Examen final 2021-02-01

95.11/75.02 - Algoritmos y Programación I - Curso Essaya

Objetivo

Implementar `actordb`, una base de datos de actores y actrices de cine y TV.

Se dispone de los siguientes archivos:

- Archivos provistos con código:
  - `actordb.h`: Declaraciones de tipos y funciones a implementar
  - `pruebas.c`: función `main` + pruebas
  - `Makefile`
- Archivos a completar con código:
  - `actordb.c`: Implementación

La idea es completar el archivo `actordb.c` de forma tal que el programa funcione.

Al compilar con `make`, se genera el archivo ejecutable `actordb_pruebas`. El mismo, al ejecutarlo, efectúa una serie de pruebas para verificar el correcto funcionamiento de `actordb.c`.

Las pruebas están divididas en 5 ejercicios. Es condición necesaria (pero no suficiente) para aprobar el examen que haya 3 ejercicios OK.

Configuración de los ejercicios

Los ejercicios no están todos habilitados por defecto (para que se pueda compilar el programa sin haber implementado todas las funciones). Para habilitar la compilación de un ejercicio, por ejemplo el ejercicio 1, descomentar en `pruebas.c` la línea que dice:

```
//#define EJERCICIO1_HABILITADO
```

es decir, que quede de la siguiente manera:

```
#define EJERCICIO1_HABILITADO
```

y volver a compilar el programa.

Salida del programa

Al ejecutar el programa (`./actordb_pruebas`), se ejecutan todos los ejercicios habilitados, y se imprime el resultado de cada uno (OK o FAIL), junto con la cantidad de ejercicios OK. Ejemplo:

```
$ ./actordb_pruebas
ejercicio 1: OK
ejercicio 2: OK
ejercicio 3: OK
Prueba fallida en pruebas.c:80: actores[i] != NULL
ejercicio 4: FAIL
```

```
ejercicio 5: OK
Cantidad de ejercicios OK: 4
```

Recordar: es condición necesaria[1] (pero no suficiente[2]) para aprobar el examen que haya al menos 3 ejercicios OK.

Recordar: Correr el programa con `valgrind --leak-check=full` para mayor seguridad de que la implementación es correcta.

TDAs

TDA actor_t

El TDA `actor_t` representa un actor en la base de datos. Un actor tiene:

- un **nombre** (máximo `NOMBRE_MAX` caracteres)
- una **fecha de nacimiento** (tipo `fecha_t`, explicado a continuación)

Las instancias de `actor_t` se operan con memoria dinámica, es decir que habrá una función para crear un actor y otra para destruirlo.

TDA fecha_t

El TDA `fecha_t` representa una fecha (año/día/mes). La estructura está declarada y definida en `actordb.h`.

Las instancias de `fecha_t` se operan con memoria estática.

Descripción de los ejercicios

EJERCICIO 1 (TDA `actor_t`) Funciones a implementar:

- `actor_crear`: Crea un actor con el nombre y fecha de nacimiento indicados.
- `actor_nombre`: Devuelve el nombre del actor. (Nota: el puntero que devuelve apunta a la cadena almacenada internamente en la estructura; es decir, esta función no devuelve una copia de la cadena.)
- `actor_nacimiento`: Devuelve la fecha de nacimiento del actor.
- `actor_destruir`: Libera la memoria asociada con la instancia del TDA.

EJERCICIO 2 (archivos) Funciones a implementar:

- `actor_escribir`: Recibe la ruta de un archivo, y escribe en el mismo los atributos del actor.
- `actor_leer`: Recibe la ruta de un archivo, y devuelve una instancia de `actor_t` con la información almacenada en el archivo, asumiendo que tiene el mismo formato que el utilizado en `actor_escribir`.

EJERCICIO 3 (ordenamiento) Funciones a implementar:

- `fecha_comparar`: Compara dos fechas `a` y `b`, devolviendo un número negativo, cero o positivo si `a` es anterior, igual o posterior a `b`, respectivamente.
- `actores_ordenar_por_fecha_nacimiento`: Recibe un vector de actores y ordena el mismo según la fecha de nacimiento, de menor a mayor (es decir, los actores más viejos primero).

  Nota: Para realizar el ordenamiento se puede implementar cualquiera de los algoritmos vistos en clase, o bien utilizar la función `qsort` de la biblioteca estándar.

[1] Es posible que un ejercicio sea considerado "bien" aun cuando la prueba informa "FAIL"; por ejemplo si hay un error trivial en el código que se arreglaría haciendo un pequeño cambio.

[2] Es posible (pero poco probable) que un ejercicio sea considerado "mal" aun cuando la prueba informa "OK"; por ejemplo si hay errores conceptuales.

EJERCICIO 4 (búsqueda binaria) Funciones a implementar:

- `actores_buscar`: Recibe un vector de actores ordenado por nombre, y un nombre a buscar. Devuelve un puntero al actor correspondiente, o `NULL` en caso de que el mismo no esté presente en el vector.

  Nota: Esta prueba asume que `actores_buscar` realiza **búsqueda binaria**. Si se hace búsqueda lineal es posible que el ejercicio dé `OK` pero aun así no se considerará resuelto correctamente.

  Nota: Para realizar la búsqueda binaria se puede implementar el algoritmo a mano, o bien utilizar la función `bsearch` de la biblioteca estándar.

EJERCICIO 5 (bits) Tenemos que importar y exportar datos de una base de datos externa, en la cual las fechas se representan en un formato antiguo llamado BCD (Binary Coded Decimal).

En este formato, se usan 4 bits para representar cada uno de los dígitos **decimales** de la fecha. Como el año tiene a lo sumo 4 dígitos decimales, se usan 16 bits para el año (4 bits para las unidades, 4 para las decenas, etc). El mes tiene a lo sumo 2 dígitos, por lo tanto se representa con 8 bits; y lo mismo para el día. Luego se empaquetan todos los dígitos en un registro de 32 bits:

```
año         mes     dia
+---------------+-------+-------+
| M | C | D | U | D | U | D | U |
+---------------+-------+-------+
31           16 15    8 7     0

M = Millares
C = Centenas
D = Decenas
U = Unidades
```

Por ejemplo, si la fecha es, 1972/8/26:

```
año         mes     dia
+---------------+-------+-------+
| 1 | 9 | 7 | 2 | 0 | 8 | 2 | 6 |
+---------------+-------+-------+
31           16 15    8 7     0
```

Funciones a implementar:

- `fecha_a_bcd`: Recibe una fecha y devuelve la representación en el formato BCD.
- `bcd_a_fecha`: Recibe la representación en el formato BCD y devuelve la fecha correspondiente.