Examen escrito de Programación 1

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

7 de septiembre de 2018

- 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.
- Tiempo máximo para realizar el examen: **3 horas**.
- 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++* y *Resumen de recursos predefinidos en C++ que son utilizados en la asignatura.*

Clasificación de una vuelta ciclista por etapas. Introducción

Ficheros utilizados

Para la gestión de la información relativa a los corredores y sus tiempos en una vuelta ciclista por etapas, se van a utilizar dos tipos de ficheros.

Por una parte, se va a trabajar con **un fichero de texto** que contiene la información de los corredores en la vuelta. En particular, cada línea del fichero contiene el número de dorsal del ciclista, sus apellidos, su nombre, el nombre del equipo en el que corre y su nacionalidad. Cada uno de estos datos está separado del resto por comas. Se muestra a continuación la estructura del fichero de texto en notación Backus-Naur:

```
<corredores> ::= { <corredores> fin_de_línea }
<corredor> ::= <dorsal> <sep> <apellidos> <sep> <nombre> <sep> <equipo>
               <sep> <nacionalidad>
<dorsal> ::= literal_entero
<apellidos> ::= literal_cadena
<nombre> ::= literal_cadena
<equipo> ::= literal_cadena
<nacionalidad> ::= literal_cadena
<sep> ::= ‘‘,’’
```

Las cadenas que representan la nacionalidad del ciclista en el fichero utiliza la codificación *ISO 3166-1 alfa-3* y utiliza siempre 3 caracteres alfabéticos en mayúsculas. El orden en el que aparecen los corredores en el fichero no está preestablecido. Se muestra a continuación parte de un fichero que sigue la notación presentada en el cuadro anterior:

```
1,Contador,Alberto,Trek-Segafredo,ESP
2,Theuns,Edward,Trek-Segafredo,BEL
3,Irizar,Markel,Trek-Segafredo,ESP
4,Bernard,Julien,Trek-Segafredo,FRA
...
39,Ventoso,Fran,BMC Racing Team,ESP
...
219,Bol,Jetse,Manzana Postobón Team,NLD
```

Por otra parte, para almacenar los datos relativos a los tiempos que cada corredor ha invertido en cada etapa o en la carrera global, se van a utilizar unos **ficheros binarios** que responden a la siguiente sintaxis:

```
<fichero_marcas> ::= { <marca_corredor> }
<marca_corredor> ::= <dorsal> <número_etapas> <tiempo>
                     <puntos_regularidad> <puntos_montaña>
<dorsal> ::= int
<número_etapas> ::= int
<tiempo> ::= int
<puntos_regularidad> ::= int
<puntos_montaña> ::= int
```

El valor del dato `<dorsal>` identifica al corredor que ha obtenido la marca (tiempo y puntos de montaña y regularidad). El dato **`int`** `<tiempo>` está expresado en segundos. Cuando los ficheros binarios que se corresponden con la sintaxis anterior se utilizan para almacenar los datos de una etapa concreta, el valor `<número_etapas>` es siempre 1. Cuando se utilizan para representar la información de la clasificación general, el valor `<número_etapas>` representa

el número de etapas que el corredor ha finalizado (y es, por tanto, menor o igual al número de etapas disputadas hasta el momento, ya que un corredor ha podido disputarlas todas o ha podido retirarse de la carrera en una etapa previa).

A modo de ejemplo, un fichero que respondiera a dicho formato y que representara las marcas obtenidas por cada corredor en una determinada etapa podría tener el siguiente contenido, donde los datos enteros se representan en base diez, agrupados por llaves y separados por espacios en blanco y comas **solo** para facilitar la comprensión del contenido del mismo:

```
{ {17, 1, 16573, 25, 0 }, { 19, 1, 16574, 24, 0 }, {201, 1, 16583, 16, 0 },
  { 2, 1, 16590, 14, 0 }, {125, 1, 16600, 12, 0 }, { 79, 1, 16600, 10, 0 },
  {35, 1, 16601,  9, 0 }, ... }
```

El contenido del fichero anterior informa, por ejemplo, de que el corredor con dorsal n.º 17 completó la etapa en 16573 segundos (4 horas, 36 minutos y 13 segundos), ganó 25 puntos para la clasificación de la regularidad y 0 puntos para la de la montaña. Así mismo, el corredor de dorsal n.º 2 (que se correspondería, de acuerdo con el ejemplo de fichero de texto de corredores presentado anteriormente, con el corredor belga Edward Theuns del equipo Trek-Segafredo), completó la etapa en 16590 segundos, ganando 14 puntos en la clasificación de la regularidad y 0 puntos en la de montaña.

Se ha establecido que los ficheros binarios correspondientes a las marcas registradas por cada ciclista en cada etapa se van a almacenar en ficheros que tendrán por nombre `clasificacion-etapa-XX.dat`, donde *XX* es el número de etapa, expresado siempre con dos dígitos (con un cero a la izquierda si fuera necesario).

El módulo de biblioteca `corredor`

Para representar la información de los corredores que participan en la vuelta ciclista por etapas, se dispone de un módulo de biblioteca denominado `corredor`, cuyo fichero de interfaz (`corredor.h`), se muestra a continuación. El tipo de datos `Corredor` y las funciones que lo acompañan en el fichero de cabecera pueden ser utilizados en la resolución de los problemas que siguen, tanto si se pide su código, como si no.

```
/* Constantes que establecen las longitudes máximas de las cadenas de
 * caracteres de los campos del tipo «Corredor».
 */
const int MAX_LONG_NOMBRE = 100;
const int MAX_LONG_APELLIDOS = 200;
const int MAX_LONG_EQUIPO = 50;
const int LONG_NACIONALIDAD = 4;

/* Tipo registro que representa la información general de un corredor
 * en una vuelta ciclista por etapas.
```

```
 */
struct Corredor {
    int dorsal;
    char nombre[MAX_LONG_NOMBRE];
    char apellidos[MAX_LONG_APELLIDOS];
    char equipo[MAX_LONG_EQUIPO];
    char nacionalidad[LONG_NACIONALIDAD];
};


/*
 * Pre:  «nombre», «apellidos», «equipo» y «nacionalidad» son cadenas de
 *       caracteres acabadas en el carácter nulo '\0'; «nombre» tiene
 *       MAX_LONG_NOMBRE caracteres como máximo; «apellidos»,
 *       MAX_LONG_APELLIDOS como máximo; «equipo», MAX_LONG_EQUIPO caracteres
 *       como máximo y «nacionalidad» tiene exactamente LONG_NACIONALIDAD
 *       caracteres.
 * Post: El registro «corredor» representa un corredor con el dorsal,
 *       nombre, apellidos, equipo y nacionalidad definidas por los valores de
 *       los parámetros «dorsal», «nombre», «apellidos», «equipo» y
 *       «nacionalidad», respectivamente.
 */
void establecerCorredor(Corredor& corredor, const int dorsal,
    const char nombre[], const char apellidos[], const char equipo[],
    const char nacionalidad[]);


/*
 * Pre:  ---
 * Post: Ha devuelto el valor del dorsal del registro «corredor».
 */
int dorsal(const Corredor corredor);

/*
 * Pre:  ---
 * Post: Ha copiado el valor del nombre del ciclista representado por el
 *       registro «corredor» en el parámetro nombre.
 */
void nombre(const Corredor corredor, char nombre[]);

/*
 * ...
 *
 * Existen funciones análogas a la inmediatamente anterior para la obtención
```

```
 * de los apellidos, equipo y nacionalidad del ciclista representado por un
 * registro de tipo «Corredor».
 */

/*
 * Pre:  «nombreFichero» es una cadena de caracteres que representa el nombre
 *       de un fichero de texto con la información sobre los corredores de una
 *       vuelta ciclista por etapas que sigue la sintaxis establecida en el
 *       enunciado de este examen; «corredores» tiene un número de componentes
 *       al menos estrictamente mayor al número de dorsal más alto de del
 *       conjunto de los números de doral de los ciclistas que aparecen en el
 *       fichero de nombre «nombreFichero».
 * Post: La componente «corredores[i]» contiene la información del ciclista de
 *       dorsal n.º «i» almacenada en el fichero de nombre «nombreFichero».
 */
void cargar(const char nombreFichero[], Corredor corredores[]);


/*
 * Pre:  «nombreCompleto» tiene las componentes necesarias para almacenar el
 *       nombre completo del ciclista cuyo nombre y apellidos están
 *       almacenados en el registro «corredor».
 * Post: «nombreCompleto» almacena el nombre completo del ciclista del
 *       registro «corredor», consistente en su nombre y sus apellidos,
 *       separados por un espacio en blanco.
 */
void nombreCompleto(char nombreCompleto[], const Corredor corredor);
```

El módulo de biblioteca `marca`

Para representar la información de los tiempos y puntos obtenidos por los corredores en cada etapa o en la clasificación general, se dispone de un módulo de biblioteca denominado `marca`, cuyo fichero de interfaz (`marca.h`), se muestra a continuación. El tipo de datos `Marca` y las funciones que lo acompañan en el fichero de cabecera pueden ser utilizados en la resolución de los problemas que siguen, tanto si se pide su código, como si no.

```
/* Tipo registro que representa los datos relativos a los tiempos y puntos
 * para la clasificación de la regularidad y de la montaña que un corredor ha
 * invertido en una etapa concreta o en la carrera global.
 */
 struct Marca {
     // número de dorsal que identifica al ciclista en la carrera
    int dorsal;
```

```
    // número total de etapas a las que hace referencia este registro (igual a
    // 1 si se trata de una única etapa, o mayor o igual que 1 si se trata de
    // una clasificación general).
    int numEtapas;

    // segundos en completar la etapa o etapas.
    int tiempo;

    // puntos obtenidos en la etapa o etapas.
    int regularidad;

    // puntos de montaña en la etapa o etapas.
    int montagna;
};

/*
 * Pre:  ---
 * Post: Ha establecido el dorsal correspondiente al parámetro «marca» como el
 *       valor del parámetro «numDorsal» y el resto de los campos
 *       («numEtapas», «tiempo», «regularidad» y «montagna»), a 0
 */
void ponerMarcaA0(Marca& marca, const int numDorsal);

/*
 * Pre:  ---
 * Post: Ha devuelto el número de dorsal correspondiente a «marcaCorredor».
 */
int dorsal(const Marca marcaCorredor);


/*
 * Pre:  ---
 * Post: Ha devuelto el número de etapas almacenado en el registro «marca».
 */
int numEtapas(const Marca marca);


/*
 * Pre:  ---
 * Post: Ha devuelto la cantidad de segundos correspondiente al tiempo
 *       almacenada en el registro «marca».
 */
int tiempo(const Marca marca);
```

```
/*
 * Pre:  ---
 * Post: Ha devuelto el número de puntos correspondientes a la clasificación
 *       de la regularidad correspondiente al registro «marca».
 */
int regularidad(const Marca marca);


/*
 * Pre:  ---
 * Post: Ha devuelto el número de puntos correspondientes a la clasificación
 *       de la montaña correspondiente al registro «marca».
 */
int montagna(const Marca marca);


/*
 * Pre:  «marcaGeneral» almacena el tiempo y los puntos de la clasificación
 *       de la regularidad y de la montaña de un determinado corredor
 *       identificado por su número de dorsal (y representa la situación de
 *       ese corredor tras haberse disputado un determinado número de etapas).
 *       «marcaEtapa» corresponde a la marca registrada por ese mismo
 *       ciclista en una etapa concreta (la etapa siguiente a la representada
 *       por «marcaGeneral»).
 * Post: «marcaGeneral» ha sido actualizado con los datos de «marcaEtapa»,
 *       sumándole a los datos que almacenaba inicialmente el tiempo y los
 *       puntos de la clasificación de la regularidad y de la montaña
 *       establecidos por el parámetro «marcaEtapa», y ha incrementado en 1 el
 *       valor del número de etapas de «marcaGeneral».
 */
void actualizar(Marca& marcaGeneral, const Marca marcaEtapa);
```

Problema 1.º (0,5 puntos)

Escribe el código de la función `actualizar` del módulo `marca`. Por comodidad, se repite a continuación su especificación:

```
/*
 * Pre:  «marcaGeneral» almacena el tiempo y los puntos de la clasificación
 *       de la regularidad y de la montaña de un determinado corredor
```

```
 *       identificado por su número de dorsal (y representa la situación de
 *       ese corredor tras haberse disputado un determinado número de etapas).
 *       «marcaEtapa» corresponde a la marca registrada por ese mismo
 *       ciclista en una etapa concreta (la etapa siguiente a la representada
 *       por «marcaGeneral»).
 * Post: «marcaGeneral» ha sido actualizado con los datos de «marcaEtapa»,
 *       sumándole a los datos que almacenaba inicialmente el tiempo y los
 *       puntos de la clasificación de la regularidad y de la montaña
 *       establecidos por el parámetro «marcaEtapa», y ha incrementado en 1 el
 *       valor del número de etapas de «marcaGeneral».
 */
void actualizar(Marca& marcaGeneral, const Marca marcaEtapa);
```

Problema 2.º (1 punto)

El cuerpo de la función `concatenar` que se presenta a continuación contiene algunos errores. La cabecera y especificación de la misma son, en todo caso, correctos y no han de ser modificados. Las funciones presentes en el módulo de biblioteca `cstring` son visibles en el código que se presenta a través de su inclusión en la directiva **`#include`** adecuada.

```
/*
 * Pre:  La cadena de caracteres «resultado» ha sido declarada con al menos
 *       strlen(prefijo) + strlen(sufijo) + 3 caracteres, «prefijo» y «sufijo»
 *       son cadenas de caracteres acabadas en el carácter nulo '\0' y
 *       «numero» es positivo y tiene como mucho dos dígitos cuando se escribe
 *       en base 10.
 * Post: «resultado» es la cadena de caracteres acabada en el carácter nulo
 *       '\0' resultante de concatenar la cadena «prefijo» con la
 *       representación decimal de dos dígitos de «numero» y con la cadena
 *       «sufijo».
 */
void concatenar(char resultado[], const char prefijo[], const int numero,
                const char sufijo[]) {
    strcmp(resultado, prefijo);
    int i = strlen(prefijo);
    resultado[i] = numero % 10 + '0';
    resultado[i + 1] = numero / 10 + '0';
    resultado[i + 2] = '\0';
    strcat(resultado, sufijo);
    return resultado;
}
```

Identifica los errores presentes en el código, indica con claridad y precisión en qué consisten y escribe el código corregido de la función `concatenar`.

Problema 3.º (0,5 puntos)

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

```
/*
 * Pre:  En la tabla «clasificación» existe exactamente una componente cuyo
 *       dorsal es igual al valor del parámetro «numDorsal».
 * Post: Ha devuelto el índice de la componente de la tabla «clasificación» en
 *       la que hay un registro de tipo «Marca» cuyo dorsal es igual al valor
 *       del parámetro «numDorsal».
 */
int buscar(const Marca clasificacion[], int numDorsal)
```

Cualquiera de las funciones definidas en los módulos `corredor` y `marca` o en los problemas anteriores pueden ser utilizadas en la solución de este problema.

Problema 4.º (3 puntos)

Escribe el código de la función `procesarEtapa` cuya especificación se muestra a continuación. En la misma se hace referencia a una constante simbólica denominada `NUM_ETAPAS` que ha sido definida como constante de tipo **`int`** con valor 21 y está en el ámbito del cuerpo de la función que se solicita.

```
/*
 * Pre:  1 <= etapa <= NUM_ETAPAS. La tabla «clasificacionGeneral» contiene
 *       una y exactamente una componente por cada uno de los ciclistas que
 *       participan en una carrera ciclista por etapas y que representan las
 *       marcas que cada uno de ellos ha registrado globalmente en las
 *       anteriores «etapa» - 1 etapas de la competición. Existe un fichero
 *       binario de nombre «clasificacion-etapa-XX.dat», donde XX es la
 *       representación decimal del valor del parámetro «etapa». Dicho fichero
 *       sigue la sintaxis establecida en el enunciado y contiene los datos de
 *       las marcas realizadas por los ciclistas en la etapa n.º «etapa» que
 *       finalizaron dicha etapa (por lo tanto, las marcas presentes en el
 *       fichero son las de un subconjunto de los corredores que participan en
 *       la carrera y todas ellas tienen correspondencia con una y exactamente
 *       una componente de la tabla «clasificacionGeneral»).
 * Post: La tabla «clasificacionGeneral» ha sido actualizada para que cada una
```

```
 *       de sus componentes represente la marca que cada corredor ha
 *       registrado en las primeras «etapa» etapas. La actualización de la
 *       tabla ha sido realizada procesando el fichero binario de nombre
 *       «clasificacion-etapa-XX.dat» mencionado en la precondición de esta
 *       función. En el caso de que el fichero mencionado en la precondición
 *       no se haya podido abrir para su lectura, la función se habrá limitado
 *       a escribir un mensaje de error en la pantalla.
 */
void procesarEtapa(const int etapa, Marca clasificacionGeneral[])
```

Cualquiera de las funciones definidas en los módulos `corredor` y `marca` o en los problemas anteriores pueden ser utilizadas en la solución de este problema y, de esta forma, simplificar el diseño de su código.

Problema 5.º (2 puntos)

Escribe el código de la función `cargar` del módulo `corredor`. Por comodidad, se repite a continuación su especificación:

```
/*
 * Pre:  «nombreFichero» es una cadena de caracteres que representa el nombre
 *       de un fichero de texto con la información sobre los corredores de una
 *       vuelta ciclista por etapas que sigue la sintaxis establecida en el
 *       enunciado de este examen; «corredores» tiene un número de componentes
 *       al menos estrictamente mayor al número de dorsal más alto del
 *       conjunto de los números de dorsal de los ciclistas que aparecen en el
 *       fichero de nombre «nombreFichero».
 * Post: La componente «corredores[i]» contiene la información del ciclista de
 *       dorsal n.º «i» almacenada en el fichero de nombre «nombreFichero».
 */
void cargar(const char nombreFichero[], Corredor corredores[]);
```

Cualquiera de las funciones definidas en los módulos `corredor` y `marca` o en los problemas anteriores pueden ser utilizadas en la solución de este problema.

Problema 6.º (3 puntos)

Escribe el código de la función `mostrarClasificacion` cuya especificación se muestra a continuación. Se sabe que el número de corredores que comenzaron la competición es igual al valor de una constante de tipo **`int`** denominada `NUM_CORREDORES` que ha sido convenientemente declarada y está en el ámbito de la función cuyo código se solicita. Similarmente, el

valor del dorsal más alto asignado a algún corredor ha sido declarado a través de otra constante simbólica denominada `NUM_DORSALES`, también declarada en el ámbito de esta función.

```
/*
 * Pre:  «clasificacionGeneral» es una tabla que contiene una y exactamente
 *       una componente por cada uno de los ciclistas que participan en una
 *       carrera ciclista por etapas y que representan las marcas que cada uno
 *       de ellos ha registrado en un determinado número de etapas. La tabla
 *       «clasificacionGeneral» YA está ordenada de forma que representa el
 *       orden de la clasificación general tras ese número determinado de
 *       etapas (es decir, está ordenada de mayor a menor número de etapas
 *       completadas y, en caso de igualdad en el número de etapas,
 *       por el tiempo invertido en completarlas, de menor a mayor tiempo).
 *       Existe un fichero de texto de nombre «corredores.txt» que contiene la
 *       información sobre los corredores de esa misma vuelta ciclista por
 *       etapas que sigue la sintaxis establecida en el enunciado de este
 *       examen.
 * Post: Ha escrito en la pantalla la clasificación general de la carrera
 *       representada por los datos del parámetro «clasificacionGeneral» y
 *       complementada por el contenido del fichero «corredores.txt» de
 *       acuerdo con el siguiente formato:
 *
 *       PUESTO  DORSAL  CORREDOR              ETAPAS    TIEMPO
 *       ------  ------  --------------------  ------  ---------
 *          1.      21   Chris Froome              21   82:30:02
 *          2.     151   Vincenzo Nibali           21   82:32:17
 *          3.     101   Ilnur Zakarin             21   82:32:53
 *          4.      61   Wilco Kelderman           21   82:33:17
 *          5.       1   Alberto Contador          21   82:33:20
 *         ...     ...   ...                      ...        ...
 *        198.      73   Michal Kolár               1    0:18:56
 */
void mostrarClasificacion(const Marca clasificacionGeneral[])
```

Como en problemas anteriores, cualquiera de las funciones definidas en los módulos `corredor` y `marca` o en los problemas anteriores pueden ser utilizadas en la solución de este problema. Se recomienda, además, utilizar una adecuada descomposición funcional del código en la solución solicitada.

Solución al problema 1.º

```
/*
 * Pre:  «marcaGeneral» almacena el tiempo y los puntos de la clasificación
 *       de la regularidad y de la montaña de un determinado corredor
 *       identificado por su número de dorsal (y representa la situación de
 *       ese corredor tras haberse disputado un determinado número de etapas).
 *       «marcaEtapa» corresponde a la marca registrada por ese mismo
 *       ciclista en una etapa concreta (la etapa siguiente a la representada
 *       por «marcaGeneral»).
 * Post: «marcaGeneral» ha sido actualizado con los datos de «marcaEtapa»,
 *       sumándole a los datos que almacenaba inicialmente el tiempo y los
 *       puntos de la clasificación de la regularidad y de la montaña
 *       establecidos por el parámetro «marcaEtapa», y ha incrementado en 1 el
 *       valor del número de etapas de «marcaGeneral».
 */
void actualizar(Marca& marcaGeneral, const Marca marcaEtapa) {
    marcaGeneral.numEtapas++;
    marcaGeneral.tiempo += marcaEtapa.tiempo;
    marcaGeneral.regularidad += marcaEtapa.regularidad;
    marcaGeneral.montagna += marcaEtapa.montagna;
}
```

Solución al problema 2.º

1. El primer error se encuentra en la línea n.º 14. En lugar de la función `strcmp` (que compara dos cadenas de caracteres), debería haberse usado la función `strcpy`.
2. El segundo error se encuentra en las líneas 16 y 17. A la cadena `resultado` se le concatena el número de etapa de forma incorrecta (primero el valor de las unidades y detrás, el de las decenas). Deben intercambiarse las expresiones que son asignadas a `resultado[i + 1]` y `resultado[i + 2]`.
3. La función `concatenar` no debe devolver ningún valor (el tipo especificado en la cabecera de la función es **`void`**). La instrucción **`return`** de la línea 20 es incorrecta y debe ser eliminada. El resultado calculado por la función ya ha sido asignado al parámetro `resultado`, que es una tabla de caracteres que ha sido pasada por referencia.

El código correcto completo se muestra a continuación:

```
/*
 * Pre:  La cadena de caracteres «resultado» ha sido declarada con al menos
 *       strlen(prefijo) + strlen(sufijo) + 3 caracteres, «prefijo» y «sufijo»
 *       son cadenas de caracteres acabadas en el carácter nulo '\0' y
 *       «numero» es positivo y tiene como mucho dos dígitos cuando se escribe
```

```
 *       en base 10.
 * Post: «resultado» es la cadena de caracteres acabada en el carácter nulo
 *       '\0' resultante de concatenar la cadena «prefijo» con la
 *       representación decimal de dos dígitos de «numero» y con la cadena
 *       «sufijo».
 */
void concatenar(char resultado[], const char prefijo[], const int numero,
                const char sufijo[]) {
    strcpy(resultado, prefijo);
    int i = strlen(prefijo);
    resultado[i] = numero / 10 + '0';
    resultado[i + 1] = numero % 10 + '0';
    resultado[i + 2] = '\0';
    strcat(resultado, sufijo);
}
```

Solución al problema 3.º

```
/*
 * Pre:  En la tabla «clasificación» existe exactamente una componente cuyo
 *       dorsal es igual al valor del parámetro «numDorsal».
 * Post: Ha devuelto el índice de la componente de la tabla «clasificación» en
 *       la que hay un registro de tipo «Marca» cuyo dorsal es igual al valor
 *       del parámetro «numDorsal».
 */
int buscar(const Marca clasificacion[], int numDorsal) {
    // Aplicación directa del esquema de búsqueda con garantía de éxito.
    int i = 0;
    while (dorsal(clasificacion[i]) != numDorsal) {
        i++;
    }
    return i;
}
```

Solución al problema 4.º

```
/*
 * Estimación de la longitud máxima del nombre del fichero.
 */
const int MAX_LONG_NOMBRE_FICHERO = 50;
```

```
/*
 * Prefijo del nombre del fichero que contiene los datos de una etapa (parte
 * fija que precede al número de la etapa).
 */
const char PREFIJO_NOMBRE_FICHERO[] = "../../Datos/clasificacion-etapa-";

/*
 * Sufijo del nombre del fichero que contiene los datos de una etapa (parte
 * fija que sucede al número de la etapa).
 */
const char SUFIJO_NOMBRE_FICHERO[] = ".dat";

/*
 * Pre:  1 <= etapa <= NUM_ETAPAS. La tabla «clasificacionGeneral» contiene
 *       una y exactamente una componente por cada uno de los ciclistas que
 *       participan en una carrera ciclista por etapas y que representan las
 *       marcas que cada uno de ellos ha registrado globalmente en las
 *       anteriores «etapa» - 1 etapas de la competición. Existe un fichero
 *       binario de nombre «clasificacion-etapa-XX.dat», donde XX es la
 *       representación decimal del valor del parámetro «etapa». Dicho fichero
 *       sigue la sintaxis establecida en el enunciado y contiene los datos de
 *       las marcas realizadas por los ciclistas en la etapa n.º «etapa» que
 *       finalizaron dicha etapa (por lo tanto, las marcas presentes en el
 *       fichero son las de un subconjunto de los corredores que participan en
 *       la carrera y todas ellas tienen correspondencia con una y exactamente
 *       una componente de la tabla «clasificacionGeneral»).
 * Post: La tabla «clasificacionGeneral» ha sido actualizada para que cada una
 *       de sus componentes represente la marca que cada corredor ha
 *       registrado en las primeras «etapa» etapas. La actualización de la
 *       tabla ha sido realizada procesando el fichero binario de nombre
 *       «clasificacion-etapa-XX.dat» mencionado en la precondición de esta
 *       función. En el caso de que el fichero mencionado en la precondición
 *       no se haya podido abrir para su lectura, la función se habrá limitado
 *       a escribir un mensaje de error en la pantalla.
 */
void procesarEtapa(const int etapa, Marca clasificacionGeneral[]) {
    // Obtención del nombre del fichero que tiene los datos de la etapa cuyo
    // valor es igual al del parámetro «etapa». Se utiliza la función
    // «concatenar» del problema 2.º
    char nombreFichero[MAX_LONG_NOMBRE_FICHERO];
    concatenar(nombreFichero, PREFIJO_NOMBRE_FICHERO, etapa,
               SUFIJO_NOMBRE_FICHERO);

    // Apertura del fichero y comprobación de la misma
```

```
    ifstream f(nombreFichero, ios::binary);
    if (f.is_open()) {

        Marca marcaLeida;

        // Intento de lectura de la primera marca del fichero
        f.read(reinterpret_cast<char*>(&marcaLeida), sizeof(marcaLeida));
        while (!f.eof()) {
            // Búsqueda del índice de la tabla «clasificacionGeneral»
            // correspondiente a la última marca leída del fichero
            // utilizando la función «buscar» del problema 3.º. La
            // precondición de esta función garantiza el éxito en la búsqueda.
            int indice = buscar(clasificacionGeneral, dorsal(marcaLeida));

            // Una vez localizado el índice, se actualiza el valor de dicha
            // componente con la función «actualizar» del problema 1.º
            actualizar(clasificacionGeneral[indice], marcaLeida);

            // Intento de lectura de una marca del fichero para la siguiente
            // iteración
            f.read(reinterpret_cast<char*>(&marcaLeida), sizeof(marcaLeida));
        }
        f.close();
    }
    else {
        cerr << "No se ha podido abrir el fichero \"" << nombreFichero
             << '"' << endl;
    }
}
```

Solución al problema 5.º

```
/*
 * Pre:  «nombreFichero» es una cadena de caracteres que representa el nombre
 *       de un fichero de texto con la información sobre los corredores de una
 *       vuelta ciclista por etapas que sigue la sintaxis establecida en el
 *       enunciado de este examen; «corredores» tiene un número de componentes
 *       al menos estrictamente mayor al número de dorsal más alto del
 *       conjunto de los números de dorsal de los ciclistas que aparecen en el
 *       fichero de nombre «nombreFichero».
 * Post: La componente «corredores[i]» contiene la información del ciclista de
 *       dorsal n.º «i» almacenada en el fichero de nombre «nombreFichero».
 */
```

```
 void cargar(const char nombreFichero[], Corredor corredores[]) {

    // Apertura del fichero y comprobación
    ifstream f(nombreFichero);
    if (f.is_open()) {

        // Intento de lectura del dorsal del corredor que aparece en la
        // primera línea del fichero
        int dorsal;
        f >> dorsal;

        while (!f.eof()) {
            // Los datos del corredor de la línea actual del fichero se
            // asignan a la componente «corredores[dorsal]»

            // El número de dorsal ya ha sido leído del fichero
            corredores[dorsal].dorsal = dorsal;

            // Se ignora la coma que separa el número de dorsal del siguiente
            // dato de la línea
            f.get();

            // Lectura de los siguientes datos de la línea, y asignación al
            // campo adecuado del registro «corredores[dorsal]»
            f.getline(corredores[dorsal].apellidos, MAX_LONG_APELLIDOS, ',');
            f.getline(corredores[dorsal].nombre, MAX_LONG_NOMBRE, ',');
            f.getline(corredores[dorsal].equipo, MAX_LONG_EQUIPO, ',');
            f.getline(corredores[dorsal].nacionalidad, LONG_NACIONALIDAD);
            f >> dorsal;
        }
        f.close();
    } else {
        cerr << "No se ha podido abrir el fichero \"" << nombreFichero << "\"."
             << endl;
    }
}
```

Solución al problema 6.º

```
const int NUM_DORSALES = 219;
const int NUM_CORREDORES = 198;
const char NOMBRE_FICHERO_CORREDORES[] = "../../Datos/corredores.txt";
```

```
const int ANCHO_PUESTO = 5;
const int ANCHO_DORSAL = 7;
const int ANCHO_CORREDOR = 20;
const int ANCHO_ETAPAS = 6;

/*
 * Pre:  tiempo >= 0 y es un tiempo expresado en segundos.
 * Post: Ha escrito en la pantalla el valor de «tiempo» en el formato h:mm:ss,
 *       donde h son las horas completas equivalentes (utilizando al menos 3
 *       caracteres, pero sin ceros a la izquierda), y mm y ss son los minutos
 *       y segundos restantes, respectivamente, que han sido escritos
 *       utilizando 2 dígitos (completando con ceros a la izquierda cuando ha
 *       sido necesario).
 */
void escribirHora(const int tiempo) {
    int horas = tiempo / 3600;
    int minutos = tiempo /60 % 60;
    int segundos = tiempo % 60;
    cout << setw(3) << horas << ":" << setfill('0')
         << setw(2) << minutos << ":"
         << setw(2) << segundos << setfill('␣');
}


/*
 * Pre:  «clasificacionGeneral» es una tabla que contiene una y exactamente
 *       una componente por cada uno de los ciclistas que participan en una
 *       carrera ciclista por etapas y que representan las marcas que cada uno
 *       de ellos ha registrado en un determinado número de etapas. La tabla
 *       «clasificacionGeneral» YA está ordenada de forma que representa el
 *       orden de la clasificación general tras ese número determinado de
 *       etapas (es decir, está ordenada de mayor a menor número de etapas
 *       completadas y, en caso de igualdad en el número de etapas,
 *       por el tiempo invertido en completarlas, de menor a mayor tiempo).
 *       Existe un fichero de texto de nombre «corredores.txt» que contiene la
 *       información sobre los corredores de esa misma vuelta ciclista por
 *       etapas que sigue la sintaxis establecida en el enunciado de este
 *       examen.
 * Post: Ha escrito en la pantalla la clasificación general de la carrera
 *       representada por los datos del parámetro «clasificacionGeneral» y
 *       complementada por el contenido del fichero «corredores.txt» de
 *       acuerdo con el siguiente formato:
 *
 *       PUESTO  DORSAL  CORREDOR             ETAPAS    TIEMPO
```

```
 *       ------  ------  ------------------  ------  ---------
 *          1.      21   Chris Froome             21   82:30:02
 *          2.     151   Vincenzo Nibali          21   82:32:17
 *          3.     101   Ilnur Zakarin            21   82:32:53
 *          4.      61   Wilco Kelderman          21   82:33:17
 *          5.       1   Alberto Contador         21   82:33:20
 *         ...     ...   ...                     ...        ...
 *        198.      73   Michal Kolár              1    0:18:56
 */
void mostrarClasificacion(const Marca clasificacionGeneral[]) {
    // Se obtienen los datos de los corredores del fichero
    // «NOMBRE_FICHERO_CORREDORES» (que se utilizarán para obtener el nombre
    // completo a partir de su dorsal) y se almacenan en una tabla.
    Corredor corredores[NUM_DORSALES + 1];
    cargar(NOMBRE_FICHERO_CORREDORES, corredores);

    // Se escribe la cabecera la clasificación general
    cout << "PUESTO␣␣DORSAL␣␣CORREDOR␣␣␣␣␣␣␣␣␣␣␣␣␣␣ETAPAS␣␣␣␣TIEMPO␣" << endl;
    cout << "------␣␣------␣␣------------------␣␣------␣␣---------" << endl;

    for (int i = 0; i < NUM_CORREDORES; i++) {
        // Se procesa la «i»-ésima marca de la tabla «clasificacionGeneral»

        // Se obtiene el número de dorsal al que corresponde la marca
        int dorsalCorredor = dorsal(clasificacionGeneral[i]);

        // Se obtiene el nombre completo del corredor, utilizando la función
        // «nombreCompleto» del módulo «corredor». El resto de los datos
        // necesarios para escribir el listado están en la propia componente
        // de tipo «Marca» de la componente «clasificacionGeneral».
        char nombreCorredor[MAX_LONG_NOMBRE + MAX_LONG_APELLIDOS + 1];
        nombreCompleto(nombreCorredor, corredores[dorsalCorredor]);

        // Se escriben los datos solicitados en el listado: puesto (valor de
        // la variable «i»), número de dorsal, nombre completo, nombre del
        // corredor, número de etapas y tiempo
        cout << setw(ANCHO_PUESTO - 1) << i + 1 << ".␣"
             << setw(ANCHO_DORSAL) << dorsalCorredor << "␣␣␣"
             << setw(ANCHO_CORREDOR) << left << nombreCorredor << "␣␣"
             << setw(ANCHO_ETAPAS) << right
             << numEtapas(clasificacionGeneral[i]) << "␣␣";
        escribirHora(tiempo(clasificacionGeneral[i]));
        cout << endl;
    }
```

```
}
```