Fundamendez - 2C 2026 - Práctica semana 10

Memoria dinámica y TDAs

Ejercicio 1

Retomando el ejercicio de la clase anterior…
Para el partido de despedida de Messi en el Monumental, la AFA dispuso que cada miembro de la delegación oficial (jugadores, cuerpo técnico, utileros, etc.) tenga derecho a llevar a su propio grupo de invitados. Para gestionar los accesos, el sistema cuenta con los siguientes structs:

```c
typedef struct invitado {
    char nombre_invitado[MAX_NOMBRE];
    int edad;
    char tipo_acceso; // 'P' (Platea), 'V' (VIP), 'C' (Campo de juego)
} invitado_t;

typedef struct miembro_plantel {
    char nombre_miembro[MAX_NOMBRE];
    char rol; // 'J' (Jugador), 'T' (Cuerpo Técnico), 'S' (Staff/Dirigente)
    invitado_t* invitados;
    int cantidad_invitados;
} miembro_plantel_t;
```

Se quiere crear un programa que use memoria dinámica para organizar la logística de los invitados del plantel. El Chiqui Tapia pidió agregar de último momento al CEO de Sancor Seguros al sector VIP.
Se pide:

- Hacer un procedimiento que reciba el plantel, el nombre de un miembro del mismo y le sume un invitado.

Ejercicio 2

La mochila de Lisa para ir todos los días a la escuela se representa con el siguiente TDA mochila:

```c
#ifndef MOCHILA_H
#define MOCHILA_H

#include <stdbool.h>
```

```c
#define MAX_NOMBRE 50
#define MAX_MATERIAS 10

typedef struct {
   char nombre[MAX_NOMBRE];
   float peso; // En Kilogramos
   char materias[MAX_MATERIAS][MAX_NOMBRE];
   int cantidad_materias;
} objeto_t;

typedef struct mochila mochila_t;

/*
* Post: Crea una mochila vacía.
* Devuelve un puntero a la mochila o NULL si ocurre un error.
*/
mochila_t* mochila_crear(void);

/*
* Pre: La mochila fue creada y no fue destruida.
* Post: Agrega una copia del objeto al inicio de la mochila.
* Devuelve true si pudo agregarlo o false si ocurrió un error.
*/
bool mochila_agregar(mochila_t* mochila, objeto_t objeto);

/*
* Pre: La mochila fue creada y no fue destruida.
* Post: Devuelve la cantidad de objetos guardados.
*/
int mochila_cantidad(mochila_t* mochila);

/*
* Pre: La mochila fue creada y no fue destruida.
* Post: Quita el último objeto agregado de la mochila.
*       Devuelve una copia del objeto quitado.
*/
objeto_t mochila_quitar(mochila_t* mochila);

/*
* Pre: La mochila fue creada y no fue destruida.
* Post: Libera toda la memoria utilizada por la mochila.
*/
void mochila_destruir(mochila_t* mochila);

#endif
```

a. Mañana Lisa no tiene clases de Física ni de Literatura, pero su mochila quedó tan llena de objetos que solo puede sacar el último que puso. Implementar una función que reciba la mochila y elimine todos los objetos que se necesiten exclusivamente para alguna de estas dos materias. Se sabe que no va a haber ningún elemento que se use para ambas materias y ninguna más (es decir, no hay un objeto que se use para dos materias solamente y que esas materias coincidan con las buscadas)
b. Implementar una función que reciba una mochila y un vector de objetos necesarios para Matemática, y agregue aquellos que no estén guardados en la mochila. Se considera que dos objetos son iguales si tienen el mismo nombre.
c. Al llevar tanto peso en la espalda Lisa empezó a sufrir dolores de espalda, es por eso que decidió pasar algunos de sus objetos más pesados al casillero. Implementar una función que pase todos los objetos que pesen más de 0.8 kg al casillero.

Ejercicio 3

Apu estuvo 96 horas seguidas trabajando en el Kwik-E-Mart, por lo que se fue a tomar un merecido descanso, sin embargo, no quiso cerrar la tienda así que le pidió ayuda a Homero. Como Homero nunca trabajó en un mercado es un poco lento para manejar la caja, debido a esto se le acumularon los clientes y están impacientes por ser atendidos para poder volver a sus actividades.
Se tiene el siguiente TDA caja:

```c
#ifndef CAJA_H
#define CAJA_H

#include <stdbool.h>

#define MAX_NOMBRE 100

typedef struct {
    char nombre[MAX_NOMBRE];
    bool escondio_mercaderia;
    int paciencia;
} cliente_t;


typedef struct caja caja_t;

/*
```

```c
Post: Crea una caja vacía. Devuelve un puntero a la caja o NULL en caso
de error.
*/
caja_t* caja_crear();

/*
Pre: `caja` fue creada usando caja_crear().
Post: Agrega un cliente al FINAL de la caja. Devuelve true si se pudo
encolar, false en caso de error.
*/
bool caja_agregar(caja_t* caja, cliente_t cliente);

/*
Pre: `caja` fue creada usando caja_crear().
Post: Atiende al cliente. Saca de la caja al PRIMER cliente y, si
`cliente_atendido` no es NULL, guarda en él una copia del cliente.
Devuelve true si se quitó correctamente, false si la caja está vacía o
hubo error.
*/
bool caja_quitar(caja_t* caja, cliente_t* cliente_atendido);

/*
Pre: `caja` fue creada usando caja_crear().
Post: Resta 1 a la paciencia de todos los clientes que están esperando en la
caja. Si algún cliente llega a paciencia 0, es eliminado de la estructura.
Devuelve la cantidad de clientes que abandonaron la caja en cada llamado.
*/
int caja_actualizar_paciencia(caja_t* caja);

/*
Pre: `caja` fue creada usando caja_crear().
Post: Destruye la caja y libera toda la memoria utilizada por los
clientes que quedaron esperando.
*/
void caja_destruir(caja_t* caja);

#endif /* CAJA_H */
```

a. Antes de comenzar a cobrar, Homero debe verificar que ningún cliente sea un ladrón, para esto Apu le dejó una lista con sospechosos. Hacer una función que recorra a los clientes y elimine de la caja a los que estén en la lista de sospechosos y además estén escondiendo mercadería.

b. Hacer un procedimiento para atender a un cliente, eliminándolo de la caja y actualizando la paciencia del resto de la misma.
c. Hacer un procedimiento que imprima por pantalla el total de clientes que había en la caja, cuántos fueron atendidos, cuántos se fueron por falta de paciencia y cuántos fueron echados por estar robando.