Guía de Ejercicios 9 - Estructuras

```
Advertencia

La resolución conjunta o grupal de los ejercicios aquí presentes no está permitida, excepto en
la medida en que puedas pedir ayuda a tus compañeros de clase y a otras personas, y siempre que
esa ayuda no se reduzca a que otro haga el trabajo por vos.

El código fuente entregado por un estudiante debe ser escrito en su totalidad por dicha persona.
```

Condiciones de entrega:

¿Qué se entrega?	¿Qué no se entrega?
Archivos fuente/source (.c)	Archivos objeto (.o)
Archivos encabezado/header (.h)	Archivos ejecutables (programa, app, a.out, etc.)
Makefile	

Se deben entregar los tres ejercicios en un zip (usar template como ayuda para el formato).

Importante: Recordá también que se evaluarán las buenas prácticas de programación con respecto al pedido de memoria. Por lo tanto, ¡no te olvides de **liberar** la memoria pedida! Recordá verificar el uso de memoria con el programa `valgrind` antes de realizar la entrega.

Tené en cuenta también que para **todos** los ejercicios, se pide también implementar (y entregar) una función main que demuestre el correcto funcionamiento del módulo.

Ejercicio 9.1

Desde la Subsecretaría de Tecnologías de la Información y Comunicación de la facultad nos solicitan implementar un módulo para el SIU Guaraní que permita almacenar y operar sobre las calificaciones finales obtenidas por los estudiantes. Para ello utilizaremos una estructura con los siguientes campos:

- Nombre (hasta 25 caracteres).
- Apellido (hasta 25 caracteres).
- Legajo (valor entero).
- Carrera (valor enumerativo).
- Código de la materia (valor entero).
- Nombre de la materia (hasta 25 caracteres).
- Calificación final (valor entero).
- Fecha de aprobación (en formato Unix epoch, ver `man 2 time`).
- Libro (hasta 10 caracteres).
- Folio (valor entero).

Se nos pide implementar un módulo capaz de agregar, eliminar (por legajo y código de la materia), ordenar (por calificación o por apellido de forma lexicográfica), e imprimir calificaciones. Las calificaciones deberán ser almacenados en un arreglo dinámico. Para ello se deberán implementar las siguientes funciones:

```c
typedef struct Calificacion
{
    /*Definir*/
} Calificacion;

// @brief Agrega una calificacion al vector de calificaciones, reservando la memoria necesaria.
Retorna EXITO o ERROR.
int calificaciones_agregar(/*Definir*/ calificaciones, /*Definir*/ cantidad_calificaciones,
```

```
/*Definir*/ calificacion);

// @brief Elimina una calificacion del vector de calificaciones (por legajo y código de la
materia), liberando la memoria asociada. Retorna EXITO o ERROR.
int calificaciones_eliminar(/*Definir*/ calificaciones, /*Definir*/ cantidad_calificaciones,
/*Definir*/ legajo, /*Definir*/ codigo_materia);

// @brief Imprime el contenido de una calificacion en pantalla con el siguiente formato ejemplo:
//
// Marty McFly - 2014321 - Ing. Electrónica - 950452 - Informática I - R0042 - 101 - 8 (ocho) -
11/02/2013
void calificacion_imprimir(/*Definir*/ calificacion);

// @brief Imprime el contenido del vector de calificaciones (reutilizar función anterior).
void calificaciones_imprimir(/*Definir*/ calificaciones, /*Definir*/ cantidad_calificaciones);

// @brief Ordena las calificaciones del vector por calificación o por apellido de forma
lexicográfica.
void calificaciones_ordenar(/*Definir*/ calificaciones, /*Definir*/ cantidad_calificaciones,
/*Definir*/ criterio_ordenamiento);
```

Importante: ¡No te olvides de implementar un main que demuestre el correcto funcionamiento del módulo!

Ejercicio 9.2

Estamos armando nuestra propia versión de Twitter, y para ello debemos ser capaces de almacenar los Tweets que cada uno de los usuario desee postear. Para ello utilizaremos una estructura con los siguientes campos:

- Nombre (hasta 25 caracteres).
- Usuario (hasta 25 caracteres).
- Fecha de creación (en formato Unix epoch, ver `man 2 time`).
- Mensaje (hasta 140 caracteres).
- Cantidad de likes (valor entero).
- Cantidad de retweets (valor entero).

Se nos pide implementar un módulo capaz de agregar, eliminar (por usuario y fecha de creación), ordenar (por cantidad de likes o retweets), e imprimir Tweets. Los Tweets deberán ser almacenados en un arreglo dinámico. Para ello se deberán implementar las siguientes funciones:

```
typedef struct Tweet
{
    /*Definir*/
} Tweet;

// @brief Agrega un Tweet al vector de Tweets, reservando la memoria necesaria. Retorna EXITO o
ERROR.
int tweets_agregar(/*Definir*/ tweets, /*Definir*/ cantidad_tweets, /*Definir*/ tweet);

// @brief Elimina un Tweet del vector de Tweets (por usuario y fecha de creación), liberando la
memoria asociada. Retorna EXITO o ERROR.
int tweets_eliminar(/*Definir*/ tweets, /*Definir*/ cantidad_tweets, /*Definir*/ usuario,
/*Definir*/ timestamp);

// @brief Imprime el contenido de un Tweet en pantalla con el siguiente formato ejemplo:
//
// 5:33 a.m. - 19 jul. 2021
// Terminator @terminatorok
// ---
// Volveré!
// ---
// Likes: 109
// Retweets: 24
void tweet_imprimir(/*Definir*/ tweet);
```

```
// @brief Imprime el contenido del vector de Tweets (reutilizar función anterior).
void tweets_imprimir(/*Definir*/ tweets, /*Definir*/ cantidad_tweets);

// @brief Ordena los Tweets del vector por cantidad de likes o de retweets.
void tweets_ordenar(/*Definir*/ tweets, /*Definir*/ cantidad_tweets, /*Definir*/
criterio_ordenamiento);
```

Importante: ¡No te olvides de implementar un main que demuestre el correcto funcionamiento del módulo!

Ejercicio 9.3

Para un proyecto de un grupo de investigación de robótica de la facultad se nos pide implementar un módulo que permita almacenar y operar sobre las poses obtenidas de un robot. La pose de un robot nos dice su ubicación, tanto en dos o tres dimensiones y también su orientación. Los mensajes de pose recibidos poseen el siguiente formato:

```
typedef struct Header
{
    char robot[25];
    time_t timestamp;
} Header;

typedef struct Posicion {
    double x, y, z;
} Posicion;

typedef struct Orientacion {
    double yaw, pitch, roll;
} Orientacion;

typedef struct Pose {
    Posicion posicion;
    Orientacion orientacion;
} Pose;

typedef struct RobotPose {
    Header header;
    Pose pose;
} RobotPose
```

Se nos pide implementar un módulo capaz de agregar, eliminar (por robot y timestamp), ordenar (por distancia al origen de coordenadas), e imprimir las poses del robot. Las poses deberán ser almacenadas en un arreglo dinámico. Para ello se deberán implementar las siguientes funciones:

```
// @brief Agrega una pose al vector de poses, reservando la memoria necesaria. Retorna EXITO o
ERROR.
int poses_agregar(/*Definir*/ poses, /*Definir*/ cantidad_poses, /*Definir*/ pose);

// @brief Elimina un pose del vector de poses (por robot y timestamp), liberando la memoria
asociada. Retorna EXITO o ERROR.
int poses_eliminar(/*Definir*/ poses, /*Definir*/ cantidad_poses, /*Definir*/ robot, /*Definir*/
timestamp);

// @brief Imprime la pose del robot en pantalla con el siguiente formato ejemplo:
//
// Robot: WALL-E
// Timestamp: 1630032476
// Posicion:
//   x: 1.5
//   y: 0.2
//   z: 0.9
// Orientacion:
//   yaw: 75
```

```
//   pitch: 32
//   roll: 63
void pose_imprimir(/*Definir*/ pose);

// @brief Imprime el contenido del vector de poses (reutilizar función anterior).
void poses_imprimir(/*Definir*/ poses, /*Definir*/ cantidad_poses);

// @brief Ordena las poses del vector por distancia al origen de coordenadas.
void poses_ordenar(/*Definir*/ poses, /*Definir*/ cantidad_poses);
```

Importante: ¡No te olvides de implementar un main que demuestre el correcto funcionamiento del módulo!

Ejercicio 9.4

Un familiar que posee una inmobiliaria nos pide implementar un módulo que permita almacenar y operar sobre las distintas publicaciones de alquiler y venta de inmuebles. Para ello utilizaremos una estructura con los siguientes campos:

- Operación (Alquiler o Venta).
- Tipo (Casa, Departamento o PH).
- Metros cuadrados (valor entero).
- Cantidad de ambientes (valor entero).
- Precio (valor entero).
- Dirección (hasta 25 caracteres).
- Barrio (hasta 25 caracteres).
- Ciudad (hasta 25 caracteres).
- Anunciante (hasta 25 caracteres).
- Fecha de publicación (en formato Unix epoch, ver `man 2 time`).

Se nos pide implementar un módulo capaz de agregar, eliminar (por anunciante y fecha de publicación), ordenar (por precio o metros cuadrados), e imprimir publicaciones. Las publicaciones deberán ser almacenadas en un arreglo dinámico. Para ello se deberán implementar las siguientes funciones:

```
typedef struct Publicacion
{
    /*Definir*/
} Publicacion;

// @brief Agrega una publicacion al vector de publicaciones, reservando la memoria necesaria.
Retorna EXITO o ERROR.
int publicaciones_agregar(/*Definir*/ publicaciones, /*Definir*/ cantidad_publicaciones,
/*Definir*/ publicacion);

// @brief Elimina una publicacion del vector de publicaciones (por anunciante y fecha de
publicación), liberando la memoria asociada. Retorna EXITO o ERROR.
int publicaciones_eliminar(/*Definir*/ publicaciones, /*Definir*/ cantidad_publicaciones,
/*Definir*/ anunciante, /*Definir*/ timestamp);

// @brief Imprime el contenido de una publicacion en pantalla con el siguiente formato ejemplo:
//
// Departamento - 64m2 - 4 ambientes
// Venta - USD 155000
// Maza al 100, Almagro, Capital Federal
// ---
// Lepore Propiedades - Publicado hace 259 días.
void publicacion_imprimir(/*Definir*/ publicacion);

// @brief Imprime el contenido del vector de publicaciones (reutilizar función anterior).
void publicaciones_imprimir(/*Definir*/ publicaciones, /*Definir*/ cantidad_publicaciones);

// @brief Ordena las publicaciones del vector por precio o metros cuadrados.
void publicaciones_ordenar(/*Definir*/ publicaciones, /*Definir*/ cantidad_publicaciones,
/*Definir*/ criterio_ordenamiento);
```

Importante: ¡No te olvides de implementar un main que demuestre el correcto funcionamiento del módulo!

Ejercicio 9.5

Un banco nos pide implementar un módulo que permita almacenar y operar sobre las distintas cuentas bancarias de un grupo de clientes. Para ello utilizaremos una estructura con los siguientes campos:

- Nombre (hasta 25 caracteres).
- Usuario (hasta 25 caracteres).
- Contraseña (hasta 25 caracteres).
- DNI (valor entero).
- Fecha de última actividad (en formato Unix epoch, ver `man 2 time`).
- Dinero en cuenta (valor real).

Se nos pide implementar un módulo capaz de agregar, eliminar (por usuario), realizar depósitos o extracciones de dinero, y mostrar el estado de cuenta. Las cuentas deberán ser almacenadas en un arreglo dinámico. Para ello se deberán implementar las siguientes funciones:

```
typedef struct Cuenta
{
    /*Definir*/
} Cuenta;

// @brief Agrega una cuenta al vector de cuentas, reservando la memoria necesaria. Retorna EXITO
o ERROR.
int cuentas_agregar(/*Definir*/ cuentas, /*Definir*/ cantidad_cuentas, /*Definir*/ cuenta);

// @brief Elimina una cuenta del vector de cuentas (por usuario), liberando la memoria asociada.
Retorna EXITO o ERROR.
int cuentas_eliminar(/*Definir*/ cuentas, /*Definir*/ cantidad_cuentas, /*Definir*/ usuario);

// @brief Realiza un depósito a una cuenta del vector de cuentas (por usuario). Retorna EXITO o
ERROR.
int cuentas_deposito(/*Definir*/ cuentas, /*Definir*/ cantidad_cuentas, /*Definir*/ usuario,
/*Definir*/ monto);

// @brief Realiza una extracción de una cuenta del vector de cuentas (por usuario). Retorna
EXITO o ERROR.
// Nota: No se admiten valores negativos en las cuentas.
int cuentas_extraccion(/*Definir*/ cuentas, /*Definir*/ cantidad_cuentas, /*Definir*/ usuario,
/*Definir*/ monto);

// @brief Imprime el estado de una cuenta en pantalla con el siguiente formato ejemplo:
//
// Banco Gringotts
// Nymphadora Tonks - @tonks
// DNI: 123456789
//
// Saldo: 654321
// Última actividad: 14:12 - 10/04/98
void cuenta_imprimir(/*Definir*/ cuenta);

// @brief Imprime el contenido del vector de cuentas (reutilizar función anterior).
void cuentas_imprimir(/*Definir*/ cuentas, /*Definir*/ cantidad_cuentas);
```

Importante: ¡No te olvides de implementar un main que demuestre el correcto funcionamiento del módulo!

Ejercicio 9.6

Un amigo dueño de una concesionaria de autos nos pide implementar un módulo que permita almacenar y operar sobre las distintas operaciones de venta de automóviles. Para ello utilizaremos una estructura con los siguientes campos:

- Marca (hasta 25 caracteres).

- Modelo (hasta 25 caracteres).
- Año de fabricación (valor entero).
- Cilindrada (valor real).
- Color (valor enumerativo).
- Patente (hasta 7 caracteres).
- Precio de venta (valor entero).
- Kilometraje (valor entero).

Se nos pide implementar un módulo capaz de agregar, eliminar (por patente), ordenar (por marca **y** modelo), e imprimir los automóviles. Los automóviles deberán ser almacenados en un arreglo dinámico. Para ello se deberán implementar las siguientes funciones:

```
typedef struct Automovil
{
    /*Definir*/
} Automovil;

// @brief Agrega una automóvil al vector de automóviles, reservando la memoria necesaria.
Retorna EXITO o ERROR.
int automoviles_agregar(/*Definir*/ automoviles, /*Definir*/ cantidad_automoviles, /*Definir*/
automovil);

// @brief Elimina un automóvil del vector de automóviles (por patente), liberando la memoria
asociada. Retorna EXITO o ERROR.
int automoviles_eliminar(/*Definir*/ automoviles, /*Definir*/ cantidad_automoviles, /*Definir*/
patente);

// @brief Imprime los detalles de un automóvil en pantalla con el siguiente formato ejemplo:
//
// Renault Clio RT - 1.4
// 1994 | 144000 km
//
// $ 330000
// Rojo - RST 123
void automovil_imprimir(/*Definir*/ automovil);

// @brief Imprime el contenido del vector de automóviles (reutilizar función anterior).
void automoviles_imprimir(/*Definir*/ automoviles, /*Definir*/ cantidad_automoviles);

// @brief Ordena los automóviles del vector por marca y modelo (es decir, deben ordenarse
lexicográficamente por marca primero y luego dentro de cada marca, deben ordenarse a su vez por
modelo).
void automoviles_ordenar(/*Definir*/ automoviles, /*Definir*/ cantidad_automoviles);
```

Importante: ¡No te olvides de implementar un main que demuestre el correcto funcionamiento del módulo!

Referencias

Algunos ejercicios fueron obtenidos y adaptados de:

- Guía de Trabajos Prácticos 2011 - Informática I - Departamento de Electrónica - UTN FRBA
- Guía de Ejercicios - Algoritmos y Programación I - UBA FIUBA