Examen final 2020-10-03

95.11/75.02 - Algoritmos y Programación I - Curso Essaya

Objetivo

Implementar `aerodb`, una base de datos de aeropuertos.

Se dispone de los siguientes archivos:

- Archivos provistos con código:
  - `aerodb.h`: Funciones `aerodb_*` (header)
  - `pruebas.c`: función `main` + pruebas
  - `Makefile`
- Archivos a completar con código:
  - `aerodb.c`: Funciones `aerodb_*` (implementación)

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

Al compilar con `make`, se genera el archivo ejecutable `aerodb_pruebas`. El mismo, al ejecutarlo, efectúa una serie de pruebas para verificar el correcto funcionamiento de `aerodb.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 (`./aerodb_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:

```
$ ./aerodb_pruebas
ejercicio 1: OK
ejercicio 2: OK
ejercicio 3: OK
Prueba fallida en pruebas.c:122: aerodb_cantidad(a) == 11
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.

Aerodb

Aerodb es un TDA que permite almacenar información acerca de todos los aeropuertos del mundo.

De cada aeropuerto queremos almacenar:

- La **designación**[3], que es una cadena de 3 caracteres que se utiliza para identificar unívocamente al aeropuerto. Por ejemplo, la designación del aeropuerto de Ezeiza es `"EZE"`.
- El **nombre**, que es una cadena de no más de 63 caracteres.
- La **ciudad** y el **país**, que son cadenas de no más de 63 caracteres.
- La **latitud** y **longitud** en la que se encuentra el aeropuerto, ambos valores de tipo `double`.

Descripción de los ejercicios

EJERCICIO 1 (funciones básicas) Funciones a implementar:

- `aerodb_crear`: Crea una instancia de Aerodb vacía (es decir, sin nungún aeropuerto).
- `aerodb_agregar`: Agrega un aeropuerto a la base de datos.
- `aerodb_cantidad`: Devuelve la cantidad de aeropuertos que hay en la base de datos.
- `aerodb_nombre`: Recibe una designación, y devuelve el nombre del aeropuerto correspondiente, o `NULL` si la designación no corresponde a ningún aeropuerto.
- `aerodb_ciudad`: Recibe una designación, y devuelve la ciudad del aeropuerto correspondiente, o `NULL` si la designación no corresponde a ningún aeropuerto.
- `aerodb_pais`: Recibe una designación, y devuelve el país del aeropuerto correspondiente, o `NULL` si la designación no corresponde a ningún aeropuerto.
- `aerodb_lat`: Recibe una designación, y devuelve la latitud del aeropuerto correspondiente, o `0` si la designación no corresponde a ningún aeropuerto.
- `aerodb_lon`: Recibe una designación, y devuelve la longitud del aeropuerto correspondiente, o `0` si la designación no corresponde a ningún aeropuerto.
- `aerodb_destruir`: Libera la memoria asociada con la instancia de Aerodb.

EJERCICIO 2 (archivos) Funciones a implementar:

- `aerodb_escribir`: Recibe la ruta de un archivo, y escribe en el mismo la información completa almacenada en la base de datos.
- `aerodb_leer`: Recibe la ruta de un archivo, y devuelve una instancia de `aerodb` con la información almacenada en el archivo, asumiendo que tiene el mismo formato que el utilizado en `aerodb_escribir`.

EJERCICIO 3 (búsqueda binaria) Funciones a implementar:

- `aerodb_buscar`: Recibe una designación, y devuelve toda la información relacionada con el aeropuerto correspondiente.

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

[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.

[3] Técnicamente se llama código IATA.

Nota: Para realizar la búsqueda binaria probablemente sea necesario modificar `aerodb_agregar`.

EJERCICIO 4 (distancia mínima) Funciones a implementar:

- `aerodb_mas_cercano`: Recibe una latitud y una longitud, y devuelve el aeropuerto más cercano a las coordenadas indicadas.

Nota: el cálculo de distancias con coordenadas geodésicas no es trivial; a los efectos de este ejercicio vamos a simplificar y calcularlas como si fueran coordenadas cartesianas en un plano. Si x es la latitud e y es la longitud:

$$d^2(P_1, P_2) = (x_1 - x_2)^2 + (y_1 - y_2)^2$$

Nota: para calcular la distancia real sería necesario aplicar la raíz cuadrada. A los efectos prácticos de este ejercicio no es necesario hacerlo, ya que solo nos interesa comparar distancias para saber cuál es la menor; no nos interesa saber la distancia exacta.

EJERCICIO 5 (bits) Las coordenadas geodésicas (latitud, longitud) se suelen expresar en notación sexagesimal: grados, minutos, segundos.

Una empresa poco conocida implementó un GPS que, para ahorrar espacio, en lugar de utilizar un `double` para almacenar una coordenada, utiliza un registro de 32 bits con el siguiente formato:

```
+-----------+----+------------+---------+----------+
|           |  N |   grados   | minutos | segundos |
+-----------+----+------------+---------+----------+
 31       21  20  19        12 11      6 5        0
```

- Los bits 0-5 (6 bits en total) corresponden a la cantidad de segundos (entero no negativo).
- Los bits 6-11 (6 bits en total) corresponden a la cantidad de minutos (entero no negativo).
- Los bits 12-19 (8 bits en total) corresponden a la cantidad de grados (entero no negativo).
- El bit 20 es 1 si la coordenada es negativa.
- Los bits 21-31 los podemos ignorar.

Funciones a implementar:

- `decodificar_angulo`: Recibe un valor sexagesimal empaquetado (el registro de 32 bits), y devuelve si la coordenada es negativa y el valor de los grados, minutos y segundos.