Examen escrito de Programación 1

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

2 de septiembre de 2020
Adaptado a los cambios hechos en el curso 2020-21

- 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 distinta** 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 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++*.

Parte I. Películas

En los problemas de esta parte, se pide trabajar con vectores de un tipo registro cuya definición se pide en el primer problema. Se recomienda leer los enunciados de los problemas 2.º y 3.º antes proporcionar la definición del registro.

Problema 1.º (1 punto)

Define un tipo registro, denominado `Pelicula`, que permita representar la siguiente información sobre una película:

- Título de la película
- Año de producción de la película
- Nombres de los principales intérpretes de la película. El número máximo de intérpretes es 10.

Junto con la definición del tipo, declara también las constantes necesarias.

Problema 2.º (2 puntos)

Escribe el código de la función `buscarPeliculaPorInterprete` cuya cabecera y especificación se muestra a continuación:

```
/* Pre:  numPeliculas >= 0 y el vector «peliculas» tiene al menos
 *       «numPeliculas» componentes. El parámetro «interprete» es una cadena
 *       de caracteres que representa el nombre de un actor o actriz.
 * Post: Si entre las «numPeliculas» primeras componentes del vector
 *       «peliculas» hay al menos una película entre cuyos intérpretes figura
 *       uno de nombre «interprete», ha devuelto el índice que ocupa en el
 *       vector una cualquiera de esas películas.
 *       En caso contrario (es decir, si no hay ninguna película entre cuyos
 *       intérpretes figure uno de nombre «interprete»), ha devuelto -1.
 */
int buscarPelicula(const Pelicula peliculas[], const unsigned int numPeliculas,
                   const string interprete);
```

Problema 3.º (1 punto)

Escribe el código de una función denominada `escribir` que, escriba en un fichero de texto el contenido de un vector de tipo `Pelicula`, definido en el problema 1.º, conforme a la sintaxis de la regla `<fichero-películas>` que se define a continuación:

```
<fichero-películas> ::= {<película> fin-de-línea }
<película> ::= <año> fin-de-línea
               <título> fin-de-línea
               <núm-intérpretes> fin-de-línea
               { <intérprete> fin-de-línea }
<año> ::= literal-entero
<título> ::= literal-cadena
<núm-intérpretes> ::= literal-entero
<intérprete> ::= literal-cadena
```

A modo de ejemplo, se muestra a continuación un fichero que cumple con dicha sintaxis:

```
1994
The Shawshank Redemption
2
Tim Robbins
Morgan Freeman

2008
The Dark Knight
4
Christian Bale
Heath Ledger
Aaron Eckhart
Michael Caine

```

La cabecera y especificación de la función que se pide diseñar se muestran a continuación:

```
/* Pre:  El vector «peliculas» tiene al menos «numPeliculas» componentes.
 * Post: Ha creado el fichero «nombreFichero» que cumple con la sintaxis de la
 *       regla <fichero-películas> y
 *       que contiene los datos de las primeras «numPeliculas» componentes del
 *       vector «peliculas». Si ha podido crear el fichero, ha devuelto
 *       «true». En caso contrario, se ha limitado a devolver «false».
 */
bool escribir(const string nombreFichero, const Pelicula peliculas[],
              const unsigned int numPeliculas);
```

Parte II. Transporte de pasajeros por carretera

Problema 4.º (6 puntos)

Una empresa dedicada al transporte de viajeros por carretera opera distintas rutas entre un determinado origen, un destino y un conjunto de paradas intermedias.

Cada una de las rutas operadas por la empresa están almacenadas en un fichero de texto con la siguiente estructura:

```
<fichero-ruta> ::= <origen> {<parada-intermedia>} <destino>
<origen> ::= literal-cadena fin-de-línea
<parada-intermedia> ::= literal-cadena fin-de-línea
<destino> ::= literal-cadena fin-de-línea
```

Por ejemplo, el fichero denominado «`ruta-1245.txt`», cuyo contenido se muestra a continuación, cumple con la sintaxis de la regla `<fichero-ruta>`:

```
Jaca
Sabiñánigo
Huesca
Zaragoza
Calatayud
Madrid
```

La misma empresa denomina *expedición* a cada viaje concreto que se realiza correspondiente a una ruta. Cada una de las expediciones programadas por la empresa están almacenadas en un fichero de texto, en el que se almacena el nombre del fichero que describe la ruta a la que corresponde la expedición, la fecha de la misma, las horas de salida del origen, paso por las paradas intermedias y llegada al destino final y una lista de pasajeros que han comprado billete para realizar el viaje correspondiente a la expedición en cuestión. La información de cada pasajero aparece en líneas independientes, en las que se indican las paradas en las que comienzan y terminan su viaje seguidas de su nombre completo y de una dirección de correo electrónico. Las paradas están identificadas con números comprendidos entre 0 y $n-1$, siendo n el número de localidades por las que pasa la expedición, incluyendo la de origen y la de destino. El valor 0 representa la localidad origen de la expedición, el valor 1, la de la primera parada intermedia y, así sucesivamente, hasta llegar al valor $n-1$, que representa la localidad de destino de la expedición. Cada expedición que opera esta compañía empieza y termina en el mismo día.

Los ficheros correspondientes a expediciones cumplen con la sintaxis de la regla `<fichero-expedición>` que se muestra a continuación:

```
<fichero-expedición> ::= <nombre-fichero-ruta> fin-de-línea
                         <horario> fin-de-línea
                         {<pasajero> fin-de-línea}
<nombre-fichero-ruta> ::= literal-cadena
<horario> ::= <fecha> {<blanco> <hora>}
<fecha> ::= literal-entero «-» literal-entero «-» literal-entero
<hora> ::= literal-entero «:» literal-entero
<pasajero> ::= <parada-subida> <blanco> <parada-bajada> <blanco>
               <nombre-completo> «;» <correo-electrónico>
<parada-subida> ::= literal-entero
<parada-bajada> ::= literal-entero
<nombre-completo> ::= literal-cadena
<correo-electrónico> ::= literal-cadena
<blanco> ::= {« »}
```

El fichero «`jaca-mad-2-9-20.txt`», cuyo contenido se muestra a continuación, cumple con la sintaxis de la regla `<fichero-expedición>`:

```
datos/ruta-1245.txt
2-9-2020 9:15 10:10 11:15 12:25 15:20
0   5   MARIA DOLORES ROMERO RUBIO;romero4@jmail.com
0   3   MANUEL RODRIGUEZ TORRES;rodriguez8@jmail.com
3   5   DANIEL GOMEZ BLANCO;gomez2@jmail.com
3   5   JAVIER PEREZ SERRANO;perez7@jmail.com
3   5   ANA MARIA DIAZ DELGADO;diaz9@jmail.com
1   4   FRANCISCO FERNANDEZ VAZQUEZ;fernandez3@jmail.com
1   3   JUAN MARTINEZ GIL;martinez2@jmail.com
```

En dicho fichero, puede observase como la pasajera María Dolores Romero Rubio ha comprado un billete para realizar el trayecto completo de la expedición: se sube al autobús en la localidad 0 (correspondiente a Jaca) y desciende en la localidad 5 (correspondiente a Madrid), mientras que Francisco Fernández Vázquez realizará el trayecto Sabiñánigo-Calatayud (primera y cuarta paradas intermedias, identificadas con los enteros 1 y 4).

La mencionada empresa cuenta con una flota de vehículos con distintas capacidades y, con el objetivo de reducir consumos y minimizar costes, está interesada en realizar sus expediciones con los vehículos más pequeños disponibles en cada momento y que garanticen plaza a todos los pasajeros que han comprado billete. Para ello, quiere calcular la *capacidad mínima* del vehículo requerido por cada una de las expediciones que realiza. Esta capacidad mínima será siempre menor o igual al número de pasajeros distintos que han comprado billete para una expedición concreta.

Así, por ejemplo, pese a que en la expedición descrita en el fichero «`jaca-mad-2-9-20.txt`» han comprado billete siete pasajeros distintos, la capacidad mínima es de 5 plazas, ya que el

número máximo de pasajeros distintos entre dos paradas consecutivas (que se alcanza entre las paradas de Zaragoza y Calatayud) es de 5. A modo ilustrativo, se muestra en la siguiente tabla el número de plazas ocupadas entre cada par de paradas consecutivas de la expedición del ejemplo, junto con los nombres de los pasajeros que los ocupan:

Paradas consecutivas	**Pasajeros**	**N.º de plazas ocupadas**
0 – 1 (Jaca–Sabiñánigo)	María, Manuel	2
1 – 2 (Sabiñánigo–Huesca)	María, Manuel, Francisco, Juan	4
2 – 3 (Huesca–Zaragoza)	María, Manuel, Francisco, Juan	4
3 – 4 (Zaragoza–Calatayud)	María, Francisco, Daniel, Javier, Ana	5
4 – ~~2~~ 5* (Calatayud–Madrid)	María, Daniel, Javier, Ana	4

* El enunciado impreso que se repartió indicaba por error 2 en lugar de 5 como identificador de la parada correspondiente a Madrid. Se advirtió verbalmente de dicha errata y de su corrección.

Escribe un programa completo que solicite al usuario el nombre del fichero correspondiente a una expedición y que informe, escribiendo en la pantalla, sobre la capacidad mínima del vehículo a utilizar para transportar a todos los pasajeros de la expedición. El programa, además, indicará el número de pasajeros que, según los datos del fichero, van a viajar entre cada par de paradas consecutivas, de acuerdo con el formato que se muestra en el siguiente ejemplo de ejecución.

```
Nombre fichero expedición: jaca-mad-2-9-20.txt
  2  Jaca-Sabiñánigo
  4  Sabiñánigo-Huesca
  4  Huesca-Zaragoza
  5  Zaragoza-Calatayud
  4  Calatayud-Madrid
La capacidad mínima del vehículo tiene que ser de 5 plazas.
```

En los casos en que alguno de los ficheros no exista o no pueda ser abierto, se mostrarán mensajes de error como los que aparecen en los siguientes ejemplos de ejecución:

```
Nombre fichero expedición: jaca-mad-2-9-01.txt
No ha podido abrirse la expedición "jaca-mad-2-9-01.txt".
```

```
Nombre fichero expedición: zgz-londres-2-9-20.txt
No ha podido abrirse la ruta "ruta-9999.txt".
```

Al resolver el problema, puede considerarse que en ningún caso se va a trabajar con rutas o expediciones de más de 50 paradas.

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 estar adecuadamente especificadas a través de una precondición y postcondición.

Solución al problema 1.º

```
/*
 * Número máximo de intérpretes principales de una película.
 */
const unsigned int MAX_INTERPRETES = 10;

/*
 * Definición del tipo registro que permite almacenar el título de una
 * película, su año de producción y entre 0 y 10 intérpretes principales.
 * El campo «interpretes» es un vector de MAX_INTERPRETES cadenas de
 * caracteres, cada una de ellas de MAX_LONG_NOMBRE caracteres como máximo.
 * El campo «numInterpretes» indica cuántas de las cadenas de «interpretes»
 * contienen información sobre los intérpretes de la película.
 */
struct Pelicula {
    string titulo;
    unsigned int agno;
    string interpretes[MAX_INTERPRETES];
    unsigned int numInterpretes;
};
```

Solución al problema 2.º

```
/* Pre:  0 <= peli.numInterpretes < MAX_INTERPRETES.
 *       peli.titulo y peli.interpretes[i] (con i entre 0 y
 *       peli.numInterpretes-1) son cadenas de menos de 200 caracteres
 *       acabadas en '\0'. El parámetro «interprete» es también una cadena de
 *       menos de 200 caracteres finalizada en '\0' que representa el nombre
 *       de un actor o actriz.
 * Post: Si entre las «peli.numInterpretes» primeras componentes del vector
 *       «peli.interpretes» hay al menos una cuyo valor es igual al de
 *       «interprete», ha devuelto true. En caso contrario, ha devuelto false.
 */
bool estaInterpreteEnPelicula(const Pelicula peli, const string interprete) {
    // Se aplica el esquema de búsqueda lineal sin garantía de éxito
    bool encontrado = false;
    unsigned int i = 0;
    while (!encontrado && i < peli.numInterpretes) {
        encontrado = (peli.interpretes[i] == interprete);
        i++;
    }
    return encontrado;
}
```

```
/*
 * Especificación en el enunciado.
 */
int buscarPelicula(const Pelicula peliculas[], const unsigned int numPeliculas,
                   const string interprete) {
    // Se aplica el esquema de búsqueda lineal sin garantía de éxito
    bool encontrado = false;
    unsigned int i = 0;
    while (!encontrado && i < numPeliculas) {
        if (estaInterpreteEnPelicula(peliculas[i], interprete)) {
            encontrado = true;
        }
        else {
            i++;
        }
    }

    // Discriminación del éxito
    if (encontrado) {
        return i;
    }
    else {
        return -1;
    }
}
```

Solución al problema 3.º

```
/*
 * Especificación en el enunciado.
 */
bool escribir(const string nombreFichero, const Pelicula peliculas[],
              const unsigned int numPeliculas) {
    // Apertura del fichero
    ofstream f(nombreFichero);

    // Comprobación de apertura
    if (f.is_open()) {
        // Cada coponente del vector «peliculas»...
        for (unsigned int i = 0; i < numPeliculas; i++) {
            // ... se escribe en el fichero campo a campo:
            f << peliculas[i].agno << endl;
            f << peliculas[i].titulo << endl;
            f << peliculas[i].numInterpretes << endl;
            for (unsigned int j = 0; j < peliculas[i].numInterpretes; j++) {
                f << peliculas[i].interpretes[j] << endl;
            }
            f << endl;
        }
        // Cierre del fichero y devolución de valor indicando ejecución correcta
        f.close();
        return true;
    }
    else {
        // Devolución de valor indicando ejecución errónea
        return false;
    }
}
```

Solución al problema 4.º

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

/* Número máximo de paradas: 50. El enunciado no indica específicamente que
 * sean paradas intermedias, así que por seguridad definimos la constante
 * con dos paradas más para considerar también el origen y el destino.
 */
const unsigned int MAX_PARADAS = 52;

/*
 * Pre:  El flujo «f» está asociado con un fichero de texto que cumple con la
 *       sintaxis de la regla <fichero-expedición> definida en el enunciado y
 *       se está en disposición de leer los datos del primer pasajero del
 *       mismo. El vector «ocupacion» tiene «MAX_PARADAS» componentes,
 *       inicializadas todas a 0.
 * Post: Cada componente «i» del vector «ocupacion» representa el número de
 *       pasajeros que, según el contenido de «f», viajarán entre las paradas
 *       de índices «i» e «i» + 1, y que, por tanto, ocuparán plaza en el
 *       vehículo que los transporte.
 */
void procesarPasajeros(ifstream& f, unsigned int ocupacion[]) {
    unsigned int origen, destino;
    f >> origen >> destino;
    while (!f.eof()) {
        // Ignoramos el nombre completo y el correo electrónico del pasajero
        string ignorarNombreCompletoYCorreoElectronico;
        getline(f, ignorarNombreCompletoYCorreoElectronico);

        // Ocupamos una plaza para el pasajero entre las estaciones «origen» y
        // «destino» - 1
        for (unsigned int i = origen; i < destino; i++) {
            ocupacion[i]++;
        }
        f >> origen >> destino;
    }
}
```

```
/*
 * Pre:  «nombreFicheroExpedicion» es el nombre de un fichero de texto que
 *       cumple con la sintaxis de la regla <fichero-expedición> definida en
 *       el enunciado. El vector «ocupacion» tiene «MAX_PARADAS» componentes,
 *       inicializadas todas a 0.
 * Post: Si «nombreFicheroExpedicion» cumple con la regla mencionada y se ha
 *       podido abrir, ha devuelto «true» y ha asignado a «nombreFicheroRuta»
 *       el nombre del fichero de texto que representa la ruta contenido en la
 *       primera línea del fichero «nombreFicheroExpedicion». Sea «n» el
 *       número de localidades presentes en el fichero
 *       «nombreFicheroExpedicion»: cada componente «i» entre 0 y «n» - 2 del
 *       vector «ocupacion» representa el número de  pasajeros que viajarán
 *       entre las paradas «i» e «i» + 1, según el contenido del fichero
 *       «nombreFicheroExpedicion».
 *       En caso contrario, ha devuelto «false».
 */
bool procesarExpedicion(const string nombreFicheroExpedicion,
                        string& nombreFicheroRuta, unsigned int ocupacion[]) {
    ifstream f(nombreFicheroExpedicion);
    if (f.is_open()) {
        getline(f, nombreFicheroRuta);
        string ignorarHorariosPaso;
        getline(f, ignorarHorariosPaso);
        procesarPasajeros(f, ocupacion);
        f.close();
        return true;
    }
    else {
        cerr << "No ha podido abrirse la expedición \""
             << nombreFicheroExpedicion << "\"." << endl;
        return false;
    }
}
```

```
/*
 * Pre:  «nombreFicheroRuta» representa el nombre de un fichero de texto que
 *       cumple con la sintaxis de la regla <fichero-ruta> definida en el
 *       enunciado. Sea «n» el número de localidades presentes en el fichero.
 *       El vector de cadenas de caracteres «paradasRuta» tiene, al menos, «n»
 *       componentes.
 * Post: En el caso de que se haya podido abrir el fichero
 *       «nombreFicheroRuta», ha asignado a las primeras «n» componentes del
 *       vector «paradasRuta» los valores correspondientes a los nombres de
 *       las localidades presentes en el fichero «nombreFicheroRuta», a
 *       «numParadasRuta» le ha asignado «n» y ha devuelto «true».
 *       En caso contrario, ha devuelto «false».
 */
bool leerRuta(const string nomFichRuta, string paradasRuta[],
              unsigned int& numParadasRuta) {
    ifstream f(nomFichRuta);
    if (f.is_open()) {
        numParadasRuta = 0;
        getline(f, paradasRuta[numParadasRuta]);
        while (!f.eof()) {
            numParadasRuta++;
            getline(f, paradasRuta[numParadasRuta]);
        }
        f.close();
        return true;
    }
    else {
        cerr << "No ha podido abrirse la ruta \""
             << nomFichRuta << "\"." << endl;
        return false;
    }
}
```

```
/*
 * Pre:  «nombreFicheroRuta» representa el nombre de un fichero de texto que
 *       cumple con la sintaxis de la regla <fichero-ruta> definida en el
 *       enunciado. Sea «n» el número de localidades presentes en el fichero.
 *       El vector «ocupacion» tiene al menos «n» componentes y cada
 *       componente «i» entre 0 y «n» - 2 del vector representa el número de
 *       pasajeros que viajarán entre las paradas «i» e «i» + 1.
 * Post: Ha escrito en la pantalla un listado que indica el número de
 *       pasajeros que, según los datos del vector «ocupacion», van a viajar
 *       entre cada par de paradas consecutivas, indicando los nombres de las
 *       localidades correspondientes a dichas paradas. Además, ha informado
 *       sobre la capacidad mínima que tendría que tener un vehículo a
 *       utilizar para transportar a todos los pasajeros indicados en el
 *       vector «ocupacion». En el caso de que no se haya podido abrir el
 *       fichero «nombreFicheroRuta», ha devuelto «false», sin escribir nada
 *       en la pantalla. En caso contrario, ha devuelto «true».
 */
bool escribirOcupacion(const string nombreFicheroRuta,
                       const unsigned int ocupacion[]) {
    string paradasRuta[MAX_PARADAS];
    unsigned int numParadasRuta;
    if (leerRuta(nombreFicheroRuta, paradasRuta, numParadasRuta)) {
        unsigned int ocupacionMaxima = 0;
        for (unsigned int i = 0; i < numParadasRuta - 1; i++) {
            if (ocupacion[i] > ocupacionMaxima) {
                ocupacionMaxima = ocupacion[i];
            }
            cout << setw(3) << ocupacion[i] << "␣␣"
                 << paradasRuta[i] << "-" << paradasRuta[i + 1] << endl;
        }
        cout << "La␣capacidad␣mínima␣del␣vehículo␣tiene␣que␣ser␣de␣"
             << ocupacionMaxima << "␣plazas." << endl;
        return true;
    }
    else {
        return false;
    }
}
```

```
/*
 * Cuando es ejecutado, este programa solicita al usuario el nombre de un
 * fichero de texto que cumpla con la sintaxis de la regla <fichero-expedición>
 * definida en el enunciado. Si dicho fichero cumple con la regla mencionada, el
 * programa escribe en la pantalla un listado que indica el número de pasajeros
 * que, según los datos del fichero, van a viajar entre cada par de paradas
 * consecutivas, indicando los nombres de las localidades correspondientes a
 * dichas paradas. Además, informa sobre la capacidad mínima del vehículo a
 * utilizar para transportar a todos los pasajeros de la expedición. El programa
 * lo hace de acuerdo con el formato del siguiente ejemplo de ejecución:
 *
 *     Nombre fichero expedición: jaca-mad-2-9-20.txt
 *       2  Jaca-Sabiñánigo
 *       4  Sabiñánigo-Huesca
 *       4  Huesca-Zaragoza
 *       5  Zaragoza-Calatayud
 *       4  Calatayud-Madrid
 *     La capacidad mínima del vehículo tiene que ser de 5 plazas.
 *
 * En caso de que no se haya podido abrir alguno de los ficheros, ha informado
 * escribiendo un mensaje de error en «cerr».
 */
int main() {
    cout << "Nombre␣fichero␣expedición:␣";
    string nombreFicheroExpedicion;
    getline(cin, nombreFicheroExpedicion);

    // Declaración de un vector de enteros para almacenar el número de
    // pasajeros que van a viajar entre cada par de paradas consecutivas. La
    // inicialización {} da valor 0 a todas las componentes del vector. Un bucle
    // en el que se inicializara cada componente de forma explícita a 0 también
    // sería correcto.
    unsigned int ocupacion[MAX_PARADAS] = {};
    string nombreFicheroRuta;

    if (procesarExpedicion(nombreFicheroExpedicion,
                           nombreFicheroRuta, ocupacion)) {
        escribirOcupacion(nombreFicheroRuta, ocupacion);
        return 0;
    }
    else {
        return 1;
    }
}
```