Examen escrito de Programación 1

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

3 de septiembre de 2019

- Tiempo máximo para realizar el examen: **3 horas**.
- Se debe disponer sobre la mesa en lugar visible un **documento de identificación** provisto de fotografía.
- Se debe escribir **nombre y dos apellidos** 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).
- No está permitido utilizar dispositivos electrónicos de ningún tipo, ni consultar libros ni apuntes, excepto los dos 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++*.

El enunciado de este examen se compone de una introducción y de seis problemas relacionados con un problema de tratamiento de información relativo a olas de calor. Se recomienda leer detenidamente tanto la introducción como los enunciados de los seis problemas antes de ponerse a resolver ninguno de ellos. En los enunciados de cada problema concreto se pide diseñar, implementar o analizar una función para solucionar una parte del problema global relativo a olas de calor. Estas funciones pueden utilizarse en cualquiera de las soluciones a los otros problemas. Cada problema concreto será corregido de forma independiente, no influyendo en su calificación el hecho de que funciones solicitadas en otros problemas hayan sido correctamente implementadas o no. En las soluciones que se escriban para cada problema, pueden definirse tantas funciones auxiliares como se estime conveniente.

Introducción

En el contexto del cambio climático, y ante las olas de calor registradas durante este verano, se desea disponer de un programa que determine, a partir de datos de estaciones meteorológicas de la Agencia Estatal de Meteorología (AEMET), episodios de olas de calor.

En el contexto de este examen, vamos a considerar una definición de ola de calor mucho más simple que la utilizada oficialmente por la AEMET. Vamos a utilizar las siguientes definiciones:

Temperatura máxima mensual de una estación: temperatura máxima de un mes determinado registrada por una estación meteorológica concreta.

Por ejemplo, la estación meteorológica del aeropuerto de Zaragoza registró en todo el mes de junio de 2019 una máxima de 43,2 °C (que se alcanzó, concretamente, el día 29).

Temperatura máxima diaria calurosa de una estación: temperatura máxima de un día determinado, registrada por una estación concreta, que supera el 95 % de la máxima mensual de esa misma estación.

Por ejemplo, las temperaturas máximas registradas en el aeropuerto de Zaragoza de los días 28 y 30 de junio (de 42,7° y 42,3°, respectivamente), son *máximas calurosas* de la estación meteorológica del aeropuerto de Zaragoza, puesto que están por encima de 41,04° (95 % de la máxima mensual de 43,2° de esa misma estación).

Día cálido: un día en el que, a nivel estatal, al menos el 10 % de las estaciones meteorológicas consideradas a nivel estatal registran una *máxima calurosa*.

Por ejemplo, el día 28 de junio fue un día cálido, ya que de un total de 792 estaciones meteorológicas, 509 registraron *máximas calurosas*.

Episodio de ola de calor: consiste en al menos tres *días cálidos* consecutivos. O lo que es lo mismo, un conjunto de al menos tres días consecutivos en los que al menos el 10 % de las estaciones meteorológicas registran máximas diarias por encima del 95 % de la temperatura máxima mensual de la estación.

Se dispone de un fichero de texto, denominado «`temperaturas-junio-2019.csv`» (cuyos datos han sido obtenidos y adaptados de la AEMET)y que cumple con la siguiente sintaxis, donde las temperaturas están expresadas en grados centígrados:

```
<fichero-temperaturas> ::= <cabecera> fin_de_línea
                           { <registro-estacion> fin_de_línea }
<cabecera> ::= «Id Estación;Estación;Día;Provincia;Máxima (ºC);Hora máx.;
                Mínima (ºC);Hora mín.;Temperatura media (ºC)»
<registro-estacion> ::= <id-estación> <delimitador> <nombre-estación>
                        <delimitador> <día-del-mes> <delimitador>
                        <nombre-provincia> <delimitador> <temperatura-máxima>
                        <delimitador> <hora-temp-máx> <delimitador>
                        <temperatura-mínima> <delimitador> <hora-temp-mín>
                        <delimitador> <temperatura-media>
<id-estación> ::= literal_entero
<nombre-estación> ::= literal_cadena
<día-del-mes> ::= literal_entero
<nombre-provincia> ::= literal_cadena
<temperatura-máxima> ::= literal_real
<hora-temp-máx> ::= <hora>
<temperatura-mínima> ::= literal_real
<hora-temp-mín> ::= <hora>
<temperatura-media> ::= literal_real
<hora> ::= <dígito> <dígito> «:» <dígito> <dígito>
<dígito> ::= «0» | «1» | «2» | «3» | «4» | «5» | «6» | «7» | «8» | «9»
<delimitador> ::= «;»
```

A modo de ejemplo, se muestra parte del contenido del fichero «`temperaturas-junio-2019.csv`»:

```
Id Estación;Nombre;Día;Prov.;T.Máx.(ºC);H.Máx.;T.Mín.(ºC);H.Mín.;T.Media(ºC)
0;Estaca de Bares;1;A Coruña;23.1;14:00;15.2;23:59;19.2
1;As Pontes;1;A Coruña;31.3;16:00;10.4;7:10;20.8
2;A Coruña;1;A Coruña;26.5;13:30;15.6;23:59;21.1
...
791;Fuencaliente;1;Ciudad Real;32.6;16:40;11.8;7:10;22.2
...
485;Zaragoza Aeropuerto;1;Zaragoza;33.0;18:00;14.9;7:00;23.9
325;Zaragoza, Valdespartera;1;Zaragoza;34.0;19:10;14.7;7:20;24.3
...
0;Estaca de Bares;2;A Coruña;20.0;14:10;13.9;3:40;16.9
1;As Pontes;2;A Coruña;24.8;16:30;9.4;7:30;17.1
2;A Coruña;2;A Coruña;21.2;15:50;14.2;6:50;17.7
...
485;Zaragoza Aeropuerto;30;Zaragoza;42.3;17:40;21.5;7:10;31.9
325;Zaragoza, Valdespartera;30;Zaragoza;42.1;17:40;21.5;8:00;31.8
```

No se puede garantizar que las líneas del fichero estén ordenadas por ningún criterio (ni por identificador de estación ascendente, ni por día del mes ascendente). **Tampoco se puede garantizar que, para cada estación meteorológica, existan datos sobre las temperaturas diarias para todos los días del mes,** puesto que en caso de avería de una estación, esta puede dejar de suministrar datos durante un periodo de varios días. En estos casos, las líneas completas correspondientes a la estación y días en los que no ha estado operativa no se hallan presentes en el fichero.

Se desea obtener un programa que, a partir de los datos del fichero «`temperaturas-junio-2019.csv`», escriba en la pantalla los episodios de ola de calor que se hayan registrado, de acuerdo con el siguiente formato:

```
Episodio de ola de calor detectado del día 26 al 30.
```

Nótese como el programa no solicita el nombre del fichero al usuario, aunque algunas de las funciones que se piden en los problemas siguientes han tenido en cuenta que el nombre del fichero pudiera no ser una constante sino un dato variable que, en una posterior actualización del programa, pudiera ser introducido por el usuario.

Nótese también como la salida del programa ha consistido en una única línea, puesto que en junio de 2019 solo hubo un episodio de ola de calor. Sin embargo, el programa debe estar preparado para informar de ninguno, uno o varios episodios de olas de calor, dependiendo del contenido concreto del fichero con el que trabaje.

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:

```
/*
 * Nombre del fichero con información sobre las temperaturas de un conjunto
 * de estaciones de la AEMET en el mes de junio de 2019, que sigue la sintaxis
 * definida en el enunciado por la regla <fichero-temperaturas>.
 */
const char NOMBRE_FICHERO[] = "temperaturas-junio-2019.csv";

/*
 * Longitud máxima de los nombres de estaciones y provincias que aparecen en
 * el fichero «NOMBRE_FICHERO».
 */
const int MAX_LONG_NOMBRES = 50;

/*
 * Longitud máxima en caracteres de la línea de cabecera del fichero
 * «NOMBRE_FICHERO».
 */
const int MAX_LONG_CABECERA = 150;
```

```
/*
 * Longitud máxima en caracteres de la cadena que representa horas en el
 * el fichero «NOMBRE_FICHERO».
 */
const int MAX_LONG_HORA = 5;

/*
 * Número de estaciones de la AEMET con las que se va a trabajar.
 */
const int NUM_ESTACIONES = 792;

/*
 * Número de componentes del mes de junio (único mes con el que se va a
 * trabajar).
 */
const int NUM_DIAS = 30;

/*
 * Tanto por ciento que permite definir, con respecto a la temperatura máxima
 * mensual de cada estación, la temperatura umbral de esa misma estación para
 * poder considerar que la estación está registrando una máxima «calurosa».
 */
const double PORCENTAJE_TEMPERATURA = 95.0;

/*
 * Tanto por ciento que permite definir el número mínimo de estaciones que
 * tienen que estar registrando una máxima «calurosa» poder considerar dicho
 * día como «cálido» a nivel estatal.
 */
const double PORCENTAJE_ESTACIONES = 10.0;

/*
 * Número mínimo de días «cálidos» consecutivos que tienen que observarse
 * para determinar que está ocurriendo un episodio de ola de calor.
 */
const double MIN_DIAS_OLA_CALOR = 3;

/*
 * Constante física que representa la mínima temperatura posible, expresada en
 * grados centígrados.
 */
const double CERO_ABSOLUTO = -273.15;
```

Problema 1.º (0,5 puntos)

Escribe el código de la función `leerLinea` cuya especificación aparece a continuación. El objetivo de esta función es el de facilitar la lectura de la información relevante al conjunto del problema de cada línea del fichero de temperaturas. Será invocada iterativamente sobre un flujo asociado con el fichero de temperaturas en el problema 2.º.

```
/*
 * Pre:  El flujo «f» está asociado con un fichero de texto que cumple con la
 *       sintaxis de la regla <fichero-temperaturas> definida en el enunciado.
 *       O bien ya no quedan datos que leer del flujo «f» o los datos
 *       pendientes de leer permiten hacer una lectura de una línea completa
 *       que cumple con la sintaxis de la regla <registro-estacion>.
 * Post: El flujo «f» sigue asociado al fichero de texto. Si no quedaban datos
 *       pendientes de leer del fichero, ha devuelto «false». En caso
 *       contrario, ha leído la línea completa a la que se hace referencia en
 *       la precondición, ha asignado a los parámetros «idEstacion»,
 *       «diaDelMes» y «temperaturaMax» los valores correspondientes al
 *       identificador de la estación, el día del mes y la temperatura máxima
 *       registrada leídos de la línea y ha devuelto «true». El flujo ha
 *       quedado en disposición de leer la siguiente línea de texto, si la
 *       hay.
 */
bool leerLinea(ifstream& f, int& idEstacion, int& diaDelMes,
                     double& temperaturaMax);
```

Problema 2.º (1,5 puntos)

Escribe el código de la función `leerTemperaturas` cuya especificación aparece a continuación:

```
/*
 * Pre:  Existe un fichero de nombre «nombreFichero» que cumple con la
 *       sintaxis de la regla <fichero-temperaturas>. La matriz
 *       «maximasDiarias» tiene «NUM_ESTACIONES» filas y NUM_DIAS + 1
 *       columnas.
 * Post: Si no se ha podido leer el fichero «nombreFichero», ha devuelto
 *       «false». En caso contrario, ha devuelto «true» después de haber leído
 *       el contenido del fichero de nombre «nombreFichero» y haber asignado a
 *       las componentes de la matriz «maximasDiarias» los valores de las
 *       temperaturas máximas de cada estación de cada día de acuerdo con
 *       dicho contenido. Como resultado, maximasDiarias[i][j] es igual al
 *       valor de la temperatura máxima registrada por la estación «i» el día
 *       «j», a no ser que no haya ninguna línea en el fichero que proporcione
 *       dicho dato (en cuyo caso, maximasDiarias[i][j] es igual al valor
 *       «CERO_ABSOLUTO»). Para cualquier valor de «i» entre 0 y
 *       NUM_ESTACIONES - 1, el valor maximasDiarias[i][0], que representa la
 *       máxima de la estación «i» en un día del mes que no existe, ha quedado
 *       establecido en «CERO_ABSOLUTO».
 */
bool leerTemperaturas(const char nombreFichero[],
                      double maximasDiarias[][NUM_DIAS + 1]);
```

El objetivo de esta función es el de leer el contenido del fichero de temperaturas disponible y cargar en memoria la información relevante de forma matricial a través del parámetro `maximasDiarias`, cuyo primer índice es el identificador de la estación meteorológica y cuyo segundo índice es el día del mes.

A efectos ilustrativos, se muestra a continuación el contenido de parte de las componentes de la matriz `maximasDiarias`, donde se ha hecho corresponder el primer índice (identificadores de estación) con las filas y, el segundo (día del mes), con las columnas. Para facilitar la interpretación de la misma, las componentes que han recibido su valor de temperatura a partir del extracto del contenido del fichero presentado en la página 3 aparecen en negrita.

	0	1	2	3	4	…	15	16	…	28	29	30
0	-273.15	**23.1**	**20.0**	16.6	18.0	…	18.9	17.8	…	21.3	20.0	20.4
1	-273.15	**31.3**	**24.8**	19.1	17.8	…	15.9	22.8	…	26.7	22.5	18.4
2	-273.15	**26.5**	**21.2**	17.6	19.2	…	18.2	18.4	…	21.5	20.8	17.6
…	…	…	…	…	…	…	…	…	…	…	…	…
325	-273.15	**34.0**	35.0	34.0	33.5	…	27.0	31.4	…	42.9	43.1	**42.1**
…	…	…	…	…	…	…	…	…	…	…	…	…
485	-273.15	**33.0**	34.4	34.1	34.8	…	26.9	31.9	…	42.7	43.2	**42.3**
…	…	…	…	…	…	…	…	…	…	…	…	…
789	-273.15	28.8	30.0	28.9	28.5	…	21.3	-273.15	…	38.4	37.6	37.0
790	-273.15	30.6	-273.15	-273.15	32.5	…	23.5	28.2	…	39.7	41.4	38.8
791	-273.15	**32.6**	33.8	32.9	29.3	…	26.4	28.2	…	39.8	39.7	36.0

Problema 3.º (0,5 puntos)

El **cuerpo** de la función `calcularMaximasMensualesPorEstacion` que se presenta a continuación contiene un **único** error **sintáctico**:

```
/*
 * Pre:  La matriz «maximasDiarias» tiene NUM_ESTACIONES filas y
 *       NUM_DIAS + 1 columnas, que representan las temperaturas máximas
 *       diarias registradas por una determinada estación (primer índice) en
 *       un determinado día (segundo índice). Para cualquier valor de «i»
 *       entre 0 y NUM_ESTACIONES - 1, el valor maximasDiarias[i][0] es
 *       igual a «CERO_ABSOLUTO».
 *       El vector «maximasMensuales» tiene «NUM_ESTACIONES» componentes.
 * Post: A cada componente del vector maximasMensuales[i], donde «i» es el
 *       índice de una estación meteorológica, se le ha asignado la mayor de
 *       las máximas diarias registradas para la estación «i», según el
 *       contenido de la matriz «maximasDiarias».
 */
void calcularMaximasMensualesPorEstacion(
        const double maximasDiarias[][NUM_DIAS + 1],
        double maximasMensuales[]) {
    for (int estacion = 0; estacion < NUM_ESTACIONES; estacion++) {
        maximasMensuales[estacion] = maximasDiarias[estacion][1];
        for (int dia = 2; dia <= NUM_DIAS; dia++) {
            if (maximasDiarias[estacion][dia] > maximasMensuales[estacion]) {
                maximasMensuales[estacion] = maximasDiarias[estacion];
            }
        }
    }
}
```

Identifica el error. Para ello, indica la línea en la que se encuentra, en qué consiste y cómo solucionarlo.

El objetivo de esta función es el de calcular, para cada estación meteorológica, la máxima mensual a partir de las máximas diarias de la propia estación, con el objeto de no tener que calcular dicha máxima más de una vez en otras partes del programa (en concreto, en el código solicitado en el problema 4).

Problema 4.º (4 puntos)

Escribe la especificación (a través de una precondición y una postcondición), la cabecera y el cuerpo de una función denominada `calcularDiasCalidos` que tenga dos parámetros:

1. El primer parámetro es de entrada y consiste en una matriz de datos de tipo **`double`** con dos índices. La matriz tiene `NUM_ESTACIONES` filas y `NUM_DIAS` + 1 columnas, que representan las temperaturas máximas diarias registradas por una determinada estación en un determinado día. No quedan definidos los valores de las componentes de la primera columna (las componentes de segundo índice igual a 0), ya que representarían temperaturas de un día del mes que no existe.
2. El segundo parámetro es de salida y consiste en un vector de datos de tipo **`bool`** de `NUM_DIAS` + 1 componentes.

Cuando la función `calcularDiasCalidos` es ejecutada, asigna a cada componente i del vector entre 1 y `NUM_DIAS`, inclusive, el valor **`true`** si el día i es un *día cálido*, de acuerdo con la definición dada en la introducción de este enunciado, es decir, si al menos un 10 % de las estaciones consideraras han registrado en el día i una temperatura máxima diaria superior al 95 % de la temperatura máxima mensual de la estación correspondiente. En caso contrario, ha asignado **`false`**. El valor de la componente indexada por 0 ha quedado indeterminado.

Tanto al escribir la especificación de la función como su cuerpo, se recomienda sustituir las constantes que representan porcentajes (10 % y 95 %), por los valores de las constantes `PORCENTAJE_ESTACIONES` y `PORCENTAJE_TEMPERATURA`)

Problema 5.º (2 puntos)

Escribe el código de la función `escribirOlasDeCalor` cuya especificación se muestra a continuación:

```
/*
 * Pre:  El vector «diaCalido» tiene NUM_DIAS + 1 componentes. La componente
 *       diaCalido[i] representa, para cualquier «i» entre 1 y NUM_DIAS, si el
 *       día «i» del mes fue un «día cálido» o no. El valor diaCalido[0] no
 *       está determinado.
 * Post: Ha escrito en la pantalla los episodios de ola de calor registrados
 *       en el mes, indicando el día inicial y el final de la ola,
 *       considerando que un episodio de ola calor consiste en tres «días
 *       cálidos» consecutivos o más.
 */
void escribirOlasDeCalor(const bool diaCalido[]);
```

Problema 6.º (1,5 puntos)

Escribe la especificación (a través de una precondición y una postcondición), la cabecera y el cuerpo de la función `main` correspondiente al programa presentado en la introducción de este enunciado.

En la solución presentada se exigirá, además de su corrección y legibilidad, la aplicación de la metodología de diseño descendente y la minimización del esfuerzo de desarrollo por haber hecho uso adecuado de las funciones cuya cabecera y especificación se han mostrado en los problemas anteriores.

Si se estima necesario, se pueden definir también otras funciones auxiliares, cuyo código y especificación deberá proporcionarse. Se puede (y recomienda) utilizar también las constantes simbólicas definidas en la introducción.

Solución al problema 1.º

```
/*
 * Pre:  El flujo «f» está asociado con un fichero de texto que cumple con la
 *       sintaxis de la regla <fichero-temperaturas> definida en el enunciado.
 *       O bien ya no quedan datos que leer del flujo «f» o los datos
 *       pendientes de leer permiten hacer una lectura de una línea completa
 *       que cumple con la sintaxis de la regla <registro-estacion>.
 * Post: El flujo «f» sigue asociado al fichero de texto. Si no quedaban datos
 *       pendientes de leer del fichero, ha devuelto «false». En caso
 *       contrario, ha leído la línea completa a la que se hace referencia en
 *       la precondición, ha asignado a los parámetros «idEstacion»,
 *       «diaDelMes» y «temperaturaMax» los valores correspondientes al
 *       identificador de la estación, el día del mes y la temperatura máxima
 *       registrada leídos de la línea y ha devuelto «true». El flujo ha
 *       quedado en disposición de leer la siguiente línea de texto, si la
 *       hay.
 */

bool leerLinea(ifstream& f, int& idEstacion, int& diaDelMes,
                    double& temperaturaMax) {
    f >> idEstacion;
    if (f.eof()) {
        return false;
    }
    else {
        f.ignore(); // delimitador ';'
        f.ignore(MAX_LONG_NOMBRES, ';'); // Nombre de la estación meteorológica

        f >> diaDelMes;

        f.ignore(); // delimitador ';'
        f.ignore(MAX_LONG_NOMBRES, ';'); // Provincia

        f >> temperaturaMax;
        f.ignore(MAX_LONG_NOMBRES, '\n'); // Lo que quede de línea

        return true;
    }
}
```

Solución al problema 2.º

```
/*
 * Pre:  Existe un fichero de nombre «nombreFichero» que cumple con la
 *       sintaxis de la regla <fichero-temperaturas>. La matriz
 *       «maximasDiarias» tiene «NUM_ESTACIONES» filas y NUM_DIAS + 1
 *       columnas.
 * Post: Si no se ha podido leer el fichero «nombreFichero», ha devuelto
 *       «false». En caso contrario, ha devuelto «true» después de haber leído
 *       el contenido del fichero de nombre «nombreFichero» y haber asignado a
 *       las componentes de la matriz «maximasDiarias» los valores de las
 *       temperaturas máximas de cada estación de cada día de acuerdo con
 *       dicho contenido. Como resultado, maximasDiarias[i][j] es igual al
 *       valor de la temperatura máxima registrada por la estación «i» el día
 *       «j», a no ser que no haya ninguna línea en el fichero que proporcione
 *       dicho dato (en cuyo caso, maximasDiarias[i][j] es igual al valor
 *       «CERO_ABSOLUTO»). Para cualquier valor de «i» entre 0 y
 *       NUM_ESTACIONES - 1, el valor maximasDiarias[i][0], que representa la
 *       máxima de la estación «i» en un día del mes que no existe, ha quedado
 *       establecido en «CERO_ABSOLUTO».
 */
bool leerTemperaturas(const char nombreFichero[],
                      double maximasDiarias[][NUM_DIAS + 1]) {
    ifstream f(nombreFichero);
    if (f.is_open()) {
        // Se ignora la línea de cabecera del fichero.
        f.ignore(MAX_LONG_CABECERA, '\n');

        // Se inicializa toda la matriz con el valor «CERO_ABSOLUTO», de forma
        // que las componentes para las que no haya un valor en el fichero
        // tendrán un valor que no interferirá en cálculos posteriores.
        for (int i = 0; i < NUM_ESTACIONES; i++) {
            for (int j = 0; j <= NUM_DIAS; j++) {
                maximasDiarias[i][j] = CERO_ABSOLUTO;
            }
        }
```

```
        // Lectura de los datos del fichero línea a línea.
        int idEstacion, diaDelMes;
        double temperaturaMax;
        while (leerLinea(f, idEstacion, diaDelMes, temperaturaMax)) {
            // Asignación del valor de temperatura a la componente
            // correspondiente a la estación y día leídas.
            maximasDiarias[idEstacion][diaDelMes] = temperaturaMax;
        }

        f.close();
        return true;
    }
    else {
        cerr << "No␣se␣ha␣podido␣leer␣el␣fichero␣\"" << nombreFichero << '\"'
             << endl;
        return false;
    }
}
```

Solución al problema 3.º

El error se encuentra en la instrucción de asignación de la línea 25 y consiste en que la parte izquierda la asignación de dicha línea es un dato de tipo **`double`** (correspondiente a una componente concreta de un vector de un índice indexado por un entero), mientras que en la parte derecha, la variable `maximasDiarias` es una matriz de dos índices, por lo que la expresión `maximasDiarias[estacion]` no hace referencia a un dato de tipo **`double`**, sino a un vector de `NUM_DIAS` + 1 componentes. Por ello, la asignación es sintácticamente incorrecta. La solución es utilizar dos índices para indexar la matriz `maximasDiarias` y acceder así a una componente de tipo **`double`**:

```
maximasMensuales[estacion] = maximasDiarias[estacion][dia];
```

Solución al problema 4.º

```
/*
 * Pre:  «dia» está comprendido entre 1 y NUM_DIAS, inclusive.
 *       La matriz «maximasDiarias» tiene exactamente NUM_ESTACIONES filas y
 *       NUM_DIAS + 1 columnas, que representan las temperaturas máximas
 *       diarias registradas por una determinada estación (primer índice) en
 *       un determinado día (segundo índice). Para cualquier valor de «i»
 *       entre 0 y NUM_ESTACIONES - 1, el valor maximasDiarias[i][0] queda
 *       indefinido.
 *       El vector «maximasMensuales» tiene exactamente «NUM_ESTACIONES»
 *       componentes, que representan la mayor de la máximas diarias
 *       registrdas por una determinada estación en el mes analizado.
 *       El vector «diaCalido» tiene exactamente NUM_DIAS + 1 componentes.
 * Post: Ha devuelto el número de estaciones que en el día «dia» registraron
 *       una temperatura máxima «calurosa», es decir, superior al
 *       PORCENTAJE_TEMPERATURA % de la temperatura máxima mensual de la
 *       estación.
 */
int contarEstacionesCalurosas(const int dia,
                              const double maximasDiarias[][NUM_DIAS + 1],
                              const double maximasMensuales[]) {
    int numEstacionesCalurosas = 0;
    for (int estacion = 0; estacion < NUM_ESTACIONES; estacion++) {
        if (maximasDiarias[estacion][dia]
                > PORCENTAJE_TEMPERATURA * maximasMensuales[estacion] / 100.0) {
            numEstacionesCalurosas++;
        }
    }
    return numEstacionesCalurosas;
}

/*
 * Pre:  La matriz «maximasDiarias» tiene exactamente NUM_ESTACIONES filas y
 *       NUM_DIAS + 1 columnas, que representan las temperaturas máximas
 *       diarias registradas por una determinada estación (primer índice) en
 *       un determinado día (segundo índice). Para cualquier valor de «i»
 *       entre 0 y NUM_ESTACIONES - 1, el valor maximasDiarias[i][0] queda
 *       indefinido.
 *       El vector «diaCalido» tiene exactamente NUM_DIAS + 1 componentes.
 * Post: A cada componente del vector diaCalido[i], donde «i» es un día del
 *       mes, se le ha asignado «true» si, al menos
 *       PORCENTAJE_ESTACIONES % de las NUM_ESTACIONES han registrado en el
```

```
 *       día «i» una temperatura máxima superior al PORCENTAJE_TEMPERATURA %
 *       de la temperatura máxima mensual de la estación. En caso contrario, ha
 *       asignado «false». El valor diaCalido[0] ha quedado indeterminado.
 */
void calcularDiasCalidos(const double maximasDiarias[][NUM_DIAS + 1],
                         bool diaCalido[]) {
    double maximasMensuales[NUM_ESTACIONES];
    calcularMaximasMensualesPorEstacion(maximasDiarias, maximasMensuales);
    const double MIN_ESTACIONES_CALIDO
            = PORCENTAJE_ESTACIONES * NUM_ESTACIONES / 100.0;
    for (int dia = 1; dia <= NUM_DIAS; dia++) {
        int numEstacionesCalurosas = contarEstacionesCalurosas(dia,
                                              maximasDiarias, maximasMensuales);
        diaCalido[dia] = numEstacionesCalurosas >= MIN_ESTACIONES_CALIDO;
    }
}
```

Solución al problema 5.º

Una posible solución, en la que el vector `diaCalido` se recorre a través de dos bucles anidados, encargándose el bucle interno de procesar episodios completos de días cálidos consecutivos:

```
/*
 * Pre:  diaInicial < diaFinal.
 * Post: Ha escrito en la pantalla un mensaje indicando que se ha detectado
 *       un episodio de ola de calor entre los días «diaInicial» y «diaFinal».
 */
void escribirMensajeOlaCalor(const int diaInicial, const int diaFinal) {
    cout << "Episodio␣de␣ola␣de␣calor␣detectado␣del␣día␣" << diaInicial
         << "␣al␣" << diaFinal << '.' << endl;
}



/*
 * Pre:  El vector «diaCalido» tiene NUM_DIAS + 1 componentes. La componente
 *       diaCalido[i] representa, para cualquier «i» entre 1 y NUM_DIAS, si el
 *       día «i» del mes fue un «día cálido» o no. El valor diaCalido[0] no
 *       está determinado.
 * Post: Ha escrito en la pantalla los episodios de ola de calor registrados
 *       en el mes, indicando el día inicial y el final de la ola,
 *       considerando que un episodio de ola calor consiste en tres «días
 *       cálidos» consecutivos o más.
 */
void escribirOlasDeCalor(const bool diaCalido[]) {
    int dia = 1;
    while(dia <= NUM_DIAS) {
        if (diaCalido[dia]) {
            // Detectado un día cálido. Puede ser el primero de una serie o no
            int diaInicialOla = dia;

            // Se itera sobre los días cálidos consecutivos, teniendo especial
            // cuidado en no iterar más allá del final del vector
            while(dia <= NUM_DIAS && diaCalido[dia]) {
                dia++;
            }
            // Se habandona el bucle porque «dia» no es cálido o porque se ha
            // acabado el mes. En cualquier caso, «dia» - 1 fue el último día
            // cálido de un posible eposodio de ola de calor.

            // Determinación de la existencia de una ola de calor
```

```
            if (dia - diaInicialOla >= MIN_DIAS_OLA_CALOR) {
                escribirMensajeOlaCalor(diaInicialOla, dia - 1);
            }
        }
        dia++;
    }
}
```

Otra solución posible , en la que el vector `diaCalido` se recorre a través de un único bucle que, en cada iteración, procesa únicamente una componente del vector:

```
/*
 * Pre:  El vector «diaCalido» tiene NUM_DIAS + 1 componentes. La componente
 *       diaCalido[i] representa, para cualquier «i» entre 1 y NUM_DIAS, si el
 *       día «i» del mes fue un «día cálido» o no. El valor diaCalido[0] no
 *       está determinado.
 * Post: Ha escrito en la pantalla los episodios de ola de calor registrados
 *       en el mes, indicando el día inicial y el final de la ola,
 *       considerando que un episodio de ola calor consiste en tres «días
 *       cálidos» consecutivos o más.
 */
void escribirOlasDeCalor(const bool diaCalido[]) {
    int diasConsecutivos = 0;
    for (int dia = 1; dia <= NUM_DIAS; dia++) {
        if (diaCalido[dia]) {
            // Detectado un día cálido. Puede ser el primero de una serie o
            // continuar una serie ya iniciada
            diasConsecutivos++;
        }
        else {
            // El día no es cálido. El día anterior podría haber terminado una
            // ola de calor si «diasConsecutivos» es mayor o igual que
            // «MIN_DIAS_OLA_CALOR».
            if (diasConsecutivos >= MIN_DIAS_OLA_CALOR) {
                escribirMensajeOlaCalor(dia - diasConsecutivos + 1, dia - 1);
            }
            diasConsecutivos = 0;
        }
    }
    // El mes ha acabado. El último día del mes podría haber terminado una
    // ola de calor si «diasConsecutivos» es mayor o igual que
    // «MIN_DIAS_OLA_CALOR».
    if (diasConsecutivos >= MIN_DIAS_OLA_CALOR) {
        escribirMensajeOlaCalor(NUM_DIAS - diasConsecutivos + 1, NUM_DIAS);
```

```
    }
}
```

Solución al problema 6.º

```
/*
 * Pre:  Existe un fichero de nombre «nombreFichero» que cumple con la
 *       sintaxis de la regla <fichero-temperaturas> definida en el enunciado.
 * Post: Ha escrito en la pantalla los episodios de ola de calor, definidos de
 *       acurerdo con el enunciado de este examen y determinados por los datos
 *       contenidos en el fichero de nombre «nombreFichero», indicando el día
 *       inicial y el final de la ola, considerando que un episodio de ola
 *       calor consiste en tres días «cálidos» consecutivos o más.
 */
int main() {
    double maximasDiarias[NUM_ESTACIONES][NUM_DIAS + 1];
    if (leerTemperaturas(NOMBRE_FICHERO, maximasDiarias)) {
        bool diaCalido[NUM_DIAS + 1];
        calcularDiasCalidos(maximasDiarias, diaCalido);
        escribirOlasDeCalor(diaCalido);
        return 0;
    }
    else {
        // No se ha podido leer el fichero.
        // Se devuelve un valor de error, disinto a 0.
        return 1;
    }
}
```