Examen escrito de Programación 1

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

7 de septiembre de 2017

- 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.*

Introducción

Uno de los problemas a los que se enfrentan en muchos centros de trabajo es la programación de reuniones de sus empleados en horas compatibles entre sí y con el resto de sus actividades. En este examen se va a trabajar con un programa que, a partir de distintos ficheros y de la entrada del usuario, va a ser capaz de encontrar un hueco en las agendas de un conjunto de personas de un determinado mes natural con el objeto de programar una reunión de una hora de duración entre todos ellos en ese hueco.

El programa con el que se va a trabajar tiene el siguiente comportamiento. Inicialmente, procesa un conjunto de ficheros (que se describirán más adelante) que contienen información sobre días laborables y las agendas de las personas involucradas en el mes en cuestión. Si

no se han producido problemas de lectura de ningún fichero, solicita al usuario un día concreto del mes e informa acerca de cuándo es posible programar la primera reunión de una hora de duración a partir de ese día de modo que todas las personas cuyas agendas se han procesado estén disponibles. Para ello, escribe el día y la hora a la que se puede programar la reunión, tanto si es en el mismo día proporcionado por el usuario (ver primer ejemplo de ejecución) o en un día posterior, pero siempre del mismo mes (segundo ejemplo). La reunión debe programarse respetando además las siguientes restricciones:

- El día de la reunión tiene que ser igual o posterior al día propuesto por el usuario y, evidentemente, menor o igual que 31.
- El día de la reunión tiene que ser un día laborable.
- La hora a la que se puede programar una reunión tiene que ser igual o posterior a las 9:00 (hora de entrada), anterior a las 18:00 (hora de salida) y distinta a las 14:00 (hora de la comida).

En el caso de que no hubiera ningún hueco para programar una reunión que cumpliera con todas las restricciones anteriores entre el día proporcionado por el usuario y el final del mes en cuestión, el programa mostrará un mensaje en la pantalla indicando dicha circunstancia (tercer ejemplo de ejecución).

Si alguno de los ficheros involucrados en el procesamiento inicial no pudiera abrirse, el programa informará del error y no llegará a pedir la fecha al usuario (ver el cuarto ejemplo de ejecución).

Se muestran a continuación los ejemplos de ejecución del programa solicitado antes referidos:

```
Día a partir del cual programar la reunión: 1

El primer hueco disponible para una reunión es el día 1 a las 15:00.
```

```
Día a partir del cual programar la reunión: 2

El primer hueco disponible para una reunión es el día 4 a las 9:00.
```

```
Día a partir del cual programar la reunión: 30

No hay ningún hueco disponible para la reunión en el mes en curso.
```

```
Error al abrir el fichero "calendario_lopez_2017_09_.txt"
```

En este examen se va a trabajar con tres tipos de ficheros de datos: un fichero que almacena información acerca de qué días son laborables, ficheros que contienen las agendas de los empleados que han de participar en la reunión y un fichero que lista los nombres de estos ficheros de agendas.

El fichero de días laborables

Existe un fichero binario denominado `laborables.dat` que permite determinar qué días del mes en cuestión son laborables. Se trata de una secuencia binaria de 31 datos de tipo **`bool`** de C++ que indican cuándo un día es laborable (a través de un valor **`true`**) o no lo es (valor **`false`**). El primer dato establece el carácter laborable o no del primer día del mes, el segundo dato, el del día 2 y así sucesivamente. Cuando el mes en cuestión tiene 30 días, el trigésimo primer dato del fichero es **`false`**, y cuando el mes en cuestión es febrero, los tres o dos últimos datos son **`false`**.

Ficheros de agenda

Se va a trabajar también con ficheros de texto que representan agendas de trabajo de distintas personas en un determinado mes natural. Estos ficheros siguen la sintaxis de la regla `<agenda>` que se define a continuación:

```
<agenda> ::= { <evento> fin_de_línea }
<evento> ::= <día> <sep> <hora_inicio> <sep> <hora_fin> <sep> <descripción>
<dia> ::= literal_entero
<hora_inicio> ::= literal_entero
<hora_fin> ::= literal_entero
<descripción> ::= literal_cadena
<sep> ::= “␣” {“␣”}
```

A modo de ejemplo, se muestran las primeras líneas de un fichero denominado `calendario_garcia_2017_09.txt`, correspondiente al mes de septiembre de 2017 de un determinado empleado:

```
 1 11 13 Tutorías
 7  9 12 Examen escrito
 7 13 15 Examen prácticas
 6 15 17 Tutorías
 8 11 13 Tutorías
12 10 18 Reunión proyecto europeo
13  9 17 Reunión proyecto europeo
13  9 13 Reunión proyecto europeo
14 11 13 Tutorías
15 11 13 Tutorías
15 16 17 Revisión
19 12 13 Tutorías
19 13 14 Presentación asignatura
19 19 20 Presentación asignatura
...
```

La primera línea indica que el día 1 del mes en cuestión hay programado un evento de dos horas de duración entre las 11:00 y las 13:00. La segunda indica un evento el día 7 de 9:00 a 12:00 y así sucesivamente. Los eventos en el fichero no tienen por qué estar ordenados cronológicamente. De hecho, nótese cómo los eventos del día 7 aparecen antes que el del día 6. En ningún fichero de agenda de una persona hay eventos que se solapen en el tiempo entre sí. En ningún caso se espera que la descripción de un evento vaya a tener más de 1000 caracteres.

Fichero listado de ficheros de agenda

Existe también un fichero de texto, denominado `listaFicheros.txt`, que contiene los nombres de otros ficheros de texto que cumplen con la sintaxis definida por la regla `<agenda>` y que corresponden todos ellos a personas distintas pero al mismo mes al que hace referencia el fichero `laborables.dat`. Se muestra a continuación un ejemplo de contenido del fichero `listaFicheros.txt`:

```
calendario_garcia_2017_09.txt
calendario_gonzalez_2017_09.txt
calendario_fernandez_2017_09.txt
calendario_lopez_2017_09.txt
```

Estructura del programa

Se dispone ya de parte del código que resuelve el problema. Se presenta a continuación la estructura del mismo (instrucciones de inclusión de bibliotecas, definiciones de constantes y declaraciones de algunas funciones). En los tres problemas que siguen, se pedirá la implementación de alguna de las funciones cuya cabecera aparecen a continuación, pero no de todas ellas.

```
#include <iostream>
#include <fstream>

using namespace std;

/* Número de horas de un día. */
const int NUM_HORAS = 24;

/* Número máximo de días de un mes. */
const int MAX_DIAS = 31;

/* Primera hora de la mañana a la que se pueden convocar reuniones.
 * Considerada como el inicio general de la jornada laboral. */
```

```
const int HORA_ENTRADA = 9;

/* Primera hora de la tarde a partir de la cual no se pueden convocar
 * reuniones. Considerada como el final general de la jornada laboral. */
const int HORA_SALIDA = 18;

/* Hora del mediodía en la que no se pueden convocar reuniones. */
const int HORA_COMIDA = 14;

/* Máxima longitud esperada de la descripción de un evento. */
const int MAX_LONG_DESCRIPCION = 1000;

/* Longitud máxima de los nombres de ficheros con los que se va a trabajar. */
const int MAX_LONG_NOMBRE_FICHERO = 512;

/* Nombre del fichero de texto que contiene un listado de los nombres de los
 * ficheros de agendas de todos los empleados que tienen que participar en la
 * reunión. */
const char NOMBRE_LISTA_FICHEROS[] = "listaFicheros.txt";

/* Nombre del fichero binario que especifica qué días son laborables del mes
 * en el que se va a plantear la reunión. */
const char NOMBRE_FICHERO_LABORABLES[] = "laborables.dat";

/* Valor constante que codifica como entero el hecho de que no se ha
 * encontrado hora para la reunión.*/
const int HORA_NO_ENCONTRADA = -1;



/*
 * Pre:  ---
 * Post: Ha solicitado al usuario un día del mes y lo ha leído del teclado
 *       asignándolo al parámetro «dia», asegurándose de que está
 *       comprendido entre 1 y 31.
 */
void pedirDatosReunion(int& dia);



/*
 * Pre:  Existe un fichero binario de nombre «NOMBRE_FICHERO_LABORABLES» que
 *       contiene una secuencia de al menos «MAX_DIAS» datos de tipo «bool»
 *       cuyo valor determina el carácter laboral («true») o no laboral
 *       («false») de cada uno de los días del mes y «laborables» es una tabla
 *       con al menos «MAX_DIAS» componentes.
```

```
 * Post: Ha devuelto «true» si la lectura del fichero denominado
 *       «NOMBRE_FICHERO_LABORABLES» se ha producido sin problemas y ha
 *       copiado los primeros «MAX_DIAS» datos del fichero denominado
 *       «NOMBRE_FICHERO_LABORABLES» a las primeras «MAX_DIAS» componentes de
 *       la tabla «laborables». En caso contrario, ha devuelto «false» y ha
 *       escrito un mensaje de error en la pantalla.
 */
bool leerFicheroDiasLaborables(bool laborables[]);


/*
 * Pre:  Existe un fichero de texto de nombre «NOMBRE_LISTA_FICHEROS» que
 *       contiene una secuencia nombres de ficheros de texto que cumplen con
 *       la sintaxis establecida por la regla <agenda> definida en
 *       el enunciado y «ocupacion» es una matriz con al menos «MAX_DIAS»
 *       filas y exactamente «NUM_HORAS» columnas.
 * Post: Las componentes ocupacion[d][h] de la matriz «ocupacion» que se
 *       corresponden con eventos recogidos en los ficheros de agenda listados
 *       en el fichero «NOMBRE_LISTA_FICHEROS» tienen un valor igual al número
 *       de agendas en las que hay un evento en el día d+1 que cubre el
 *       intervalo temporal entre las h:00 y h:59 horas. Las componentes valen
 *       0 cuando no se corresponden con ningún evento en ninguna de las
 *       agendas. Ha devuelto true si las lecturas del fichero denominado
 *       «NOMBRE_LISTA_FICHEROS» y de todos en él contenidos se han producido
 *       sin problemas. En caso contrario, ha devuelto false y ha escrito un
 *       mensaje de error en la pantalla.
 */
bool obtenerOcupacionTotal(int ocupacion[][NUM_HORAS]);


/*
 * Pre:  «ocupacion» es una matriz con al menos «MAX_DIAS» filas y exactamente
 *       «NUM_HORAS» columnas; «laborables» es una tabla con al menos
 *       «MAX_DIAS» componentes, «dia» está inicialmente entre 1 y 31 y el
 *       valor de «hora» no está definido.
 * Post: Tras acabar la ejecución, se ha buscado el primer hueco disponible a
 *       partir del valor inicial del parámetro «dia», y los parámetros «dia»
 *       y «hora» se han modificado para indicar cuál es ese primer hueco. El
 *       hueco tiene que cumplir, además, estar en una hora igual o posterior
 *       a «HORA_ENTRADA», anterior a «HORA_SALIDA» y ser distinto a
 *       «HORA_COMIDA». El día del hueco, además, tiene que ser laborable,
 *       según la información de la tabla «laborables» (laborables[i] es true
       si y solo si el día i+1 es laborable).
 *       Si no había ningún hueco que cumpla con las condiciones anteriores a
```

```
 *       partir del valor inicial del parámetro «dia», el valor del parámetro
 *       «hora» se ha establecido en «HORA_NO_ENCONTRADA».
 */
void buscarHueco(const int ocupacion[][NUM_HORAS], const bool laborables[],
                 int& dia, int& hora);


/*
 * Pre:  ¿?
 * Post: ¿?
 */
int main();
```

Problema 1.º (2 puntos)

Escribe la especificación y el código de la función `main`, de forma que resuelva el problema planteado en la introducción.

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 la introducción. Estas funciones pueden ser utilizadas tanto si se pide su código en los problemas 2.º y 3.º como si no. Si lo estimas necesario, puedes definir también otras funciones auxiliares, cuyo código sí que tendrás que proporcionar.

Se puede (y recomienda) utilizar también las constantes simbólicas definidas en la sección anterior.

Problema 2.º (4 puntos)

Escribe el código de la función `obtenerOcupacionTotal`, cuya especificación aparece en el esquema de código presentado en la introducción. Está función se encarga de leer los ficheros de agenda cuyos nombres están en el fichero `listaFicheros.txt` y rellenar adecuadamente una matriz que indique la ocupación de las horas de los distintos días: La matriz `ocupacion` está indexada por un primer índice que indica el día (con valores de 0 a $31 - 1$) y un segundo índice que indica las horas (de 0 a 23). La matriz debe ser calculada de forma que la componente `ocupacion[d][h]` represente el número de agendas que tienen algún evento programado para el día $d + 1$ entre las h:00 y h:59.

El problema que tiene que resolver la función `obtenerOcupacionTotal` tiene la suficiente entidad como para requerir la descomposición del mismo en otros de menor tamaño. Se exige

por tanto que te apoyes en al menos una función auxiliar, adecuadamente especificada, de la que también deberás proporcionar el código.

Problema 3.º (4 puntos)

Escribe el código de la función `buscarHueco`, encargada de buscar un hueco para una reunión de una hora de duración, de acuerdo con la información de una matriz de ocupación del tiempo de los empleados como la que se utiliza como parámetro en la función solicitada en el problema 2.º.

La función `buscarHueco` debe proporcionar el día y la hora a la que se puede programar dicha reunión, teniendo en cuenta, además del contenido de la matriz de ocupación de los empleados, las restricciones que aparecen en la introducción del enunciado: el día de la reunión tiene que ser igual o posterior al valor inicial del parámetro `dia` y menor o igual que 31; el día tiene que ser laborable, según el contenido de la tabla `laborables`; y la hora a la que se puede programar una reunión tiene que ser igual o posterior a la hora de entrada, anterior a la hora de salida y distinta a la hora de la comida.

En el caso de que no hubiera ningún hueco para programar una reunión que cumpla con todas las restricciones anteriores, al acabar la ejecución de la función `buscarHueco` el parámetro `hora` valdrá `HORA_NO_ENCONTRADA`.

El problema que tiene que resolver la función `buscarHueco` tiene la suficiente entidad como para requerir la descomposición del mismo en otros de menor tamaño. Se exige por tanto que te apoyes en al menos una función auxiliar, adecuadamente especificada, de la que también deberás proporcionar el código.

Solución al problema 1.º

```
/*
 * Pre:  Existe un fichero binario de nombre «NOMBRE_FICHERO_LABORABLES» que
 *       contiene una secuencia de al menos «MAX_DIAS» datos de tipo «bool» y
 *       un fichero de texto de nombre «NOMBRE_LISTA_FICHEROS» que
 *       contiene una secuencia nombres de ficheros de texto que cumplen con
 *       la sintaxis establecida por la regla <agenda> definida en
 *       el enunciado.
 * Post: Ha solicitado un día al usuario y ha informado en pantalla de cuándo
 *       podría celebrarse una reunión de una hora de duración a partir del día
 *       introducido por el usuario y hasta final de mes en la que todas
 *       las personas cuyos ficheros de agenda figuran en el fichero
 *       «NOMBRE_LISTA_FICHEROS» estén disponibles, de acuerdo con la
 *       información contenida en el fichero «NOMBRE_FICHERO_LABORABLES» y los
 *       listados en el fichero «NOMBRE_LISTA_FICHEROS». El día y hora elegidos
 *       cumplen las siguientes restricciones: el día es un día laborable, y la
 *       hora es posterior o igual a las 9:00 (hora de entrada), anterior a las
 *       18:00 (hora de salida) y distinta a las 14:00 (hora de la comida). En
 *       el caso de que no hubiera ningún hueco para programar una reunión que
 *       cumpliera con todas las restricciones anteriores entre el día
 *       proporcionado por el usuario y el final del mes, el programa ha
 *       mostrado un mensaje en la pantalla indicando dicha circunstancia.
 *       Si alguno de los ficheros no pudiera abrirse, el programa se ha
 *       limitado a informar del error.
 */
#include <cstring>
int main() {
    bool laborables[MAX_DIAS];
    if (leerFicheroDiasLaborables(laborables)) {
        int ocupacion[MAX_DIAS][NUM_HORAS];
        if (obtenerOcupacionTotal(ocupacion)) {
            int dia, hora;
            pedirDatosReunion(dia);
            buscarHueco(ocupacion, laborables, dia, hora);
            cout << endl;
            if (hora != HORA_NO_ENCONTRADA) {
                cout << "El␣primer␣hueco␣disponible␣para␣una␣reunión␣es␣el␣día␣"
                     << dia << "␣a␣las␣" << hora << ":00." << endl;
            }
            else {
                cout << "No␣hay␣ningún␣hueco␣disponible." << endl;
            }
```

```
        }
    }

    return 0;
}
```

Solución al problema 2.º

```
/*
 * Pre:  «ocupacion» es una matriz con al menos «MAX_DIAS» filas y exactamente
 *       «NUM_HORAS» columnas y «nombreFichero» es el nombre de un fichero de
 *       texto que sigue la sintaxis establecida por la regla <agenda> definida
 *       en el enunciado.
 * Post: Al ejecutarse esta función, las componentes de la matriz
 *       «ocupacion» que se corresponden con eventos recogidos en el
 *       fichero de agenda denominado «nombreFichero» han visto incrementados
 *       sus valores en una unidad, teniendo en cuenta que ocupacion[d][h]
 *       representa el número de personas en cuyas agendas hay un evento en el
 *       día d+1 que cubre el intervalo temporal entre las h:00 y h:59 horas.
 *       Ha devuelto true si la lectura del fichero denominado
 *       «nombreFichero» se ha producido sin problemas. En caso contrario, ha
 *       devuelto false y ha escrito un mensaje de error en la pantalla.
 */
bool actualizarOcupacion(int ocupacion[][NUM_HORAS],
                         const char nombreFichero[]) {
    ifstream f;
    f.open(nombreFichero);
    if (f.is_open()) {
        int dia, horaInicio, horaFin;
        f >> dia;
        while (!f.eof()) {
            f >> horaInicio >> horaFin;
            f.ignore(MAX_LONG_DESCRIPCION, '\n');
            for (int hora = horaInicio; hora < horaFin; hora++) {
                ocupacion[dia - 1][hora] += 1;
            }
            f >> dia;
        }
        f.close();
        return true;
    }
    else {
        cerr << "Error al abrir el fichero \"" << nombreFichero << "\"" << endl;
```

```
        return false;
    }
}

/*
 * Pre:  Existe un fichero de texto de nombre «NOMBRE_LISTA_FICHEROS» que
 *       contiene una secuencia nombres de ficheros de texto que cumplen con
 *       la sintaxis establecida por la regla <agenda> definida en
 *       el enunciado y «ocupacion» es una matriz con al menos «MAX_DIAS»
 *       filas y exactamente «NUM_HORAS» columnas.
 * Post: Las componentes ocupacion[d][h] de la matriz «ocupacion» que se
 *       corresponden con eventos recogidos en los ficheros de agenda listados
 *       en el fichero «NOMBRE_LISTA_FICHEROS» tienen un valor igual al número
 *       de agendas en las que hay un evento en el día d+1 que cubre el
 *       intervalo temporal entre las h:00 y h:59 horas. Las componentes valen
 *       0 cuando no se corresponden con ningún evento en ninguna de las
 *       agendas. Ha devuelto true si las lecturas del fichero denominado
 *       «NOMBRE_LISTA_FICHEROS» y de todos en él contenidos se han producido
 *       sin problemas. En caso contrario, ha devuelto false y ha escrito un
 *       mensaje de error en la pantalla.
 */
bool obtenerOcupacionTotal(int ocupacion[][NUM_HORAS]) {
    // Inicialización a 0 de todas las componentes de la tabla «ocupacion»
    for (int dia = 1; dia <= MAX_DIAS; dia++) {
        for (int hora = 0; hora < NUM_HORAS; hora++) {
            ocupacion[dia - 1][hora] = 0;
        }
    }

    // Procesamiento de las agendas que figuran en «NOMBRE_LISTA_FICHEROS»
    ifstream f;
    f.open(NOMBRE_LISTA_FICHEROS);
    if (f.is_open()) {
        bool todosFicherosOK = true;
        char nombreFichero[MAX_LONG_NOMBRE_FICHERO];
        f.getline(nombreFichero, MAX_LONG_NOMBRE_FICHERO);
        while (!f.eof() && todosFicherosOK) {
            todosFicherosOK = actualizarOcupacion(ocupacion, nombreFichero);
            f.getline(nombreFichero, MAX_LONG_NOMBRE_FICHERO);
        }
        f.close();
        return todosFicherosOK;
    }
    else {
```

```
        cerr << "Error al abrir el fichero \"" << NOMBRE_LISTA_FICHEROS
             << "\"" << endl;
        return false;
    }
}
```

Solución al problema 3.º

```
/*
 * Pre:  «ocupacion» es una matriz con al menos «MAX_DIAS» filas y exactamente
 *       «NUM_HORAS» columnas y «dia» está comprendido entre 1 y «MAX_DIAS».
 * Post: Ha devuelto el valor de la hora correspondiente al primer hueco
 *       disponible en la matriz «ocupacion» correspondiente al día «dia» en el
 *       que se puede celebrar una reunión. El valor devuelto cumple, además,
 *       estar en una hora igual o posterior a «HORA_ENTRADA», anterior a
 *       «HORA_SALIDA» y ser distinto a
 *       «HORA_COMIDA».
 *       Si no hay ninguna hora que cumpla con todas las condiciones anteriores,
 *       ha devuelto «HORA_NO_ENCONTRADA».
 */
int buscarHuecoEnUnDia(const int ocupacion[][NUM_HORAS], const int dia) {
    bool encontrado = false;
    int hora = HORA_ENTRADA;
    while (!encontrado && hora < HORA_SALIDA) {
        if (ocupacion[dia - 1][hora] == 0 && hora != HORA_COMIDA) {
            encontrado = true;
        }
        else {
            hora++;
        }
    }
    if (encontrado) {
        return hora;
    }
    else {
        return HORA_NO_ENCONTRADA;
    }
}

/*
 * Pre:  «ocupacion» es una matriz con al menos «MAX_DIAS» filas y exactamente
 *       «NUM_HORAS» columnas; «laborables» es una tabla con al menos «MAX_DIAS»
 *       componentes (laborables[i] es true  si y solo si el día i+1 es
```

```
 *       laborable), «dia» está inicialmente entre 1 y 31 y el valor de «hora»
 *       no está definido.
 * Post: Tras acabar la ejecución, se ha buscado el primer hueco disponible a
 *       partir del valor inicial del parámetro «dia», y los parámetros «dia»
 *       y «hora» se han modificado para indicar cuál es ese primer hueco. El
 *       hueco tiene que cumplir, además, estar en una hora igual o posterior
 *       a «HORA_ENTRADA», anterior a «HORA_SALIDA» y ser distinto a
 *       «HORA_COMIDA». El día del hueco, además, tiene que ser laborable,
 *       según la información de la tabla «laborables».
 *       Si no había ningún hueco que cumpla con las condiciones anteriores a
 *       partir del valor inicial del parámetro «dia», el valor del parámetro
 *       «hora» se ha establecido en «HORA_NO_ENCONTRADA».
 */
void buscarHueco(const int ocupacion[][NUM_HORAS], const bool laborables[],
                 int& dia, int& hora) {
    hora = HORA_NO_ENCONTRADA;
    while (hora == HORA_NO_ENCONTRADA && dia <= MAX_DIAS) {
        if (laborables[dia - 1]) {
            hora = buscarHuecoEnUnDia(ocupacion, dia);
        }
        if (hora == HORA_NO_ENCONTRADA) {
            dia++;
        }
    }
}
```