Examen escrito de Programación 1

Escuela de Ingeniería y Arquitectura
Departamento de Informática e Ingeniería de Sistemas

30 de enero de 2020
Adaptado a los cambios hechos en el curso 2020-21

- Se debe disponer sobre la mesa en lugar visible un **documento de identificación** provisto de fotografía.
- Se debe escribir **apellidos y nombre** en cada una de las hojas de papel que haya sobre la mesa.
- Se debe comenzar a resolver cada uno de los problemas del examen **en una hoja diferente** para facilitar su corrección por profesores diferentes.
- Solo debe entregarse a los profesores aquello que deba ser corregido (no deben entregarse borradores ni soluciones en sucio).
- Tiempo máximo para realizar el examen: **3 horas**.
- No está permitido utilizar dispositivos electrónicos de ningún tipo, ni consultar libros ni apuntes, excepto los documentos facilitados por los profesores de la asignatura: *Guía de sintaxis ANSI/ISO estándar C++*, *Resumen de recursos predefinidos en C++ que son utilizados en la asignatura* y *Guía de estilo para programar en C++*.

Introducción

El Instituto Geográfico Nacional (IGN) pone a disposición del público información detallada sobre los últimos terremotos ocurridos en el área de la Península Ibérica e Islas Canarias a través de su página web[1]. Esta información va a ser el punto de partida de los dos programas que cuya implementación se va a pedir en este examen.

[1] `https://www.ign.es/web/ultimos-terremotos`

En ambos se va a trabajar con ficheros de texto con información sobre terremotos procedente del IGN, aunque no exactamente con el mismo formato que el de los ficheros que este ofrece. Los ficheros con los que se va a trabajar en este examen siguen la sintaxis de la regla `<fichero-terremotos>` cuya definición se reproduce a continuación:

```
<fichero-terremotos> ::= <cabecera> fin_de_línea
                         { <terremoto> fin_de_línea }
<cabecera> ::= «Evento;Tiempo;Coordenadas;Profundidad;Magnitud;Provincia»
<terremoto> ::= <cod-terremoto> <delimitador> <fecha-y-hora> <delimitador>
                <coordenadas> <delimitador> <profundidad> <delimitador>
                <magnitud> <delimitador> <cod-provincia>
<cod-terremoto> ::= literal-cadena
<fecha-y-hora> ::= <fecha> <blanco> <hora>
<fecha> ::= literal-entero «-» literal-entero «-» literal-entero
<hora> ::= literal-entero «:» literal-entero
<coordenadas> ::= <latitud> <blanco> <longitud>
<latitud> ::= literal-real
<longitud> ::= literal-real
<profundidad> ::= [literal-entero]
<magnitud> ::= literal-real
<cod-provincia> ::= literal-entero
<delimitador> ::= «;»
<blanco> ::= « »
```

Los datos del fichero están ordenados cronológicamente (de más antiguo a más reciente) y los códigos de los eventos registrados (regla `<cod-terremoto>`) están compuestos exclusivamente de letras minúsculas del alfabeto inglés y dígitos y siguen el orden alfabético. Los datos correspondientes a la regla `<cod-provincia>` son los correspondientes a los códigos del Instituto Nacional de Estadística de las 50 provincias españolas y las dos ciudades autónomas y están comprendidos entre 1 y 52 (más detalles en el enunciado del problema 3).

A modo de ejemplo, se muestra parte del contenido de un fichero que sigue la sintaxis de la regla `<fichero-terremotos>`:

```
Evento;Fecha y hora;Coordenadas;Profundidad;Magnitud;Provincia
es2020axtss;14-1-2020 0:01;36.5194 -5.0705;19;2.3;29
es2020aybae;14-1-2020 3:42;39.8266 -0.5759;4;2.3;12
es2020aybom;14-1-2020 3:58;37.4486 -3.7974;8;2.0;23
es2020ayigq;14-1-2020 7:21;36.6523 -4.602;79;2.2;29
es2020ayjyh;14-1-2020 8:12;36.294 -7.5929;11;2.4;21
es2020aylxg;14-1-2020 9:12;28.3585 -14.5367;8;2.2;35
es2020aypiz;14-1-2020 10:56;38.548 -8.0102;;1.9;6
es2020azhhb;14-1-2020 20:00;28.9706 -14.2202;13;2.3;35
es2020azrec;15-1-2020 0:59;39.3023 -10.312;27;3.5;10
...
```

Para mejorar la legibilidad del ejemplo anterior, se han destacado en negrita los valores de cada línea correspondientes a las reglas `<cod-terremoto>`, `<magnitud>` y `<nombre-provincia>`.

Obsérvese cómo en las reglas sintácticas del fichero se ha indicado que el contenido de la regla `<profundidad>` es optativo y como, efectivamente, en el ejemplo de contenido del fichero aparecen líneas en las que dicha información no aparece.

En este examen se va a solicitar el diseño de dos programas distintos: uno relativo a la gestión del fichero de datos (problemas 1.º y 2.º) y otro relativo a la identificación de las provincias españolas con mayor actividad sísmica (problema 3.º), que es el objetivo principal del problema que se pretende resolver. **Se recomienda la lectura completa de los enunciados de todos los problemas antes de comenzar a resolver cada uno de ellos.**

En el conjunto de los problemas que se presenta a continuación, puede hacerse uso de las siguientes constantes globales, que pueden considerarse definidas en el ámbito de todas y cada una de las funciones solicitadas en los problemas que siguen:

```
const string FICH_PROVINCIAS = "provincias.txt";
const string FICH_TERREMOTOS = "terremotos.csv";
const string FICH_ULTIMOS = "nuevos-terremotos.csv";
const char DELIMITADOR = ';';

const unsigned int RESULTADO_OK = 0;
const unsigned int RESULTADO_ERROR_FICH_TERREMOTOS = 1;
const unsigned int RESULTADO_ERROR_OTRO_FICH = 2;

const unsigned int NUM_PROVINCIAS = 52;
```

Problema 1.º (1 punto)

La función `obtenerUltimoCodigo` que se presenta a continuación está dando problemas. Es sintácticamente correcta, y sin embargo, cuando se ejecuta, en lugar de asignar al parámetro `codEvento` el código del último terremoto del fichero `nombreFicheroTerremotos`, termina asignándole la cadena vacía. Alguien sugirió que el problema estaba en que era necesario leer los códigos de los terremotos del fichero en una variable local distinta al parámetro `codEvento` y, aunque no iba mal encaminado, la función sigue sin funcionar correctamente.

```
/*
 * Pre:  Si el fichero «nombreFicheroTerremotos» existe, es un fichero de
 *       texto cuyo contenido cumple con la regla sintáctica
 *       <fichero-terremotos> y el resto de restricciones establecidas en el
 *       enunciado.
 * Post: Si el fichero «nombreFicheroTerremotos» existe, ha asignado a
 *       «codEvento» el valor del código del terremoto correspondiente a la
 *       última línea del fichero y ha devuelto «true». En caso contrario, ha
 *       devuelto «false».
 */
bool obtenerUltimoCodigo(const string nombreFicheroTerremotos,
                         string& codEvento) {
    ifstream f(nombreFicheroTerremotos);
    if (f.is_open()) {
        string cabecera;
        getline(f, cabecera);

        string codEventoLeido;
        getline(f, codEventoLeido, DELIMITADOR);
        while (!f.eof()) {
            string restoLinea;
            getline(f, restoLinea);
            getline(f, codEventoLeido, DELIMITADOR);
        }
        f.close();
        codEvento = codEventoLeido;
        return true;
    }
    else {
        cerr << "No ha podido abrirse el fichero \""
             << nombreFicheroTerremotos << "\"." << endl;
        return false;
    }
}
```

Explica por qué la función `obtenerUltimoCodigo` no funciona correctamente e indica cómo corregir dicho error. La solución más simple al problema consiste en mover una única instrucción de sitio.

Problema 2.º (3 puntos)

Se dispone de un fichero denominado «terremotos.csv» que sigue la sintaxis de la regla <fichero-terremotos>, donde se tiene almacenado un histórico de datos sobre terremotos. Para actualizarlo con nuevos terremotos, el responsable del mantenimiento del fichero descarga periódicamente la información de los terremotos de los últimos diez días publicados en la página web del IGN y modifica ligeramente su estructura para adaptarla a la sintaxis de la regla <fichero-terremotos> y la almacena en un fichero denominado «nuevos-terremotos.csv».

Como el responsable realiza esta operación con una frecuencia aproximada pero no exactamente semanal y la información descargada corresponde a los diez últimos días, suele ser habitual que haya información sobre terremotos replicada en los ficheros «terremotos.csv» y «nuevos-terremotos.csv».

Se pide escribir un **programa completo**, con su función main, que actualice los datos del fichero «terremotos.csv» con los datos de los terremotos del «nuevos-terremotos.csv». Evidentemente, solo se desea que al fichero «terremotos.csv» se le añada la información de los terremotos del fichero «nuevos-terremotos.csv» que no se encontraran ya almacenados en «terremotos.csv».

Por ejemplo, si el contenido del fichero «terremotos.csv» fuese el siguiente:

```
Evento;Fecha y hora;Coordenadas;Profundidad;Magnitud;Provincia
es2020axtss;14-1-2020 0:01;36.5194 -5.0705;19;2.3;29
es2020aybae;14-1-2020 3:42;39.8266 -0.5759;4;2.3;12
es2020aybom;15-1-2020 2:58;37.4486 -3.7974;8;2.0;23
es2020ayigq;15-1-2020 7:21;36.6523 -4.602;79;2.2;29
```

Y el contenido del fichero «nuevos-terremotos.csv» fuese el que sigue:

```
Evento;Fecha y hora;Coordenadas;Profundidad;Magnitud;Provincia
es2020aybom;15-1-2020 2:58;37.4486 -3.7974;8;2.0;23
es2020ayigq;15-1-2020 7:21;36.6523 -4.602;79;2.2;29
es2020ayjyh;16-1-2020 0:12;36.294 -7.5929;11;2.4;21
es2020aylxg;16-1-2020 1:12;28.3585 -14.5367;8;2.2;35
```

Tras la ejecución del programa solicitado, al contenido del fichero «terremotos.csv» se le habrían añadido los datos de los dos terremotos de «nuevos-terremotos.csv» que no estaban contenidos en él inicialmente:

```
Evento;Fecha y hora;Coordenadas;Profundidad;Magnitud;Provincia
es2020axtss;14-1-2020 0:01;36.5194 -5.0705;19;2.3;29
es2020aybae;14-1-2020 3:42;39.8266 -0.5759;4;2.3;12
es2020aybom;15-1-2020 2:58;37.4486 -3.7974;8;2.0;23
es2020ayigq;15-1-2020 7:21;36.6523 -4.602;79;2.2;29
es2020ayjyh;16-1-2020 0:12;36.294 -7.5929;11;2.4;21
es2020aylxg;16-1-2020 1:12;28.3585 -14.5367;8;2.2;35
```

Se recuerda que los ficheros «`terremotos.csv`» y «`nuevos-terremotos.csv`», además de cumplir con la sintaxis de la regla `<fichero-terremotos>`, satisfacen el resto de las restricciones establecidas en la introducción: los datos de ambos ficheros siguen un orden cronológico creciente y los códigos de los eventos registrados están ordenados alfabéticamente.

El programa solicitado también debe actualizar correctamente el fichero incluso en el caso de que no haya eventos repetidos en los ficheros «`terremotos.csv`» y «`nuevos-terremotos.csv`». Si este fuera el caso, sería porque todos los eventos de «`nuevos-terremotos.csv`» son posteriores a cualquier evento (incluido, por tanto, el último) del fichero «`terremotos.csv`».

El programa solicitado no solicita ningún tipo de información (los nombres de los ficheros) al usuario, sino que trabaja directamente con «`terremotos.csv`» y «`nuevos-terremotos.csv`». Si puede abrir correctamente ambos ficheros, informará al usuario del número de terremotos añadidos al fichero «`terremotos.csv`», de acuerdo con el siguiente ejemplo de ejecución:

```
Se han añadido 2 eventos al fichero "terremotos.csv".
```

En caso de que no se haya podido abrir alguno de los ficheros, informará escribiendo un mensaje de error en `cerr`.

Puede hacerse uso de la función `obtenerUltimoCodigo` tanto si se ha identificado el error de dicha función como si no.

Problema 3.º (6 puntos)

En este problema se va a trabajar con el fichero «terremotos.csv» presentado en el problema anterior y con un fichero denominado «provincias.txt» que sigue la sintaxis de la regla <fichero-provincias> que se define a continuación:

```
<fichero-provincias> ::= { <provincia> fin_de_línea }
<provincia> ::= <cod-provincia> <blanco> <nombre-provincia>
<cod-provincia> ::= literal-entero
<blanco> ::= « »
<nombre-provincia> ::= literal-cadena
```

A modo de ejemplo, se muestra parte del contenido del fichero «provincias.txt»:

```
2   Albacete
3   Alicante/Alacant
4   Almería
1   Araba/Álava
33  Asturias
...
38  Santa Cruz de Tenerife
47  Valladolid
49  Zamora
50  Zaragoza
51  Ceuta
52  Melilla
```

En el fichero aparecen los nombres oficiales de las 50 provincias españolas y las dos ciudades autónomas, precedidas por el código numérico asignado por el Instituto Nacional de Estadística. Dichos códigos están comprendidos entre el 1 y el 52 y, evidentemente, no se repiten. No se puede suponer nada acerca del orden en el que aparecen las provincias en el fichero.

Se pide escribir un **programa completo** que, a partir de los datos de los ficheros «terremotos.csv» y «provincias.txt», escriba un listado en la pantalla en el que aparezcan los nombres de las provincias españolas en las que haya registrado al menos un terremoto en el fichero «terremotos.csv». Se debe indicar el número de terremotos por provincia que aparecen en el fichero y la magnitud media de los mismos. El listado debe aparecer ordenado por número de terremotos de forma descendente, de acuerdo con el siguiente formato:

```
    Provincia     Nº terremotos  Magnitud media
================================================
 1. Huelva                   18           2.49
 2. Cádiz                    12           2.33
 3. Granada                   7           2.16
 4. Málaga                    7           1.97
 5. Valencia/València         7           2.19
 6. Badajoz                   5           2.12
 7. Castellón/Castelló        5           2.36
 8. Girona                    5           1.74
 9. Santa Cruz de Tenerife    5           1.78
10. Melilla                   5           2.64
11. Balears, Illes            4           3.40
12. Huesca                    3           2.13
13. Palmas, Las               3           2.17
14. Barcelona                 2           1.80
15. Jaén                      2           2.05
16. Murcia                    2           1.75
17. Ciudad Real               2           1.85
18. Sevilla                   2           2.15
19. Albacete                  1           2.40
20. Coruña, A                 1           2.60
21. Almería                   1           2.10
22. Cuenca                    1           2.20
23. Ourense                   1           1.90
24. Cáceres                   1           3.50
```

Como en el caso del problema anterior, el programa solicitado en este caso no es interactivo y no tiene que solicitar los nombres de los fichero, sino trabajar directamente con los ficheros «`terremotos.csv`» y «`provincias.txt`».

En caso de que no se haya podido abrir alguno de los ficheros, informará escribiendo un mensaje de error en `cerr`.

Debe plantearse un diseño modular, en el que la función `main` correspondiente a este programa se apoye en otras para realizar su labor. Todas las funciones utilizadas, incluida la función `main`, deben estár adecuadamente especificadas a través de una precondición y postcondición. Debe escribirse todo el código necesario para la implementación de problema propuesto. La única excepción consiste en que puede suponerse que está disponible una función de ordenación que podrá utilizarse sin necesidad de escribir su código, aunque se deberá escribir su cabecera y especificación.

Notas

Declaraciones y definiciones comunes a los problemas 2.º y 3.º

No es necesario declarar elementos o definir funciones comunes a los problemas 2.º y 3.º para reutilizarlos en estos. No obstante, si optaras por ello, hazlo en hojas distintas a las de los citados problemas y entrégalas por separado.

Anexo al documento de recursos predefinidos en C++ usados en la asignatura

En la asignatura se han visto las siguientes constantes del espacio de nombres `std::ios` que pueden utilizarse como segundo parámetro del método `open` de las clases `ifstream`, `ofstream` y `fstream` o de sus constructores:

ios::binary Las operaciones de lectura o escritura se realizan en modo binario (sin realizar ningún tipo de conversión de los *bytes* leídos o escritos) en lugar de en modo texto (donde puede realizarse algún tipo de conversión en función del *byte* leído o escrito y el valor del mismo como carácter, en particular en el caso del carácter `'\n'`).

ios::app Las operaciones de escritura se inician a partir del final del fichero, preservando el contenido previo que el fichero pudiera tener.

ios::in Indica de forma explícita que se abre un flujo para leer el contenido del fichero asociado al mismo. Este modo está implícito en el caso de los flujos de la clase `ifstream`.

ios::out Indica de forma explícita que se abre un flujo para escribir datos en el fichero asociado al mismo. Este modo está implícito en el caso de los flujos de la clase `ofstream`.

Solución al problema 1.º

La función `obtenerUltimoCodigo` no funciona correctamente debido a que hace una copia del último código leído del flujo `f` demasiado tarde, cuando ya se ha intentado leer la cadena de caracteres correspondiente al código de un evento de una línea que no existe. Ese último intento de lectura es necesario para que el método `eof` del flujo `f` comience a devolver **`true`**, pero hace que a `codEventoLeido` se le asigne la cadena vacía (`""`).

La solución consiste en mover la instrucción de copia de cadenas de la línea 26 al interior del bucle, cuando, tras el intento de lectura de un nuevo código de terremoto, se tiene la certeza de que la lectura se efectuó correctamente porque la incocación al método `eof` del flujo `f` ha devuelto **`false`** (es decir, antes de cualquiera de las líneas 21, 22 o 23, precediendo al intento de lectura de un nuevo código dentro del bucle).

El código correcto se muestra a continuación:

```
/*
 * Especificación en el enunciado
 */
bool obtenerUltimoCodigo(const string nombreFicheroTerremotos, string& codEvento) {
    ifstream f(nombreFicheroTerremotos);
    if (f.is_open()) {
        string cabecera;
        getline(f, cabecera);

        string codEventoLeido;
        getline(f, codEventoLeido, DELIMITADOR);
        while (!f.eof()) {
            codEvento = codEventoLeido;
            string restoLinea;
            getline(f, restoLinea);
            getline(f, codEventoLeido, DELIMITADOR);
        }
        f.close();
        return true;
    }
    else {
        cerr << "No␣ha␣podido␣abrirse␣el␣fichero␣\"" << nombreFicheroTerremotos
             << "\"." << endl;
        return false;
    }
}
```

Solución al problema 2.º

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

/* Declaraciones de las constantes definidas en el enunciado */

/* Declaración y definición de la función «obtenerUltimoCodigo» */


/*
 * Pre:  Los flujos «terremotos» y «ultimos» están abiertos y asociados con
 *       ficheros de texto cuyos contenidos cumplen con la regla sintáctica
 *       <fichero-terremotos> y el resto de restricciones establecidas en el
 *       enunciado. Se está en disposición de leer desde el principio de ambos
 *       flujos. Además, el primer terremoto del flujo «terremotos» es igual o
 *       posterior al primer evento del flujo «ultimos». Análogamente, el
 *       último terremoto de «ultimos» es también igual o posterior al primer
 *       evento del flujo «terremotos».
 * Post: Tras ser ejecucada, esta función ha actualizado los datos del flujo
 *       «terremotos» con los datos de los terremotos del flujo «ultimos»,
 *       añadiendo únicamente la información de los terremotos del flujo
 *       «ultimos» que no se encontraran inicialmente en «terremotos». Ha
 *       asignado al parámetro «numSismosNuevos» el número de terremotos
 *       añadidos al flujo «ultimos».
 */
void actualizar(ostream& terremotos, istream& ultimos, const string codEvento,
                unsigned int& numSismosNuevos) {
    numSismosNuevos = 0;
    string codEventoLeido;
    getline(ultimos, codEventoLeido, DELIMITADOR);
    while (!ultimos.eof()) {
        string restoLinea;
        getline(ultimos, restoLinea);
        if (codEventoLeido > codEvento) {
            terremotos << codEventoLeido << DELIMITADOR << restoLinea << endl;
            numSismosNuevos++;
        }
        getline(ultimos, codEventoLeido, DELIMITADOR);
    }
}
```

```
/*
 * Pre:  Si los ficheros «nombreFicheroTerremotos» y «nombreFicheroUltimos»
 *       existen, son ficheros de texto cuyos contenidos cumplen con la regla
 *       sintáctica <fichero-terremotos> y el resto de restricciones
 *       establecidas en el enunciado. Además, el primer terremoto de
 *       «nombreFicheroUltimos» es igual o posterior al primer evento de
 *       «nombreFicheroTerremotos». Análogamente, el último terremoto de
 *       «nombreFicheroUltimos» es también igual o posterior al primer
 *       evento de «nombreFicheroTerremotos».
 * Post: Tras ser ejecucada, esta función ha actualizado los datos del fichero
 *       «nombreFicheroTerremotos» con los datos de los terremotos del fichero
 *       «nombreFicheroUltimos», añadiendo únicamente los terremotos de
 *       «nombreFicheroUltimos» que no se encontraran inicialmente en
 *       «nombreFicheroTerremotos». Si ha podido abrir correctamente ambos
 *       ficheros, ha asignado al parámetro «numSismosNuevos» el número de
 *       terremotos añadidos al fichero «nombreFicheroTerremotos» y ha
 *       devuelto «true». En caso contrario, ha escrito un mensaje de error en
 *       «cerr» y ha devuelto «false».
 */
bool actualizar(const string nombreFicheroTerremotos,
                const string nombreFicheroUltimos, const string codEvento,
                unsigned int& numSismosNuevos) {
    ofstream terremotos(nombreFicheroTerremotos, ios::app);
    if (terremotos.is_open()) {
        ifstream ultimos(nombreFicheroUltimos);
        if (ultimos.is_open()) {
            actualizar(terremotos, ultimos, codEvento, numSismosNuevos);
            terremotos.close();
            return true;
        }
        else {
            cerr << "No ha podido abrirse el fichero \"" << nombreFicheroUltimos
                 << "\"." << endl;
            return false;
        }
    }
    else {
        cerr << "No ha podido abrirse el fichero \"" << nombreFicheroTerremotos
             << "\" para la concatenación de datos." << endl;
        return false;
    }
}
```

```
/*
 * Pre:  Si los ficheros «FICH_TERREMOTOS» y «FICH_ULTIMOS» existen, son
 *       ficheros de texto cuyos contenidos cumplen con la regla sintáctica
 *       <fichero-terremotos> y el resto de restricciones establecidas en el
 *       enunciado. Además, el primer terremoto de «FICH_ULTIMOS» es igual
 *       o posterior al primer evento de «FICH_TERREMOTOS». Análogamente, el
 *       último terremoto de «FICH_ULTIMOS» es también igual o posterior al
 *       primer evento de «FICH_TERREMOTOS».
 *
 * Post: Tras ser ejecucado, este programa ha actualizado los datos del
 *       fichero «FICH_TERREMOTOS» con los datos de los terremotos del
 *       fichero «FICH_ULTIMOS», añadiendo únicamente la información de los
 *       terremotos del fichero «FICH_ULTIMOS» que no se encontraran
 *       inicialmente en «FICH_TERREMOTOS». Si ha podido abrir correctamente
 *       ambos ficheros, ha informado al usuario del número de terremotos
 *       añadidos al fichero «FICH_TERREMOTOS» escribiendo un mensaje en la
 *       pantalla. En caso de que no se haya podido abrir alguno de los
 *       ficheros, ha informado escribiendo un mensaje de error en «cerr».
 */
int main() {
    string codEvento;
    if (obtenerUltimoCodigo(FICH_TERREMOTOS, codEvento)) {
        unsigned int numSismosNuevos;
        if (actualizar(FICH_TERREMOTOS, FICH_ULTIMOS, codEvento,
                       numSismosNuevos)) {
            cout << "Se␣han␣añadido␣" << numSismosNuevos
                 << "␣eventos␣al␣fichero␣\"" << FICH_TERREMOTOS << "\"."
                 << endl;
            return RESULTADO_OK;
        }
        else {
            return RESULTADO_ERROR_FICH_TERREMOTOS;
        }
    }
    else {
        return RESULTADO_ERROR_OTRO_FICH;
    }
}
```

Solución al problema 3.º

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;


/* Declaraciones de las constantes definidas en el enunciado */

/*
 * Registro para almacenar la información sísmica de cada una de las
 * provincias según el contenido de los ficheros «terremotos.csv» y
 * «provincias.txt":
 *  - el nombre de la provincia (como una cadena de caracteres de tipo string),
 *  - el número de terremotos registrados en la provincia según el fichero
 *    «terremotos.csv» y
 *  - la suma de sus magnitudes, para luego poder calcular su media.
 *
 * No es necesario almacenar el código de la provincia puesto que en la
 * solución que se plantea a continuación, se va a utilizar un vector de
 * registros de este tipo y los datos de cada provincia se almacenarán en la
 * componente indexada por su código de provincia. Esto será así hasta que se
 * ordene el vector de mayor a menor número de terremotos, momento en el que
 * dejará de ser necesaria la relación entre las provincias y sus códigos.
 */
struct SismicidadProvincial {
    string provincia;
    unsigned int numTerremotos = 0;
    double sumaMagnitudes = 0.0;
};
```

```
/*
 * Pre:  Si el fichero «nombreFicheroProvincias» existe, se trata de un
 *       fichero de texto cuyo contenido cumple con la regla sintáctica
 *       <fichero-provincias> y con el resto de restricciones establecidas en
 *       el enunciado.
 *       El vector «sismicidad» tiene NUM_PROVINCIAS + 1 componentes.
 * Post: Ha asignado, al campo «provincia» de cada una de las componentes
 *       indexadas entre 1 y NUM_PROVINCIAS del vector «sismicidad», ambas
 *       inclusive, el nombre de la provincia cuyo código sirve de índice a la
 *       componente, según los datos del fichero «nombreFicheroProvincias».
 *       Los valores de los otros campos no se han modificado. El valor de
 *       sismicidad[0] no está determinado.
 */
bool inicializarProvincias(const string nombreFicheroProvincias,
                           SismicidadProvincial sismicidad[]) {
    ifstream f(nombreFicheroProvincias);
    if (f.is_open()) {
        unsigned int codProvincia;
        f >> codProvincia;
        while (!f.eof()) {
            f.ignore(); // Separador entre código y nombre de la provincia
            // Lectura del nombre de la provincia a través de getline y no
            // del operado de extracción, debido a que hay provincias cuyo
            // nombre consta de más de una palabra.
            getline(f, sismicidad[codProvincia].provincia);
            f >> codProvincia;
        }
        f.close();
        return true;
    }
    else {
        cerr << "No␣ha␣podido␣abrirse␣el␣fichero␣\""
             << nombreFicheroProvincias << "\"." << endl;
        return false;
    }
}
```

```
/*
 * Pre:  El flujo «f» está abierto y asociado con un fichero de texto cuyo
 *       contenido cumple con la regla sintáctica <fichero-terremotos> y el
 *       resto de restricciones establecidas en el enunciado. Se está en
 *       disposición de leer de «f» desde el principio de una línea distinta a
 *       la de cabecera.
 * Post: Ha extraído una línea completa del flujo «f», asignando a los
 *       parámetros «magnitud» y «codProvincia» los valores correspondientes
 *       leídos de «f».
 */
void leerTerremoto(istream& f, double& magnitud, unsigned int& codProvincia) {
    // También se podría haber optado por
    // repetir la instrucción 4 veces a través de un bucle for.
    string ignorar;
    getline(f, ignorar, DELIMITADOR);    // Código del terremoto
    getline(f, ignorar, DELIMITADOR);    // Fecha y hora
    getline(f, ignorar, DELIMITADOR);    // Coordenadas
    getline(f, ignorar, DELIMITADOR);    // Profundidad
    f >> magnitud;
    f.ignore();                          // DELIMITADOR
    f >> codProvincia;
    getline(f, ignorar);                 // Carácter de fin de línea
}
```

```
/*
 * Pre:  Si el fichero «nombreFicheroTerremotos» existe, se trata de un
 *       fichero de texto cuyo contenido cumple con la regla sintáctica
 *       <fichero-terremotos> y con el resto de restricciones establecidas en
 *       el enunciado.
 *       El vector «sismicidad» tiene NUM_PROVINCIAS + 1 componentes en cada
 *       una de sus componentes indexadas entre 1 y NUM_PROVINCIAS los valores
 *       de los campos «numTerremotos» y «sumaMagnitudes» son iguales a 0.
 * Post: En cada una de las componentes indexadas entre 1 y NUM_PROVINCIAS del
 *       vector «sismicidad», ambas inclusive, el valor del campo
 *       «numTerremotos» es igual al número de terremotos registrados en la
 *       provincia cuyo código sirve de índice a la componente, según los
 *       datos del fichero «nombreFicheroTerremotos»; análogamente, el valor
 *       del campo «sumaMagnitudes» es igual a la suma de las magnitudes de
 *       los terremotos registrados en la provincia cuyo código sirve de
 *       índice a la componente, de nuevo, según los datos del fichero
 *       «nombreFicheroTerremotos». Los valores de los nombres de las
 *       provincias (campo «provincia») no se han modificado.
 *       El valor de sismicidad[0] no está determinado.
 */
bool leerTerremotos(const string nombreFicheroTerremotos,
                    SismicidadProvincial sismicidad[]) {
    ifstream f(nombreFicheroTerremotos);
    if (f.is_open()) {
        // Extracción de la cabecera del fichero
        string cabecera;
        getline(f, cabecera);

        double magnitud;
        unsigned int codProvincia;
        leerTerremoto(f, magnitud, codProvincia);
        while (!f.eof()) {
            sismicidad[codProvincia].numTerremotos++;
            sismicidad[codProvincia].sumaMagnitudes += magnitud;
            leerTerremoto(f, magnitud, codProvincia);
        }
        f.close();
        return true;
    }
    else {
        cerr << "No ha podido abrirse el fichero \""
             << nombreFicheroTerremotos << "\"." << endl;
        return false;
    }
```

```
}

/*
 * Pre:  ---
 * Post: Ha intercambiado los valores iniciales de los parámetros «a» y «b».
 * Nota: Esta función es utilizada por la función como «ordenar» que figura a
 *       continuación y cuyo código no se solicitaba. Por lo tanto, esta
 *       función «intercambiar» podría no formar parte de la solución
 *       proporcionada por los estudiantes a este problema.
 */
void intercambiar(SismicidadProvincial& a, SismicidadProvincial& b) {
    SismicidadProvincial temp = a;
    a = b;
    b = temp;
}

/*
 * Pre:  El vector «sismicidad» tiene NUM_PROVINCIAS + 1 componentes. Las
 *       componentes indexadas entre 1 y NUM_PROVINCIAS, ambas inclusive,
 *       almacenan información sísmica de una provincia diferente. El valor de
 *       sismicidad[0] no está determinado.
 * Post: Los datos de las componentes indexadas entre 1 y NUM_PROVINCIAS del
 *       vector «sismicidad», ambas inclusive, están ordenados de mayor a
 *       menor número de terremotos y son una permutación de las que contenía
 *       el vector «sismicidad» al inicio de la ejecución de esta función.
 * Nota: El enunciado permitía utilizar una función de ordenación como
 *       «ordenar» sin necesidad de escribir el código que sigue. No obstante,
 *       se exigía proporcionar su cabecera y especificación.
 */
void ordenar(SismicidadProvincial sismicidad[]) {
    // Algoritmo de ordenación por selección directa.
    for (unsigned int i = 1; i < NUM_PROVINCIAS; i++) {
        unsigned int iMayor = i;
        for (unsigned int j = i + 1; j <= NUM_PROVINCIAS; j++) {
            if (sismicidad[j].numTerremotos > sismicidad[iMayor].numTerremotos) {
                iMayor = j;
            }
        }
        intercambiar(sismicidad[i], sismicidad[iMayor]);
    }
}
```

```
/*
 * Pre:  El vector «sismicidad» tiene NUM_PROVINCIAS + 1 componentes. Las
 *       componentes indexadas entre 1 y NUM_PROVINCIAS, ambas inclusive,
 *       almacenan información sísmica de una provincia diferente, ordenada de
 *       mayor a menor número de terremotos. El valor de sismicidad[0] no está
 *       determinado.
 * Post: Esta función ha escrito en la pantalla un listado en en el que
 *       aparecen los nombres de las provincias españolas en las que
 *       se haya registrado al menos un terremoto según los datos del vector
 *       «sismicidad». En el listado se ha indicado el número de terremotos
 *       por provincia y la magnitud media estos y se ha escrito ordenado por
 *       número de terremotos de forma descendente, de acuerdo con el
 *       siguiente formato:
 *
 *           Provincia     Nº terremotos  Magnitud media
 *       ================================================
 *        1. Huelva                    18           2.49
 *        2. Cádiz                     12           2.33
 *        3. Granada                    7           2.16
 *       ... ...                      ...            ...
 */
void mostrar(const SismicidadProvincial sismicidad[]) {
    cout << "␣␣␣␣Provincia␣␣␣␣␣Nº␣terremotos␣␣Magnitud␣media" << endl;
    cout << "================================================" << endl;
    cout << fixed << setprecision(2);
    unsigned int i = 1;
    while (i <= NUM_PROVINCIAS && sismicidad[i].numTerremotos > 0) {
        cout << setw(2) << i << ".␣"
             << left << setw(25) << sismicidad[i].provincia
             << right << setw(3) << sismicidad[i].numTerremotos
             << setw(15)
             << sismicidad[i].sumaMagnitudes / sismicidad[i].numTerremotos
             << endl;
        i++;
    }
}
```

```
/* Pre:  Si los ficheros «FICH_TERREMOTOS» y «FICH_PROVINCIAS» existen, son
 *       ficheros de texto cuyos contenidos cumplen, respectivamente, con las
 *       reglas sintácticas <fichero-terremotos> y <fichero-provincias> y con
 *       el resto de restricciones establecidas en el enunciado.
 * Post: Cuando es ejecutado, este programa escribe en la pantalla un listado
 *       en el que aparecen los nombres de las provincias españolas en las que
 *       se haya registrado al menos un terremoto según los datos del fichero
 *       «FICH_TERREMOTOS». En el listado se indica el número de terremotos
 *       por provincia que aparecen en el citado fichero y la magnitud media
 *       de los mismos y aparece ordenado por número de terremotos de forma
 *       descendente, de acuerdo con el siguiente formato:
 *           Provincia     Nº terremotos  Magnitud media
 *       ================================================
 *        1. Huelva                    18          2.49
 *        2. Cádiz                     12          2.33
 *       ... ...                      ...           ...
 *
 *       En caso de que no se haya podido abrir alguno de los
 *       ficheros, ha informado escribiendo un mensaje de error en «cerr».
 */
int main() {
    // Declaración de un vector de 53 componentes para almacenar la información
    // sísmica de cada una de las provincias. La información de cada provincia
    // estará indexada en la componente correspondiente a su código INE. La
    // componente indexada por 0 va a tener un valor no determinado y no va a
    // ser utilizada. Los valores iniciales de los campos «numTerremotos» y
    // «sumaMagnitudes» de cada componente son 0, puesto que así se ha indicado
    // en la definición del tipo registro «SismicidadProvincial».
    SismicidadProvincial sismicidad[NUM_PROVINCIAS + 1];

    if (inicializarProvincias(FICH_PROVINCIAS, sismicidad)) {
        if (leerTerremotos(FICH_TERREMOTOS, sismicidad)) {
            ordenar(sismicidad);
            mostrar(sismicidad);
            return RESULTADO_OK;
        }
        else {
            return RESULTADO_ERROR_FICH_TERREMOTOS;
        }
    }
    else {
        return RESULTADO_ERROR_OTRO_FICH;
    }
}
```