# Algoritmos 1

Primer cuatrimestre 2012

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

## TP Imperativo

Grupo 10



| Integrante | LU | Correo electrónico |
|---|---|---|
| Gastón de Orta | 244/11 | gaston.deorta@hotmail.com |
| Leandro Lovisolo | 645/11 | leandro@leandro.me |
| María Candela Capra Coarasa | 234/11 | canduh_27@hotmail.com |
| Lautaro Jose Petaccio | 443/11 | lausuper@gmail.com |



Código y scripts utilizados para generar este imprimible disponibles en:
`https://github.com/LeandroLovisolo/TPI`

# ojota.cpp

```cpp
#include <iostream>
#include "interfaz.h"
#include "lista.h"

using namespace std;

// problema obtenerCampeones(j: JJOO) = result: [Atleta] {
//     asegura result == [cab(ranking(c)) | c <- competencias(j), finalizada(c) && |ranking(c)| > 0];
// }

Lista<Atleta> obtenerCampeones(const JJOO & j) {
        Lista<Competencia> competenciasOro = j.competenciasFinalizadasConOroEnPodio();
        Lista<Atleta> campeones;
        int i = 0;

        // Vale Pc: competenciasOro == [c | c <- competencias(j), finalizada(c) && |ranking(c)| > 0] &&
        //          |campeones| == 0 &&
        //          i == 0
        // Implica  0 <= i <= |competenciasOro|
        // Implica  campeones == [] == [cab(ranking(c)) | c <- competenciasOro[0..i)]
        //
        // Luego, Pc -> I.
        while(i < competenciasOro.longitud()) {
                // Guarda B:     i < |competenciasOro|
                // Invariante I: 0 <= i <= |competenciasOro| &&
                //               campeones = [cab(ranking(c)) | c <- competenciasOro[0..i)]
                // Variante V:   |competenciasOro| - i
                // Cota C:       0

                // Estado E1
                // Vale I && B
                campeones.agregarAtras(competenciasOro.iesimo(i).ranking().cabeza());

                // Estado E2
                // Vale    i = i@E1 && campeones == campeones@E1 ++ [cab(ranking(competenciasOro[i]))]
                // Implica campeones == [cab(ranking(c)) | c <- competenciasOro[0..i)] ++
                //                      [cab(ranking(competenciasOro[i]))]
                // Implica campeones == [cab(ranking(c)) | c <- competenciasOro[0..i]]
                i++;

                // Estado E3
                // Vale    i == i@E2 + 1 && campeones == campeones@E2
                // Implica 0 < i
                // Implica i == i@E2 + 1 < |competenciasOro| + 1
                // Implica i == i@E2 + 1 <= |competenciasOro|
                // Implica i <= |competenciasOro|
                // Implica 0 <= i <= |competenciasOro|
                // Implica campeones == [cab(ranking(c)) | c <- competenciasOro[0..i@E2]]
                // Implica i@E2 == i - 1
                // Implica campeones == [cab(ranking(c)) | c <- competenciasOro[0..i - 1]]
                // Implica campeones == [cab(ranking(c)) | c <- competenciasOro[0..i)]
                // Implica 0 <= i <= |competenciasOro| &&
                //         campeones == [cab(ranking(c)) | c <- competenciasOro[0..i)]
                //
                // Luego, E3 -> I.
                //
                // Vale    V == |competenciasOro| - i@E3
                // Implica V == |competenciasOro| - (i@E1 + 1)
                // Implica V == |competenciasOro| - i@E1 - 1 < |competenciasOro| - i@E1 == V@E1
                //
                // Luego, V@E3 < V@E1.
                //
                // Supongo V <= C.
                // Vale    I && (V <= C)
                // Implica 0 <= i <= |competenciasOro| &&
                //         campeones = [cab(ranking(c)) | c <- competenciasOro[0..i)] &&
                //         (|competenciasOro| - i <= 0)
                // Implica |competenciasOro| <= i
                // Implica i == |competenciasOro|
                // Implica ¬(i < |competenciasOro|)
                //
                // Luego, I && (V <= C) -> ¬B.
        }

        // Vale Qc: i == |competenciasOro| &&
        //          campeones = [cab(ranking(c)) | c <- competencias(j), finalizada(c) && |ranking(c)| > 0]
        //
        // Vale     I && ¬B
        // Implica  0 <= i <= |competenciasOro| &&
        //          campeones = [cab(ranking(c)) | c <- competenciasOro[0..i)] &&
        //          i >= |competenciasOro|
        // Implica  i <= |competenciasOro| && i >= |competenciasOro|
        // Implica  i == |competenciasOro|
        // Implica  campeones = [cab(ranking(c)) | c <- competenciasOro[0..|competenciasOro|)]
        // Implica  campeones = [cab(ranking(c)) | c <- competenciasOro]
        // Implica  campeones = [cab(ranking(c)) | c <- competencias(j), finalizada(c) && |ranking(c)| > 0]
```

```
        // Implica  i == |competenciasOro| &&
        //          campeones = [cab(ranking(c)) | c <- competencias(j), finalizada(c) && |ranking(c)| > 0]
        //
        // Luego, (I && ¬B) -> Qc.

        return campeones;

        // Vale    result == campeones@Qc
        // Implica result == [cab(ranking(c)) | c <- competencias(j), finalizada(c) && |ranking(c)| > 0]
}


// problema atletaProdigio (j: JJOO) = result: Atleta {
//     requiere algunaVezSeCompitio: |competenciasConOroEnPodio(j)| > 0;
//     asegura esCampeon(result, j);
//     asegura (∀c ∈ competenciasConOroEnPodio(j)) añoNacimiento(campeon(c)) ≤ añoNacimiento(result);
//
//     aux esCampeon(a: Atleta, j: JJOO): Bool =
//         (∃c ∈ competenciasConOroEnPodio(j)) a == campeon(c);
//     aux campeon (c: Competencia): Atleta = cab(ranking(c));
//     aux competenciasConOroEnPodio(j: JJOO): [Competencia] =
//         [c|c ← competencias(j), finalizada(c) ∧ |ranking(c)| > 0];
// }

Atleta atletaProdigio(const JJOO & j) {
        Lista<Atleta> campeones = obtenerCampeones(j);
        Atleta prodigio = campeones.cabeza();
        int i = 1;

        // Vale Pc: campeones == [cab(ranking(c)) | c <- competencias(j), finalizada(c) && |ranking(c)| > 0] &&
        //          |campeones| > 0 &&
        //          prodigio == cab(campeones) &&
        //          i == 1
        // Implica  prodigio ∈ campeones
        // Implica  1 <= i <= |campeones|
        // Implica  (∀c ∈ campeones[0..1)) añoNacimiento(prodigio) >= añoNacimiento(c)
        // Implica  (∀c ∈ campeones[0..i)) añoNacimiento(prodigio) >= añoNacimiento(c)
        // Implica  1 <= i <= |campeones| && prodigio ∈ campeones &&
        //          (∀c ∈ campeones[0..i)) añoNacimiento(prodigio) >= añoNacimiento(c)
        //
        // Luego, Pc -> I.
        while(i < campeones.longitud()) {
                // Guarda B:     i < |campeones|
                // Invariante I: 1 <= i <= |campeones| && prodigio ∈ campeones &&
                //               (∀c ∈ campeones[0..i)) añoNacimiento(prodigio) >= añoNacimiento(c)
                // Variante V:   |campeones| - i
                // Cota C:       0

                // Estado E1
                // Vale I && B

                if(campeones.iesimo(i).anioNacimiento() > prodigio.anioNacimiento()) {
                        // Estado F
                        // Vale i == i@E1 && campeones == campeones@E1 &&
                        //      añoNacimiento(campeones[i]) > añoNacimiento(prodigio@E1)
                        prodigio = campeones.iesimo(i);

                        // Estado G
                        // Vale i == i@E1 && campeones == campeones@E1 &&
                        //      añoNacimiento(campeones[i]) > añoNacimiento(prodigio@E1) &&
                        //      prodigio == campeones[i]
                } else {

                        // Estado H
                        // Vale i == i@E1 && campeones == campeones@E1 &&
                        //      añoNacimiento(campeones[i]) <= añoNacimiento(prodigio) &&
                        //      prodigio == prodigio@E1
                }

                // Estado E2
                // Vale      G || H
                // Implica  (añoNacimiento(campeones[i]) >  añoNacimiento(prodigio@E1) && prodigio == campeones[i]) ||
                //          (añoNacimiento(campeones[i]) <= añoNacimiento(prodigio)    && prodigio == prodigio@E1)
                // Implica   añoNacimiento(prodigio) >= añoNacimiento(campeones[i])
                // Implica   prodigio ∈ campeones
                // Implica   (∀c ∈ campeones[0..i]) añoNacimiento(prodigio) >= añoNacimiento(c)

                i++;
                // Estado E3
                // Vale    i == i@E2 + 1 && prodigio ∈ campeones &&
                //         (∀c ∈ campeones[0..i@E2]) añoNacimiento(prodigio) >= añoNacimiento(c)
                // Implica 1 < i
                // Implica i == i@E2 + 1 < |campeones| + 1 <= |campeones|
                // Implica 1 <= i <= |campeones|
                // Implica i@E2 == i - 1
                // Implica (∀c ∈ campeones[0..i - 1]) añoNacimiento(prodigio) >= añoNacimiento(c)
                // Implica (∀c ∈ campeones[0..i)) añoNacimiento(prodigio) >= añoNacimiento(c)
                // Implica 1 <= i <= |campeones| && prodigio ∈ campeones &&
                //         (∀c ∈ campeones[0..i)) añoNacimiento(prodigio) >= añoNacimiento(c)
                //
```

```
                // Luego, E3 -> I.
                //
                // Vale    V = |campeones| - i@E3
                // Implica V = |campeones| - (i@E1 + 1)
                // Implica V = |campeones| - i@E1 - 1 < |campeones| - i@E1 == V@E1
                //
                // Luego, V@E3 < V@E1.
                //
                // Supongo V <= C.
                // Vale    I && (V <= C)
                // Implica 1 <= i <= |campeones| && prodigio ∈ campeones &&
                //         ((∀c ∈ campeones[0..i)) añoNacimiento(prodigio) >= añoNacimiento(c)) &&
                //         (|campeones| - i <= 0)
                // Implica |campeones| <= i
                // Implica i == |campeones|
                // Implica ¬(i < |campeones|)
                //
                // Luego, I && (V <= C) -> ¬B.
        }

        // Vale Qc: i == |campeones| && prodigio ∈ campeones &&
        //          (∀c ∈ campeones) añoNacimiento(prodigio) >= añoNacimiento(c)
        //
        // Vale     I && ¬B
        // Implica  1 <= i <= |campeones| &&
        //          prodigio ∈ campeones &&
        //          ((∀c ∈ campeones[0..i)) añoNacimiento(prodigio) >= añoNacimiento(c)) &&
        //          ¬(i < |campeones|)
        // Implica  i >= |campeones|
        // Implica  i == |campeones|
        // Implica  (∀c ∈ campeones[0..|campeones|)) añoNacimiento(prodigio) >= añoNacimiento(c)
        // Implica  (∀c ∈ campeones) añoNacimiento(prodigio) >= añoNacimiento(c)
        // Implica  i == |campeones| && prodigio ∈ campeones &&
        //          (∀c ∈ campeones) añoNacimiento(prodigio) >= añoNacimiento(c)
        //
        // Luego, (I && ¬B) -> Qc.

        return prodigio;

        // Vale      result == prodigio
        // Implica   result ∈ campeones && (∀c ∈ campeones) añoNacimiento(result) >= añoNacimiento(c)
        // Implica   result ∈ [cab(ranking(c)) | c <- competencias(j), finalizada(c) && |ranking(c)| > 0] &&
        //          (∀c ∈ [cab(ranking(c)) | c <- competencias(j), finalizada(c) && |ranking(c)| > 0])
        //               añoNacimiento(result) >= añoNacimiento(c)
        // Implica ((∃c ∈ [cab(ranking(c)) | c <- competencias(j), finalizada(c) && |ranking(c)| > 0])
        //               result == cab(ranking(c))) &&
        //         ((∀c ∈ [cab(ranking(c)) | c <- competencias(j), finalizada(c) && |ranking(c)| > 0])
        //               añoNacimiento(result) >= añoNacimiento(c))
}

int main(){
    MenuPrincipal();
    return 0;
}
```

# tipos.h

```
#ifndef TIPOS_H
#define TIPOS_H

#include <stdlib.h>

#include <string>
#include <iostream>

using namespace std;
typedef string Deporte;
typedef string Pais;
enum Sexo{Femenino, Masculino};
typedef pair<Deporte, Sexo> Categoria;

#endif /*TIPOS_H*/
```

# atleta.h

```
#ifndef ATLETA_H
#define ATLETA_H

#include <iostream>
#include <fstream>

#include "tipos.h"
#include "lista.h"
```

```
 9
10  class Atleta{
11      public:
12
13          Atleta();
14          Atleta(const string nombre,const Sexo sexo, const int anio, const Pais pais, const int ciaNumber);
15
16          string nombre() const;
17          Sexo sexo() const;
18          int anioNacimiento() const;
19          Pais nacionalidad() const;
20          int ciaNumber() const;
21          Lista<Deporte> deportes() const;
22          int capacidad(const Deporte d) const;
23
24          void entrenarNuevoDeporte(const Deporte deporte, const int capacidad);
25          bool operator==(const Atleta& a) const;
26
27          void mostrar(std::ostream& os) const;
28          void guardar(std::ostream& os) const;
29          void cargar (std::istream& is);
30
31      private:
32          string _nombre;
33          Sexo _sexo;
34          int _anioNacimiento;
35          Pais _nacionalidad;
36          int _ciaNumber;
37          Lista<pair<Deporte, int> > _deportes;
38
39          enum {ENCABEZADO_ARCHIVO = 'A'};
40  };
41
42  std::ostream & operator<<(std::ostream & os,const Atleta & a);
43
44  #endif // ATLETA_H
```

# atleta.cpp

```
 1   #include <iostream>
 2   #include "tipos.h"
 3   #include "lista.h"
 4   #include "atleta.h"
 5
 6   using namespace std;
 7   Atleta::Atleta() {}
 8   Atleta::Atleta(const string nombre,const Sexo sexo, const int anio, const Pais pais, const int ciaNumber) {
 9       _nombre = nombre;
10       _sexo = sexo;
11       _anioNacimiento = anio;
12       _nacionalidad = pais;
13       _ciaNumber = ciaNumber;
14   }
15
16       //string nombre() const;
17   string Atleta::nombre() const {
18           return _nombre;
19   }
20
21       //Sexo sexo() const;
22   Sexo Atleta::sexo() const {
23       return _sexo;
24   }
25
26       //int anioNacimiento() const;
27   int Atleta::anioNacimiento() const {
28       return _anioNacimiento;
29   }
30
31       //Pais nacionalidad() const;
32   Pais Atleta::nacionalidad() const {
33       return _nacionalidad;
34   }
35
36       //int ciaNumber() const;
37   int Atleta::ciaNumber() const {
38       return _ciaNumber;
39   }
40
41       //Lista<Deporte> deportes() const;
42   Lista<Deporte> Atleta::deportes() const {
43       int i=0;
44       Lista<Deporte> deportes;
45       while(i<_deportes.longitud()) {
46           deportes.agregarAtras(_deportes.iesimo(i).first);
47           i++;
48       }
```

```
49      return deportes;
50  }
51      //int capacidad(const Deporte d) const;
52  int Atleta::capacidad(const Deporte d) const {
53      int i=0;
54      int capacidad;
55      while(i<_deportes.longitud()) {
56          if(_deportes.iesimo(i).first == d) {
57              capacidad = _deportes.iesimo(i).second;
58           }
59           i++;
60      }
61      return capacidad;
62  }
63
64      //void entrenarNuevoDeporte(const Deporte deporte, const int capacidad);
65  void Atleta::entrenarNuevoDeporte(const Deporte deporte, const int capacidad) {
66      Lista<pair<Deporte, int> > newDeportes;
67      bool agregado = false;
68      int i = 0;
69      while(i < _deportes.longitud()) {
70          if(deporte < _deportes.iesimo(i).first && !agregado) {
71              newDeportes.agregarAtras(make_pair(deporte, capacidad));
72              newDeportes.agregarAtras(_deportes.iesimo(i));
73              agregado = true;
74          } else if(deporte == _deportes.iesimo(i).first) {
75              newDeportes.agregarAtras(make_pair(deporte, capacidad));
76              agregado = true;
77          } else {
78              newDeportes.agregarAtras(_deportes.iesimo(i));
79          }
80          i++;
81      }
82      if(!agregado) {
83          newDeportes.agregarAtras(make_pair(deporte, capacidad));
84      }
85      _deportes = newDeportes;
86  }
87
88  bool Atleta::operator==(const Atleta& a) const {
89      bool igual = _nombre         == a._nombre         &&
90                   _sexo           == a._sexo           &&
91                   _anioNacimiento == a._anioNacimiento &&
92                   _nacionalidad   == a._nacionalidad   &&
93                   _ciaNumber      == a._ciaNumber;
94
95      int i=0;
96      if(a.deportes() == this->deportes()) {
97          while(i<a.deportes().longitud()) {
98              if(!(this->capacidad(a.deportes().iesimo(i)) == a.capacidad(a.deportes().iesimo(i)))) {
99                  igual = false;
100             }
101             i++;
102         }
103     }
104     else {
105         igual = false;
106     }
107     return igual;
108 }
109
110 void Atleta::mostrar(std::ostream& os) const {
111     string sexo = "Masculino";
112     if(_sexo == Femenino) sexo = "Femenino";
113
114     os << _nombre << " (#" << _ciaNumber << "). " <<
115           sexo << ". Año nac.: " << _anioNacimiento << ". " <<
116           "Nacionalidad: " << _nacionalidad << ". " << "Deportes: ";
117
118     if(_deportes.longitud() == 0) {
119         os << "ninguno.";
120     } else {
121         int i = 0;
122         while(i < _deportes.longitud()) {
123             os << endl << "   " << _deportes.iesimo(i).first << " (" <<
124                                   _deportes.iesimo(i).second << ")";
125             i++;
126         }
127     }
128
129     os << endl;
130 }
131
132 void Atleta::guardar(std::ostream& os) const {
133     //Hay que guardar A |Liu Song| |Masculino| 1972 |China| 123 [(|Tenis de Mesa|, 90)]
134     // A |Pepe| |Masculino| 1991 |Arg| 1 [(|Football|, 13),(|Karate|, 44),(|Zunga|, 17)]
135     os << "A |" << nombre() <<"| |";
136     if(sexo() == Masculino) {
137         os << "Masculino";
138     }
```

```
139       else {
140           os << "Femenino";
141       }
142       os << "| " << anioNacimiento() << " |" << nacionalidad() << "| " << ciaNumber();
143       os << " [";
144       int i=0;
145       while(i<deportes().longitud()) {
146           os << "(|" << deportes().iesimo(i) << "|, " << capacidad(deportes().iesimo(i)) << ")";
147           if((i+1)<deportes().longitud()) {
148               os << ",";
149           }
150           i++;
151       }
152       os << "]";
153   }
154
155   void Atleta::cargar (std::istream& is) {
156       //A |Pepe| |Masculino| 1991 |Arg| 1 [(|Football|, 13),(|Karate|, 44),(|Zunga|, 17)]
157       char c;
158       int capacidad;
159       string deporte, sexo;
160       is >> c;
161       is >> c;
162       getline(is, _nombre, '|');
163       is >> c;
164       getline(is, sexo, '|');
165       if(sexo == "Masculino") {
166           _sexo = Masculino;
167       }
168       else {
169           _sexo = Femenino;
170       }
171       is >> _anioNacimiento;
172       is >> c;
173       getline(is, _nacionalidad, '|');
174       is >> _ciaNumber;
175       //Empiezo con la lista de deportes, agarro [
176       is >> c;
177       Lista<pair<Deporte, int> > tempDeportes;
178       if(is.peek() != ']') {
179           bool looper = true;
180           while(looper) {
181               //Agarro (
182               is >> c;
183               //Agarro |
184               is >> c;
185               //Agarro el deporte
186               getline(is, deporte, '|');
187               //Agarro la ,
188               is >> c;
189               //Agarro capacidad
190               is >> capacidad;
191               //Agarro )
192               is >> c;
193               //Peek se fija sin agarrar el caracter, cual es el siguiente
194               if(is.peek() != ',') {
195                   looper = false;
196               }
197               else {
198                   //Saco la , que delimita otro deporte, ej, [(|Tenis de Mesa|, 90),(|Bmx|, 90)]
199                   is >> c;
200               }
201               entrenarNuevoDeporte(deporte, capacidad);
202           }
203       }
204       //Saco el ultimo ]
205       is >> c;
206   }
```

# competencia.h

```
1   #ifndef COMPETENCIA_H
2   #define COMPETENCIA_H
3
4   #include "atleta.h"
5   #include "lista.h"
6
7   class Competencia{
8       public:
9           Competencia();
10          Competencia(const Deporte d, const Sexo s, const Lista<Atleta>& participantes);
11          Categoria categoria() const;
12
13          Lista<Atleta> participantes() const;
14          bool finalizada() const;
15
16          Lista<Atleta> ranking() const;
```

```
17
18          Lista<Atleta> lesTocoControlAntidoping() const;
19          bool leDioPositivo(const Atleta& a) const;
20          void finalizar(const Lista<int>& posiciones, const Lista<pair<int, bool> >& control);
21          void linfordChristie(const int ciaNum);
22          bool gananLosMasCapaces() const;
23          void sancionarTramposos();
24
25          bool operator==(const Competencia& c) const;
26
27          void mostrar(std::ostream& os) const;
28          void guardar(std::ostream& os) const;
29          void cargar (std::istream& is);
30
31      private:
32          Categoria _categoria;
33          Lista<Atleta> _participantes;
34          bool _finalizada;
35          Lista<int> _ranking;
36          Lista<pair<int, bool> > _controlAntidoping;
37
38          enum {ENCABEZADO_ARCHIVO = 'C'};
39
40          Atleta atletaConCia(const int ciaNumber) const;
41          bool mismoDoping(const Competencia& c) const;
42          bool mismosParticipantes(const Competencia& c) const;
43  };
44
45  std::ostream & operator<<(std::ostream & os,const Competencia & c);
46
47  #endif // COMPETENCIA_H
```

# competencia.cpp

```
1   #include <iostream>
2   #include "competencia.h"
3   #include "lista.h"
4   #include "tipos.h"
5
6   Competencia::Competencia() {}
7   Competencia::Competencia(const Deporte d, const Sexo s, const Lista<Atleta>& participantes) {
8       _categoria = make_pair(d, s);
9       _participantes = participantes;
10      _finalizada = false;
11  }
12  Categoria Competencia::categoria() const {
13      return _categoria;
14  }
15
16  Lista<Atleta> Competencia::participantes() const {
17      return _participantes;
18  }
19
20  bool Competencia::finalizada() const {
21      return _finalizada;
22  }
23
24  Lista<Atleta> Competencia::ranking() const {
25      int i=0;
26      Lista<Atleta> atletas;
27      while(i<_ranking.longitud()) {
28          atletas.agregarAtras(this->atletaConCia(_ranking.iesimo(i)));
29          i++;
30      }
31      return atletas;
32  }
33
34  Lista<Atleta> Competencia::lesTocoControlAntidoping() const {
35      Lista<Atleta> atletas;
36      int i = 0;
37      while(i<_controlAntidoping.longitud()) {
38          atletas.agregarAtras(atletaConCia(_controlAntidoping.iesimo(i).first));
39          i++;
40      }
41      return atletas;
42  }
43
44  bool Competencia::leDioPositivo(const Atleta& a) const {
45      int i = 0;
46      bool leDio = false;
47      while(i<_controlAntidoping.longitud()) {
48          if(_controlAntidoping.iesimo(i).first == a.ciaNumber() && _controlAntidoping.iesimo(i).second) {
49              leDio = true;
50          }
51          i++;
52      }
53      return leDio;
```

```
54  }
55
56  void Competencia::finalizar(const Lista<int>& posiciones, const Lista<pair<int, bool> >& control) {
57      _finalizada = true;
58      _ranking = posiciones;
59      _controlAntidoping = control;
60  }
61
62  void Competencia::linfordChristie(const int ciaNum) {
63      Atleta atle = atletaConCia(ciaNum);
64      _participantes.sacar(atle);
65  }
66
67  bool Competencia::gananLosMasCapaces() const {
68      bool ret = true;
69      int i = 0;
70      while(_ranking.longitud() >= 2 && i < _ranking.longitud() - 1) {
71          int capacidadPuestoActual    = atletaConCia(_ranking.iesimo(i)).capacidad(_categoria.first);
72          int capacidadPuestoSiguiente = atletaConCia(_ranking.iesimo(i + 1)).capacidad(_categoria.first);
73          ret = ret && (capacidadPuestoActual >= capacidadPuestoSiguiente);
74          i++;
75      }
76      return ret;
77  }
78
79  void Competencia::sancionarTramposos() {
80      int i=0;
81      while(i<_controlAntidoping.longitud()) {
82          if(_controlAntidoping.iesimo(i).second) {
83              _ranking.sacar(_controlAntidoping.iesimo(i).first);
84          }
85          i++;
86      }
87  }
88
89  bool Competencia::operator==(const Competencia& c) const {
90      bool iguales = mismosParticipantes(c)     &&
91                     _categoria == c._categoria &&
92                     _finalizada == c._finalizada;
93      if(_finalizada && c._finalizada) {
94          iguales = iguales                  &&
95                    _ranking == c._ranking &&
96                    mismoDoping(c);
97      }
98      return iguales;
99  }
100
101 void Competencia::mostrar(std::ostream& os) const {
102     os << "Competencia:" << endl << "Categoria: " << _categoria.first << " ";
103     if(_categoria.second == Masculino) {
104         os << "Masculino";
105     }
106     else {
107         os << "Femeninio";
108     }
109     os << endl << "Participantes:";
110     int i=0;
111     while(i<participantes().longitud()) {
112         os << endl;
113         os << "  " << participantes().iesimo(i).nombre() <<
114               " (#" << participantes().iesimo(i).ciaNumber() << "). " <<
115               "Capacidad: " << participantes().iesimo(i).capacidad(_categoria.first);
116         i++;
117     }
118     os << endl << "Finalizada: ";
119     if(_finalizada) {
120         os << "Si" << endl;
121         int i = 0;
122         os << "Ranking: [";
123         while(i<ranking().longitud()) {
124             os << ranking().iesimo(i).nombre();
125             i++;
126             if(i<ranking().longitud()) {
127                 os << ",";
128             }
129         }
130         os << "]" << endl;
131         i=0;
132         os << "Dopping : [";
133         while(i<lesTocoControlAntidoping().longitud()) {
134             os << "<" << lesTocoControlAntidoping().iesimo(i).nombre() << ",";
135             os << leDioPositivo(lesTocoControlAntidoping().iesimo(i)) << ">";
136             i++;
137             if(i<lesTocoControlAntidoping().longitud()) {
138                 os << ",";
139             }
140         }
141         os << "]" << endl;
142     }
143     else {
```

```
144             os << "No" << endl;
145         }
146 
147     }
148 
149 
150     void Competencia::guardar(std::ostream& os) const {
151         os << "C (|" << _categoria.first << "|, |";
152         if(_categoria.second == Masculino) {
153             os << "Masculino";
154         }
155         else {
156             os << "Femenino";
157         }
158         os << "|) |";
159         if(finalizada()) {
160             os << "True";
161         }
162         else {
163             os << "False";
164         }
165         os << "| [";
166         int i=0;
167         while(i<participantes().longitud()) {
168             os << "(";
169             participantes().iesimo(i).guardar(os);
170             os << ")";
171             i++;
172             if(i<participantes().longitud()) {
173                 os << ",";
174             }
175         }
176         os << "] ";
177         if(finalizada()) {
178             os << "[";
179             i=0;
180             while(i<ranking().longitud()) {
181                 os << ranking().iesimo(i).ciaNumber();
182                 i++;
183                 if(i<ranking().longitud()) {
184                     os << ",";
185                 }
186             }
187             os << "] [";
188             i=0;
189             while(i<lesTocoControlAntidoping().longitud()) {
190                 os << "(" << lesTocoControlAntidoping().iesimo(i).ciaNumber() << ", " << "|";
191                 if(leDioPositivo(lesTocoControlAntidoping().iesimo(i))) {
192                     os << "True" << "|)";
193                 }
194                 else {
195                     os << "False" << "|)";
196                 }
197                 i++;
198                 if(i<lesTocoControlAntidoping().longitud()) {
199                     os << ",";
200                 }
201             }
202             os << "]";
203         }
204         else {
205             os << "[] []";
206         }
207     }
208 
209     /*
210     C (|Rugby|, |Masculino|) |True|
211     [(A |Juan| |Masculino| 1920 |Argentina| 1 [(|Football|, 35), (|Rugby|, 10)]),
212     (A |Jorge| |Masculino| 1930 |Argentina| 2 [(|Football|, 32), (|Rugby|, 20)]),
213     (A |Jackson| |Masculino| 1935 |Escocia| 6 [(|Basket|, 25), (|Football|, 40), (|Rugby|, 5)])]
214     [1, 6] [(1, |True|), (6, |True|)]
215      */
216 
217     void Competencia::cargar (std::istream& is) {
218         char c;
219         string stringDeporte;
220         string stringSexo;
221         Sexo sexo;
222         string stringFinalizada;
223         //Saco C , ( y |
224         is >> c;
225         is >> c;
226         is >> c;
227         getline(is, stringDeporte, '|');
228         //Saco , y |
229         is >> c >> c;
230         getline(is, stringSexo, '|');
231         if(stringSexo == "Masculino") {
232             sexo = Masculino;
233         }
```

```
234         else {
235             sexo = Femenino;
236         }
237         //Saco ) y |
238         is >> c >> c;
239         getline(is, stringFinalizada, '|');
240         if(stringFinalizada == "True") {
241             _finalizada = true;
242         }
243         else {
244             _finalizada = false;
245         }
246         //Saco [
247         is >> c;
248         Lista<Atleta> atletas;
249         //Me fijo si atletas no es vacio
250         if(is.peek() != ']') {
251             bool loop = true;
252             while(loop) {
253                 //Saco (
254                 is >> c;
255                 Atleta atle;
256                 atle.cargar(is);
257                 atletas.agregarAtras(atle);
258                 //Saco )
259                 is >> c;
260                 if(is.peek() != ',') {
261                     loop = false;
262                 }
263                 else {
264                     //Saco la ,
265                     is >> c;
266                 }
267             }
268         }
269         //Saco ]
270         is >> c;
271         _participantes = atletas;
272         _categoria = make_pair(stringDeporte, sexo);
273         if(_finalizada) {
274             Lista<int> ranking;
275             Lista<pair<int, bool> > doping;
276             bool loop = true;
277             //Saco [
278             is >> c;
279             //Me fijo si el ranking no esta vacio
280             if(is.peek() != ']') {
281                 while(loop) {
282                     int ciaNumber;
283                     is >> ciaNumber;
284                     if(is.peek() != ',') {
285                         loop = false;
286                     }
287                     else {
288                         //Saco la ,
289                         is >> c;
290                     }
291                     ranking.agregarAtras(ciaNumber);
292                 }
293             }
294             //Saco ] y [
295             is >> c >> c;
296             //Me fijo si el doping no esta vacio
297             if(is.peek() != ']') {
298                 loop = true;
299                 while(loop) {
300                     int ciaNumber;
301                     bool positive;
302                     string dopingStatus;
303                     //Saco (, ciaNumber, la , y |
304                     is >> c >> ciaNumber >> c >> c;
305                     getline(is, dopingStatus, '|');
306                     //Saco )
307                     is >> c;
308                     if(is.peek() != ',') {
309                         loop = false;
310                     }
311                     else {
312                         //Saco la ,
313                         is >> c;
314                     }
315                     if(dopingStatus == "True") {
316                         positive = true;
317                     }
318                     else {
319                         positive = false;
320                     }
321                     doping.agregarAtras(make_pair(ciaNumber, positive));
322                 }
323             }
```

```
324          //Saco ]
325          is >> c;
326          finalizar(ranking, doping);
327      }
328      else {
329          //Saco [] []
330          is >> c >> c >> c >> c;
331      }
332  }
333
334  /*******************************
335   *          AUXILIARES         *
336   *******************************/
337
338  Atleta Competencia::atletaConCia(const int ciaNumber) const {
339      int i = 0;
340      Atleta atle;
341      while(i<_participantes.longitud()) {
342          if(_participantes.iesimo(i).ciaNumber() == ciaNumber) {
343              atle = _participantes.iesimo(i);
344          }
345          i++;
346      }
347      return atle;
348  }
349
350  bool Competencia::mismosParticipantes(const Competencia& c) const {
351      bool iguales = _participantes.longitud() == c._participantes.longitud();
352      int i = 0;
353      while(i < _participantes.longitud()) {
354          iguales = iguales && c._participantes.pertenece(_participantes.iesimo(i));
355          i++;
356      }
357      return iguales;
358  }
359
360  bool Competencia::mismoDoping(const Competencia& c) const {
361      bool iguales = _controlAntidoping.longitud() == c._controlAntidoping.longitud();
362      int i = 0;
363      while(i < _controlAntidoping.longitud()) {
364          iguales = iguales && c._controlAntidoping.pertenece(_controlAntidoping.iesimo(i));
365          i++;
366      }
367      return iguales;
368  }
```

# jjoo.h

```
1   #ifndef JJOO_H
2   #define JJOO_H
3
4   #include "competencia.h"
5
6   typedef pair<int, pair<int, int> > infoM;
7
8   class JJOO{
9       public:
10
11          JJOO();
12          // |cronograma|==cantDias. Si un día no hay competencias, que esté la lista vacía.
13          JJOO(const int anio, const Lista<Atleta>& atletas, const Lista<Lista<Competencia> >& competenciasPorDia);
14          int anio() const;
15          Lista<Atleta> atletas() const;
16          int cantDias() const;
17          int jornadaActual() const;
18          Lista<Competencia> cronograma(const int dia) const;
19
20          Lista<Competencia> competencias() const;
21          Lista<Competencia> competenciasFinalizadasConOroEnPodio() const;
22
23          Lista<Atleta> dePaseo() const;
24          Lista<pair<Pais,Lista<int> > > medallero() const;
25          int boicotPorDisciplina(const Categoria cat, const Pais p);
26          Lista<Atleta> losMasFracasados(const Pais p) const;
27          void liuSong(const Atleta& a, const Pais p);
28          Atleta stevenBradbury() const;
29          bool uyOrdenadoAsiHayUnPatron() const;
30          Lista<Pais> sequiaOlimpica() const;
31          void transcurrirDia();
32
33          bool operator==(const JJOO& j) const;
34
35          void mostrar(std::ostream& os) const;
36          void guardar(std::ostream& os) const;
37          void cargar (std::istream& is);
38
39      private:
```

```
40          int _anio;
41          Lista<Atleta> _atletas;
42          int _jornadaActual;
43          Lista<Lista<Competencia> > _competenciasPorDia;         // En la i-ésima posición de la lista, las competencias del día
44
45          enum {ENCABEZADO_ARCHIVO = 'J'};
46
47          // Auxiliares generales
48          Lista<Pais> paises() const;
49          Lista<Atleta> participantes() const;
50
51          // Auxiliares JJOO::medallero()
52          Lista<pair<Pais,Lista<int> > > ordenarMedallero(const Lista<pair<Pais,Lista<int> > > &) const;
53
54          // Auxiliares JJOO::sequiaOlimpica()
55          bool ganoMedallasEseDia(Pais p, int x) const;
56          int maximaDistanciaEntreJornadas(Lista<int> jornadas) const;
57
58          // Auxiliares JJOO::transcurrirDia()
59          Competencia finalizarCompetencia(const Competencia& competencia) const;
60          Lista<int> generarRanking(const Competencia& competencia) const;
61          Lista<pair<int, bool> > generarAntidoping(const Competencia& competencia) const;
62          void reemplazarCronogramaJornadaActual(Lista<Competencia> nuevoCronograma);
63
64          // Auxiliares JJOO::operator==()
65          bool mismosAtletas(const JJOO& c) const;
66          bool mismoCronograma(const JJOO& j) const;
67  };
68
69  std::ostream & operator<<(std::ostream & os,const JJOO & j);
70
71  #endif // JJOO_H
```

# jjoo.cpp

```
1   #include <iostream>
2   #include "tipos.h"
3   #include "lista.h"
4   #include "jjoo.h"
5
6   JJOO::JJOO () {
7       _jornadaActual = 1;
8       _competenciasPorDia.agregar(Lista<Competencia>());
9   }
10
11  JJOO::JJOO (const int anio, const Lista<Atleta>& atletas, const Lista<Lista<Competencia> >& competenciasPorDia){
12      _anio = anio;
13      _atletas = atletas;
14      _jornadaActual = 1;
15      _competenciasPorDia = competenciasPorDia;
16  }
17
18  int JJOO::anio() const{
19      return _anio;
20  }
21
22  Lista<Atleta> JJOO::atletas() const {
23      return _atletas;
24  }
25
26  int JJOO::cantDias() const {
27      return _competenciasPorDia.longitud();
28  }
29
30  int JJOO::jornadaActual() const {
31      return _jornadaActual;
32  }
33
34  Lista<Competencia> JJOO::cronograma(const int dia) const {
35      return _competenciasPorDia.iesimo(dia - 1);
36  }
37
38  Lista<Competencia> JJOO::competencias() const {
39      Lista<Competencia> competencias;
40      int i = 0;
41      while (i < cantDias()) {
42          competencias.concatenar(_competenciasPorDia.iesimo(i));
43          i++;
44      }
45      return competencias;
46  }
47
48  Lista<Competencia> JJOO::competenciasFinalizadasConOroEnPodio() const {
49      Lista<Competencia> result;
50      int i = 0;
51      while (i < competencias().longitud())   {
52          Competencia actual = competencias().iesimo(i);
```

```
53          if (actual.finalizada() && actual.ranking().longitud() > 0) {
54              result.agregar(actual);
55          }
56          i++;
57      }
58      return result;
59  }
60
61  Lista<Atleta> JJOO::dePaseo() const {
62      Lista<Atleta> result;
63      int i = 0;
64      while (i < _atletas.longitud()) {
65          if (!participantes().pertenece(_atletas.iesimo(i))) {
66              result.agregar(_atletas.iesimo(i));
67          }
68          i++;
69      }
70      return result;
71  }
72
73  // Devuelve una lista de atletas donde cada atleta aparece tantas
74  // veces como la cantidad de competencias en las que participa.
75  Lista<Atleta> JJOO::participantes() const {
76      Lista<Atleta> participantes;
77      int i = 0;
78      while (i < competencias().longitud()) {
79          participantes.concatenar(competencias().iesimo(i).participantes());
80          i++;
81      }
82      return participantes;
83  }
84
85  Lista<pair<Pais, Lista<int> > > JJOO::medallero() const {
86      Lista<Pais> paises = this->paises();
87      Lista<pair<Pais, Lista<int> > > medallero;
88
89      // Recorro la lista de países.
90      int i = 0;
91      while(i < paises.longitud()) {
92          Pais pais = paises.iesimo(i);
93          int oros = 0;
94          int platas = 0;
95          int bronces = 0;
96
97          // Recorro los rankings de todas las competencias y acumulo las medallas del país actual.
98          int j = 0;
99          while(j < competencias().longitud()) {
100             Lista<Atleta> ranking = competencias().iesimo(j).ranking();
101             if(ranking.longitud() > 0 && ranking.iesimo(0).nacionalidad() == pais) oros++;
102             if(ranking.longitud() > 1 && ranking.iesimo(1).nacionalidad() == pais) platas++;
103             if(ranking.longitud() > 2 && ranking.iesimo(2).nacionalidad() == pais) bronces++;
104             j++;
105         }
106
107         // Si el país ganó alguna medalla, agrego al medallero la tupla país/medallas.
108         if(oros + platas + bronces > 0) {
109             Lista<int> medallas;
110             medallas.agregarAtras(oros);
111             medallas.agregarAtras(platas);
112             medallas.agregarAtras(bronces);
113             medallero.agregar(make_pair(pais, medallas));
114         }
115
116         i++;
117     }
118
119     // Devuelvo el medallero ordenado
120     return ordenarMedallero(medallero);
121 }
122
123 Lista<Pais> JJOO::paises() const {
124     Lista<Pais> paises;
125     int i = 0;
126     while(i < _atletas.longitud()) {
127         Pais actual = _atletas.iesimo(i).nacionalidad();
128         if(!paises.pertenece(actual)) {
129             paises.agregar(actual);
130         }
131         i++;
132     }
133     return paises;
134 }
135
136 Lista<pair<Pais,Lista<int> > > JJOO::ordenarMedallero(const Lista<pair<Pais,Lista<int> > > & medallero) const {
137
138     // Guardo acá mi copia ordenada del medallero.
139     Lista<pair<Pais, Lista<int> > > ordenado;
140
141     // Recorro el medallero.
142     int i = 0;
```

```
143     while(i < medallero.longitud()) {
144
145         // Obtengo la tupla país/medallas actual y las medallas (para más legibilidad.)
146         pair<Pais, Lista<int> > tuplaActual = medallero.iesimo(i);
147         int oros    = tuplaActual.second.iesimo(0);
148         int platas  = tuplaActual.second.iesimo(1);
149         int bronces = tuplaActual.second.iesimo(2);
150
151         // Guardo acá una copia de las tuplas país/medallas que ya ordené, junto con
152         // la tupla país/medallas actual en la posición correcta.
153         Lista<pair<Pais, Lista<int> > > nuevoOrdenado;
154         bool agregado = false;
155
156         // Recorro las tuplas país/medallas que ya ordené.
157         int j = 0;
158         while(j < ordenado.longitud()) {
159             int orosOtro    = ordenado.iesimo(j).second.iesimo(0);
160             int platasOtro  = ordenado.iesimo(j).second.iesimo(1);
161             int broncesOtro = ordenado.iesimo(j).second.iesimo(2);
162
163             // Ubico a la tupla país/medallas actual en la posición correcta dentro de la lista ordenada.
164             if( !agregado &&
165                ((oros >  orosOtro) ||
166                 (oros == orosOtro && platas >  platasOtro) ||
167                 (oros == orosOtro && platas == platasOtro && bronces >= broncesOtro))) {
168                 nuevoOrdenado.agregarAtras(tuplaActual);
169                 agregado = true;
170             }
171
172             // Dejo la tupla que ya había ordenado en la posición que le corresponde.
173             nuevoOrdenado.agregarAtras(ordenado.iesimo(j));
174             j++;
175         }
176
177         // Si luego de recorrer las tuplas país/medallas ya ordenadas resulta que todavía no agregué
178         // la tupla país/medallas actual, es porque la tupla actual es la que menos medallas tiene, y
179         // por lo tanto su posición es al final de la lista.
180         if(!agregado) nuevoOrdenado.agregarAtras(tuplaActual);
181
182         // Finalmente, reemplazo el medallero ordenado actual por mi nuevo medallero ordenado, que
183         // incluye a la tupla país/medallas actual.
184         ordenado = nuevoOrdenado;
185
186         i++;
187     }
188
189     return ordenado;
190 }
191
192 int JJOO::boicotPorDisciplina(const Categoria categoria, const Pais pais) {
193     int sacados = 0;
194
195     // Guardo acá las nuevas competencias por día después de boicotear al país.
196     Lista<Lista<Competencia> > nuevaCompetenciasPorDia;
197
198     // Recorro todas las jornadas.
199     int i = 0;
200     while(i < _competenciasPorDia.longitud()) {
201
202         // Guardo acá las competencias finales de la jornada i-ésima.
203         Lista<Competencia> competenciasEnElDia;
204
205         // Recorro las competencias de la jornada i-ésima.
206         int j = 0;
207         while(j < _competenciasPorDia.iesimo(i).longitud()) {
208                 Competencia competencia = _competenciasPorDia.iesimo(i).iesimo(j);
209
210             // Si es la competencia de la categoría buscada, la boicoteo.
211             if(competencia.categoria() == categoria) {
212
213                 // Guardo acá el ciaNumber de los atletas boicoteados
214                 Lista<int> ciaNumberDeSacados;
215
216                 // Guardo acá los participantes que no boicoteé.
217                 Lista<Atleta> participantes;
218
219                 // Recorro la lista de participantes de la competencia actual.
220                 int h = 0;
221                 while(h < competencia.participantes().longitud()) {
222                         Atleta participante = competencia.participantes().iesimo(h);
223
224                         // Si el participante actual es de la nacionalidad a boicotear, lo dejo
225                         // fuera de la nueva lista de participantes y guardo su ciaNumber.
226                     if(participante.nacionalidad() == pais) {
227                         ciaNumberDeSacados.agregarAtras(participante.ciaNumber());
228                         sacados++;
229                     }
230
231                     // En caso contrario, lo agrego a la lista de nuevos participantes.
232                     else {
```

```
                        participantes.agregarAtras(participante);
                    }

                    h++;
                }

                // Creo la nueva competencia con los participantes boicoteados.
                Competencia competenciaBoicoteada(competencia.categoria().first,
                                                   competencia.categoria().second,
                                                   participantes);

                // Finalizo la nueva competencia de ser necesario.
                if(competencia.finalizada()) {

                        // Guardo acá el ranking de la nueva competencia.
                    Lista<int> ranking;

                    // Recorro el ranking de la competencia original.
                    int h = 0;
                    while(h < competencia.ranking().longitud()) {
                        int ciaNumberAtletaActual = competencia.ranking().iesimo(h).ciaNumber();

                        // Si el atleta actual no fue boicoteado, lo agrego al ranking en la posición que estaba.
                        if(!ciaNumberDeSacados.pertenece(ciaNumberAtletaActual)) {
                            ranking.agregarAtras(ciaNumberAtletaActual);
                        }

                        h++;
                    }

                    // Guardo acá el control antidoping de la nueva competencia.
                    Lista<pair<int, bool> > antidoping;

                    // Recorro el control antidoping de la competencia original.
                    h = 0;
                    while(h < competencia.lesTocoControlAntidoping().longitud()) {
                        Atleta controladoActual = competencia.lesTocoControlAntidoping().iesimo(h);

                        // Si el atleta controlado actual no fue boicoteado, lo agrego al control antidoping.
                        if(!ciaNumberDeSacados.pertenece(controladoActual.ciaNumber())) {
                            antidoping.agregarAtras(make_pair(controladoActual.ciaNumber(),
                                                              competencia.leDioPositivo(controladoActual)));
                        }

                        h++;
                    }

                    // Finalizo la competencia con el ranking y antidoping boicoteados.
                    competenciaBoicoteada.finalizar(ranking, antidoping);
                }

                // Agrego la competencia boicoteada a la nueva lista de competencias para la jornada i-ésima.
                competenciasEnElDia.agregarAtras(competenciaBoicoteada);
            }

            // Si no es la que quiero boicotear, la dejo como está.
            else {
                competenciasEnElDia.agregarAtras(competencia);
            }

            j++;
        }

        // Agrego las nuevas competencias del día actual a la nueva lista de competencias por día.
        nuevaCompetenciasPorDia.agregarAtras(competenciasEnElDia);

        i++;
    }

    _competenciasPorDia = nuevaCompetenciasPorDia;
    return sacados;
}

Lista<Atleta> JJOO::losMasFracasados(const Pais p) const {
//Crea una lista con los atletas que ganaron medallas
    int h=0;
    Lista<Atleta> rank;
    while(h<competenciasFinalizadasConOroEnPodio().longitud()){
        if (competenciasFinalizadasConOroEnPodio().iesimo (h).ranking().longitud()<3){
            rank.concatenar(competenciasFinalizadasConOroEnPodio().iesimo(h).ranking());
        }else{
            // 0, 1 y 2 en vez de 1,2,3
            rank.agregar(competenciasFinalizadasConOroEnPodio().iesimo (h).ranking().iesimo(0));
            rank.agregar(competenciasFinalizadasConOroEnPodio().iesimo (h).ranking().iesimo(1));
            rank.agregar(competenciasFinalizadasConOroEnPodio().iesimo (h).ranking().iesimo(2));
        }
        h++;
    }
//Este ciclo me da una lista de atletas del pais p que no ganaron ninguna medalla
    Lista<Atleta> atles;
```

```
323      int n = 0;
324      while (n < atletas().longitud()){
325          Atleta competidor = atletas().iesimo(n);
326          if ((competidor.nacionalidad()== p) && !rank.pertenece(competidor)) {
327              atles.agregar(competidor);
328          }
329          n++;
330      }
331  //Divido en dos casos, si la lista es vacia o si tiene al menos un elemento
332      Lista<Atleta> atlesFracasados;
333      if (atles.longitud()==0){
334          atlesFracasados=atles;
335      }
336      else{
337  //Acá me fijo cual es el atleta que participó en mas competencias
338          int k = 0, j=0;
339          Atleta maxAp = atles.iesimo(k);
340          Lista<Atleta> atlecomp= participantes();
341          while (j < atles.longitud()){
342              if (atlecomp.cantidadDeApariciones(maxAp)>atlecomp.cantidadDeApariciones(atles.iesimo(j))){
343                      maxAp = atles.iesimo(j);
344                  }
345              j++;
346          }
347  //Acá creo la lista con todos los atletas que aparecen tantas veces como maxAp
348          atlesFracasados.agregar(maxAp);
349          int m=0;
350          while (m < atles.longitud()){
351              //No estabas viendo si el que tenia la misma cantidad de apariciones era el mismo
352              if(   atlecomp.cantidadDeApariciones(maxAp) == atlecomp.cantidadDeApariciones(atles.iesimo(m)) &&
353                  !(atles.iesimo(m) == maxAp)) {
354                  atlesFracasados.agregar(atles.iesimo(m));
355              }
356              m++;
357          }
358      }
359    return atlesFracasados;
360  }
361
362  void JJOO::liuSong(const Atleta& a, const Pais p) {
363      Atleta atletaNacionalizado(a.nombre(), a.sexo(), a.anioNacimiento(), p, a.ciaNumber());
364      int i = 0;
365      while(i<a.deportes().longitud()) {
366          atletaNacionalizado.entrenarNuevoDeporte(a.deportes().iesimo(i), a.capacidad(a.deportes().iesimo(i)));
367          i++;
368      }
369
370      //Saco el atleta y lo agrego nacionalizado en atletas;
371      _atletas.sacar(a);
372      _atletas.agregarAtras(atletaNacionalizado);
373
374      i=0;
375      int j = 0;
376      //Lista que va a reemplazar a la vieja lista de listas competencias
377      Lista<Lista<Competencia> > nuevaCompetenciasPorDia;
378      while(i<_competenciasPorDia.longitud()) {
379          //Lista de competencias que va a reemplazar a la anterior en ese dia
380          Lista<Competencia> nuevaCompetenciasEnDia;
381          j = 0;
382          while(j<_competenciasPorDia.iesimo(i).longitud()) {
383              Competencia viejaCompe = _competenciasPorDia.iesimo(i).iesimo(j);
384              //Me fijo si esa competencia tiene al atleta, si no, la dejo como estaba
385              if(viejaCompe.participantes().pertenece(a)) {
386                  //Creo la nueva lista de participantes
387                  Lista<Atleta> nuevosParticipantes = viejaCompe.participantes();
388                  nuevosParticipantes.sacar(a);
389                  nuevosParticipantes.agregarAtras(atletaNacionalizado);
390                  //
391                  //Guardo la categoria
392                  Deporte dep = viejaCompe.categoria().first;
393                  Sexo sex = viejaCompe.categoria().second;
394                  //
395                  //Creo lo que va a ser la nueva competencia con el atleta cambiado
396                  //
397                  Competencia nuevaCompe(dep, sex, nuevosParticipantes);
398
399                  //Si esta finalizada, asigno el ranking y el doping
400                  if(viejaCompe.finalizada()) {
401                      int h = 0;
402                      Lista<int> ranking;
403                      Lista<pair<int, bool> > control;
404                      while(h<viejaCompe.ranking().longitud()) {
405                          ranking.agregarAtras(viejaCompe.ranking().iesimo(h).ciaNumber());
406                          h++;
407                      }
408                      h = 0;
409                      while(h<viejaCompe.lesTocoControlAntidoping().longitud()) {
410                          bool leToco = viejaCompe.leDioPositivo(viejaCompe.lesTocoControlAntidoping().iesimo(h));
411                          int suCiaNumber = viejaCompe.lesTocoControlAntidoping().iesimo(h).ciaNumber();
412                          pair<int, bool> par = make_pair(suCiaNumber, leToco);
```

```
                        control.agregarAtras(par);
                        h++;
                    }
                    //Finalizo
                    nuevaCompe.finalizar(ranking, control);
                }
                //Agrego a lo que va a ser mi nueva lista de competencias en ese dia
                nuevaCompetenciasEnDia.agregarAtras(nuevaCompe);
            }
            else {
                nuevaCompetenciasEnDia.agregarAtras(viejaCompe);
            }
            j++;
        }
        i++;
        nuevaCompetenciasPorDia.agregarAtras(nuevaCompetenciasEnDia);
    }
    _competenciasPorDia = nuevaCompetenciasPorDia;
}

Atleta JJOO::stevenBradbury() const {
    // Guardo el atleta con menor capacidad, y la menor de sus
    // capacidades de los deportes en los que salió campeón.
    Atleta atletaMenosCapaz;
    int capacidadMenosCapaz;

    // Recorro la lista de competencias finalizadas con oro.
    int i = 0;
    while(i < competenciasFinalizadasConOroEnPodio().longitud()) {

        // Guardo el campeón de la competencia actual y
        // su capacidad en el deporte de la misma.
        Competencia competencia = competenciasFinalizadasConOroEnPodio().iesimo(i);
        Atleta campeon = competencia.ranking().cabeza();
        int capacidadCampeon = campeon.capacidad(competencia.categoria().first);

        // Si es la primera competencia que recorro, entonces tomo al campeón
        // de la misma como el atleta menos capaz hasta el momento.
        if(i == 0) {
            atletaMenosCapaz = campeon;
            capacidadMenosCapaz = capacidadCampeon;
        }

        // En caso contrario, si el campeón de esta competencia tiene una menor capacidad
        // que el atleta menos capaz hasta el momento, lo tomo como el menso capaz.
        else {
            if(capacidadCampeon < capacidadMenosCapaz) {
                atletaMenosCapaz = campeon;
                capacidadMenosCapaz = capacidadCampeon;
            }
        }

        i++;
    }

    return atletaMenosCapaz;
}

bool JJOO::uyOrdenadoAsiHayUnPatron() const {

    // Guardo acá la secuencia de mejores países.
    Lista<Pais> mejoresPaises;

    // Recorro los juegos día por día.
    int dia = 1;
    while(dia <= cantDias()) {

        // Guardo acá la cantidad de oros por país.
        Lista<pair<Pais, int> > oros;

        // Recorro el las competencias de este día.
        int i = 0;
        while(i < cronograma(dia).longitud()) {
            Competencia actual = cronograma(dia).iesimo(i);

            // Si la competencia actual está finalizada y alguien se llevó el oro,
            // voy a buscar la nacionalidad del medallista y voy a sumarle un oro a
            // ese país en mi lista de oros por país. En caso contrario, ignoro
            // esta competencia.
            if(actual.finalizada() && actual.ranking().longitud() > 0) {
                Pais pais = actual.ranking().iesimo(0).nacionalidad();

                // Busco el país del medallista en mi lista de oros por país.
                bool encontrado = false;
                int j = 0;
                while(j < oros.longitud() && !encontrado) {

                    // Si encontré el país del medallista en mi lista de oros por país,
                    // aumento en uno la cantidad de oros de ese país.
                    if(oros.iesimo(j).first == pais) {
```

```
503                         pair<Pais, int> nuevaTupla = make_pair(oros.iesimo(j).first, oros.iesimo(j).second + 1);
504                         oros.eliminarPosicion(j);
505                         oros.agregarAtras(nuevaTupla);
506                         encontrado = true;
507                     }
508
509                     j++;
510                 }
511
512                 // Si el país del medallista no estaba en la lista de oros por país,
513                 // agrego ese país a la lista, con cantidad de oros uno.
514                 if(!encontrado) {
515                     oros.agregarAtras(make_pair(pais, 1));
516                 }
517             }
518
519             i++;
520         }
521
522         // Si hubieron oros este día, busco al mejor país y lo agrego a mejoresPaises.
523         if(oros.longitud() > 0) {
524
525             // Guardo acá el mejor país hasta el momento (tupla país/oros.)
526             pair<Pais, int> mejorPais;
527
528             // Recorro la lista de oros por país.
529             i = 0;
530             while(i < oros.longitud()) {
531
532                 // Si el país i-ésimo es el primero de la lista de oros, o si tiene más oros que el mejor país
533                 // hasta el momento, o si tiene igual cantidad de oros pero es  lexicográficamente menor,
534                 // entonces convierto al país i-ésimo en el mejor país hasta el momento.
535                 if( i == 0 ||
536                     oros.iesimo(i).second > mejorPais.second ||
537                    (oros.iesimo(i).second == mejorPais.second && oros.iesimo(i).first < mejorPais.first)) {
538                     mejorPais = oros.iesimo(i);
539                 }
540
541                 i++;
542             }
543
544             // Finalmente, agrego al mejor país del día a la lista de mejores países.
545             mejoresPaises.agregarAtras(mejorPais.first);
546         }
547
548         dia++;
549     }
550
551     bool hayPatron = true;
552
553     // Busco patrón si y sólo si hay tres o más mejores países.
554     if(mejoresPaises.longitud() >= 3) {
555
556         // Recorro la lista de mejores países hasta el anteúltimo elemento.
557         int i = 0;
558         while(i < mejoresPaises.longitud() - 1) {
559             Pais actual    = mejoresPaises.iesimo(i);
560             Pais siguiente = mejoresPaises.iesimo(i + 1);
561
562             // Recorro todos los países a la derecha del país actual
563             int j = i + 1;
564             while(j < mejoresPaises.longitud() - 1) {
565
566                 // Si el país j-ésimo es el mismo que el actual, verifico que el patrón
567                 // se cumpla (el elemento siguiente siempre debe ser el mismo.)
568                 if(mejoresPaises.iesimo(j) == actual) {
569                     hayPatron = hayPatron && (mejoresPaises.iesimo(j + 1) == siguiente);
570                 }
571
572                 j++;
573             }
574
575             i++;
576         }
577     }
578
579     return hayPatron;
580 }
581
582 Lista<Pais> JJOO::sequiaOlimpica() const {
583     Lista<Pais> paises = this->paises();
584
585     // Guardo acá una lista de tuplas país/máxima cantidad de días que pasaron sin ganar medallas.
586     Lista<pair<Pais,int> > paisDiasSinGanar;
587
588     // Recorro todos los países.
589     int i = 0;
590     while (i < paises.longitud()) {
591         Pais pais = paises.iesimo(i);
592
```

```
593         // Armo una lista con las jornadas ganadoras de ese país.
594         Lista<int> jornadasGanadoras;
595
596         // Agrego un cero al principio de la lista para poder calcular
597         // las diferencias entre días (como en la especificación.)
598         jornadasGanadoras.agregar(0);
599
600         // Recorro las jornadas hasta la actual, excluyéndola.
601         int j = 1;
602         while (j < jornadaActual()){
603
604             // Si el país ganó alguna medalla en la jornada
605             // actual, agrego la jornada a la lista.
606             if(ganoMedallasEseDia(pais, j)) {
607                 jornadasGanadoras.agregarAtras(j);
608             }
609             j++;
610         }
611
612         // Agrego la jornada actual al final de la lista, para poder calcular
613         // las diferencias entre días (otra vez, como en la especificación.)
614         jornadasGanadoras.agregarAtras(jornadaActual());
615
616         // Calculo la máxima diferencia entre días, y agrego
617         // la tupla país/diferencia de días a la lista.
618         paisDiasSinGanar.agregarAtras(make_pair(pais, maximaDistanciaEntreJornadas(jornadasGanadoras)));
619
620         i++;
621     }
622
623     // Busco la máxima cantidad de días que algún país pasó sin ganar medallas.
624     int maximosDiasSinGanar;
625     i = 0;
626     while(i < paisDiasSinGanar.longitud()) {
627         if(i == 0 || paisDiasSinGanar.iesimo(i).second > maximosDiasSinGanar) {
628             maximosDiasSinGanar = paisDiasSinGanar.iesimo(i).second;
629         }
630         i++;
631     }
632
633     // Me quedo con los países cuya mayor cantidad de días sin ganar medallas
634     // es mayor o igual que la del resto de los países.
635     Lista<Pais> secos;
636     i = 0;
637     while (i < paisDiasSinGanar.longitud()) {
638         if(paisDiasSinGanar.iesimo(i).second == maximosDiasSinGanar) {
639             secos.agregar(paisDiasSinGanar.iesimo(i).first);
640         }
641         i++;
642     }
643
644     return secos;
645 }
646
647 int JJOO::maximaDistanciaEntreJornadas(Lista<int> jornadas) const {
648     // Puedo asumir que recibo dos o más jornadas.
649
650     // Guardo acá las distancias calculadas.
651     Lista<int> distancias;
652
653     // Calculo todas las distancias entre jornadas.
654     int i = 1;
655     while (i < jornadas.longitud()){
656         distancias.agregarAtras(jornadas.iesimo(i) - jornadas.iesimo(i - 1));
657         i++;
658     }
659
660     // Busco la máxima distancia.
661     int maximaDistancia;
662     i = 0;
663     while(i < distancias.longitud()) {
664         if(i == 0 || distancias.iesimo(i) > maximaDistancia) {
665             maximaDistancia = distancias.iesimo(i);
666         }
667         i++;
668     }
669
670     return maximaDistancia;
671 }
672
673 bool JJOO::ganoMedallasEseDia(Pais pais, int dia) const{
674     bool gano = false;
675
676     // Recorro el cronograma del día.
677     int i = 0;
678     while(i < cronograma(dia).longitud()) {
679         Competencia competencia = cronograma(dia).iesimo(i);
680
681         // Recorro el ranking de la competencia actual hasta el tercer puesto.
682         int j = 0;
```

```
683         while(j < competencia.ranking().longitud() && j < 3) {
684
685             // El valor de retorno es true sólo si el país ganó alguna medalla.
686             gano = gano || competencia.ranking().iesimo(j).nacionalidad() == pais;
687             j++;
688         }
689
690         i++;
691     }
692
693     return gano;
694 }
695
696 void JJOO::transcurrirDia() {
697     // Guardo acá el nuevo cronograma de la jornada actual.
698     Lista<Competencia> nuevoCronogramaJornadaActual;
699
700     // Recorro las competencias de la jornada actual.
701     int i = 0;
702     while(i < cronograma(jornadaActual()).longitud()) {
703         Competencia competencia = cronograma(jornadaActual()).iesimo(i);
704
705         // Si la competencia no estaba finalizada, la finalizo.
706         if (!competencia.finalizada()) {
707             competencia = finalizarCompetencia(competencia);
708         }
709
710         // Agrego competencia al nuevo cronograma.
711         nuevoCronogramaJornadaActual.agregarAtras(competencia);
712         i++;
713     }
714
715     // Reemplazo el cronograma de la jornada actual por el nuevo cronograma finalizado.
716     reemplazarCronogramaJornadaActual(nuevoCronogramaJornadaActual);
717
718     // Transcurro el día.
719     _jornadaActual++;
720 }
721
722 Competencia JJOO::finalizarCompetencia(const Competencia& competencia) const  {
723     // Creo una copia de la competencia.
724     Competencia finalizada(competencia.categoria().first,
725                            competencia.categoria().second,
726                            competencia.participantes());
727
728     // Finalizo la competencia generando un ranking y un control antidoping.
729     finalizada.finalizar(generarRanking(competencia), generarAntidoping(competencia));
730
731     return finalizada;
732 }
733
734 Lista<int> JJOO::generarRanking(const Competencia& competencia) const {
735     // Guardo acá el ranking construído.
736     Lista<int> ranking;
737
738     Lista<Atleta> participantesSinRankear = competencia.participantes();
739
740     // Rankeo todos los participantes hasta quedarme sin participantes para rakear.
741     while(participantesSinRankear.longitud() > 0) {
742
743         // Busco el participante con menos capacidad entre los que aún no fueron rankeados.
744         Atleta peorParticipante;
745         int i = 0;
746         while(i < participantesSinRankear.longitud()) {
747             Deporte deporte = competencia.categoria().first;
748             Atleta participante = participantesSinRankear.iesimo(i);
749
750             // En caso de ser el peor hasta el momento, me quedo con el participante i-ésimo.
751             if(i == 0 || participante.capacidad(deporte) < peorParticipante.capacidad(deporte)) {
752                 peorParticipante = participante;
753             }
754
755             i++;
756         }
757
758         // Agrego el peor participante hallado al principio del ranking.
759         ranking.agregar(peorParticipante.ciaNumber());
760
761         // Lo elimino de la lista de participantes sin rankear.
762         participantesSinRankear.eliminarPosicion(participantesSinRankear.posicion(peorParticipante));
763     }
764
765     return ranking;
766 }
767
768 Lista<pair<int,bool> > JJOO::generarAntidoping(const Competencia& competencia) const {
769     Lista<pair<int,bool> > antidoping;
770
771     // Tomo al primer participante y lo agrego al control, con resultado falso.
772     if(competencia.participantes().longitud() > 0) {
```

```
773             antidoping.agregar(make_pair(competencia.participantes().cabeza().ciaNumber(), false));
774         }
775 
776         return antidoping;
777     }
778 
779     void JJOO::reemplazarCronogramaJornadaActual(Lista<Competencia> nuevoCronograma) {
780         Lista<Lista<Competencia> > nuevasCompetenciasPorDia;
781 
782         // Recorro todas las jornadas.
783         int i = 0;
784         while (i < _competenciasPorDia.longitud()) {
785 
786             // Si la i-ésima jornada es la actual, reemplazo
787             // su cronograma por el recibido como parámetro.
788             if(i == _jornadaActual - 1) {
789                 nuevasCompetenciasPorDia.agregarAtras(nuevoCronograma);
790             } else {
791                 nuevasCompetenciasPorDia.agregarAtras(_competenciasPorDia.iesimo(i));
792             }
793 
794             i++;
795         }
796 
797         // Reemplazo por las nuevas competencias.
798         _competenciasPorDia = nuevasCompetenciasPorDia;
799     }
800 
801     bool JJOO::operator==(const JJOO& j) const {
802         return _anio == j.anio() && _jornadaActual == j.jornadaActual() && mismosAtletas(j) && mismoCronograma(j);
803     }
804 
805     bool JJOO::mismosAtletas(const JJOO& j) const {
806         bool igual = true;
807 
808         // Verifico misma cantidad de atletas.
809         if(_atletas.longitud() == j.atletas().longitud()) {
810 
811             // Recorro lista de atletas de instancia a comparar.
812             int i = 0;
813             while(i < _atletas.longitud()) {
814 
815                 // Verfico que ambas instancias tengan los mismos atletas.
816                 if(!j.atletas().pertenece(_atletas.iesimo(i))) {
817                     igual = false;
818                 }
819                 i++;
820             }
821         } else {
822             igual = false;
823         }
824 
825         return igual;
826     }
827 
828     bool JJOO::mismoCronograma(const JJOO& j) const {
829         bool igual = true;
830 
831         // Verifico misma cantidad de días.
832         if(cantDias() == j.cantDias()) {
833 
834             // Recorro el cronograma día por día.
835             int i = 1;
836             while(i < cantDias()) {
837 
838                 // Verifico misma cantidad de competencias en la jornada i-ésima.
839                 if(cronograma(i).longitud() == j.cronograma(i).longitud()) {
840 
841                     // Recorro competencias de la instancia de JJOO a comparar.
842                     int k = 0;
843                     while(k < j.cronograma(i).longitud()) {
844 
845                         // Verifico que ambos cronogramas tengan las mismas competencias.
846                         if(!cronograma(i).pertenece(j.cronograma(i).iesimo(k))) {
847                             igual = false;
848                         }
849 
850                         k++;
851                     }
852                 } else {
853                     igual = false;
854                 }
855                 i++;
856             }
857         } else {
858             igual = false;
859         }
860 
861         return igual;
862     }
```

```
863
864 void JJOO::mostrar(std::ostream& os) const {
865     os << "Juego olimpico" << endl << "Anio: " << _anio << " Dia: " << _jornadaActual << "/" << cantDias() << endl;
866     os << "Participantes:" << endl;
867     int i = 0;
868     while(i<_atletas.longitud()) {
869         _atletas.iesimo(i).mostrar(os);
870         i++;
871     }
872     os << endl << "Cronograma:" << endl;
873     i=0;
874     while(i<_competenciasPorDia.longitud()) {
875         os << "Dia: " << (i+1) << "/" << cantDias() << endl;
876         int j=0;
877         while(j<_competenciasPorDia.iesimo(i).longitud()) {
878             _competenciasPorDia.iesimo(i).iesimo(j).mostrar(os);
879             os << endl;
880             j++;
881         }
882         i++;
883     }
884
885 }
886
887 void JJOO::guardar(std::ostream& os) const {
888     os << "J " << _anio << " " << _jornadaActual << " [";
889     int i=0;
890     while(i<_atletas.longitud()) {
891         os << "(";
892         _atletas.iesimo(i).guardar(os);
893         os << ")";
894         i++;
895         if(i<_atletas.longitud()) {
896             os << ",";
897         }
898     }
899     os << "] [";
900     i=0;
901     while(i<_competenciasPorDia.longitud()) {
902         os << "[";
903         int j=0;
904         while(j<_competenciasPorDia.iesimo(i).longitud()) {
905             os << "(";
906             _competenciasPorDia.iesimo(i).iesimo(j).guardar(os);
907             os << ")";
908             j++;
909             if(j<_competenciasPorDia.iesimo(i).longitud()) {
910                 os << ",";
911             }
912         }
913         os << "]";
914         i++;
915         if(i<_competenciasPorDia.longitud()) {
916             os << ",";
917         }
918     }
919     os << "]";
920 }
921
922 void JJOO::cargar (std::istream& is) {
923     char c;
924     //Saco J anio jornada [
925     is >> c >> _anio >> _jornadaActual >> c;
926     bool loop = true;
927     Lista<Atleta> nuevosAtletas;
928     if(is.peek() != ']') {
929         while(loop) {
930             //Saco (
931             is >> c;
932             Atleta atle;
933             atle.cargar(is);
934             nuevosAtletas.agregarAtras(atle);
935             //Saco )
936             is >> c;
937             if(is.peek() != ',') {
938                 loop = false;
939             }
940             else {
941                 //Saco la coma
942                 is >> c;
943             }
944         }
945     }
946     _atletas = nuevosAtletas;
947     //Saco ] y [
948     is >> c >> c;
949     Lista<Lista<Competencia> > nuevasCompetenciasPorDia;
950     if(is.peek() != ']') {
951         loop = true;
952         while(loop) {
```

```
953             //Saco [
954             is >> c;
955             Lista<Competencia> nuevaCompetenciasEnElDia;
956             if(is.peek() != ']') {
957                 Competencia compe;
958                 bool secLoop = true;
959                 while(secLoop) {
960                     //Saco (
961                     is >> c;
962                     compe.cargar(is);
963                     nuevaCompetenciasEnElDia.agregarAtras(compe);
964                     //Saco )
965                     is >> c;
966                     if(is.peek() != ',') {
967                         secLoop = false;
968                     }
969                     else {
970                         //Saco la coma
971                         is >> c;
972                     }
973                 }
974             }
975             //Saco ]
976             is >> c;
977             nuevasCompetenciasPorDia.agregarAtras(nuevaCompetenciasEnElDia);
978             if(is.peek() != ',') {
979                 loop = false;
980             }
981             else {
982                 //Saco la coma
983                 is >> c;
984             }
985         }
986         //Saco ]
987         is >> c;
988         _competenciasPorDia = nuevasCompetenciasPorDia;
989     }
990
991
992 }
```