

```

1  #ifndef _LISTAEQUIPO_H
2  #define _LISTAEQUIPO_H
3  #include "Equipo.h"
4  #ifndef NULL
5  #define NULL      0
6  #endif
7  /*-----*/
8  //                                ESTRUCTURAS
9  /*-----*/
10 enum ResultadoComparacionEquipo {
11     MAYOR_EQUIPO,
12     IGUAL_EQUIPO,
13     MENOR_EQUIPO
14 };
15
16 struct NodoListaEquipo{
17     Equipo equipo;
18     NodoListaEquipo* siguiente;
19
20 };
21
22 typedef NodoListaEquipo* PtrNodoListaEquipo;
23
24 struct ListaEquipo{
25     PtrNodoListaEquipo primero;
26
27 };
28 /*-----*/
29 //                                PRIMITIVAS
30 /*-----*/
31 /*
32     pre : la lista no debe haber sido creada.
33     post: lista queda creada y preparada para ser usada.
34
35     lista : estructura de datos a ser creado.
36 */
37 void crearListaEquipo(ListaEquipo &lista);
38
39 /*-----*/
40 /*
41     pre : lista Creada con crearLista().
42     post: Devuelve true si lista esta vacia, sino devuelve false.
43
44     lista : lista sobre la cual se invoca la primitiva.
45 */
46 bool listaVaciaEquipo(ListaEquipo &lista);
47 /*-----*/
48 /*
49     pre : lista Creada con crearLista().
50     post: devuelve la representacion de lo Siguiente al último Nodo de la lista,
51           o sea el valor Null, que en esta implementacion representa el final de
52           la lista.
53
54     return representación del fin de la lista.
55 */
56 PtrNodoListaEquipo finEquipo();
57 /*-----*/
58 /*
59     pre : lista Creada con crearLista().
60     post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si
61           esta vacia
62
63     lista : lista sobre la cual se invoca la primitiva.
64     return puntero al primer nodo.
65 */
66 PtrNodoListaEquipo primeroEquipo(ListaEquipo &lista);

```

```

67  /*-----*/
68  /*
69   pre : lista Creada con crearLista().
70   post: devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si
71         ptrNodo apuntaba a fin() o si lista esta vacia.
72
73   lista : lista sobre la cual se invoca la primitiva.
74   prtNodo : puntero al nodo a partir del cual se requiere el siguiente.
75   return puntero al nodo siguiente.
76  */
77  PtrNodoListaEquipo siguienteEquipo(ListaEquipo &lista, PtrNodoListaEquipo ptrNodo);
78  /*-----*/
79  /*
80   pre : lista Creada con crearLista().
81         ptrNodo es un puntero a un nodo de lista.
82   post: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si
83         ptrNodo apuntaba al primero o si lista esta vacia.
84
85   lista : lista sobre la cual se invoca la primitiva.
86   prtNodo : puntero al nodo a partir del cual se requiere el anterior.
87   return puntero al nodo anterior.
88  */
89  PtrNodoListaEquipo anteriorEquipo(ListaEquipo &lista,PtrNodoListaEquipo ptrNodo);
90  /*-----*/
91  /*
92   pre : lista creada con crearLista().
93   post: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si
94         si lista esta vacia.
95
96   lista : lista sobre la cual se invoca la primitiva.
97   return puntero al último nodo.
98  */
99  PtrNodoListaEquipo ultimoEquipo(ListaEquipo &lista);
100 /*-----*/
101 /*
102 pre : lista creada con crearLista().
103 post: crea el nodo de la lista.
104
105 dato : elemento a almacenar en el nodo.
106 return puntero al nodo creado.
107 */
108 PtrNodoListaEquipo crearNodoListaEquipo(Equipo equipo);
109 /*-----*/
110 /*
111 pre : lista creada con crearLista().
112 post: agrega un nodo nuevo al principio de la lista con el dato proporcionado
113       y devuelve un puntero a ese elemento.
114
115 lista : lista sobre la cual se invoca la primitiva.
116 dato : elemento a adicionar al principio de la lista.
117 return puntero al nodo adicionado.
118 */
119 PtrNodoListaEquipo adicionarAlPrincipio(ListaEquipo &lista, Equipo equipo);
120 /*-----*/
121 /*
122 pre : lista creada con crearLista().
123 post: agrega un nodo despues del apuntado por ptrNodo con el dato
124       proporcionado y devuelve un puntero apuntado al elemento insertado.
125       Si la lista esta vacía agrega un nodo al principio de esta y devuelve
126       un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta
127       nada y devuelve fin().
128
129 lista : lista sobre la cual se invoca la primitiva.
130 dato : elemento a adicionar.
131 ptrNodo : puntero al nodo después del cual se quiere adicionar el dato.
132 return puntero al nodo adicionado.

```

```

133 */
134 PtrNodoListaEquipo adicionarDespues(ListaEquipo &lista, Equipo equipo, PtrNodoListaEquipo ptrNodo);
135 /*-----*/
136 /*
137     pre : lista creada con crearLista().
138     post: agrega un nodo al final de la lista con el dato proporcionado y devuelve
139           un puntero al nodo insertado.
140
141     lista : lista sobre la cual se invoca la primitiva.
142     dato : elemento a adicionar al final de la lista.
143     return puntero al nodo adicionado.
144 */
145 PtrNodoListaEquipo adicionarFinal(ListaEquipo &lista, Equipo equipo);
146 /*-----*/
147 /*
148     pre : lista creada con crearLista().
149     post: agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo
150           y devuelve un puntero al nodo insertado. Si la lista esta vacia no
151           inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo
152           insertado sera el nuevo primero.
153
154     lista : lista sobre la cual se invoca la primitiva.
155     dato : elemento a adicionar.
156     ptrNodo : puntero al nodo antes del cual se quiere adicionar el dato.
157     return puntero al nodo adicionado.
158 */
159 PtrNodoListaEquipo adicionarAntes(ListaEquipo &lista, Equipo equipo, PtrNodoListaEquipo ptrNodo);
160 /*-----*/
161 /*
162     pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
163     post: coloca el dato proporcionado en el nodo apuntado por ptrNodo.
164
165     lista : lista sobre la cual se invoca la primitiva.
166     dato : elemento a colocar.
167     ptrNodo : puntero al nodo del cual se quiere colocar el dato.
168 */
169 void colocarDato(ListaEquipo &lista, Equipo equipo, PtrNodoListaEquipo ptrNodo);
170 /*-----*/
171 /*
172     pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
173     post: devuelve el dato del nodo apuntado por ptrNodo.
174
175     lista : lista sobre la cual se invoca la primitiva.
176     dato : elemento obtenido.
177     ptrNodo : puntero al nodo del cual se quiere obtener el dato.
178 */
179 void obtenerDato(ListaEquipo &lista, Equipo &equipo, PtrNodoListaEquipo ptrNodo);
180
181 /*-----*/
182 /*
183     pre : lista creada con crearLista().
184     post: elimina el nodo apuntado por ptrNodo. No realiza accion si la lista
185           esta vacia o si ptrNodo apunta a fin().
186
187     lista : lista sobre la cual se invoca la primitiva.
188     ptrNodo : puntero al nodo que se desea eliminar.
189 */
190 void eliminarNodo(ListaEquipo &lista, PtrNodoListaEquipo ptrNodo);
191
192 /*-----*/
193 /*
194     pre : lista creada con crearLista().
195     post: si la lista no esta vacia, elimina su nodo primero, sino no realiza
196           accion alguna.
197
198     lista : lista sobre la cual se invoca la primitiva.

```

```

199  */
200  void eliminarNodoPrimero(ListaEquipo &lista);
201
202  /*-----*/
203  /*
204     pre : lista creada con crearLista().
205     post: si la lista no esta vacia elimina su nodo ultimo,
206           sino no realiza accion.
207
208     lista : lista sobre la cual se invoca la primitiva.
209  */
210  void eliminarNodoUltimo(ListaEquipo &lista);
211
212  /*-----*/
213  /*
214     pre : lista creada con crearLista().
215     post: elimina todos los Nodos de la lista quedando destruida e inhabilitada
216           para su uso.
217
218     lista : lista sobre la cual se invoca la primitiva.
219  */
220  void eliminarLista(ListaEquipo &lista);
221  /*-----*/
222  /*
223     pre : ninguna.
224     post: compara ambos dato1 y dato2, devuelve
225           mayor si dato1 es mayor que dato2,
226           igual si dato1 es igual a dato2,
227           menor si dato1 es menor que dato2.
228
229     dato1 : dato a comparar.
230     dato2 : dato a comparar.
231     return resultado de comparar dato1 respecto de dato2.
232  */
233  ResultadoComparacionEquipo compararDatoEquipo(Equipo equipo1, Equipo equipo2);
234  /*-----*/
235  /*
236     pre : lista fue creada con crearLista().
237     post: si el dato se encuentra en la lista, devuelve el puntero al primer nodo
238           que lo contiene. Si el dato no se encuentra en la lista devuelve fin().
239
240     lista : lista sobre la cual se invoca la primitiva.
241     dato : elemento a localizar.
242     return puntero al nodo localizado o fin().
243  */
244  PtrNodoListaEquipo localizarDato(ListaEquipo &lista, Equipo equipo);
245  /*-----*/
246  /*
247     pre : la lista fue creada con crearLista().
248     post : elimina el dato de la lista, si el mismo se encuentra.
249
250     lista : lista sobre la cual se invoca la primitiva.
251     dato : elemento a eliminar.
252  */
253  void eliminarDato(ListaEquipo &lista, Equipo equipo);
254  /*-----*/
255  /*
256     pre : lista fue creada con crearLista() y cargada con datos ordenados de
257           menor a mayor respecto del sentido progresivo.
258     post: agrega a la lista el dato manteniendo el orden pero con multiples
259           valores iguales y devuelve un puntero al nodo insertado.
260
261     lista : lista sobre la cual se invoca la primitiva.
262     dato : elemento a insertar.
263     return puntero al nodo insertado.
264  */

```

```

265 PtrNodoListaEquipo insertarDato(ListaEquipo &lista, Equipo equipo);
266 /*-----*/
267 /*
268  pre : la lista fue creada con crearLista().
269  post : reordena la lista.
270
271  lista : lista sobre la cual se invoca la primitiva.
272 */
273 void reordenar(ListaEquipo &lista);
274 /*-----*/
275 /*
276  pre : ninguna.
277  post: compara ambos datol y dato2, devuelve
278         mayor si datol es mayor que dato2,
279         igual si datol es igual a dato2,
280         menor si datol es menor que dato2.
281
282  datol : dato a comparar.
283  dato2 : dato a comparar.
284  return resultado de comparar datol respecto de dato2.
285 */
286 ResultadoComparacionEquipo compararGolesEquipo(Equipo equipo1, Equipo equipo2) ;
287 /*****/
288 #endif // _LISTAEQUIPO_H

```

```

1  #ifndef _LISTAGRUPO_H_
2  #define _LISTAGRUPO_H_
3  #include "Grupo.h"
4  #ifndef NULL
5  #define NULL      0
6  #endif
7  /*-----*/
8  //                                ESTRUCTURAS
9  /*-----*/
10 enum ResultadoComparacionGrupo{
11     MAYOR_GRUPO,
12     IGUAL_GRUPO,
13     MENOR_GRUPO
14 };
15
16 struct NodoGrupo{
17     Grupo grupo;
18     NodoGrupo* siguiente;
19 };
20 typedef NodoGrupo* PtrNodoGrupo;
21
22 struct ListaGrupo{
23     PtrNodoGrupo primero;
24 };
25 /*-----*/
26 //                                PRIMITIVAS
27 /*-----*/
28 /*
29     pre : la lista no debe haber sido creada.
30     post: lista queda creada y preparada para ser usada.
31
32     lista : estructura de datos a ser creado.
33 */
34 void crearListaGrupo(ListaGrupo &lista);
35 /*-----*/
36 /*
37     pre : lista Creada con crearLista().
38     post: devuelve la representacion de lo Siguiete al último Nodo de la lista,
39           o sea el valor Null, que en esta implementacion representa el final de
40           la lista.
41
42     return representación del fin de la lista.
43 */
44 PtrNodoGrupo finGrupo();
45 /*-----*/
46 /*
47     pre : lista Creada con crearLista().
48     post: Devuelve true si lista esta vacia, sino devuelve false.
49
50     lista : lista sobre la cual se invoca la primitiva.
51 */
52 bool listaVaciaGrupo(ListaGrupo lista);
53 /*-----*/
54 /*
55     pre : lista Creada con crearLista().
56     post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si
57           esta vacia
58
59     lista : lista sobre la cual se invoca la primitiva.
60     return puntero al primer nodo.
61 */
62 PtrNodoGrupo primeroListaGrupo(ListaGrupo &lista);
63 /*-----*/
64 /*
65     pre : lista Creada con crearLista().
66     post: devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si

```

```

67         ptrNodo apuntaba a fin() o si lista esta vacia.
68
69     lista : lista sobre la cual se invoca la primitiva.
70     prtNodo : puntero al nodo a partir del cual se requiere el siguiente.
71     return puntero al nodo siguiente.
72 */
73 PtrNodoGrupo siguienteListaGrupo(ListaGrupo &lista,PtrNodoGrupo ptrNodoGrupo);
74 /*-----*/
75 /*
76     pre : lista Creada con crearLista().
77         ptrNodo es un puntero a un nodo de lista.
78     post: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si
79         ptrNodo apuntaba al primero o si lista esta vacia.
80
81     lista : lista sobre la cual se invoca la primitiva.
82     prtNodo : puntero al nodo a partir del cual se requiere el anterior.
83     return puntero al nodo anterior.
84 */
85 PtrNodoGrupo anteriorListaGrupo(ListaGrupo &lista,PtrNodoGrupo ptrNodoGrupo);
86 /*-----*/
87 /*
88     pre : lista creada con crearLista().
89     post: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si
90         si lista esta vacia.
91
92     lista : lista sobre la cual se invoca la primitiva.
93     return puntero al último nodo.
94 */
95 PtrNodoGrupo ultimoListaGrupo(ListaGrupo &lista);
96 /*-----*/
97 /*
98     pre : lista creada con crearLista().
99     post: crea el nodo de la lista.
100
101     dato : elemento a almacenar en el nodo.
102     return puntero al nodo creado.
103 */
104 PtrNodoGrupo crearNodoGrupo(Grupo grupo);
105 /*-----*/
106 /*
107     pre : lista creada con crearLista().
108     post: agrega un nodo nuevo al principio de la lista con el dato proporcionado
109         y devuelve un puntero a ese elemento.
110
111     lista : lista sobre la cual se invoca la primitiva.
112     dato : elemento a adicionar al principio de la lista.
113     return puntero al nodo adicionado.
114 */
115 PtrNodoGrupo adicionarAlPrincipio(ListaGrupo &lista,Grupo grupo);
116 /*-----*/
117 /*
118     pre : lista creada con crearLista().
119     post: agrega un nodo despues del apuntado por ptrNodo con el dato
120         proporcionado y devuelve un puntero apuntado al elemento insertado.
121         Si la lista esta vacía agrega un nodo al principio de esta y devuelve
122         un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta
123         nada y devuelve fin().
124
125     lista : lista sobre la cual se invoca la primitiva.
126     dato : elemento a adicionar.
127     ptrNodo : puntero al nodo después del cual se quiere adicionar el dato.
128     return puntero al nodo adicionado.
129 */
130 PtrNodoGrupo adicionarDespues(ListaGrupo &lista, Grupo grupo,PtrNodoGrupo ptrNodo);
131 /*-----*/
132 /*

```

```

133     pre : lista creada con crearLista().
134     post: agrega un nodo al final de la lista con el dato proporcionado y devuelve
135           un puntero al nodo insertado.
136
137     lista : lista sobre la cual se invoca la primitiva.
138     dato : elemento a adicionar al final de la lista.
139     return puntero al nodo adicionado.
140 */
141 PtrNodoGrupo adicionarFinal(ListaGrupo &lista, Grupo grupo);
142 /*-----*/
143 /*
144     pre : lista creada con crearLista().
145     post: agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo
146           y devuelve un puntero al nodo insertado. Si la lista esta vacia no
147           inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo
148           insertado sera el nuevo primero.
149
150     lista : lista sobre la cual se invoca la primitiva.
151     dato : elemento a adicionar.
152     ptrNodo : puntero al nodo antes del cual se quiere adicionar el dato.
153     return puntero al nodo adicionado.
154 */
155 PtrNodoGrupo adicionarAntes(ListaGrupo &lista, Grupo grupo, PtrNodoGrupo ptrNodoGrupo);
156 /*-----*/
157 /*
158     pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
159     post: coloca el dato proporcionado en el nodo apuntado por ptrNodo.
160
161     lista : lista sobre la cual se invoca la primitiva.
162     dato : elemento a colocar.
163     ptrNodo : puntero al nodo del cual se quiere colocar el dato.
164 */
165 void colocarDato(ListaGrupo &lista, Grupo grupo, PtrNodoGrupo ptrNodoGrupo);
166 /*-----*/
167 /*
168     pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
169     post: devuelve el dato del nodo apuntado por ptrNodo.
170
171     lista : lista sobre la cual se invoca la primitiva.
172     dato : elemento obtenido.
173     ptrNodo : puntero al nodo del cual se quiere obtener el dato.
174 */
175 void obtenerDato(ListaGrupo &lista, Grupo &grupo, PtrNodoGrupo ptrNodo);
176
177 /*-----*/
178 /*
179     pre : lista creada con crearLista().
180     post: elimina el nodo apuntado por ptrNodo. No realiza accion si la lista
181           esta vacia o si ptrNodo apunta a fin().
182
183     lista : lista sobre la cual se invoca la primitiva.
184     ptrNodo : puntero al nodo que se desea eliminar.
185 */
186 void eliminarNodo(ListaGrupo &lista, PtrNodoGrupo ptrNodo);
187
188 /*-----*/
189 /*
190     pre : lista creada con crearLista().
191     post: si la lista no esta vacia, elimina su nodo primero, sino no realiza
192           accion alguna.
193
194     lista : lista sobre la cual se invoca la primitiva.
195 */
196 void eliminarNodoPrimero(ListaGrupo &lista);
197
198 /*-----*/

```



```

199  /*
200     pre : lista creada con crearLista().
201     post: si la lista no esta vacia elimina su nodo ultimo,
202           sino no realiza accion.
203
204     lista : lista sobre la cual se invoca la primitiva.
205  */
206  void eliminarNodoUltimo(ListaGrupo &lista);
207
208  /*-----*/
209  /*
210     pre : lista creada con crearLista().
211     post: elimina todos los Nodos de la lista quedando destruida e inhabilitada
212           para su uso.
213
214     lista : lista sobre la cual se invoca la primitiva.
215  */
216  void eliminarLista(ListaGrupo &lista);
217  /*-----*/
218  /*
219     pre : ninguna.
220     post: compara ambos datol y dato2, devuelve
221           mayor si datol es mayor que dato2,
222           igual si datol es igual a dato2,
223           menor si datol es menor que dato2.
224
225     datol : dato a comparar.
226     dato2 : dato a comparar.
227     return resultado de comparar datol respecto de dato2.
228  */
229  ResultadoComparacionGrupo compararDatoGrupo(GGrupo grupo1, GGrupo grupo2);
230  /*-----*/
231  /*
232     pre : lista fue creada con crearLista().
233     post: si el dato se encuentra en la lista, devuelve el puntero al primer nodo
234           que lo contiene. Si el dato no se encuentra en la lista devuelve fin().
235
236     lista : lista sobre la cual se invoca la primitiva.
237     dato : elemento a localizar.
238     return puntero al nodo localizado o fin().
239  */
240  PtrNodoGrupo localizarDato(ListaGrupo &lista, GGrupo grupo);
241  /*-----*/
242  /*
243     pre : la lista fue creada con crearLista().
244     post : elimina el dato de la lista, si el mismo se encuentra.
245
246     lista : lista sobre la cual se invoca la primitiva.
247     dato : elemento a eliminar.
248  */
249  void eliminarDato(ListaGrupo &lista, GGrupo grupo);
250  /*-----*/
251  /******
252
253  #endif // _LISTAGRUPO_H

```

```

1  #ifndef __LISTAJUGADORES_H_
2  #define __LISTAJUGADORES_H_
3  #include "Jugador.h"
4  #ifndef NULL
5  #define NULL      0
6  #endif
7  /*-----*/
8  //                                ESTRUCTURAS
9  /*-----*/
10 enum ResultadoComparacionJugador{
11     MAYOR_JUGADOR,
12     IGUAL_JUGADOR,
13     MENOR_JUGADOR
14 };
15
16
17 struct NodoListaJugador{
18     Jugador jugador;
19     NodoListaJugador* sgte;
20 };
21
22
23 typedef NodoListaJugador* PtrNodoListaJugador;
24
25 struct ListaJugador{
26     PtrNodoListaJugador primero;
27 };
28 /*-----*/
29 //                                PRIMITIVAS
30 /*-----*/
31 /*
32     pre : la lista no debe haber sido creada.
33     post: lista queda creada y preparada para ser usada.
34
35     lista : estructura de datos a ser creado.
36 */
37 void crearListaJugador(ListaJugador &lista);
38
39 /*-----*/
40 /*
41     pre : lista Creada con crearLista().
42     post: Devuelve true si lista esta vacia, sino devuelve false.
43
44     lista : lista sobre la cual se invoca la primitiva.
45 */
46 bool listaVaciaJugador(ListaJugador &lista);
47
48 /*-----*/
49 /*
50     pre : lista Creada con crearLista().
51     post: devuelve la representacion de lo Siguiente al último Nodo de la lista,
52           o sea el valor Null, que en esta implementacion representa el final de
53           la lista.
54
55     return representación del fin de la lista.
56 */
57 PtrNodoListaJugador finJugador();
58
59 /*-----*/
60 /*
61     pre : lista Creada con crearLista().
62     post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si
63           esta vacia
64
65     lista : lista sobre la cual se invoca la primitiva.
66     return puntero al primer nodo.

```

```

67  */
68  PtrNodoListaJugador primeroJugador(ListaJugador &lista);
69
70  /*-----*/
71  /*
72   pre : lista Creada con crearLista().
73   post: devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si
74         ptrNodo apuntaba a fin() o si lista esta vacia.
75
76   lista : lista sobre la cual se invoca la primitiva.
77   prtNodo : puntero al nodo a partir del cual se requiere el siguiente.
78   return puntero al nodo siguiente.
79  */
80  PtrNodoListaJugador siguienteJugador(ListaJugador &lista, PtrNodoListaJugador ptrNodo);
81
82  /*-----*/
83  /*
84   pre : lista Creada con crearLista().
85         ptrNodo es un puntero a un nodo de lista.
86   post: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si
87         ptrNodo apuntaba al primero o si lista esta vacia.
88
89   lista : lista sobre la cual se invoca la primitiva.
90   prtNodo : puntero al nodo a partir del cual se requiere el anterior.
91   return puntero al nodo anterior.
92  */
93  PtrNodoListaJugador anteriorJugador(ListaJugador &lista, PtrNodoListaJugador ptrNodo);
94
95  /*-----*/
96  /*
97   pre : lista creada con crearLista().
98   post: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si
99         si lista esta vacia.
100
101   lista : lista sobre la cual se invoca la primitiva.
102   return puntero al último nodo.
103  */
104  PtrNodoListaJugador ultimoJugador(ListaJugador &lista);
105  /*-----*/
106  /*
107   pre : lista creada con crearLista().
108   post: crea el nodo de la lista.
109
110   dato : elemento a almacenar en el nodo.
111   return puntero al nodo creado.
112  */
113  PtrNodoListaJugador crearNodoListaJugador(Jugador jugador);
114  /*-----*/
115  /*
116   pre : lista creada con crearLista().
117   post: agrega un nodo nuevo al principio de la lista con el dato proporcionado
118         y devuelve un puntero a ese elemento.
119
120   lista : lista sobre la cual se invoca la primitiva.
121   dato : elemento a adicionar al principio de la lista.
122   return puntero al nodo adicionado.
123  */
124  PtrNodoListaJugador adicionarPrincipio(ListaJugador &lista, Jugador jugador);
125
126  /*-----*/
127  /*
128   pre : lista creada con crearLista().
129   post: agrega un nodo despues del apuntado por ptrNodo con el dato
130         proporcionado y devuelve un puntero apuntado al elemento insertado.
131         Si la lista esta vacía agrega un nodo al principio de esta y devuelve
132         un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta

```

```

133         nada y devuelve fin().
134
135     lista : lista sobre la cual se invoca la primitiva.
136     dato : elemento a adicionar.
137     ptrNodo : puntero al nodo después del cual se quiere adicionar el dato.
138     return puntero al nodo adicionado.
139 */
140 PtrNodoListaJugador adicionarDespues(ListaJugador &lista, Jugador jugador, PtrNodoListaJugador ptrNodo);
141
142 /*-----*/
143 /*
144     pre : lista creada con crearLista().
145     post: agrega un nodo al final de la lista con el dato proporcionado y devuelve
146           un puntero al nodo insertado.
147
148     lista : lista sobre la cual se invoca la primitiva.
149     dato : elemento a adicionar al final de la lista.
150     return puntero al nodo adicionado.
151 */
152 PtrNodoListaJugador adicionarFinal(ListaJugador &lista, Jugador jugador);
153
154 /*-----*/
155 /*
156     pre : lista creada con crearLista().
157     post: agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo
158           y devuelve un puntero al nodo insertado. Si la lista esta vacia no
159           inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo
160           insertado sera el nuevo primero.
161
162     lista : lista sobre la cual se invoca la primitiva.
163     dato : elemento a adicionar.
164     ptrNodo : puntero al nodo antes del cual se quiere adicionar el dato.
165     return puntero al nodo adicionado.
166 */
167 PtrNodoListaJugador adicionarAntes(ListaJugador &lista, Jugador jugador, PtrNodoListaJugador ptrNodo);
168
169 /*-----*/
170 /*
171     pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
172     post: coloca el dato proporcionado en el nodo apuntado por ptrNodo.
173
174     lista : lista sobre la cual se invoca la primitiva.
175     dato : elemento a colocar.
176     ptrNodo : puntero al nodo del cual se quiere colocar el dato.
177 */
178 void colocarDato(ListaJugador &lista, Jugador &jugador, PtrNodoListaJugador ptrNodo);
179
180 /*-----*/
181 /*
182     pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
183     post: devuelve el dato del nodo apuntado por ptrNodo.
184
185     lista : lista sobre la cual se invoca la primitiva.
186     dato : elemento obtenido.
187     ptrNodo : puntero al nodo del cual se quiere obtener el dato.
188 */
189 void obtenerDato(ListaJugador &lista, Jugador &jugador, PtrNodoListaJugador ptrNodo);
190
191 /*-----*/
192 /*
193     pre : lista creada con crearLista().
194     post: elimina el nodo apuntado por ptrNodo. No realiza accion si la lista
195           esta vacia o si ptrNodo apunta a fin().
196
197     lista : lista sobre la cual se invoca la primitiva.
198     ptrNodo : puntero al nodo que se desea eliminar.

```

```

199 */
200 void eliminarNodo(ListaJugador &lista, PtrNodoListaJugador ptrNodo);
201
202 /*-----*/
203 /*
204     pre : lista creada con crearLista().
205     post: si la lista no esta vacia, elimina su nodo primero, sino no realiza
206           accion alguna.
207
208     lista : lista sobre la cual se invoca la primitiva.
209 */
210 void eliminarNodoPrimero(ListaJugador &lista);
211
212 /*-----*/
213 /*
214     pre : lista creada con crearLista().
215     post: si la lista no esta vacia elimina su nodo ultimo,
216           sino no realiza accion.
217
218     lista : lista sobre la cual se invoca la primitiva.
219 */
220 void eliminarNodoUltimo(ListaJugador &lista);
221
222 /*-----*/
223 /*
224     pre : lista creada con crearLista().
225     post: elimina todos los Nodos de la lista quedando destruida e inhabilitada
226           para su uso.
227
228     lista : lista sobre la cual se invoca la primitiva.
229 */
230 void eliminarLista(ListaJugador &lista);
231 /*-----*/
232 /*
233     pre : ninguna.
234     post: compara ambos datol y dato2, devuelve
235           mayor si datol es mayor que dato2,
236           igual si datol es igual a dato2,
237           menor si datol es menor que dato2.
238
239     datol : dato a comparar.
240     dato2 : dato a comparar.
241     return resultado de comparar datol respecto de dato2.
242 */
243 ResultadoComparacionJugador compararDatoJugador(Jugador jugador1, Jugador jugador2);
244 /*-----*/
245 /*
246     pre : lista fue creada con crearLista().
247     post: si el dato se encuentra en la lista, devuelve el puntero al primer nodo
248           que lo contiene. Si el dato no se encuentra en la lista devuelve fin().
249
250     lista : lista sobre la cual se invoca la primitiva.
251     dato : elemento a localizar.
252     return puntero al nodo localizado o fin().
253 */
254 PtrNodoListaJugador localizarDato(ListaJugador &lista, Jugador jugador);
255 /*-----*/
256 /*
257     pre : la lista fue creada con crearLista().
258     post : elimina el dato de la lista, si el mismo se encuentra.
259
260     lista : lista sobre la cual se invoca la primitiva.
261     dato : elemento a eliminar.
262 */
263 void eliminarDato(ListaJugador &lista, Jugador jugador);
264 /*-----*/

```

```
265  /******  
266  #endif
```

```

1  #ifndef _LISTAPARTIDO_H_
2  #define _LISTAPARTIDO_H_
3  #include "Partido.h"
4  #ifndef NULL
5  #define NULL      0
6  #endif
7  /*-----*/
8  //                                ESTRUCTURAS
9  /*-----*/
10 enum ResultadoComparacionPartido{
11     MAYOR_PARTIDO,
12     IGUAL_PARTIDO,
13     MENOR_PARTIDO
14 };
15 struct NodoListaPartido{
16     Partido partido;
17     NodoListaPartido* siguiente;
18 }
19 };
20
21 typedef NodoListaPartido* PtrNodoPartido;
22
23 struct ListaPartido{
24     PtrNodoPartido primero;
25 };
26 /*-----*/
27 //                                PRIMITIVAS
28 /*-----*/
29 /*
30     pre : la lista no debe haber sido creada.
31     post: lista queda creada y preparada para ser usada.
32
33     lista : estructura de datos a ser creado.
34 */
35 void crearListaPartido(ListaPartido &lista);
36 /*-----*/
37 /*
38     pre : lista Creada con crearLista().
39     post: Devuelve true si lista esta vacia, sino devuelve false.
40
41     lista : lista sobre la cual se invoca la primitiva.
42 */
43 bool listaVaciaPartido(ListaPartido &lista);
44 /*-----*/
45 /*
46     pre : lista Creada con crearLista().
47     post: devuelve la representacion de lo Siguiete al último Nodo de la lista,
48           o sea el valor Null, que en esta implementacion representa el final de
49           la lista.
50
51     return representación del fin de la lista.
52 */
53 PtrNodoPartido finListaPartido();
54 /*-----*/
55 /*
56     pre : lista Creada con crearLista().
57     post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si
58           esta vacia
59
60     lista : lista sobre la cual se invoca la primitiva.
61     return puntero al primer nodo.
62 */
63 PtrNodoPartido primeroPartido(ListaPartido &lista);
64 /*-----*/
65 /*
66     pre : lista Creada con crearLista().

```

```

67     post: devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si
68         ptrNodo apuntaba a fin() o si lista esta vacia.
69
70     lista : lista sobre la cual se invoca la primitiva.
71     prtNodo : puntero al nodo a partir del cual se requiere el siguiente.
72     return puntero al nodo siguiente.
73 */
74 PtrNodoPartido siguientePartido(ListaPartido &lista, PtrNodoPartido ptrNodo);
75 /*-----*/
76 /*
77     pre : lista Creada con crearLista().
78         ptrNodo es un puntero a un nodo de lista.
79     post: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si
80         ptrNodo apuntaba al primero o si lista esta vacia.
81
82     lista : lista sobre la cual se invoca la primitiva.
83     prtNodo : puntero al nodo a partir del cual se requiere el anterior.
84     return puntero al nodo anterior.
85 */
86 PtrNodoPartido anteriorPartido(ListaPartido &lista, PtrNodoPartido ptrNodo);
87 /*-----*/
88 /*
89     pre : lista creada con crearLista().
90     post: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si
91         si lista esta vacia.
92
93     lista : lista sobre la cual se invoca la primitiva.
94     return puntero al último nodo.
95 */
96 PtrNodoPartido ultimoPartido(ListaPartido &lista);
97 /*-----*/
98 /*
99     pre : lista creada con crearLista().
100    post: crea el nodo de la lista.
101
102    dato : elemento a almacenar en el nodo.
103    return puntero al nodo creado.
104 */
105 PtrNodoPartido crearNodoListaPartido(Partido partido);
106 /*-----*/
107 /*
108     pre : lista creada con crearLista().
109     post: agrega un nodo nuevo al principio de la lista con el dato proporcionado
110         y devuelve un puntero a ese elemento.
111
112     lista : lista sobre la cual se invoca la primitiva.
113     dato : elemento a adicionar al principio de la lista.
114     return puntero al nodo adicionado.
115 */
116 PtrNodoPartido adicionarPrincipio(ListaPartido &lista, Partido partido);
117 /*-----*/
118 /*
119     pre : lista creada con crearLista().
120     post: agrega un nodo despues del apuntado por ptrNodo con el dato
121         proporcionado y devuelve un puntero apuntado al elemento insertado.
122         Si la lista esta vacía agrega un nodo al principio de esta y devuelve
123         un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta
124         nada y devuelve fin().
125
126     lista : lista sobre la cual se invoca la primitiva.
127     dato : elemento a adicionar.
128     ptrNodo : puntero al nodo después del cual se quiere adicionar el dato.
129     return puntero al nodo adicionado.
130 */
131 PtrNodoPartido adicionarDespues(ListaPartido &lista, Partido partido, PtrNodoPartido ptrNodo);
132 /*-----*/

```



```

133  /*
134  pre : lista creada con crearLista().
135  post: agrega un nodo al final de la lista con el dato proporcionado y devuelve
136        un puntero al nodo insertado.
137
138  lista : lista sobre la cual se invoca la primitiva.
139  dato : elemento a adicionar al final de la lista.
140  return puntero al nodo adicionado.
141  */
142  PtrNodoPartido adicionarFinal(ListaPartido &lista, Partido partido);
143  /*-----*/
144  /*
145  pre : lista creada con crearLista().
146  post: agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo
147        y devuelve un puntero al nodo insertado. Si la lista esta vacia no
148        inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo
149        insertado sera el nuevo primero.
150
151  lista : lista sobre la cual se invoca la primitiva.
152  dato : elemento a adicionar.
153  ptrNodo : puntero al nodo antes del cual se quiere adicionar el dato.
154  return puntero al nodo adicionado.
155  */
156  PtrNodoPartido adicionarAntes(ListaPartido &lista, Partido partido, PtrNodoPartido ptrNodo);
157  /*-----*/
158  /*
159  pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
160  post: coloca el dato proporcionado en el nodo apuntado por ptrNodo.
161
162  lista : lista sobre la cual se invoca la primitiva.
163  dato : elemento a colocar.
164  ptrNodo : puntero al nodo del cual se quiere colocar el dato.
165  */
166  void colocarDato(ListaPartido &lista, Partido &partido, PtrNodoPartido ptrNodo);
167  /*-----*/
168  /*
169  pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
170  post: devuelve el dato del nodo apuntado por ptrNodo.
171
172  lista : lista sobre la cual se invoca la primitiva.
173  dato : elemento obtenido.
174  ptrNodo : puntero al nodo del cual se quiere obtener el dato.
175  */
176  void obtenerDato(ListaPartido &lista, Partido &partido, PtrNodoPartido ptrNodo);
177  /*-----*/
178  /*
179  pre : lista creada con crearLista().
180  post: elimina el nodo apuntado por ptrNodo. No realiza accion si la lista
181        esta vacia o si ptrNodo apunta a fin().
182
183  lista : lista sobre la cual se invoca la primitiva.
184  ptrNodo : puntero al nodo que se desea eliminar.
185  */
186  void eliminarNodo(ListaPartido &lista, PtrNodoPartido ptrNodo);
187  /*-----*/
188  /*
189  pre : lista creada con crearLista().
190  post: si la lista no esta vacia, elimina su nodo primero, sino no realiza
191        accion alguna.
192
193  lista : lista sobre la cual se invoca la primitiva.
194  */
195  void eliminarNodoPrimero(ListaPartido &lista);
196  /*-----*/
197  /*
198  pre : lista creada con crearLista().

```

```

199     post: si la lista no esta vacia elimina su nodo ultimo,
200         sino no realiza accion.
201
202     lista : lista sobre la cual se invoca la primitiva.
203 */
204 void eliminarNodoUltimo(ListaPartido &lista);
205 /*-----*/
206 /*
207     pre : lista creada con crearLista().
208     post: elimina todos los Nodos de la lista quedando destruida e inhabilitada
209         para su uso.
210
211     lista : lista sobre la cual se invoca la primitiva.
212 */
213 void eliminarLista(ListaPartido &lista);
214 /*-----*/
215 /*
216     pre : ninguna.
217     post: compara ambos dato1 y dato2, devuelve
218         mayor si dato1 es mayor que dato2,
219         igual si dato1 es igual a dato2,
220         menor si dato1 es menor que dato2.
221
222     dato1 : dato a comparar.
223     dato2 : dato a comparar.
224     return resultado de comparar dato1 respecto de dato2.
225 */
226 ResultadoComparacionPartido compararDatoPartido(Partido partido1, Partido partido2);
227 /*-----*/
228 /*
229     pre : lista fue creada con crearLista().
230     post: si el dato se encuentra en la lista, devuelve el puntero al primer nodo
231         que lo contiene. Si el dato no se encuentra en la lista devuelve fin().
232
233     lista : lista sobre la cual se invoca la primitiva.
234     dato : elemento a localizar.
235     return puntero al nodo localizado o fin().
236 */
237 PtrNodoPartido localizarDato(ListaPartido &lista, Partido partido);
238 /*-----*/
239 /*
240     pre : la lista fue creada con crearLista().
241     post : elimina el dato de la lista, si el mismo se encuentra.
242
243     lista : lista sobre la cual se invoca la primitiva.
244     dato : elemento a eliminar.
245 */
246 void eliminarDato(ListaPartido &lista, Partido partido);
247 /*-----*/
248 /******
249 #endif

```

```

1  #ifndef _PARTIDO_H
2  #define _PARTIDO_H
3  #ifndef NULL
4  #define NULL      0
5  #endif
6  /*-----*/
7  //                                ESTRUCTURAS
8  /*-----*/
9  typedef struct{
10     int id;
11     int idEquipoL;
12     int idEquipoV;
13     int golesL;
14     int golesV;
15 }Partido;
16 /*-----*/
17 //                                PRIMITIVAS
18 /*-----*/
19 /*
20  pre : el partido no debe haber sido creado.
21  post: el partido queda creado y preparado para ser usado.
22
23  partido : estructura de datos a ser creado.
24 */
25 void crearPartido(Partido &partido);
26 /*-----*/
27 /*
28  pre : el partido debe haber sido creado.
29  post: se obtiene el id del partido creado pasado por parametro.
30
31  partido : estructura de datos a ser usada.
32  return: id obtenida del partido.
33 */
34 int getId(Partido &partido);
35 /*-----*/
36 /*
37  pre : el partido debe haber sido creado.
38  post: se obtiene el id del equipoL del partido creado pasado por parametro.
39
40  partido : estructura de datos a ser usada.
41  return: id del equipoL obtenido del partido.
42 */
43 int getIdEquipoL(Partido &partido);
44 /*-----*/
45 /*
46  pre : el partido debe haber sido creado.
47  post: se obtiene el id del equipoV del partido creado pasado por parametro.
48
49  partido : estructura de datos a ser usada.
50  return: id del equipoV obtenido del partido.
51 */
52 int getIdEquipoV(Partido &partido);
53 /*-----*/
54 /*
55  pre : el partido debe haber sido creado.
56  post: se obtiene los golesL del partido creado pasado por parametro.
57
58  partido : estructura de datos a ser usada.
59  return: golesL obtenidos del partido.
60 */
61 int getGolesL(Partido &partido);
62 /*-----*/
63 /*
64  pre : el partido debe haber sido creado.
65  post: se obtiene los golesV del partido creado pasado por parametro.
66

```

```

67     partido : estructura de datos a ser usada.
68     return: golesV obtenidos del partido.
69 */
70 int getGolesV(Partido &partido);
71 /*-----*/
72 /*
73     pre : el partido debe haber sido creado.
74     post: se setea el id del partido creado pasado por parametro.
75
76     partido : estructura de datos a ser usada.
77     id: valor a cargar en el partido.
78 */
79 void setId(Partido &partido, int id);
80 /*-----*/
81 /*
82     pre : el partido debe haber sido creado.
83     post: se setea el id del equipoL del partido creado pasado por parametro.
84
85     partido : estructura de datos a ser usada.
86     idEquipoL: valor a cargar en el partido.
87 */
88 void setIdEquipoL(Partido &partido, int idEquipoL);
89 /*-----*/
90 /*
91     pre : el partido debe haber sido creado.
92     post: se setea el id del equipoV del partido creado pasado por parametro.
93
94     partido : estructura de datos a ser usada.
95     idEquipoV: valor a cargar en el partido.
96 */
97 void setIdEquipoV(Partido &partido, int idEquipoV);
98 /*-----*/
99 /*
100     pre : el partido debe haber sido creado.
101     post: se setea los golesL del partido creado pasado por parametro.
102
103     partido : estructura de datos a ser usada.
104     golesL: valor a cargar en el partido.
105 */
106 void setGolesL(Partido &partido, int golesL);
107 /*-----*/
108 /*
109     pre : el partido debe haber sido creado.
110     post: se setea los golesV del partido creado pasado por parametro.
111
112     partido : estructura de datos a ser usada.
113     golesV: valor a cargar en el partido.
114 */
115 void setGolesV(Partido &partido, int golesV);
116 /*-----*/
117 /*
118     pre : el partido debe haber sido creado.
119     post: el partido queda destruido (iniciado a su origen).
120
121     partido : estructura de datos a ser destruido.
122 */
123 void destructor(Partido &partido);
124 /*-----*/
125 /*****
126 #endif // _PARTIDO_H

```

```

1  #ifndef _EQUIPO_H
2  #define _EQUIPO_H
3  #ifndef NULL
4  #define NULL      0
5  #endif
6  #include "ListaJugadores.h"
7  #include <string>
8  using namespace std;
9  /*-----*/
10 //                                ESTRUCTURAS
11 /*-----*/
12 typedef struct {
13     int id;
14     string nombre;
15     int golesAFavor;
16     int golesEnContra;
17     int puntos;
18     ListaJugador listaJugadores;
19 }Equipo;
20 /*-----*/
21 //                                PRIMITIVAS
22 /*-----*/
23 /*
24     pre : el equipo no debe haber sido creado.
25     post: el equipo queda creado y preparado para ser usado.
26
27     equipo : estructura de datos a ser creado.
28 */
29 void crearEquipo(Equipo &equipo);
30 /*-----*/
31 /*
32     pre : el equipo debe haber sido creado.
33     post: se obtiene el id del equipo creado pasado por parametro.
34
35     equipo : estructura de datos a ser usada.
36     return: id obtenida del equipo.
37 */
38 int getId(Equipo equipo);
39 /*-----*/
40 /*
41     pre : el equipo debe haber sido creado.
42     post: se setea el id del equipo creado pasado por parametro.
43
44     equipo : estructura de datos a ser usada.
45     id: valor a cargar en el equipo.
46 */
47
48 void setId(Equipo &equipo, int id);
49 /*-----*/
50 /*
51     pre : el equipo debe haber sido creado.
52     post: se obtiene el nombre del equipo creado pasado por parametro.
53
54     equipo : estructura de datos a ser usada.
55     return: nombre obtenido del equipo.
56 */
57 string getNombre(Equipo equipo);
58 /*-----*/
59 /*
60     pre : el equipo debe haber sido creado.
61     post: se setea el nombre del equipo creado pasado por parametro.
62
63     equipo : estructura de datos a ser usada.
64     nombre: valor a cargar en el equipo.
65 */
66 void setNombre(Equipo &equipo, string nombre);

```

```

67  /*-----*/
68  /*
69   pre : el equipo debe haber sido creado.
70   post: se setea los goles a favor del equipo creado pasado por parametro.
71
72   equipo : estructura de datos a ser usada.
73   golesAFavor: valor a cargar en el equipo.
74  */
75  void setGolesAFavor(Equipo &equipo,int goles);
76  /*-----*/
77  /*
78   pre : el equipo debe haber sido creado.
79   post: se obtiene los goles a favor del equipo creado pasado por parametro.
80
81   equipo : estructura de datos a ser usada.
82   return:goles a favor obtenidos del equipo.
83  */
84  int getGolesAFavor(Equipo equipo);
85  /*-----*/
86  /*
87   pre : el equipo debe haber sido creado.
88   post: se setea los goles en contra del equipo creado pasado por parametro.
89
90   equipo : estructura de datos a ser usada.
91   golesEnContra: valor a cargar en el equipo.
92  */
93  void setGolesEnContra(Equipo &equipo,int golesEnContra);
94  /*-----*/
95  /*
96   pre : el equipo debe haber sido creado.
97   post: se obtiene los goles en contra del equipo creado pasado por parametro.
98
99   equipo : estructura de datos a ser usada.
100  return:goles en contra obtenidos del equipo.
101  */
102  int getGolesEnContra(Equipo equipo);
103  /*-----*/
104  /*
105   pre : el equipo debe haber sido creado.
106   post: se setea puntos del equipo creado pasado por parametro.
107
108   equipo : estructura de datos a ser usada.
109   puntos: valor a cargar en el equipo.
110  */
111  void setPuntos(Equipo &equipo,int puntos);
112  /*-----*/
113  /*
114   pre : el equipo debe haber sido creado.
115   post: se obtiene los puntos del equipo creado pasado por parametro.
116
117   equipo : estructura de datos a ser usada.
118   return:puntos obtenidos del equipo.
119  */
120  int getPuntos(Equipo equipo);
121  /*-----*/
122  /*
123   pre : el equipo debe haber sido creado.
124   post: se obtiene la lista de jugador del equipo creado pasado por parametro.
125
126   equipo : estructura de datos a ser usada.
127   return: lista de jugadores obtenidos del equipo.
128  */
129  ListaJugador getListado(Equipo equipo);
130  /*-----*/
131  /*
132   pre : el equipo debe haber sido creado.

```

```
133     post: el equipo queda destruido (iniciado a su origen).
134
135     equipo : estructura de datos a ser destruido.
136 */
137 void destructor(Equipo &equipo);
138 /*-----*/
139 /*****/
140 #endif // _EQUIPO_H
```

```

1  #ifndef _GRUPO_H
2  #define _GRUPO_H_
3  #ifndef NULL
4  #define NULL      0
5  #endif
6  #include <string>
7  using namespace std;
8  /*-----*/
9  //                                ESTRUCTURAS
10 /*-----*/
11 typedef struct{
12     char id;
13     string nombre;
14     int idEquipo1;
15     int idEquipo2;
16     int idEquipo3;
17     int idEquipo4;
18 }Grupo;
19 /*-----*/
20 //                                PRIMITIVAS
21 /*-----*/
22 /*
23     pre : el grupo no debe haber sido creado.
24     post: el grupo queda creado y preparado para ser usado.
25
26     grupo : estructura de datos a ser creado.
27 */
28 void crearGrupo(Gruo &grupo);
29 /*-----*/
30 /*
31     pre : el grupo debe haber sido creado.
32     post: se setea el id del grupo creado pasado por parametro.
33
34     grupo : estructura de datos a ser usada.
35     id: valor a cargar en el grupo.
36 */
37 void setId(Gruo &grupo,char id);
38 /*-----*/
39 /*
40     pre : el grupo debe haber sido creado.
41     post: se obtiene el id del grupo creado pasado por parametro.
42
43     grupo : estructura de datos a ser usada.
44     return: id obtenida del grupo.
45 */
46 char getId(Gruo grupo);
47 /*-----*/
48 /*
49     pre : el grupo debe haber sido creado.
50     post: se setea el nombre del grupo creado pasado por parametro.
51
52     jugador : estructura de datos a ser usada.
53     nombre: valor a cargar en el grupo.
54 */
55 void setNombre(Gruo &grupo,string nombre);
56 /*-----*/
57 /*
58     pre : el grupo debe haber sido creado.
59     post: se obtiene el nombre del grupo creado pasado por parametro.
60
61     grupo : estructura de datos a ser usada.
62     return: nombre obtenido del grupo.
63 */
64 string getNombre(Gruo grupo);
65 /*-----*/
66 /*

```



```

67     pre : el grupo debe haber sido creado.
68     post: se setea el id del equipo 1 del grupo creado pasado por parametro.
69
70     grupo : estructura de datos a ser usada.
71     id: valor del equipo 1 a cargar en el grupo.
72 */
73 void setIdEquipo1(Grupo &grupo,int id);
74 /*-----*/
75 /*
76     pre : el grupo debe haber sido creado.
77     post: se obtiene el id del equipo 1 del grupo creado pasado por parametro.
78
79     grupo : estructura de datos a ser usada.
80     return: id del equipo 1 obtenida del grupo.
81 */
82 int getIdEquipo1(Grupo grupo);
83 /*-----*/
84 /*
85     pre : el grupo debe haber sido creado.
86     post: se setea el id del equipo 2 del grupo creado pasado por parametro.
87
88     grupo : estructura de datos a ser usada.
89     id: valor del equipo 2 a cargar en el grupo.
90 */
91 void setIdEquipo2(Grupo &grupo,int id);
92 /*-----*/
93 /*
94     pre : el grupo debe haber sido creado.
95     post: se obtiene el id del equipo 2 del grupo creado pasado por parametro.
96
97     grupo : estructura de datos a ser usada.
98     return: id del equipo 2 obtenida del grupo.
99 */
100 int getIdEquipo2(Grupo grupo);
101 /*-----*/
102 /*
103     pre : el grupo debe haber sido creado.
104     post: se setea el id del equipo 3 del grupo creado pasado por parametro.
105
106     grupo : estructura de datos a ser usada.
107     id: valor del equipo 3 a cargar en el grupo.
108 */
109 void setIdEquipo3(Grupo &grupo,int id);
110 /*-----*/
111 /*
112     pre : el grupo debe haber sido creado.
113     post: se obtiene el id del equipo 3 del grupo creado pasado por parametro.
114
115     grupo : estructura de datos a ser usada.
116     return: id del equipo 3 obtenida del grupo.
117 */
118 int getIdEquipo3(Grupo grupo);
119 /*-----*/
120 /*
121     pre : el grupo debe haber sido creado.
122     post: se setea el id del equipo 4 del grupo creado pasado por parametro.
123
124     grupo : estructura de datos a ser usada.
125     id: valor del equipo 4 a cargar en el grupo.
126 */
127 void setIdEquipo4(Grupo &grupo,int id);
128 /*-----*/
129 /*
130     pre : el grupo debe haber sido creado.
131     post: se obtiene el id del equipo 4 del grupo creado pasado por parametro.
132

```

```

133     grupo : estructura de datos a ser usada.
134     return: id del equipo 4 obtenida del grupo.
135 */
136 int getIdEquipo4(Grupo grupo);
137 /*-----*/
138 /*
139     pre : el grupo debe haber sido creado.
140     post: el grupo queda destruido (iniciado a su origen).
141
142     grupo : estructura de datos a ser destruido.
143 */
144 void destructor(Grupo &grupo);
145 /*-----*/
146 /*****/
147 #endif // _GRUPO_H

```

```

1  #ifndef _JUGADOR_H_
2  #define _JUGADOR_H_
3  #ifndef NULL
4  #define NULL      0
5  #endif
6  #include <string>
7  using namespace std;
8  /*-----*/
9  //                                ESTRUCTURAS
10 /*-----*/
11 typedef struct{
12     int id;
13     string nombre;
14     int goles;
15     int idEquipo;
16 }Jugador;
17 /*-----*/
18 //                                PRIMITIVAS
19 /*-----*/
20 /*
21     pre : el jugador no debe haber sido creado.
22     post: el jugador queda creado y preparado para ser usado.
23
24     jugador : estructura de datos a ser creado.
25 */
26 void crearJugador(Jugador &jugador);
27 /*-----*/
28 /*
29     pre : el jugador debe haber sido creado.
30     post: se obtiene el id del jugador creado pasado por parametro.
31
32     jugador : estructura de datos a ser usada.
33     return: id obtenida del jugador.
34 */
35 int getId(Jugador &jugador);
36 /*-----*/
37 /*
38     pre : el jugador debe haber sido creado.
39     post: se obtiene el nombre del jugador creado pasado por parametro.
40
41     jugador : estructura de datos a ser usada.
42     return: nombre obtenido del jugador.
43 */
44 string getNombre(Jugador &jugador);
45 /*-----*/
46 /*
47     pre : el jugador debe haber sido creado.
48     post: se obtiene los goles del jugador creado pasado por parametro.
49
50     jugador : estructura de datos a ser usada.
51     return: goles obtenidos del jugador.
52 */
53 int getGoles(Jugador &jugador);
54 /*-----*/
55 /*
56     pre : el jugador debe haber sido creado.
57     post: se setea el nombre del jugador creado pasado por parametro.
58
59     jugador : estructura de datos a ser usada.
60     nombre: valor a cargar en el jugador.
61 */
62 void setNombre(Jugador &jugador, string nombre);
63 /*-----*/
64 /*
65     pre : el jugador debe haber sido creado.
66     post: se setea el id del jugador creado pasado por parametro.

```

```

67
68     jugador : estructura de datos a ser usada.
69     id: valor a cargar en el jugador.
70 */
71 void setId(Jugador &jugador, int id);
72 /*-----*/
73 /*
74     pre : el jugador debe haber sido creado.
75     post: se setea los goles del jugador creado pasado por parametro.
76
77     jugador : estructura de datos a ser usada.
78     goles: valor a cargar en el jugador.
79 */
80 void setGoles(Jugador &jugador, int goles);
81 /*-----*/
82 /*
83     pre : el jugador debe haber sido creado.
84     post: se setea los id del equipo del jugador creado pasado por parametro.
85
86     jugador : estructura de datos a ser usada.
87     id: valor a cargar en el jugador.
88 */
89 void setIdEquipo(Jugador &jugador,int id);
90 /*-----*/
91 /*
92     pre : el jugador debe haber sido creado.
93     post: se obtiene los id del equipo del jugador creado pasado por parametro.
94
95     jugador : estructura de datos a ser usada.
96     return: id del equipo obtenido del jugador.
97 */
98 int getIdEquipo(Jugador &jugador);
99 /*-----*/
100 /*
101     pre : el jugador debe haber sido creado.
102     post: el jugador queda destruido (iniciado a su origen).
103
104     jugador: estructura de datos a ser destruido.
105 */
106 void destructor(Jugador &jugador);
107 /*-----*/
108 /*****/
109 #endif // _JUGADOR_H_

```