso
 * (se habrán mezclado 0 canciones o 1 canción) */

int Cancionero_Shuffle(Cancionero*, int n);

/*
 * Ordena las canciones según el orden en que fueron insertadas originalmente
 *
 * PRE: Cancionero creado
 * POST: Las canciones contenidas en el cancionero han quedado en su orden de
 * inserción original
 *
 * Nota de implementación:
 * Reestablece el orden de las canciones según su número de orden original.
 * Por ejemplo
 * 2-Autor2;1-Autor1;4-Autor4;3-Autor3;5-Autor5
 * quedaría como
 * 1-Autor1;2-Autor2;3-Autor3;4-Autor4;5-Autor5
 * De esta forma ahora una llamada a Cancionero_ObtenerProxima devolvería
 * la canción 1, luego la 2, etc.
 *
 * Si no hay canciones suficientes en el cancionero el resultado es exitoso
 * (se habrán ordenado 0 canciones) */

int Cancionero_Ordenar(Cancionero*);

```

Se pide:

1. Implemente la primitiva Cancionero_Shuffle
2. Implemente la primitiva Cancionero_Ordenar

Ejercicios 3 y 4) IMPLEMENTACIÓN - en estos ejercicios no puede utilizar otros TDAs ni llamar a primitivas dentro de su implementación (sí a nuevas funciones auxiliares que defina). Esto quiere decir que todo trabajo de lógica, algoritmia, uso de punteros, etc. deberá ser desarrollado por el alumno como parte de la resolución.

Problema Marco:

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo		Parcial 17/10/2011
Padrón	Apellido y Nombre	Curso

Un conjunto matemático es una colección de entidades que no contiene elementos duplicados. En general no garantiza un orden de los elementos, pero puede hacerlo. A continuación se propone una definición para un TDA ConjuntoOrdenado (que sí garantiza el orden):

```
#include "lista/ListaSimple.h"

/*
 * Todas las primitivas devuelven OK si su ejecución ha sido exitosa
 * y no hay un valor de retorno específico
 * o ERROR si no lo ha sido (y dejan inalterado el TDA) y no hay un
 * valor de retorno específico
 */
#define OK 0
#define ERROR -1
#define EXISTE -2

#define IGUALES 1
#define DISTINTOS 0

/* Compara los elementos pasados por parámetro
 * Devuelve 0 si son iguales, <0 si el primer elemento es menor que
 * el segundo y >0 si el primer elemento es mayor que el segundo */
typedef int (*fcomp)(void*,void*);
/* Clona el elemento dado en el primer parámetro, en el segundo */
typedef int (*fclonarElem)(void*,void*);
/* Libera los recursos utilizados por el elemento dado (pero no el
 * propio elemento) */
typedef int (*fdestruirElem)(void*);

typedef struct {
    /* A COMPLETAR POR EL ALUMNO */
} ConjuntoOrdenado;

/* PRE: ninguna; POST: Conjunto creado */
int ConjuntoOrdenado_Crear(ConjuntoOrdenado* , fcomp comparar, fclonarElem clonar,
fdestruirElem destruir, int tamañoElemento);

/* PRE: Conjunto creado; POST: Conjunto destruido */
int ConjuntoOrdenado_Destruir(ConjuntoOrdenado*);

/* PRE: Conjunto creado
 * POST: Se ha agregado el elemento dado si no estaba en el conjunto,
 * devuelve OK si se ha agregado el elemento o EXISTE si ya existía en
 * el conjunto (y no lo altera) */
int ConjuntoOrdenado_Agregar(ConjuntoOrdenado*, void* elemento);

/* PRE: Conjunto creado; POST: Se ha quitado el elemento dado */
int ConjuntoOrdenado_Quitar(ConjuntoOrdenado*, void* elemento);

/* PRE: Conjuntos creados; POST: Se han agregado al conjunto los elementos del
 * conjunto dado por parámetro */
int ConjuntoOrdenado_UnirCon(ConjuntoOrdenado* afectado, ConjuntoOrdenado* aUnir);

/* PRE: Conjuntos creados; POST: Se han removido del conjunto los elementos del
 * conjunto dado por parámetro */
int ConjuntoOrdenado_Remove(ConjuntoOrdenado* afectado, ConjuntoOrdenado* aRemove);

/* PRE: Conjuntos creados; POST: Se han removido del conjunto los elementos no contenidos
en el conjunto dado por parámetro */
int ConjuntoOrdenado_Interseccion(ConjuntoOrdenado* afectado, ConjuntoOrdenado*
aInterseccion);

/* PRE: Conjuntos creados;
 * POST: Devuelve IGUALES si los conjuntos contienen la misma cantidad
 * de elementos y dichos elementos son los mismos (por la condición de
 * conjunto ordenado, se desprende que estarán en el mismo orden), o
 * DISTINTOS si dichas condiciones no se cumplen */
int ConjuntoOrdenado_Igual(ConjuntoOrdenado*, ConjuntoOrdenado*);
```

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo		Parcial 17/10/2011
Padrón	Apellido y Nombre	Curso

```

/* PRE: Conjunto creado, ListaSimple creada
 * POST: puebla la lista dada con los elementos del conjunto, en el mismo orden
 * en que están en el conjunto
 *
 * Nota de implementación: esta es la única primitiva que utiliza otro TDA, y lo
 * hace mediante sus primitivas, en forma abstracta */
int ConjuntoOrdenado_DevolverEnLista(ConjuntoOrdenado*, TListaSimple* lista);

```

3. Complete la definición del TDA ConjuntoOrdenado. Puede definir más typedef si lo considera adecuado.
4. Implemente la primitiva ConjuntoOrdenado_Agregar

Nota:

En todos los casos en que no se indique explícitamente lo contrario, los elementos con los que cuenta son las estructuras del lenguaje C, el TDA ListaSimple explicado en clase, el TDA Pila explicado en clase, y el TDA Cola explicado en clase. Toda funcionalidad por encima de estos elementos deberá ser desarrollada como parte del examen.

Para aprobar el examen, debe (las condiciones son aditivas, ninguna es opcional):

- a) resolver el ejercicio 1 o el 2, y el ejercicio 4
- b) demostrar claramente que ha aprendido el concepto de abstracción
- c) demostrar claramente que ha aprendido a utilizar y a implementar las estructuras de datos que se enseñaron
- d) demostrar claramente que ha aprendido los elementos de programación que se enseñaron
- e) resolver correctamente al menos el 60% del examen