Algo1Mendez - Final - 10/09/2020

Ejercicio 1

Versión 1

En el largo viaje custodiando el anillo, Legolas, quien es muy cuidadoso con su largo y lacio cabello, recorrió muchos pueblos y dentro de ellos las barberias locales, donde aprovechó los momentos de ocio para arreglar su hermosa cabellera.

Como registro de su experiencia, creó un **archivo de texto**, **ordenado por nombre del pueblo y luego por nombre de barbería**, en el que puntuó a cada una de las que visitó con un número del 0 al 5 y con un -1 a aquellas que no tuvo placer de visitar.

Cada línea del archivo contiene la siguiente información:

```
pueblo;nombre de la barbería;tipo de corte que se realizó;puntuación
```

Los campos están separados por **;** (punto y coma).

Se pide

1. Listar todas las barberías visitadas por Legolas, agrupadas por pueblo, mostrando al finalizar cada uno, un promedio de la puntuación de las barberías del mismo, como se muestra a continuación:

Ejemplo

```
Gondolin
    Capitán Barba: 4 puntos.
    Pelito pelón: 5 puntos.
    Tres pelos tiene mi barba: 2 puntos.
    ...
Promedio barberías Gondolin: 4 puntos.
-----------------------------------------
Rivendel
    El enano charlatán: 3 puntos.
    Un corte y una quebrada: 5 puntos.
    ...
Promedio barberías Rivendel: 3 puntos.
-----------------------------------------
```

Versión 2

En el largo viaje custodiando el anillo, Gimli, tiene un amor profundo (y correspondido) por la cerveza artesanal, recorrió muchos pueblos y dentro de ellos probó las cervezas de los bares locales.

Como registro de su experiencia, creó un vector, **ordenado por nombre del pueblo y luego por nombre del bar** (no está ordenado por tipo de cerveza), donde guardó, una a una, todas las cervezas que fue tomando.

Cada elemento del vector es del tipo:

```
typedef struct cerveza {
    char pueblo[MAX_PUEBLO];
    char bar[MAX_BAR];
    int id_tipo_cerveza;
} cerveza_t;
```

Se sabe que no hay más de 5 tipos de cerveza en la actualidad, los tipos de cerveza van del 1 al 5.

Se pide

1. Listar la cantidad de cervezas bebidas por Gimli, agrupando por pueblo y luego por bar, mostrando por cada bar, la cantidad de cervezas de cada tipo tomadas en él y al finalizar cada pueblo, la cantidad de cervezas totales tomadas en él, como se muestra a continuación.

Ejemplo

```
Gondolin
    Capitan Cerveza:
        Cervezas tipo 1: 5.
        Cervezas tipo 2: 0.
        Cervezas tipo 3: 10.
        Cervezas tipo 4: 3.
        Cervezas tipo 5: 8.
    Cervelon:
        Cervezas tipo 1: 6.
        Cervezas tipo 2: 8.
        Cervezas tipo 3: 3.
        Cervezas tipo 4: 2.
        Cervezas tipo 5: 9.
    ...
Total cervezas en Gondolin: 121.
-----------------------------------------
Rivendel
    Drusia:
        Cervezas tipo 1: 5.
        Cervezas tipo 2: 0.
        Cervezas tipo 3: 0.
        Cervezas tipo 4: 30.
        Cervezas tipo 5: 4.
    Una pinta y una quebrada:
        Cervezas tipo 1: 12.
        Cervezas tipo 2: 4.
        Cervezas tipo 3: 7.
        Cervezas tipo 4: 11.
        Cervezas tipo 5: 4.
    ...
Total cervezas en Rivendel: 201.
-----------------------------------------
...
```

Ejercicio 2

Versión 1

Gandalf y Saruman fueron los 2 condidatos a quedarse con la presidencia del ministerio de Magia.

Luego de una reñida y siempre polémica votación, quedaron empatados en 35.

Decidieron, sin mucha discusión, que el próximo presidente se determinará tirando los dardos. Para ponerle suspenso, el blanco será especial, los concursantes solo saben a que sección del blanco le pegaron, pero no saben el puntaje que les otorga.

Las tiradas de Gandalf se volcaron a un **archivo binario de acceso secuencial** llamado **gandalf.dat** y los de saruman a un **archivo binario de acceso secuencial** llamado **saruman.dat**.

Ambos archivos están ordenados por número de tiro y están compuestos de registros del tipo **tiro_t**:

```
typedef struct tiro {
    int tiro_numero;
    int id_seccion;
} tiro_t;
```

Por otro lado, se cuenta con un **archivo binario de acceso directo** ordenado por id_seccion, **secciones.dat** que cuenta con los puntajes de las secciones del blanco. En este archivo, las secciones son números enteros correlativos que comienzan desde el 1 y tiene registros del tipo **seccion_t**:

```
typedef struct seccion {
    int id_seccion;
    int puntaje;
} seccion_t;
```

En caso de no haberle acertado al blanco, se guardará **-1** en ese tiro en los archivos de Gandalf y Saruman, y ese tiro no sumará puntos.

Se pide

1. Determinar quien se consagrará presidente del ministerio de magia.

Versión 2

Gandalf y Saruman fueron los 2 condidatos a quedarse con la presidencia del ministerio de Magia.

Luego de una reñida y siempre polémica votación, quedaron empatados en 35.

Decidieron, sin mucha discusión, que el próximo presidente se determinará tirando los dardos. Para ponerle suspenso, el blanco será especial, los concursantes solo saben a que sección del blanco le pegaron, pero no saben el puntaje que les otorga.

Las tiradas de Gandalf se volcaron a una **archivo binario de acceso secuencial** llamado **gandalf.dat** y los de Saruman a un **archivo binario de acceso secuencial** llamado **saruman.dat**.

Ambos archivos están ordenados por número de tiro y están compuestos de registros del tipo **tiro_t**:

```
typedef struct tiro {
    int tiro_numero;
    int id_seccion;
} tiro_t;
```

Por otro lado, se cuenta con un **archivo binario de acceso directo** llamado **secciones.dat** que cuenta con los puntajes de las secciones del blanco. En este archivo, las secciones no están ordenadas, pero se sabe que no son más de 10 son registros del tipo **seccion_t**:

```
typedef struct seccion {
    int id_seccion;
    int puntaje;
} seccion_t;
```

En caso de no haberle acertado al blanco, se guardará -1 en ese tiro en los archivos de Gandalf y Saruman, y ese tiro no sumará puntos.

Se pide

1. Determinar quien se consagrará presidente del ministerio de magia.

Ejercicio 3

Versión 1

Al volver se su laaaaaarga travesía para destruir el anillo, la vida de Sam se volvió un poco aburrida. No es que este disconforme con la tranquilidad que consiguió, solo le gustaría tener algunos hobbies.

Probó zumba, yoga, running, pero nada lo conformaba. Pippin le recomendó que pruebe resolviendo rompecabezas, que lo iban a entretener.

Empezó con uno de elefantes y le encantó el desafío, hoy se encuentra resolviendo uno con casitas de colores y está un poco trabado, porque puso todas las fichas pero como que no le cierra.

El rompecabezas se puede ver como una matriz, donde cada celda es una ficha con la siguiente estructura:

```
typedef struct ficha {
    int id_ficha;
    int id_ficha_derecha;
    int id_ficha_izquierda;
    int id_ficha_arriba;
    int id_ficha_abajo;
} ficha_t;
```

Si una ficha está en uno de los bordes, tendrá un **-1** en los campos correspondientes.

1. Determinar cuántas fichas deberían estar en un borde y no lo están.

Versión 2

Al volver se su laaaaaarga travesía para destruir el anillo, la vida de Sam se volvió un poco aburrida. No es que este disconforme con la tranquilidad que consiguió, solo le gustaría tener algunos hobbies.

Probó zumba, yoga, runnnig, pero nada lo conformaba. Pippin le recomendó que pruebe resolviendo rompecabezas, que lo iban a entretener.

Empezó con uno de elefantes y le encantó el desafío, hoy se encuentra resolviendo uno con casitas de colores y está un poco trabado, porque puso todas las fichas pero como que no le cierra.

El rompecabezas se puede ver como una matriz, donde cada celda es una ficha con la siguiente estructura:

```
typedef struct ficha {
    int id_ficha;
    int id_ficha_derecha;
    int id_ficha_izquierda;
    int id_ficha_arriba;
    int id_ficha_abajo;
} ficha_t;
```

Si una ficha está en uno de los bordes, tendrá un **-1** en los campos correspondientes.

1. Determinar cuántas fichas no tienen ninguna de sus fichas vecinas (derecha, izquierda, arriba y abajo) correctamente puestas.

Ejercicio 4

Versión 1

Sam está con algunas incertidumbres sobre su pasado, es por esto que creó 2 **archivos binarios de acceso secuencial**, uno llamado mama_sam.dat y otro papa_sam.dat, en donde almacenó los antepasados de cada uno de sus padres.

Ambos archivos están ordenados por dni de hobbit y cuentan con registros del tipo **hobbit_t**:

```
typedef struct hobbit {
    char nombre[MAX_NOMBRE];
    char fecha_nacimiento[MAX_FECHA_NACIMIENTO];
    char pareja_de[MAX_NOMBRE];
    int dni;
}hobbit_t;
```

Se pide: 1. Crear un archivo que incluya aquellos parientes, que figuran entre los antepasados de su madre y de su padre. 2. Devolver como retorno de la función la cantidad de parientes, que figuran entre los antepasados de su madre y de su padre, pero que tienen diferente pareja en cada archivo.

Versión 2

Sam está con algunas incertidumbres sobre su pasado, es por esto que creó 2 **archivos binarios de acceso secuencial**, uno llamado mama_sam.dat y otro papa_sam.dat, en donde almacenó los antepasados de cada uno de sus padres.

Ambos archivos están ordenados por dni de hobbit y cuentan con registros del tipo **hobbit_t**:

```
typedef struct hobbit {
    char nombre[MAX_NOMBRE];
    char fecha_nacimiento[MAX_FECHA_NACIMIENTO];
    char pareja_de[MAX_NOMBRE];
    int dni;
}hobbit_t;
```

Se pide: 1. Crear un archivo que incluya aquellos parientes, que son antepasados de su madre y no de su padre o que son antepasados de su padre y no de su madre. 2. Devolver como retorno de la función la cantidad de parientes, que no tienen pareja y que son antepasados de su madre y no de su padre o que son antepasados de su padre y no de su madre.

Ejercicio 5

Versión 1

Determine si las siguientes afirmaciones son verdaderas o falsas, justificando la respuesta con algún ejemplo o comentario. 1. La unión de conjuntos solo puede hacerse entre archivos del mismo tipo. 2. Siempre es mejor trabajar con vectores antes que con archivos. 3. Pasar parámetros por referencia nos permite modificar las variables y conservar los cambios. 4. Siempre que un vector está ordenado es conveniente usar búsqueda binaria.

Versión 2

Determine si las siguientes afirmaciones son verdaderas o falsas, justificando la respuesta con algún ejemplo o comentario. 1. La interserccción de conjuntos puede hacerse entre archivos y vectores. 2. Siempre es mejor trabajar con archivos antes que con vectores. 3. Pasar parámetros por valor nos permite modificar las variables y conservar los cambios. 4. Siempre que un vector está ordenado es conveniente usar búsqueda lineal.