

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo			Parcial 11/06/2012
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.

Problema Marco:

Para el sistema de expendio de un restaurant, se desarrolla un TDA Cava que permite almacenar y operar con el stock de vinos disponibles. Como sucede habitualmente en muchos restaurantes, si el comensal ordena un vino del cual no hay stock disponible, se le ofrecerá el siguiente más caro al precio del que el cliente había solicitado.

A continuación se presenta el diseño del TDA, que cuenta con una lista interna de stock ordenada por la clave compuesta (bodega, cepa) y una lista interna de precios, ordenada por la clave simple precioCtvs. Los ordenamientos los realiza el cliente de la ListaSimple, que en este caso es el TDA Cava. Los varietales (bodega, cepa) de vinos disponibles siempre están presente en las listas (si no hay ítems, con stock 0).

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

/* Todas las primitivas del TDA Cava devuelven OK u ERROR salvo que se indique lo contrario. */

#define OK 0
#define ERROR -1
#define MODIFICADO 2

typedef struct {
    ListaSimple stockXvarietal; /* lista de ItemStock, ordenada por (bodega, cepa) */
    ListaSimple vinosXprecio; /* lista de PrecioVino, ordenada por precioCtvs */
} Cava;

typedef struct {
    char bodega[50];
    char cepa[50];
    int stock;
} ItemStock;

typedef struct {
    int precioCtvs;
    char bodega[50];
    char cepa[50];
} PrecioVino;

/* El siguiente es un tipo de dato de intercambio entre el TDA Cava y sus clientes */

typedef struct {
    char bodega[50];
    char cepa[50];
    int precioCtvs;
} Vino;

/* Las siguientes son las primitivas del TDA Cava */

/* PRE: ninguna, POST: Cava creada */
int Cava_Crear(Cava*);

/* PRE: Cava creada, POST: Cava destruida */
int Cava_Destruir(Cava*);
```

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo		Parcial 11/06/2012
Padrón	Apellido y Nombre	Curso

```

/* PRE: Cava creada, POST: Varietal registrado en la Cava, si ya existía devuelve ERROR */
int Cava_RegistrarVarietal(Cava*, char *bodega, char *cepa, int precioCtvos);

/* PRE: Cava creada POST: Se han agregado la cantidad dada de items al stock del
varietal, si no existe tal varietal devuelve ERROR*/
int Cava_ReponerStock(Cava*, char *bodega, char *cepa, int unidades);

/* PRE: Cava creada , POST: si no existe el varietal, devuelve ERROR; si existe el
varietal y tiene stock, resta una unidad al stock disponible y puebla el parámetro
vinoOrdenado con los datos del vino, devuelve OK; si existe el varietal pero no tiene
stock, encuentra el vino inmediatamente más caro con stock, resta una unidad al stock
disponible y puebla sus datos en vinoOrdenado, devuelve OK; si existe el varietal y no
tiene stock y no hay ningún vino más caro con stock, devuelve ERROR */
int Cava_OrdenarVino(Cava*, char *bodega, char *cepa, Vino *vinoOrdenado);

/* PRE: Cava creada, POST: si no existe el varietal, devuelve ERROR; si existe el
varietal modifica su precio y devuelve OK */
int Cava_ModificarPrecio(Cava*, char *bodega, char *cepa, int nuevoPrecioCtvos);

/* PRE: Cava creada, POST: desregistra el varietal si existe, si no devuelve ERROR */
int Cava_DesregistrarVarietal(Cava *, char *bodega, char *cepa);

```

Nota: Todos los vinos tienen precios distintos

Se pide:

1. Implemente la primitiva Cava_OrdenarVino
2. Implemente la primitiva Cava_RegistrarVarietal

Ejercicios 3, 4 y 5) 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:

Se desea contar con una Lista genérica que permita almacenar cualquier tipo de datos, y desplazarse no sólo hacia adelante sino también hacia atrás de manera eficiente (esto es, desde un nodo debería poder irse al anterior con el mismo costo que al siguiente). A continuación se recuerdan las primitivas del TDA Lista dadas en clase (con la primitiva MoverCorriente modificada para este ejercicio).

```

/* ls_Crear
Pre: Ls no fue creada.
Post: Ls creada y vacía */
void ls_Crear(TListaSimple *pLs, int TamanoDato);

/* ls_Vaciar
Pre: Ls creada.
Post: Ls vacía.*/
void ls_Vaciar(TListaSimple *pLs);

/* ls_Vacia
Pre: Ls creada.
Post: Si Ls tiene elementos devuelve FALSE sino TRUE.*/
int ls_Vacia(TListaSimple Ls);

/* ls_ElemCorriente
Pre: Ls creada y no vacía.
Post: Se devuelve en E el elemento corriente de la lista.*/
void ls_ElemCorriente(TListaSimple Ls, void* pE);

```

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo		Parcial 11/06/2012
Padrón	Apellido y Nombre	Curso

```

/* ls_ModifCorriente
Pre: Ls creada y no vacía.
Post: El contenido del elemento actual quedo actualizado con E. */
void ls_ModifCorriente(TListaSimple *pLs, void* pE);

/* ls_MoverCorriente
Pre: Ls creada y no vacía.
Post: Si Ls está vacía, devuelve FALSE. Sino:
Si M = LS_PRIMERO, el nuevo elemento corriente es el primero. Devuelve TRUE
Si M = LS_SIGUIENTE, el nuevo elemento corriente es el siguiente al
corriente. Si estaba en el último elemento, devuelve FALSE, sino TRUE.
Si M = LS_ANTERIOR, el nuevo elemento corriente es el anterior al
corriente. Si estaba en el primer elemento, devuelve FALSE, sino TRUE.
*/
int ls_MoverCorriente(TListaSimple *pLs, TMovimiento_Ls M);

/* ls_BorrarCorriente
Pre: Ls creada y no vacía.
Post: Se eliminó el elemento corriente, El nuevo elemento es el siguiente o
el anterior si el corriente era el último elemento.*/
void ls_BorrarCorriente(TListaSimple *pLs);

/* ls_Insertar
Pre: Ls creada.
Post: E se agregó a la lista y es el actual elemento corriente.
Si M=LS_PRIMERO: se insertó como primero de la lista.
Si M=LS_SIGUIENTE: se insertó después del elemento corriente.
Si M=LS_ANTERIOR: se insertó antes del elemento corriente.
Si pudo insertar el elemento devuelve TRUE, sino FALSE.*/
int ls_Insertar(TListaSimple *pLs, TMovimiento_Ls M, void* E);

```

- Indique todos los typedef correspondientes a la nueva versión de la Lista.
- Implemente la nueva versión de la primitiva Insertar que aprovecha la definición modificada en el caso que se indique ANTERIOR.
- Implemente la nueva versión de la primitiva MoverCorriente que aprovecha la definición modificada en el caso que se indique ANTERIOR.

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):

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