Examen escrito de Programación 1

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

2 de septiembre de 2015

- 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.
- El tiempo total previsto para realizar el examen es de **tres 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.*

Problema 1.º (3 puntos)

Un número entero positivo se dice *pandigital* si, expresado en base 10, contiene exactamente una vez todos los dígitos del 1 al 9 y no contiene ninguna el número 0. Por ejemplo, el número 546321987 es un número pandigital. No son números pandigitales el 14725836393 (porque el dígito 3 aparece más de una vez), el 7481590263 (porque contiene el dígito 0); ni el 12435768 (porque el dígito 9 no aparece ninguna vez).

Se debe escribir el código de la función `esPandigital` cuya especificación se muestra a continuación:

```
/*
 * Pre:  n > 0
 * Post: Ha devuelto true si y solo si n es pandigital en base 10, es
 *       decir, si no contiene la cifra 0 y contiene exactamente una
 *       vez todas las demás cifras entre 1 y 9.
 */
bool esPandigital(int n);
```

Problema 2.º (3 puntos)

El sistema de facturación de una determinada compañía telefónica utiliza un módulo de biblioteca denominado `resumenLlamadas` cuyo objetivo es el de definir un tipo registro que permita representar, al emitir la factura de cada abonado, el número total de veces que un determinado número de teléfono ha sido llamado por dicho abonado durante ese mes, y el importe total de esas llamadas (expresado en céntimos de euro).

A continuación se presenta el contenido del fichero de interfaz del módulo de biblioteca `resumenLlamadas`:

```
/* Fichero resumenLlamadas.h de interfaz del módulo «resumenLlamadas» */

/* Un dato de tipo ResumenLlamadas representa el número total de veces que un
 * determinado número de teléfono ha sido llamado por un determinado abonado
 * durante un mes, y el importe total de las llamadas (expresado en céntimos
 * de euro) ese número durante ese mes.
 */
struct ResumenLlamadas {
    int telefonoLlamado;    // Número de teléfono llamado
    int numeroLlamadas;     // Número de llamadas a «telefonoLlamado»
    int importeLlamadas;    // Importe total de las llamadas realizadas a
                            // «telefonoLlamado», en céntimos de euro
};

/*
 * Pre:  ---
 * Post: Ha devuelto un registro de tipo ResumenLlamadas que representa
 *       0 llamadas al teléfono «telefonoLlamado», por un importe
 *       de 0 céntimos de euro.
 */
ResumenLlamadas establecerResumenLlamadas(int telefonoLlamado);
```

```
/*
 * Pre:  ---
 * Post: Ha devuelto el número de teléfono correspondiente al registro
 *       «resumen».
 */
int telefonoLlamado(ResumenLlamadas resumen);

/*
 * Pre:  ---
 * Post: Ha devuelto el número de llamadas realizadas al número de teléfono
 *       correspondiente al registro «resumen», según la
 *       información almacenada en este.
 */
int numeroLlamadas(ResumenLlamadas resumen);

/*
 * Pre:  ---
 * Post: Ha devuelto, expresado en céntimos de euro, el importe de las
 *       llamadas realizadas al número de teléfono correspondiente al
 *       registro «resumen», según la información almacenada en este.
 */
int importeLlamadas(ResumenLlamadas resumen);

/*
 * Pre:  importe >= 0
 * Post: Ha modificado el registro «resumen» de dos formas:
 *        - incrementado en una unidad el número de llamadas realizadas al
 *          número de teléfono correspondiente al registro «resumen»
 *        - incrementando el importe total de las llamadas realizadas a dicho
 *          número de teléfono en la cantidad de «importe» céntimos de euro.
 */
void incrementarLlamadas(ResumenLlamadas& resumen, int importe);
```

```
/*
 * Pre:  Las primeras «numResumenes» componentes de la tabla «resumenes»
 *       almacenan datos válidos de resúmenes de llamadas realizadas a números
 *       de teléfono diferentes y «numResumenes» >= 0.
 * Post: Si entre las primeras «numResumenes» componentes de la tabla
 *       «resumenes» hay una que representa un resumen de llamadas al teléfono
 *       «telefono», ha incrementado en una unidad el número de llamadas a
 *       dicho número y en «importe» céntimos de euro el importe total de las
 *       llamadas realizads a dicho número.
 *       En caso contrario, habrá colocado en la siguiente componente de
 *       «resumenes» un resumen inicial de las llamadas a «telefono», con una
 *       primera llamada por valor de «importe» céntimos de euro.
 *       Al finalizar, «numResumenes» indica el número de resúmenes válidos
 *       que hay en las primeras componentes de la tabla tras contabilizar la
 *       llamada, es decir, el mismo que cuando se invocó a la función si ya
 *       había un resumen para el número «telefono», o incrementado en 1 en
 *       caso contrario.
 */
void contabilizarLlamada(ResumenLlamadas resumenes[], int& numResumenes,
                         int telefono, int importe);
```

Se pide el código de las siguientes (y solo las siguientes) funciones del módulo `resumenLlamadas`:

- `incrementarLlamadas`.
- `contabilizarLlamada`.

Problema 3.º (4 puntos)

El operador telefónico mencionado en el problema anterior gestiona, por cada abonado, un fichero de tarifas asociado a dicho abonado y un conjunto de ficheros mensuales en los que se almacenan los datos de cada llamada realizada por el abonado durante el mes correspondiente.

El **fichero de tarifas** del abonado es un **fichero de texto** y cada línea corresponde con la tarifa específica aplicable a cada llamada a un operador concreto que tiene contratada el abonado.

La línea comienza con un código numérico comprendido entre 0 y 99, que identifica el operador al que corresponde la tarifa que sigue en la línea. Los dos siguientes datos enteros son el coste de establecimiento y el precio por minuto de la llamada expresados en céntimos de euro. Cada dato está separado al menos por un espacio en blanco. La sintaxis del fichero expresada en notación de Backus-Naur es la siguiente:

```
<fichero_tarifas> ::= { <tarifa> fin_de_línea }
<tarifa> ::= <operador> <separador> <establecmnto> <separador> <precio_min>
<operador> ::= literal_entero
<establecmnto> ::= literal_entero
<precio_min> ::= literal_entero
<separador> ::= "␣" {"␣"}
```

A continuación se muestra el contenido de un fichero de texto de tarifas a modo de ejemplo:

```
10  18  15
 0   0   0
 1  20  15
11  20   0
 2  18  15
22   0  15
 3   9   8
 4  60   0
 5  20   0
 6  20  15
66  20   0
 7  10   8
```

Los **ficheros de llamadas** mensuales de un abonado son **ficheros binarios**, que almacenan una secuencia de datos que representan llamadas de acuerdo con la siguiente estructura expresada con notación de Backus-Naur:

```
<fichero_llamadas_mes> ::= { <llamada> }
<llamada> ::= <número_destino> <código_operador> <duración_llamada>
<número_destino> ::= int
<código_operador> ::= int
<duración_llamada> ::= int
```

La duración de la llamada está expresada en segundos.

A modo de ejemplo, se muestra el contenido de un fichero mensual de llamadas, en el que, para facilitar la comprensión del mismo en este enunciado, los datos se han representado en base diez, se han agrupado con llaves y han sido separados por espacios en blanco y comas:

```
{ {976468379, 10,  67}, {678453955,  0, 125}, {976248036, 10,  60},
  {976284650, 22,   2}, {976543315,  1, 109}, {678593724, 11, 125},
  {976905676, 11,  82}, {678453955,  0, 120}, {976543315,  1, 167},
  {678453955,  0, 143}, {976905676, 11, 130}, {976905676,  6, 167},
  {678453955,  0,  61}, {976284650, 22, 137}, {976905676, 10,  32},
  {976905676, 11,  37}, {678453955,  0, 147}, {976660569, 11, 168},
  {976727800,  2,  73}, {976727800, 22,  99}, {678453955,  0,  49},
  {976727800,  2, 124}, {976727800,  2,  13} }
```

Para la solución del problema que se plantea más adelante, puede utilizarse todo el contenido del módulo de biblioteca `resumenLlamadas` definido en el problema anterior, con independencia de que se haya solicitado o no implementar su código en este examen. También puede considerarse que se dispone de un módulo de biblioteca de nombre `tarifa`, cuyo fichero de interfaz `tarifa.h` se reproduce a continuación. El fichero de implementación `tarifa.cc` ya ha sido desarrollado y, por tanto, los tipos y funciones que aparecen a continuación pueden utilizarse sin necesidad de escribir su código.

```
/* Fichero tarifa.h de interfaz del módulo «tarifa» */

/* El tipo Tarifa permite representar las distintas tarifas que una compañía
 * telefónica ofrece a sus clientes. En estos momentos, cada tarifa está
 * definida únicamente por el coste de establecimiento de llamada y por el
 * precio por minuto que tiene cada llamada (expresado en céntimos de euro).
 */
struct Tarifa {
    int costeEstablecimiento;
    int precioMinuto;
};

/*
 * Pre:  costeEstablecimiento >= 0 y precioMinuto >= 0
 *       Ambos valores están expresados en céntimos de euro.
 * Post: Ha definido la tarifa «t» con un coste de establecimiento de llamada
 *       de «costeEstablecimiento» céntimos de euro y con un precio por minuto
 *       de «precioMinuto» céntimos de euro.
 */
void establecerTarifa(Tarifa& t, int costeEstablecimiento, int precioMinuto);
```

```
/*
 * Pre:  ---
 * Post: Ha devuelto el coste de establecimiento de llamda correspondiente a
 *       la tarifa «t», expresado en céntimos de euro.
 */
int costeEstablecimiento(Tarifa t);

/*
 * Pre:  ---
 * Post: Ha devuelto el precio por minuto correspondiente a la tarifa «t»,
 *       expresado en céntimos de euro.
 */
int precioMinuto(Tarifa t);

/*
 * Pre:  duracion >= 0
 * Post: Ha devuelto el valor, expresado en céntimos de euro, de una llamada
 *       de «duracion» segundos de duración, según los costes y precios
 *       definidos por la tarifa «t».
 */
double importeLlamada(Tarifa t, int duracion);
```

Se pide la escritura de un programa que solicite al operador el nombre del fichero de tarifas de un determinado abonado y el nombre de un fichero mensual del mismo abonado y que, a continuación, muestre en la pantalla un listado con los números de teléfono llamados por el abonado, el número de veces que ha sido llamado y el coste total de las llamadas a cada número de teléfono.

En el listado, cada número de teléfono distinto aparecerá únicamente una vez. El orden en el que aparezcan es irrelevante. Se muestra a continuación un ejemplo de la ejecución del programa en la que los contenidos de los ficheros introducidos se corresponden con los de los ejemplos anteriores:

```
Escriba el nombre del fichero de tarifas del abonado: tarifas.txt
Escriba el nombre del fichero mensual de llamadas del abonado: llamadas.dat

FACTURA

 Teléfono  Nº llamadas  Importe total
---------  -----------  -------------
976468379            1           0.34
678453955            6           0.00
976248036            1           0.33
976284650            2           0.34
976543315            2           1.08
678593724            1           0.20
976905676            5           1.47
976660569            1           0.20
976727800            4           1.30

Importe total: 5.26
```

Se garantiza que todos los códigos de operador que aparecen en los ficheros de llamadas también aparecen en los ficheros de tarifas.

En la solución presentada se exigirá, además de su corrección, las siguientes propiedades:

1. Independencia del código de la forma en que hayan sido definidos los tipos `ResumenLlamadas` y `Tarifa`.
2. Diseño descendente.
3. Legibilidad del código.
4. Minimización del esfuerzo de desarrollo por haber hecho uso adecuado de los módulos facilitados.

Solución al problema 1.º

```
/*
 * Pre:  n > 0
 * Post: Ha devuelto true si y solo si n es pandigital en base 10, es decir,
 *       si no contiene la cifra 0 y contiene exactamente una vez todas las
 *       demás cifras entre 1 y 9.
 */
bool esPandigital(int n) {
    // Definición de la base en la que se va a trabajar
    const int BASE = 10;

    // Definición e inicialización de una tabla para contar cuántas veces
    // aparece cada dígito
    int vecesDigito[BASE];
    for (int i = 0; i < BASE; i++) {
        vecesDigito[i] = 0;
    }

    // Exploración de los dígitos del número y actualización del número de
    // veces que han aparecido
    while (n != 0) {
        int ultimoDigito = n % BASE;
        vecesDigito[ultimoDigito]++;
        n = n / BASE;
    }

    // Determinación del resultado de la función: comprobación de que el
    // dígito 0 no ha aparecido y de que el resto han aparecido exactamente
    // una vez (esquema de búsqueda lineal sin garantía de éxito)
    bool estanVecesOK = (vecesDigito[0] == 0);
    int i = 1;
    while (estanVecesOK && i < BASE) {
        estanVecesOK = estanVecesOK && vecesDigito[i] == 1;
        i++;
    }
    return estanVecesOK;
}
```

Solución al problema 2.º

```
/*
 * Pre:  importe >= 0
 * Post: Ha modificado el registro «resumen» de dos formas:
 *        - incrementado en una unidad el número de llamadas realizadas al
 *          número de teléfono correspondiente al registro «resumen»
 *        - incrementando el importe total de las llamadas realizadas a dicho
 *          número de teléfono en la cantidad de «importe» céntimos de euro.
 */
void incrementarLlamadas(ResumenLlamadas& resumen, int importe) {
    resumen.numeroLlamadas++;
    resumen.importeLlamadas += importe;
}

/*
 * Pre:  Las primeras «numResumenes» componentes de la tabla «resumenes»
 *       almacenan datos válidos de resúmenes de llamadas realizadas a números
 *       de teléfono diferentes y «numResumenes» >= 0.
 * Post: Si entre las primeras «numResumenes» componentes de la tabla
 *       «resumenes» hay una que representa un resumen de llamadas al teléfono
 *       «telefono», ha incrementado en una unidad el número de llamadas a
 *       dicho número y en «importe» céntimos de euro el importe total de las
 *       llamadas realizads a dicho número.
 *       En caso contrario, habrá colocado en la siguiente componente de
 *       «resumenes» un resumen inicial de las llamadas a «telefono», con una
 *       primera llamada por valor de «importe» céntimos de euro.
 *       Al finalizar, «numResumenes» indica el número de resúmenes válidos
 *       que hay en las primeras componentes de la tabla tras contabilizar la
 *       llamada, es decir, el mismo que cuando se invocó a la función si ya
 *       había un resumen para el número «telefono», o incrementado en 1 en
 *       caso contrario.
 */
void contabilizarLlamada(ResumenLlamadas resumenes[], int& numResumenes,
                         int telefono, int importe) {
    // Búsqueda de un resumen correspondiente a «telefono»
    int i = 0;
    bool encontrado = false;
    while (!encontrado && i < numResumenes) {
        if (telefonoLlamado(resumenes[i]) == telefono) {
            encontrado = true;
        }
        else {
```

```
            i++;
        }
    }
    // encontrado || i >= numResumenes

    // Discriminación del resultado de la búsqueda
    if (encontrado) {
        // En la componente «i» de la tabla «resumenes» hay una
        // correspondiente a «telefonoLlamado»: se incrementa el número de
        // llamadas al teléfono y no se aumenta el número de resúmenes.
        incrementarLlamadas(resumenes[i], importe);
    }
    else {
        // Entre las primeras «numResumenes» componentes de la tabla
        // «resumenes» no hay ninguna que represente un resumen de llamadas
        // al teléfono «telefonoLlamado»: se añade un nuevo resumen en la
        // componente «numResumenes» y se aumenta el número de resúmenes.
        resumenes[numResumenes] = establecerResumenLlamadas(telefono);
        incrementarLlamadas(resumenes[numResumenes], importe);
        numResumenes++;
    }
}
```

Solución al problema 3.º

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include "tarifa.h"
#include "resumenLlamadas.h"
using namespace std;

// Número máximo de operadores (establecido en enunciado)
const int MAX_OPERADORES = 100;

// Estimación del número máximo de llamadas de un abonado en un mes
const int MAX_LLAMADAS = 10000;



/*
 * Pre:  «nombreFicheroTarifas» representa el nombre de un fichero existente
 *       de tarifas con el formato establecido en el enunciado del examen.
 * Post: Ha leído el fichero de tarifas de nombre «nombreFicheroTarifas» y en
```

```
 *       las componentes de la tabla
 *       «tarifas» indexadas por cada código de tarifa leído del fichero ha
 *       establecido la tarifa correspondiente, definida por los datos del
 *       fichero que acompañan a cada código de tarifa leído.
 */
void leerTarifas(const char nombreFicheroTarifas[], Tarifa tarifas[]) {
    ifstream f(nombreFicheroTarifas);
    if (f.is_open) {
        // Intenta empezar a leer la primera tarifa
        int operador, establecimiento, precioMinuto;
        f >> operador;
        while (!f.eof()) {
            // Termina de leer la tarifa
            f >> establecimiento;
            f >> precioMinuto;

            // Almacena la tarifa al operador «operador» en la componente
            // indexada por «operador» de la tabla «tarifas», estableciéndola
            // a través de la función «establecerTarifa» del módulo «tarifa».
            establecerTarifa(tarifas[operador], establecimiento, precioMinuto);

            // Intenta empezar a leer la siguiente tarifa
            f >> operador;
        }
        f.close();
    }
}

/*
 * Pre:  nombreFicheroLlamadas representa el nombre de un fichero binario de
 *       llamadas existente y con el formato establecido en el enunciado. Los
 *       códigos de tarifas que aparecen en el fichero de llamadas tienen
 *       establecida una tarifa en la componente de la tabla «tarifas»
 *       indexada por el código de la tarifa.
 * Post: Ha leído el contenido del fichero de llamadas de nombre
 *       «nombreFicheroLlamadas» y, por cada llamada leída, ha calculado su
 *       importe de acuerdo con la tarifa asociada a la llamada leída y el
 *       contenido de la tabla «tarifas» y la ha contabilizado en la tabla
 *       «resumenes», incrementando el valor de «numResumenes» si ha sido
 *       necesario.
 */
void leerFacturas(const Tarifa tarifas[],
                  const char nombreFicheroLlamadas[],
                  ResumenLlamadas resumenes[], int& numResumenes) {
```

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

        // Intenta leer la primera llamada
        int telefono, operador, duracion;
        f.read(reinterpret_cast<char*>(&telefono), sizeof(telefono));

        while (!f.eof()) {
            // Termina de leer la llamada
            f.read(reinterpret_cast<char*>(&operador), sizeof(operador));
            f.read(reinterpret_cast<char*>(&duracion), sizeof(duracion));

            // Procesa la ultima llamada leída
            int importe = importeLlamada(tarifas[operador], duracion);
            contabilizarLlamada(resumenes, numResumenes, telefono, importe);

            // Intenta empezar a leer la siguiente llamada
            f.read(reinterpret_cast<char*>(&telefono), sizeof(telefono));
        }

        f.close();
    }
}

/*
 * Pre:  0 <= numResumenes < MAX_LLAMADAS
 * Post: Ha escrito en la pantalla un listado de los resúmenes de llamadas
 *       almacenados en las primeras «numResumenes» componentes de la tabla
 *       «resumenes», así como el importe total de las llamadas, de acuerdo
 *       con el formato que se muestra en el siguiente ejemplo:
 *
 *           FACTURA
 *
 *            Teléfono  Nº llamadas  Importe total
 *           ---------  -----------  -------------
 *           976468379            1           0.34
 *           678453955            6           0.00
 *           976248036            1           0.33
 *           976284650            2           0.34
 *           976543315            2           1.08
 *           678593724            1           0.20
 *           976905676            5           1.47
 *           976660569            1           0.20
 *           976727800            4           1.30
```

```
 *
 *           Importe total: 5.26
 */
void escribirFactura(ResumenLlamadas resumenes[], int numResumenes) {
    // Escritura de la cabecera
    cout << endl;
    cout << "FACTURA" << endl;
    cout << endl;
    cout << "␣Teléfono␣␣Nº␣llamadas␣␣Importe␣total" << endl;
    cout << "---------␣␣----------␣␣-------------" << endl;
    cout << fixed << setprecision(2);

    int total = 0;
    for (int i = 0; i < numResumenes; i++) {
        cout << setw(9)  << telefonoLlamado(resumenes[i]) << "␣␣"
             << setw(11) << numeroLlamadas(resumenes[i]) << "␣␣"
             << setw(13) << importeLlamadas(resumenes[i]) / 100.0 << endl;
        total += importeLlamadas(resumenes[i]);
    }
    cout << endl;
    cout << "Importe␣total:␣" << total / 100.0 << endl;
}


/*
 * Pre:  ---
 * Post: El programa ha solicitado al operador el nombre del fichero de
 *       tarifas de un determinado abonado y el nombre de un fichero mensual
 *       del mismo abonado y, a continuación, ha mostrado en la pantalla un
 *       listado con los números de teléfono llamados por el abonado, el
 *       número de veces que ha sido llamado y el coste total de las llamadas
 *       a cada número de teléfono. En el listado, cada número de teléfono
 *       distinto ha aparecido únicamente una vez. Los ficheros de tarifas y
 *       de llamadas cuyo nombre ha introducido el usuario tienen los formatos
 *       establecidos en el enunciado del examen.
 *       La interacción con el usuario y el formato de los resultados se
 *       muestran en el siguiente ejemplo de ejecución:
 *
 *   Escriba el nombre del fichero de tarifas del abonado: tarifas.txt
 *   Escriba el nombre del fichero mensual de llamadas del abonado: llamadas.dat
 *
 *   FACTURA
 *
 *    Teléfono  Nº llamadas  Importe total
```

```
 *   ---------  -----------  -------------
 *   976468379            1           0.34
 *   678453955            6           0.00
 *   976248036            1           0.33
 *   976284650            2           0.34
 *   976543315            2           1.08
 *   678593724            1           0.20
 *   976905676            5           1.47
 *   976660569            1           0.20
 *   976727800            4           1.30
 *
 *   Importe total: 5.26
 *
 */
int main() {
    // Petición y lectura de los nombres de los ficheros
    cout << "Escriba␣el␣nombre␣del␣fichero␣de␣tarifas␣del␣abonado:␣" << flush;
    char nombreFicheroTarifas[80];
    cin >> nombreFicheroTarifas;

    cout << "Escriba␣el␣nombre␣del␣fichero␣mensual␣de␣llamadas␣del␣abonado:␣"
         << flush;
    char nombreFicheroLlamadas[80];
    cin >> nombreFicheroLlamadas;

    // Declaración de las estructuras de datos necesarias para almacenar las
    // tarifas y el resumen de llamadas
    Tarifa tarifas[MAX_OPERADORES];
    ResumenLlamadas resumenes[MAX_LLAMADAS];
    int numResumenes = 0;

    // Lectura de las tarifas
    leerTarifas(nombreFicheroTarifas, tarifas);

    // Lectura de las llamadas para generar el resumen
    leerFacturas(tarifas, nombreFicheroLlamadas, resumenes, numResumenes);

    // Escritura de la factura en la pantalla
    escribirFactura(resumenes, numResumenes);
}
```