

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo			Parcial 09/05/2011
Tema	Página 1	Ejercicio	Puntaje
Apellido y Nombre		1 (2 pts)	
Padrón		2 (3 pts)	
		3 (3 pts)	
Nota Final	Corrigió	4 (2 pts)	

Práctica

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

Se desea implementar un TDA TAlmacen que guarde datos de productos a la venta.

Se pide:

1. Complete la estructura del TDA TAlmacen tal que pueda implementarse el TDA con las primitivas descriptas en el contrato que se muestra. Justifique brevemente sus decisiones.

```
typedef struct {
    int rubro;
    int subrubro;
    int codigo;
    char descripcion[50];
    int precioEnCentavos;
} SProducto;

/* Estructura del TDA TAlmacen
Entre sus campos debe contener los productos, que deberán estar ordenados por la
siguiente clave compuesta: rubro, subrubro, codigo
*/

typedef struct {

    /* A completar por el alumno */

} TAlmacen;

/* Todas las primitivas devuelven 0 en caso de exito y 1 en caso contrario */

/* PRE: almacen no creado; POST: almacen creado */
int TAlmacen_Crear(TAlmacen *almacen);

/* PRE: almacen creado; POST: almacen destruido */
int TAlmacen_Destruir (TAlmacen *almacen);

/* PRE: almacen creado; POST: se ha agregado el producto indicado, si ya se encontraba
devuelve error */
int TAlmacen_AgregarProducto(TAlmacen *almacen, SProducto producto);

/* PRE: almacen creado; POST: se ha modificado el producto del rubro, subrubro, y codigo
indicados pisandolo con el provisto por parametro; si el producto no se encuentra
devuelve error */
int TAlmacen_ModificarProducto(TAlmacen *almacen, int rubro, int subrubro, int codigo,
SProducto producto);

/* PRE: almacen creado; POST: se ha removido el producto del rubro, subrubro y codigo
indicados; si no se encuentra devuelve error */
int TAlmacen_RemoverProducto(TAlmacen *almacen, int rubro, int subrubro, int codigo);

/* PRE: almacen creado, producto tiene suficiente memoria para alojar un SProducto;
POST: se ha devuelto el producto buscado (por rubro, subrubro, código) en la posición
indicada por el puntero a producto, si no se encuentra devuelve error */
int TAlmacen_ConsultarProducto(TAlmacen almacen, int rubro, int subrubro, int codigo,
SProducto* producto);
```

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo		Parcial 25/10/2010
Padrón	Apellido y Nombre	Tema

```

/* PRE: almacen creado, codigodto entre 0 y 4; POST: registra una funcion de descuento
sobre un producto, bajo el codigo de descuento provisto; si ya habia un descuento con
dicho codigo lo reemplaza */
int TAlmacen_RegistrarDescuento(TAlmacen *almacen, int codigodto, void
(*fdescuento)(SProducto *producto));

/* PRE: almacen creado, codigodto se corresponde con un codigo previamente pasado al
llamar a la primitiva TAlmacen_RegistrarDescuento y toma valores entre 0 y 4; POST: se ha
aplicado el descuento de codigo indicado a todos los productos del almacen cuyo (rubro,
subrubro) se encuentren entre (rubroDD, subrubroDD) y (rubroHH, subrubroHH)

Por ejemplo si rubroDD=2, subrubroDD=3, rubroHH=4, subrubroHH=1 el descuento sera
aplicado a productos de (rubro=2, subrubro=3), (rubro=2, subrubro=4), (rubro=3,
subrubro=2), (rubro=4, subrubro=1)
pero no a productos de (rubro=1, subrubro=2), (rubro=2, subrubro=1), (rubro=4,
subrubro=2), (rubro=5, subrubro=1)
*/

int TAlmacen_AplicarDescuento(TAlmacen *almacen, int codigodto, int rubroDD, int
subrubroDD, int rubroHH, int subrubroHH);

```

Nota: tal como se desprende de las pre condiciones, hay sólo 5 códigos de descuento posibles (del 0 al 4).

2. Implemente la primitiva **TAlmacen_AplicarDescuento** del TDA TAlmacen:
3. Se agrega una nueva primitiva para el TDA ListaSimple: **LS_Aplanar**. Esta tiene como funcionalidad convertir una lista de sublistas en una lista con los elementos de las sublistas, pero directamente en el primer nivel. Dada la firma de la primitiva, sus PRE y sus POST, realizar su implementación.

```

/*
* Llena una lista con los elementos de otra lista. La lista de origen es una lista de
listas.
Por ejemplo, si la lista origen tiene 3 elementos que son [ [1 2] [3] [4 5 6] ]
la lista destino quedara con los siguientes 6 elementos [ 1 2 3 4 5 6 ]

PRE: lista destino creada y vacia; lista origen creada, se sabe que el elemento de lista
origen es de tipo TListaSimple; el tamaño del dato de la lista destino es igual al tamaño
del dato de las sublistas de la lista origen

POST: llena lista destino con los elementos de cada una de las sublistas de lista origen
el corriente de la lista destino queda en su primer elemento */

int LS_Aplanar(TListaSimple *destino, TListaSimple origen);

```

Notas:

- **no está permitido en este ejercicio utilizar ninguna primitiva existente**
- **para la implementación debe basarse en la versión de lista simple con punteros explicada en clase**
- **para simplificar el problema, no es necesario verificar el retorno de los malloc**
- **si bien el retorno de la primitiva es int (por extensibilidad futura) no hay situaciones de error, por lo que debe devolver 0 en todos los casos.**

4. Dada la siguiente función en C, indicar qué escribe por pantalla. Explicar el razonamiento realizado para arribar al resultado.

```

1. void seguimiento() {
2.     char* aux1;
3.     int* aux2;
4.     char aux3;

```

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo		Parcial 25/10/2010
Padrón	Apellido y Nombre	Tema

```

5.  char destino[11];
6.  TPila pila;
7.  Tcola cola;

8.  P_Crear(&pila, sizeof(char));
9.  C_Crear(&cola, sizeof(char));

10. char* origen = "ni ser genio";

11. destino[10]='\0';

12. aux3= 'h';
13. P_Agregar(&pila, &aux3);
14. P_Agregar(&pila, origen);
15. P_Agregar(&pila, origen+1);

16. P_Sacar(&pila, &aux3);
17. destino[0] = aux3;

18. P_Sacar(&pila, &aux3);
19. destino[1] = aux3;

20. aux1 = &origen[3];
21. aux2= ((int*)aux1)+1;
22. destino[2]=0;
23. strncat(destino, (char*)aux2, 4);

24. C_Agregar(&cola, &origen[4]);
25. C_Agregar(&cola, &origen[5]);

26. C_Sacar(&cola, &aux3);
27. destino[6] = aux3;

28. C_Sacar(&cola, &aux3);
29. destino[7] = aux3;

30. destino[8] = origen[strlen(origen)-1];

31. memcpy(destino+9, origen+3, 1);

32. P_Vaciar(&pila);
33. C_Vaciar(&cola);

34. printf("Resultado: %s\n", destino);
35. }

```

Nota: considere char de 1 byte y enteros (int) de 4 bytes.

Consideraciones generales:

En todos los casos, 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):

- 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