LENGUAJES DE PROGRAMACIÓN

Solución al Trabajo Práctico - Convocatoria ordinaria de 2024

Ejercicio 1

En el plano cartesiano X-Y se encuentran tres puntos diferentes A, B y C. Sean P, Q y R puntos definidos de la forma siguiente:

- El punto P está situado sobre la recta que use los puntos A y B, a la misma distancia de ambos.
- El punto Q está situado sobre la recta que une los puntos A y C, y su distancia a A es el doble que su distancia a C.
- El punto R está situado sobre la recta que use B y C, y su distancia a B es el triple que su distancia a C.

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

1. Escribir un mensaje en la consola solicitando al usuario que introduzca las coordenadas cartesianas X-Y de los puntos A, B y C.
2. Leer los valores introducidos por consola. Comprobar si los tres puntos son distintos. Si no son distintos, escribir un mensaje en la consola indicándolo y terminar.
3. Calcular las coordenadas cartesianas X-Y de los puntos P, Q y R y escribirlas en la consola, en formato fijo con 4 dígitos decimales.
4. Terminar.

Muestre el resultado de la ejecución de su programa para cada uno de los cuatro casos siguientes:

- Caso 1: $A = \{4, 8\}$, $B = \{0.5, -3\}$, $C = \{4, 8\}$
- Caso 2: $A = \{0, 0\}$, $B = \{2, 0\}$, $C = \{0, 3\}$
- Caso 3: $A = \{0, 2\}$, $B = \{0, 0\}$, $C = \{0, 4\}$
- Caso 4: $A = \{4.5, -0.8\}$, $B = \{1.4, -5.8\}$, $C = \{-4.9, 3\}$

Solución al Ejercicio 1

```cpp
#include <iostream>
#include <iomanip>
#include <complex>

int main() {
   // Entrada por consola
   double x, y;
   std::cout << "Coordenadas X, Y del punto A: ";
   std::cin >> x >> y;
   std::complex<double> A(x,y);
   std::cout << "Coordenadas X, Y del punto B: ";
   std::cin >> x >> y;
   std::complex<double> B(x,y);
   std::cout << "Coordenadas X, Y del punto C: ";
   std::cin >> x >> y;
   std::complex<double> C(x,y);
   // Igualdad
   if ( A == B || A == C || B == C ) {
      std::cerr << "Los puntos no son distintos";
      return 0;
   }
   // Puntos P, Q, R
   std::complex<double> P = A + 1./2 * ( B - A );
   std::complex<double> Q = A + 2./3 * ( C - A );
   std::complex<double> R = B + 3./4 * ( C - B );
    std::cout << std::fixed << std::setprecision(4)
              << "P = ( " << P.real() << ", " << P.imag() << " )\n"
              << "Q = ( " << Q.real() << ", " << Q.imag() << " )\n"
              << "R = ( " << R.real() << ", " << R.imag() << " )\n";
    return 0;
}
```

Ejercicio 2

Un objeto puntual se mueve en línea recta en el espacio con velocidad constante conocida $\vec{v}$. Sabiendo que en el instante t_A el objeto se encuentra en la posición $\vec{A}$, se desea calcular en qué posición $\vec{B}$ del espacio se encuentra el objeto en el instante t_B, aplicando para ello la ecuación: $\vec{B} = \vec{A} + \vec{v} \cdot (t_B - t_A)$. Escriba un programa en C++ que realice las acciones indicadas a continuación.

1. Declarar una constante global del programa que sea un array de tres componentes llamado `v` y asignarle el valor $\{0.1,\ 1.5,\ -2.6\}$. Esta constante describe las coordenadas cartesianas X-Y-Z de la velocidad del objeto.

2. Mediante un mensaje escrito en la consola, solicitar al usuario que introduzca por consola el instante de tiempo t_A, así como las tres coordenadas cartesianas X-Y-Z de la posición $\vec{A}$. Suponemos que el tiempo está expresado en segundos y que las coordenadas están expresadas en metros.

3. Leer los valores introducidos por consola, almacenándolos en una variable llamada `tA` y un array de tres componentes llamado `A`, respectivamente.

4. Mediante un mensaje escrito en la consola, solicitar al usuario que introduzca por consola el instante de tiempo t_B expresado en segundos, minutos u horas. El formato en el cual el usuario debe escribir el instante de tiempo en la consola es el siguiente: debe escribir un número, seguido de uno o más espacios en blanco, a continuación una cadena de caracteres de entre el conjunto {s, min, hora} que indica las unidades, y a continuación debe pulsar la tecla *enter*.

5. Leer el valor introducido por consola. Calcular la posición $\vec{B}$ del objeto en el instante t_B y escribir sus tres coordenadas cartesianas X-Y-Z expresadas en metros, así como la distancia entre las posiciones $\vec{A}$ y $\vec{B}$ expresada en metros. Los resultados deben escribirse en formato científico con 3 dígitos decimales.

6. Terminar.

Al escribir el programa puede asumir que el texto introducido por el usuario tiene el formato correcto. Muestre el resultado de la ejecución de su programa para cada uno de los tres casos siguientes:

- Caso 1: $t_A = 1.3$, $\vec{A} = \{4.5, -0.8, 7\}$, $t_B = 8586$ s
- Caso 2: $t_A = 1.3$, $\vec{A} = \{4.5, -0.8, 7\}$, $t_B = 143.1$ min
- Caso 3: $t_A = 1.3$, $\vec{A} = \{4.5, -0.8, 7\}$, $t_B = 2.385$ hora

Solución al Ejercicio 2

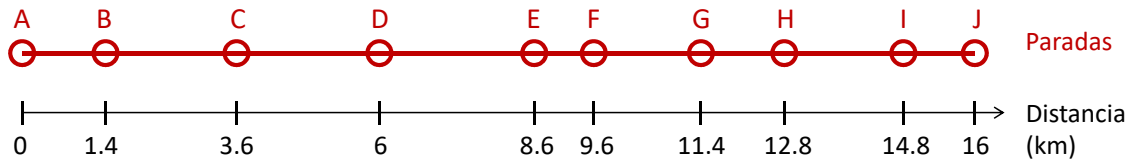
```cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>

const double v[3] =  { 0.1, 1.5, -2.6 };

int main() {
   double tA, A[3];
   std::cout << "Instante tA: ";
   std::cin >> tA;
   std::cout << "Coordenadas X-Y-Z de A: ";
   std::cin >> A[0] >> A[1] >> A[2];
   std::cout << "Instante tB y unidades (s, min, hora): ";
   double tB;
   std::string units;
   std::cin >> tB >> units;
   if ( units!="s" && units!="min" && units!="hora" ) {
      std::cerr << "Unidades desconocidas" << std::endl;
      return 0;
   }
   if ( units == "min" )
      tB *= 60;
   else if ( units == "hora" )
      tB *= 3600;
   double B[3];
   for ( int i=0; i<3; i++ )
      B[i] = A[i] + v[i] * ( tB - tA );
   double distancia_A_B = std::sqrt(
                             std::pow( A[0] - B[0], 2 ) +
                             std::pow( A[1] - B[1], 2 ) +
                             std::pow( A[2] - B[2], 2 ) );
   std::cout << std::scientific << std::setprecision(3)
             << "B (m) = (  " << B[0] << ", " << B[1]
             << ", " << B[2] << " )\n"
             << "Distancia A-B (m): "
             << distancia_A_B << std::endl;
   return 0;
}
```

Ejercicio 3

Una línea de autobús urbano tiene 10 paradas, a las cuales se identifica mediante las letras A–J tal como se muestra en la figura mostrada a continuación.



El autobús realiza trayectos recorriendo la línea en ambos sentidos. Si en un trayecto la cabecera de línea es A y el final de línea es J, en el siguiente trayecto la cabecera de línea es J y el final de línea es A. El trayecto A-J se realiza por el mismo camino que el J-A, por lo cual las distancias son las mismas con independencia del sentido en el que circule el autobús. En la figura se indica la distancia que debe recorrer el autobús para llegar a las paradas, tomando como origen de parada A. Por ejemplo, la distancia entre la parada A y la parada E son 8.6 km.

La velocidad del autobús depende del estado del tráfico en cada tramo entre dos paradas consecutivas. Esta información está almacenada en un fichero de texto llamado *trafico.txt*, que tiene el formato indicado a continuación. En la primera columna está concatenado el nombre de la parada origen y destino del tramo. En la segunda, la velocidad media a la que el autobús circulará en ese tramo, en el sentido indicado, expresada en km/hora. A continuación se muestra un ejemplo del contenido del fichero descriptivo del estado del tráfico.

```
AB   35.5
BA   41.4
BC   29.3
CB   46.8
DC   39.9
CD   40.4
DE   24.9
EF   37.4
FG   29.8
GH   32.5
HI   42.7
IJ   22.7
JI   40.4
IH   42.1
HG   33.2
GF   45.6
FE   35.6
ED   42.8
```

Así, por ejemplo, el autobús puede desplazarse de la parada D a la E a 24.9 km/h, y de la parada E a la D a 42.8 km/h. Dado que ambas paradas distan 2.6 km entre

sí (véase la figura), el autobús tardará $2.6 \cdot 60/24.9$ minutos en ir de la parada D a la E y $2.6 \cdot 60/42.8$ minutos en ir de la parada E a la D.

Se estima que, con independencia del sentido de circulación del autobús, éste se detiene durante 2 minutos en cada una de las paradas intermedias (B, C, D, E, F, G, H e I) para que los viajeros suban y bajen.

Escriba un programa en C++ que, a partir del contenido del fichero *trafico.txt*, calcule el tiempo (en minutos) que transcurrirá desde el instante actual hasta que el autobús llegue a cada una de las paradas de la línea en los dos casos siguientes:

1. En el instante actual el autobús sale de la parada A.
2. En el instante actual el autobús sale de la parada J.

Debe mostrarse en la consola solo la parte entera del tiempo calculado. Por ejemplo, si el tiempo calculado es 13.87 minutos, debe escribirse en la consola 13 minutos.

Puede asumir, al escribir el programa, que el fichero *trafico.txt* no contiene errores de formato y que contiene todos los tramos entre paradas consecutivas, en ambos sentidos. No asuma que las líneas del fichero están ordenadas.

Muestre el resultado de la ejecución de su programa para el fichero de estado del tráfico empleado como ejemplo en el enunciado.

Solución al Ejercicio 3

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

const char nombreFich[] = "trafico.txt";

struct DistParada {
   char parada;
   double dist_km;
};

// Vector ordenado crecientemente por la distancia
const std::vector<DistParada> distParadas = {
      { 'A',  0 },   { 'B',  1.4 },
      { 'C',  3.6 }, { 'D',  6 },   { 'E',  8.6 }, { 'F',  9.6 },
      { 'G', 11.4 }, { 'H', 12.8 }, { 'I', 14.8 }, { 'J', 16 }
};

struct TraficoTramo {
   char pOrigen, pDestino;
   double veloc_km_h;
};


int main() {
   // Apertura del fichero
   std::ifstream inFich(nombreFich, std::ios::in);
   if (!inFich) {
      std::cerr << "Error al abrir el fichero " << nombreFich;
      return 0;
   }
   // Lectura del fichero
   std::vector<TraficoTramo> vTraficoTramos;
   TraficoTramo trafTramo;
   std::string identTramo;
   while ( inFich >> identTramo >>  trafTramo.veloc_km_h ) {
      trafTramo.pOrigen  = identTramo[0];
      trafTramo.pDestino = identTramo[1];
      vTraficoTramos.push_back(trafTramo);
   }
   // Cerrar el fichero
   inFich.close();
   // El autobus sale de A
   std::cout << "El autobus sale de la parada A" << std::endl;
   double instanteLlegada = 0;
   for (unsigned int i=0; i<distParadas.size()-1; i++) {
      char parada_inicioTramo = distParadas[i].parada;
      char parada_finTramo    = distParadas[i+1].parada;
      double distanciaTramo   =
          distParadas[i+1].dist_km - distParadas[i].dist_km;
      double velocidadTramo_km_h;
```

```
53        for (unsigned int j=0; j<vTraficoTramos.size(); j++)
54           if ( vTraficoTramos[j].pOrigen  == parada_inicioTramo &&
55                vTraficoTramos[j].pDestino == parada_finTramo     ) {
56              velocidadTramo_km_h = vTraficoTramos[j].veloc_km_h;
57              break;
58           }
59        instanteLlegada += 60 * distanciaTramo / velocidadTramo_km_h;
60        if (i>0)
61            instanteLlegada += 2;
62        std::cout << "Tiempo hasta llegada a parada "
63                  << parada_finTramo << ": "
64                  << (int)instanteLlegada << " min" << std::endl;
65     }
66     // El autobus sale de J
67     std::cout << "El autobus sale de la parada J" << std::endl;
68     instanteLlegada = 0;
69     for (unsigned int i=distParadas.size()-1; i>0; i--) {
70        char parada_inicioTramo = distParadas[i].parada;
71        char parada_finTramo = distParadas[i-1].parada;
72        double distanciaTramo =
73            distParadas[i].dist_km - distParadas[i-1].dist_km;
74        double velocidadTramo_km_h;
75        for (unsigned int j=0; j<vTraficoTramos.size(); j++)
76           if ( vTraficoTramos[j].pOrigen == parada_inicioTramo &&
77                vTraficoTramos[j].pDestino == parada_finTramo    ) {
78              velocidadTramo_km_h = vTraficoTramos[j].veloc_km_h;
79              break;
80           }
81        instanteLlegada += 60 * distanciaTramo / velocidadTramo_km_h;
82        if (i<distParadas.size()-1)
83           instanteLlegada += 2;
84        std::cout << "Tiempo hasta llegada a parada "
85                  << parada_finTramo << ": "
86                  << (int)instanteLlegada << " min" << std::endl;
87     }
88     return 0;
89  }
```

Ejercicio 4

Se plantea el problema de decidir a qué personas se permite el acceso a unas determinadas carreras universitarias, basándose para ello en la nota que dichas personas han obtenido en la Prueba de Acceso a la Universidad (PAU) y en sus preferencias respecto a la carrera que desean estudiar, debiéndose en cualquier caso satisfacer que el número de personas admitidas en cada carrera no puede superar un número máximo preestablecido.

La información acerca de cuántas personas pueden ser admitidas en cada carrera está almacenada en un fichero de texto llamado *carreras.txt*. En cada línea del fichero se encuentra la información correspondiente a una carrera. En la primera columna está escrito el identificador de la carrera, que es una cadena de caracteres que está compuesta de letras y opcionalmente por el símbolo guión bajo (_). En la segunda columna se indica el número máximo de personas que pueden ser admitidas en la carrera. A continuación se muestra un ejemplo del contenido del fichero, según el cual, por citar solo dos casos, el número máximo de personas a las cuales se les puede conceder el acceso a la carrera `Biologia` es 10 y a la carrera `Bioquimica` es 5.

```
Biologia                                10
Bioquimica                               5
Ciencia_y_Tecnologia_de_los_Alimentos   10
Fisica                                   5
Geologia                                10
Matematicas                             10
Matematicas_y_Ciencia_de_Datos          10
Quimica                                  5
```

La información acerca de las personas aspirantes a acceder a una de las carreras se encuentra en un fichero de texto llamado *personas.txt*. La información de cada persona está escrita en una línea del fichero. En la primera columna está escrito un número entero mayor que cero, que se emplea como identificador unívoco de la persona durante este proceso de admisión. En la segunda columna se encuentra la puntuación obtenida por la persona en la PAU. Dicha puntuación es un número real del intervalo $[5, 14]$. La tercera columna contiene las preferencias de la persona: carreras ordenadas de mayor a menor preferencia y separadas por coma.

En la página siguiente se muestra un ejemplo del contenido del fichero. En este caso, la persona identificada mediante el número 1 ha obtenido una nota en la PAU de 13.20 puntos y desea acceder a la carrera `Bioquimica`, mientras que la persona identificada mediante el número 2 ha obtenido una nota en la PAU de 9.60 puntos y desea acceder como primera opción a `Geologia` y, si esto no es posible, como segunda opción a `Biologia`.

```
1    13.20   Bioquimica
2     9.60   Geologia,Biologia
3     7.10   Bioquimica
4     9.20   Bioquimica
5    11.00   Bioquimica,Quimica
6    10.80   Geologia,Biologia
7     9.30   Bioquimica,Quimica
8    13.00   Fisica,Matematicas
9     5.50   Fisica,Matematicas
10    6.00   Bioquimica,Quimica
11    5.00   Bioquimica,Quimica,Fisica,Matematicas,Geologia
12   11.00   Matematicas_y_Ciencia_de_Datos,Matematicas
13   11.60   Bioquimica,Quimica,Fisica,Matematicas,Geologia
14   12.50   Bioquimica,Quimica
15    7.00   Bioquimica,Quimica
16    9.00   Bioquimica,Quimica,Fisica,Matematicas,Geologia
17    7.00   Bioquimica
18    7.70   Fisica,Bioquimica,Quimica
19    5.00   Fisica,Bioquimica,Quimica
20    5.00   Bioquimica,Ciencia_y_Tecnologia_de_los_Alimentos
21   12.10   Bioquimica,Quimica
22   13.60   Bioquimica,Quimica,Fisica,Matematicas,Geologia
23    7.50   Matematicas
24   12.40   Matematicas,Matematicas_y_Ciencia_de_Datos
25   12.40   Matematicas,Matematicas_y_Ciencia_de_Datos,Fisica
26    9.10   Matematicas_y_Ciencia_de_Datos,Matematicas
27   10.00   Bioquimica,Quimica
28    6.60   Bioquimica,Quimica
29   12.10   Matematicas_y_Ciencia_de_Datos,Matematicas
30    6.40   Matematicas,Matematicas_y_Ciencia_de_Datos
31   11.40   Fisica,Bioquimica,Quimica
32   13.25   Fisica,Bioquimica,Quimica
33    9.60   Matematicas,Matematicas_y_Ciencia_de_Datos
34   12.00   Matematicas,Matematicas_y_Ciencia_de_Datos,Fisica
35    5.50   Geologia
36    9.00   Geologia
37   11.10   Matematicas_y_Ciencia_de_Datos,Matematicas
38    8.50   Quimica,Bioquimica,Fisica,Matematicas
39   10.70   Geologia,Biologia,Matematicas
40   13.00   Fisica
41   13.00   Fisica
42    5.00   Matematicas
43    9.50   Matematicas
44   13.30   Fisica
45   13.60   Fisica,Bioquimica,Quimica
46   10.80   Quimica
47    9.00   Fisica,Bioquimica,Quimica
48   11.90   Matematicas,Matematicas_y_Ciencia_de_Datos
49    5.40   Fisica,Bioquimica,Quimica
50   13.00   Fisica,Bioquimica,Quimica
51   12.60   Geologia,Biologia
52    6.50   Fisica,Matematicas
53    9.50   Fisica,Matematicas,Quimica
54    5.50   Quimica,Bioquimica,Fisica,Matematicas
55   13.00   Fisica,Matematicas,Quimica
56    5.00   Fisica,Matematicas,Quimica,Ciencia_y_Tecnologia_de_los_Alimentos
57   11.20   Geologia
58    9.10   Geologia
59   11.20   Geologia,Biologia
60    6.00   Geologia,Biologia
61    7.00   Quimica,Bioquimica
62   11.20   Geologia,Biologia
63    6.00   Geologia,Biologia,Bioquimica
64    8.50   Geologia,Biologia,Bioquimica
65   11.20   Geologia,Biologia
66   11.40   Geologia,Biologia,Bioquimica,Fisica
67    6.40   Geologia,Biologia,Bioquimica,Matematicas
68   12.00   Quimica
69   13.00   Quimica,Bioquimica
70    6.60   Quimica,Bioquimica
```

El proceso de adjudicación de las plazas comienza ordenando las personas en orden descendente de su nota en la PAU (primero los de mayor nota). Si varias personas han obtenido la misma nota, su ordenación relativa se hace en orden ascendente del número de identificador (primero los de menor identificador).

Una vez todas las personas han sido ordenadas, se realiza persona a persona el proceso de asignación, siguiendo esta ordenación. El proceso se realiza de la forma siguiente. Si la carrera preferida por la persona no está completa (el número de personas asignadas a la carrera es inferior al máximo), entonces se asigna a la persona esa carrera y finaliza el proceso para esa persona. En caso contrario, es decir, si la carrera está completa (el número de personas asignadas a la carrera es igual al máximo posible), entonces se repite el proceso con la siguiente carrera de la lista de preferencias de la persona, y así sucesivamente. Una vez se asigna una carrera a la persona, finaliza el proceso de asignación de esa persona. Si todas las carreras de la lista de preferencias de la persona están completas, la persona se clasifica como "No admitida", completándose el proceso de asignación de la persona.

Escriba un programa en C++ que lea los ficheros *carreras.txt* y *personas.txt* y escriba en la consola, para cada carrera, los identificadores de las personas admitidas en dicha carrera, por orden de admisión. Asimismo, el programa debe mostrar los identificadores de las personas no admitidas, ordenados de mayor a menor nota y, a igualdad de nota, de menor a mayor identificador.

El programa debe además escribir en la consola la "nota de corte" de cada carrera, calculada de la forma siguiente: si la carrera está completa, la nota de corte es la menor nota en la PAU de los alumnos asignados a dicha carrera, y si la carrera no está completa, la nota de corte es 5 puntos.

Al escribir el programa puede asumir que el formato de los ficheros *carreras.txt* y *personas.txt* es el indicado en el enunciado, y que ambos ficheros contendrán al menos una carrera y una persona respectivamente. No realice ninguna suposición acerca del nombre de las carreras, ya que éste debe obtenerse del fichero *carreras.txt*, y tampoco del número de carreras y del número de personas, ya que dependerán del número de líneas escritas en los ficheros *carreras.txt* y *personas.txt* respectivamente. El identificador asociado a cada persona es un número entero mayor que cero, que debe leerse de la primera columna del fichero *personas.txt*.

Muestre en la memoria el resultado de ejecutar su programa para los dos ficheros *carreras.txt* y *personas.txt* usados como ejemplo en el enunciado.

Solución al Ejercicio 4

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

const char nombreFich_Carreras[] = "carreras.txt";
const char nombreFich_Personas[] = "personas.txt";

struct Carrera {
   std::string nombre;
   unsigned int capacidad;
   std::vector<int> vIdentAdmitidos;
};

struct Persona {
   int ident;
   double nota;
   std::vector<std::string> vPreferencias;
};


int main() {
   // Apertura del fichero de carreras
   std::ifstream inFich1(nombreFich_Carreras, std::ios::in);
   if (!inFich1) {
      std::cerr << "Error al abrir el fichero " << nombreFich_Carreras;
      return 0;
   }
   // Lectura del fichero de carreras
   std::vector<Carrera> vCarreras;
   Carrera carrera;
   while (inFich1 >> carrera.nombre >>  carrera.capacidad) {
      vCarreras.push_back(carrera);
   }
   // Cerrar el fichero de carreras
   inFich1.close();
   // Apertura del fichero de personas
   std::ifstream inFich2(nombreFich_Personas, std::ios::in);
   if (!inFich2) {
      std::cerr << "Error al abrir el fichero " << nombreFich_Personas;
      return 0;
   }
   // Lectura del fichero de personas
   std::vector<Persona> vPersona;
   Persona persona;
   std::string preferencias;
   while (inFich2 >> persona.ident >> persona.nota >> preferencias) {
      persona.vPreferencias.clear();
      int indI = 0;
      for (unsigned int i=0; i<preferencias.length(); i++)
         if (preferencias[i] == ',') {
```

```
            persona.vPreferencias.push_back(
                preferencias.substr(indI,i-indI) );
            indI = i+1;
         }
      persona.vPreferencias.push_back(
           preferencias.substr(indI,preferencias.length()-indI) );
      vPersona.push_back(persona);
   }
   // Cerrar el fichero de personas
   inFich2.close();
   // Ordenar personas
   bool desordenado;
   do {
      desordenado = false;
      for (unsigned int i=0; i<vPersona.size()-1; i++)
         if ( ( vPersona[i].nota < vPersona[i+1].nota ) ||
              ( vPersona[i].nota == vPersona[i+1].nota &&
                vPersona[i].ident > vPersona[i+1].ident ) ) {
            Persona aux = vPersona[i+1];
            vPersona[i+1] = vPersona[i];
            vPersona[i] = aux;
            desordenado = true;
         }
   } while (desordenado);
   // Asignar carreras
   std::vector<int> noAdmitidos;
   for (unsigned int i=0; i<vPersona.size(); i++) {
      bool asignado = false;
      for (unsigned int j=0;
           !asignado && j<vPersona[i].vPreferencias.size(); j++) {
         for (unsigned int k=0; k<vCarreras.size(); k++) {
            if ( vCarreras[k].nombre ==
                              vPersona[i].vPreferencias[j] &&
                 vCarreras[k].vIdentAdmitidos.size() <
                              vCarreras[k].capacidad ) {
               asignado = true;
               vCarreras[k].vIdentAdmitidos.push_back(
                                                 vPersona[i].ident );
            }
         }
      }
      if (!asignado)
         noAdmitidos.push_back(vPersona[i].ident);
   }
   // Escritura en consola de admitidos por carrera
   for (unsigned int i=0; i<vCarreras.size(); i++) {
      std::cout << vCarreras[i].nombre << ": ";
      for (unsigned int j=0;
           j<vCarreras[i].vIdentAdmitidos.size(); j++)
         std::cout << vCarreras[i].vIdentAdmitidos[j] << " ";
      std::cout << std::endl;
   }
   // Escritura en consola de no admitidos
   std::cout << "No admitidos: ";
```

```
106    for (unsigned int i=0; i<noAdmitidos.size(); i++)
107       std::cout << noAdmitidos[i] << " ";
108    std::cout << std::endl;
109    // Escritura en consola de nota de corte por carrera
110    std::cout << "\nNotas de corte\n";
111    for (unsigned int i=0; i<vCarreras.size(); i++) {
112       double notaCorte;
113       unsigned int numAdmitidos = vCarreras[i].vIdentAdmitidos.size();
114       if (vCarreras[i].capacidad > numAdmitidos) {
115          notaCorte = 5;
116       } else {
117          int identUltimoAdmitido =
118                vCarreras[i].vIdentAdmitidos[numAdmitidos-1] ;
119          for (unsigned int j=0; j<vPersona.size(); j++)
120             if (vPersona[j].ident == identUltimoAdmitido) {
121                notaCorte = vPersona[j].nota;
122                break;
123             }
124       }
125       std::cout << vCarreras[i].nombre << ": "
126                 << notaCorte << std::endl;
127    }
128 /*
129    // Escritura en consola de las personas ordenadas
130    for (unsigned int i=0; i<vPersona.size(); i++) {
131       std::cout << "\n" << vPersona[i].ident << " "
132                 << vPersona[i].nota << " ";
133       for (unsigned int j=0; j<vPersona[i].vPreferencias.size(); j++)
134          std::cout << vPersona[i].vPreferencias[j] << "  ";
135    }
136 */
137     return 0;
138 }
```