LENGUAJES DE PROGRAMACIÓN

Solución al Trabajo Práctico - Septiembre de 2020

EJERCICIO 1

Consideremos el polinomio de grado n siguiente

$$p_n(x) = a_0 + a_1 \cdot x + \cdots + a_n \cdot x^n$$

cuyos coeficientes a_0, a_1, $\ldots$, a_n son números reales.

Escriba un programa en C++ que permita al usuario especificar el orden y el valor de los coeficientes del polinomio, y que calcule y escriba en la consola los valores obtenidos de evaluar el polinomio para un determinado conjunto de valores de x.

El programa debe realizar las acciones siguientes:

1. Escribir un mensaje en la consola indicando al usuario que introduzca el orden del polinomio.
2. Leer el valor introducido por el usuario, almacenándolo en una variable de tipo entero llamada `n`. Si el valor de n es menor que cero, terminar.
3. Escribir un mensaje en la consola solicitando al usuario que introduzca por consola el valor de los $n + 1$ coeficientes a_0, a_1, $\ldots$, a_n. Los valores deben ser leídos de la consola y almacenados en un vector de tipo **double** llamado `coefs`.
4. Calcular y escribir en la consola, en formato científico con 8 dígitos de precisión, el valor del polinomio para cada uno de los valores de x siguientes:

$$x_k = \cos\left(\frac{2 \cdot \pi \cdot k}{N}\right) \qquad \text{con} \quad k = 0, \ldots, N-1$$

donde N es una constante global del programa cuyo valor es 10.

5. Terminar.

Solución al Ejercicio 1

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>

const int N = 10;
const double PI = std::acos(-1);

int main() {
      // Entrada por consola
      std::cout << "Orden del polinomio:" << std::endl;
      int n;
      std::cin >> n;
      if (n<0) return 0;
      std::cout << "Introduzca los coeficientes:"
                << std::endl;
      std::vector<double> coefs(n+1,0);
      for (int i=0; i<=n; i++)
            std::cin >> coefs[i];
      // Cálculo del valor del polinomio en x0, ..., xN
      for (unsigned int k=0; k<N; k++) {
            double xk = std::cos(2*PI*k/N);
            double b = coefs[n];
            for (int i = n - 1; i >= 0; i--)
                  b = b * xk + coefs[i];
            // Salida por consola
            std::cout << "p(" << xk << ") = " <<
                        std::setprecision(8) << std::scientific <<
                        b << std::endl;
      }
      return 0;
}
```

EJERCICIO 2

Consideremos una magnitud física escalar (por ejemplo, la temperatura), que llamaremos U, dependiente de la coordenada espacial x y del tiempo t. El valor de U en cada punto (x, t), tal que $0 < x < 1$ y $t > 0$, puede calcularse resolviendo la ecuación de difusión en una dimensión:

$$\frac{\partial^2 U}{\partial x^2} = \frac{\partial U}{\partial t}, \qquad 0 < x < 1, \quad t > 0$$

con las condiciones de contorno:

$$U(0, t) = 0 = U(1, t), \qquad t > 0$$

y la condición inicial:

$$U(x, 0) = 100, \qquad 0 < x < 1$$

Nuestro objetivo es estimar el valor de U en un determinado punto (x^*, t^*). Para ello, discretizamos la coordenada espacial en el intervalo $[0, 1]$:

$$x_i = \frac{i}{I} \qquad \text{con} \qquad i = 0, 1, \ldots, I$$

donde I es un número natural, y discretizamos el tiempo:

$$t_n = n \cdot \Delta t \qquad \text{con} \qquad n = 0, 1, \ldots$$

donde el paso de avance en el tiempo, Δt, es un número real mayor que cero.

El punto (x^*, t^*) debe ser uno de los puntos de la discretización:

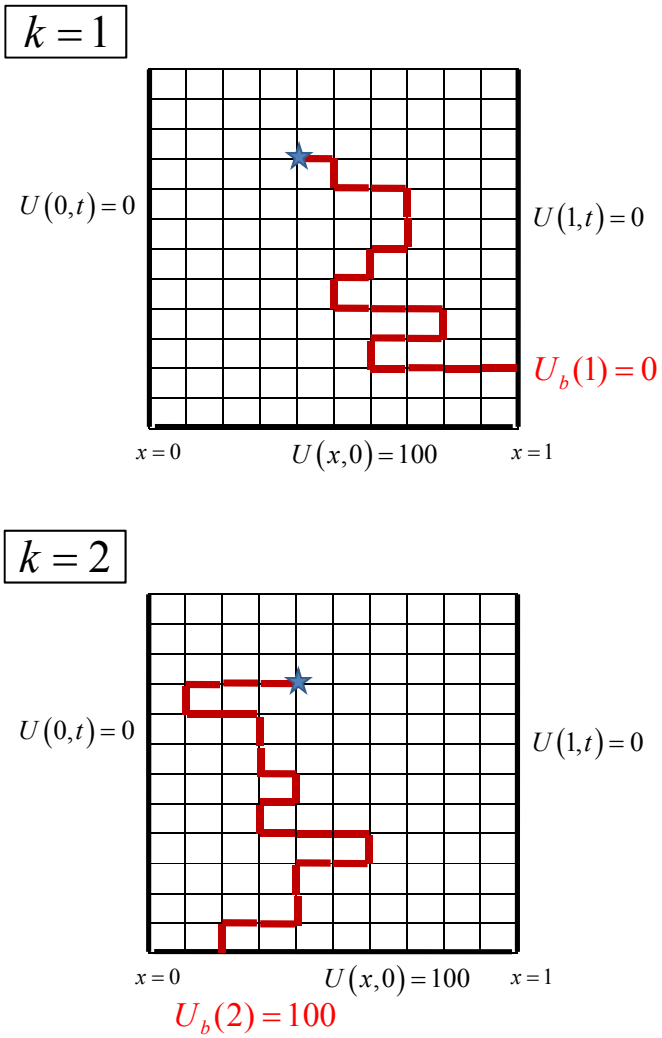
$$x^* = \frac{i^*}{I}, \qquad t^* = n^* \cdot \Delta t$$

donde i^* y n^* son números enteros mayores que cero, y además de satisface $i^* < I$.

El algoritmo que emplearemos para estimar $U(x^*, t^*)$ es un método de Monte Carlo. Consiste en generar gran número de caminos aleatorios en el espacio bidimensional $x-t$ discretizado. Cada camino aleatorio comienza en (x^*, t^*) y termina cuando alcanza un punto cualquiera del contorno (x vale 0 ó 1) o del instante inicial (t vale 0). El algoritmo contabiliza el valor de U en el punto terminal del camino, ya que la media de dichos valores es una estimación de $U(x^*, t^*)$. El algoritmo es descrito a continuación.

Algoritmo

1. Seleccionar el punto (x^*, t^*).
2. Discretizar la coordenada espacial y el tiempo:
   a) Asignar un valor entero positivo a la constante I.
   b) Asignar
   $$\Delta t = \frac{1}{2 \cdot I^2}$$
   Obsérvese que de esta manera se satisface $\frac{(\Delta x)^2}{\Delta t} = 2$.
3. Asignar valor a N_C. Esta constante entera es el número de caminos aleatorios que recorrerá el algoritmo.
4. Asignar $k = 1$.
5. Situarse en el punto inicial del camino: $(x, t) = (x^*, t^*)$.
6. Avanzar un paso en el camino. Para ello, generar un número pseudoaleatorio, r, y emplearlo para decidir a qué punto nos movemos:
   $$\begin{array}{lllll} \text{de} & (x,t) & \text{a} & (x+\Delta x, t) & \text{si} \quad 0 \leq r < 0.25 \\ \text{de} & (x,t) & \text{a} & (x-\Delta x, t) & \text{si} \quad 0.25 \leq r < 0.5 \\ \text{de} & (x,t) & \text{a} & (x, t-\Delta t) & \text{si} \quad 0.5 \leq r \leq 1 \end{array}$$
   Obsérvese que el avance en el tiempo, de producirse, siempre es hacia atrás, nunca hacia adelante.
7. Si el punto al que nos hemos movido no pertenece ni al contorno, ni al instante inicial, ir al Paso 6 del algoritmo.
8. Asignar a $U_b(k)$ el valor que tiene U en el punto del contorno o inicial que hemos alcanzado en este camino aleatorio k-ésimo.
9. Si $k < N_C$, asignar $k = k + 1$ e ir al Paso 5 del algoritmo.
10. Calcular $U(x^*, t^*)$ de la forma siguiente:
    $$U(x^*, t^*) = \frac{1}{N_C} \cdot \sum_{k=1}^{N_C} U_b\,(k)$$
11. Terminar.



Como ilustración, sobre estas líneas se muestran dos caminos aleatorios. Obsérvese que cada camino aleatorio es una sucesión de nodos de la cuadrícula obtenida de discretizar la coordenada espacial (eje horizontal) y el tiempo (eje vertical, creciente hacia arriba). Todo camino comienza en el punto (x^*, t^*), que está señalado mediante una estrella en la figura, y termina cuando alcanza el contorno (como en el camino $k = 1$) o el instante inicial (como en el camino $k = 2$). El valor terminal del camino $k = 1$ es $U_b(1) = 0$ y el del camino $k = 2$ es $U_b(2) = 100$.

Escriba un programa en C++ que estime el valor de U empleando el algoritmo anterior y escriba dicho valor en la consola. Ejecútelo para estimar, empleando $I = 200$ y $N_C = 10000$, el valor de U en los dos puntos siguientes:

$$(x^* = 0.4,\ t^* = 0.04) \qquad\qquad (x^* = 0.4,\ t^* = 0.1)$$

Solución al Ejercicio 2

```
#include <iostream>
#include <cstdlib>      /* RAND_MAX, rand */
#include <time.h>       /* time */

const int Nc = 10000;         // Numero de caminos aleatorios
const int I  =   200;         // Numero de intervalos en x
const double delta_x = 1.0/I;           // Tamaño paso en x
const double delta_t = 1.0/(2*I*I); // Tamaño paso en t
// Condiciones de contorno e iniciales
const double U_x0 = 0;     // U(0,t)
const double U_x1 = 0;     // U(1,t)
const double U_t0 = 100; // U(x,0)

struct Punto {
      int i; // x = i * delta_x
      int n; // t = n * delta_t
};

int main() {
   srand(time(NULL));
   // Punto donde se quiere calcular U
   double x_ast = 0.4;
   double t_ast = 0.1;   // 0.04;

   double suma_Ub = 0;
   for (int k=0; k<Nc; k++) {
         // Situarse en el inicio del camino
         Punto p ={(int)(I*x_ast), (int)(t_ast/delta_t)};
         // Recorrer el camino
         while (p.i > 0 && p.i < I && p.n > 0) {
               double r = (double)rand() / RAND_MAX; // r en el rango 0 a 1;
               //std::cout << "r = " << r << std::endl;
               if (r < 0.25) {
                     p.i++;
               } else if ( r < 0.5 ) {
                     p.i--;
               } else {
                     p.n--;
               }
         }
         // Añadir a suma_Ub el valor de U en el punto terminal
         if (p.i <= 0)     {
               suma_Ub += U_x0;
         } else if (p.i >= I) {
               suma_Ub += U_x1;
         } else {
               suma_Ub += U_t0;
         }
   }
   std::cout << "U(" << x_ast << ", " << t_ast << ") = "
         << suma_Ub/Nc << std::endl;
return 0;
}
```

EJERCICIO 3

El método de Newton de las diferencias divididas es una técnica de interpolación polinómica. Se describe a continuación.

Supongamos que conocemos el valor de una función $f(x)$ en $n+1$ puntos diferentes.

$$\begin{array}{ll} x_0 & f(x_0) \\ x_1 & f(x_1) \\ \vdots & \\ x_i & f(x_i) \\ \vdots & \\ x_n & f(x_n) \end{array}$$

El polinomio de interpolación de grado n es el siguiente

$$\begin{aligned} p_n(x) \;=\; & f\,[x_0] \\ & +(x-x_0)\cdot f\,[x_1,x_0] \\ & +(x-x_0)\cdot(x-x_1)\cdot f\,[x_2,x_1,x_0] \\ & \dots \\ & +(x-x_0)\cdot(x-x_1)\cdot(x-x_2)\;\cdots\;(x-x_{n-1})\cdot f\,[x_n,x_{n-1},\dots,x_0] \end{aligned}$$

donde las diferencias divididas $f\,[\dots]$ se calculan de la forma indicada a continuación

Orden	Notación	Definición
0	$f\,[x_0]$	$f(x_0)$
1	$f\,[x_1,x_0]$	$\frac{f[x_1]-f[x_0]}{x_1-x_0}$
2	$f\,[x_2,x_1,x_0]$	$\frac{f[x_2,x_1]-f[x_1,x_0]}{x_2-x_0}$
3	$f\,[x_3,x_2,x_1,x_0]$	$\frac{f[x_3,x_2,x_1]-f[x_2,x_1,x_0]}{x_3-x_0}$
$\vdots$		
n	$f\,[x_n,x_{n-1},\dots,x_0]$	$\frac{f[x_n,x_{n-1},\dots,x_1]-f[x_{n-1},x_{n-2},\dots,x_0]}{x_n-x_0}$

A continuación se muestra un ejemplo. Supongamos que conocemos el valor de una función $f(x)$ en 4 puntos diferentes.

i	x_i	$f(x_i)$
0	0	-5
1	1	1
2	3	25
3	4	55

Se desea calcular el polinomio de interpolación de grado 3. Para ello, calculamos las diferencias divididas.

$$\begin{aligned}
f[x_0] &= f(x_0) &&= -5\\
f[x_1] &= f(x_1) &&= 1\\
f[x_2] &= f(x_2) &&= 25\\
f[x_3] &= f(x_3) &&= 55\\
f[x_3,x_2] &= \tfrac{f[x_3]-f[x_2]}{x_3-x_2} = \tfrac{55-25}{3-2} &&= 30\\
f[x_2,x_1] &= \tfrac{f[x_2]-f[x_1]}{x_2-x_1} = \tfrac{25-1}{3-1} &&= 12\\
f[x_1,x_0] &= \tfrac{f[x_1]-f[x_0]}{x_1-x_0} = \tfrac{1-(-5)}{1-0} &&= 6\\
f[x_3,x_2,x_1] &= \tfrac{f[x_3,x_2]-f[x_2,x_1]}{x_3-x_1} = \tfrac{30-12}{4-1} &&= 6\\
f[x_2,x_1,x_0] &= \tfrac{f[x_2,x_1]-f[x_1,x_0]}{x_2-x_0} = \tfrac{12-6}{3-0} &&= 2\\
f[x_3,x_2,x_1,x_0] &= \tfrac{f[x_3,x_2,x_1]-f[x_2,x_1,x_0]}{x_3-x_0} = \tfrac{6-2}{4-0} &&= 1
\end{aligned}$$

El polinomio de interpolación de grado 3 se muestra a continuación.

$$\begin{aligned}
p_3(x) &= f[x_0]\\
&\quad +(x-x_0)\cdot f[x_1,x_0]\\
&\quad +(x-x_0)\cdot(x-x_1)\cdot f[x_2,x_1,x_0]\\
&\quad +(x-x_0)\cdot(x-x_1)\cdot(x-x_2)\cdot f[x_3,x_2,x_1,x_0]\\
&= -5+x\cdot 6+x\cdot(x-1)\cdot 2+x\cdot(x-1)\cdot(x-3)
\end{aligned}$$

La interpolación para un determinado valor de x se calcula sustituyendo dicho valor en el polinomio. Por ejemplo, el valor interpolado en $x=0.5$ es

$$\begin{aligned}
p_3(0.5) &= -5+0.5\cdot 6+0.5\cdot(0.5-1)\cdot 2+0.5\cdot(0.5-1)\cdot(0.5-3)\\
&= -1.875
\end{aligned}$$

Escriba un programa en C++ que realice las acciones siguientes.

1. Abrir para lectura un fichero de texto llamado `puntos.txt`. Si se produce error, terminar.

2. Leer el contenido del fichero `puntos.txt`. En este fichero están escritos, en dos columnas, los valores x_i (primera columna) y $f(x_i)$ (segunda columna). Estos valores son, en general, números reales.

   El fichero `puntos.txt` correspondiente al ejemplo anterior es:

   ```
   0  -5
   1  1
   3  25
   4  55
   ```

   Los valores leídos deben almacenarse en un vector de estructuras `Punto`, la cual debe definirse de la forma siguiente:

   ```
   struct Punto {
      double x;
      double fx;
   };
   ```

3. Si el número de puntos es menor que dos, o si alguno de los puntos está repetido, mostrar un mensaje indicándolo y terminar.

4. Solicitar al usuario, mediante un mensaje en la consola, que introduzca el valor de x para el cual quiere evaluar el polinomio de interpolación.

5. Si el valor introducido es mayor que todos los valores x_i leídos del fichero o si es menor que todos ellos, mostrar un mensaje en la consola indicándolo y terminar.

6. Mostrar en la consola el valor obtenido de sustituir x en el polinomio de interpolación de grado n calculado de los valores leídos del fichero, donde el número de valores leídos es $n + 1$.

7. Terminar.

Muestre en la memoria el resultado obtenido al ejecutar su programa para cada uno de los dos ficheros `puntos.txt` siguientes, interpolando en $x = 0.5$.

Primer fichero de prueba

```
0  -5
1  1
3  25
4  55
```

Segundo fichero de prueba

```
0.0  1.000
0.2  0.980
0.3  0.956
0.4  0.921
0.6  0.825
0.7  0.765
0.9  0.622
1.0  0.540
```

Solución al Ejercicio 3

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

const std::string puntosFich = "puntos.txt";

struct Punto {
    double x;
    double fx;
};

double difDiv(unsigned int indSup, unsigned int indInf,
        std::vector<Punto> &vPuntos) {
    if ( indSup == indInf ) return vPuntos[indSup].fx;
    return ( difDiv(indSup,indInf+1, vPuntos) -
             difDiv(indSup-1,indInf, vPuntos) ) /
           (vPuntos[indSup].x - vPuntos[indInf].x);
}


int main() {
    // Abrir fichero para lectura
    std::ifstream inFich(puntosFich.c_str(), std::ios::in);
    if (!inFich) {
        std::cout << "Error al abrir el fichero" << std::endl;
        return 0;
    }
    // Lectura de fichero y comprobación de punto repetido
    std::vector<Punto> vPuntos;
    while ( !inFich.eof() ) {
        Punto punto;
        inFich >> punto.x;
        if (inFich.eof()) break;
        // Comprueba si el punto está repetido
        for (unsigned int i=0; i<vPuntos.size(); i++) {
            if (punto.x == vPuntos[i].x) {
                std::cout << "Punto repetido en x = "
                      << punto.x << std::endl;
                inFich.close();
                return 0;
            }
        }
        inFich >> punto.fx;
        vPuntos.push_back(punto);
    }
    inFich.close();
```

```
    // Comprueba si el número de puntos es menor que dos
    if (vPuntos.size() < 2) {
        std::cout << "Num. puntos menor que dos" << std::endl;
        return 0;
    }
     // Entrada por consola del valor de x donde interpolar
    double x;
    std::cout << "Introduzca x" << std::endl;
    std::cin >> x;
    // Comprobación del valor de x
    double valorMax = vPuntos[0].x;
    double valorMin = vPuntos[0].x;
    for (unsigned int i=1; i<vPuntos.size(); i++) {
        if (vPuntos[i].x > valorMax) valorMax = vPuntos[i].x;
        if (vPuntos[i].x < valorMin) valorMin = vPuntos[i].x;
    }
    if (x > valorMax) {
        std::cout << "x demasiado grande" << std::endl;
        return 0;
    }
    if (x < valorMin) {
        std::cout << "x demasiado pequeno" << std::endl;
        return 0;
    }
    // Interpolación
    double pol = difDiv(0, 0, vPuntos);
    double coef = 1;
    for (unsigned int i=0; i<vPuntos.size()-1; i++) {
        coef *= (x - vPuntos[i].x);
        pol += coef*difDiv(i+1, 0, vPuntos);
    }
    std::cout << "p(" << x << ") = " << pol << std::endl;
    return 0;
}
```

EJERCICIO 4

Un grafo $G = (V, E)$ consiste en un conjunto de vértices $V = \{v_1, v_2, \dots\}$ y un conjunto de arcos $E = \{e_1, e_2, \dots\}$, tal que cada arco es identificado mediante una pareja no ordenada (v_i, v_j) de vértices. La forma más común de representar un grafo es mediante un diagrama, en el cual cada vértice es representado mediante un punto y cada arco mediante un segmento de línea conectando sus vértices.

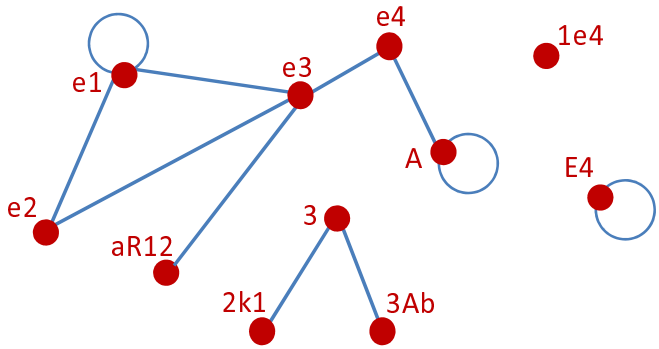
Supongamos que se emplean dos ficheros de texto para describir el grafo:

- En el primero, llamado `vertices.txt`, están escritos en columna los identificadores de los vértices. El identificador de un vértice es una cadena de caracteres compuesta por letras y/o números. El identificador de cada vértice debe ser único, es decir, no puede haber dos vértices con el mismo identificador.
- En el segundo, llamado `arcos.txt`, están escritas las parejas de vértices de cada arco. Cada pareja de vértices está escrita en una línea del fichero.

Escriba un programa en C++ que realice las acciones siguientes:

1. Leer los dos ficheros de texto `vertices.txt` y `arcos.txt`. Si se produce error, indicarlo mediante un mensaje en la consola y terminar.
2. Comprobar que ninguno de los identificadores del fichero `vertices.txt` está repetido. Si alguno está repetido, indicarlo mediante un mensaje en la consola y terminar.
3. Comprobar que todos los identificadores que aparecen en `arcos.txt`, aparecen en el fichero `vertices.txt`. Si no es así, indicarlo mediante un mensaje en la consola y terminar.
4. Escribir en consola, para cada vértice del grafo, qué vértices están directamente conectados a él mediante un arco. La información de cada vértice debe escribirse en una línea, con este formato: identificador del vértice, símbolo dos puntos (:), identificadores de los vértices directamente conectados.
5. Terminar.

Como ejemplo, a continuación se muestra el diagrama de un grafo. Junto a cada vértice se ha escrito su identificador. Debajo del diagrama, se muestra una forma de describir el grafo empleando los dos ficheros de texto. Finalmente se muestra la salida por consola del programa.



Fichero `vertices.txt`

```
e3
E4
e2
e4
A
1e4
3Ab
2k1
e1
3
aR12
```

Fichero `arcos.txt`

```
e1 e2
e1 e1
e1 e3
e3 e4
e2 e3
aR12 e3
A e4
E4 E4
A A
3 2k1
3Ab 3
```

Salida por consola del programa

```
1e4 :
2k1 : 3
3 : 2k1  3Ab
3Ab : 3
A : e4  A
E4 : E4
aR12 : e3
e1 : e2  e1  e3
e2 : e1  e3
e3 : e1  e4  e2  aR12
e4 : e3  A
```

Solución al Ejercicio 4

```
#include <iostream>
#include <fstream>
#include <list>
#include <string>
#include <map>

const std::string fichVertices = "vertices.txt";
const std::string fichArcos = "arcos.txt";

int main() {
    // Abrir fichero de vértices para lectura
    std::ifstream inFichV(fichVertices.c_str(),std::ios::in);
    if (!inFichV) {
        std::cout << "Error al abrir " << fichVertices
                  << std::endl;
        return 0;
    }
    // Mapa grafo
    //    Clave: id del vértive
    //    Valor: lista con id de vértices directamente conectados
    std::map<std::string,std::list<std::string> > grafo;
    while ( !inFichV.eof() ) {
        std::string idVertice;
        inFichV >> idVertice;
        if (inFichV.eof()) break;
        // Comprueba si el vértice ya ha sido leído
        if ( grafo.find(idVertice) == grafo.end() ) {
            std::list<std::string> idVecinos;
            grafo[idVertice] = idVecinos;
        } else {
            std::cout << "Error: el vertice " << idVertice
                  << " esta repetido" << std::endl;
            inFichV.close();
            return 0;
        }
    }
    inFichV.close();
```

```
    // Abrir fichero de arcos para lectura
    std::ifstream inFichA(fichArcos.c_str(),std::ios::in);
    if (!inFichA) {
        std::cout << "Error al abrir " << fichArcos
                  << std::endl;
        return 0;
    }
    while ( !inFichA.eof() ) {
        std::string idVertice1,idVertice2;
        inFichA >> idVertice1;
        if (inFichA.eof()) break;
        std::map<std::string,std::list<std::string> >::iterator p1 =
              grafo.find(idVertice1);
        if (p1 == grafo.end()) {
            std::cout << "Error arcos: vertice " << idVertice1 <<
                  " desconocido" << std::endl;
            inFichA.close();
            return 0;
        }
        inFichA >> idVertice2;
        std::map<std::string,std::list<std::string> >::iterator p2 =
              grafo.find(idVertice2);
        if (p2 == grafo.end()) {
            std::cout << "Error arcos: vertice " << idVertice2 <<
                  " desconocido" << std::endl;
            inFichA.close();
            return 0;
        }
        p1->second.push_back(idVertice2);
        if (idVertice1 != idVertice2)
            p2->second.push_back(idVertice1);
    }
    inFichA.close();
    // Salida por consola de vértices conectados a cada vértice
    std::map<std::string,std::list<std::string> >::iterator p =
          grafo.begin();
    while (p  != grafo.end()) {
        std::cout << "\n" << p->first << " : ";
        std::list<std::string>::iterator pL = p->second.begin();
        while (pL!=p->second.end()) {
            std::cout << *pL << "  ";
            pL++;
        }
        p++;
    }
    return 0;
}
```