Archivo ListaEquipo.h

```
#ifndef _LISTAEQUIPO_H
#define _LISTAEQUIPO_H
#include "Equipo.h"
#ifndef NULL
#define NULL      0
#endif
/*--------------------------------------------------------------------------*/
//                              ESTRUCTURAS
/*--------------------------------------------------------------------------*/
enum ResultadoComparacionEquipo {
  MAYOR_EQUIPO,
  IGUAL_EQUIPO,
  MENOR_EQUIPO
};

struct NodoListaEquipo{
Equipo equipo;
NodoListaEquipo* siguiente;

};

typedef NodoListaEquipo* PtrNodoListaEquipo;

struct ListaEquipo{
PtrNodoListaEquipo primero;

};
/*--------------------------------------------------------------------------*/
//                              PRIMITIVAS
/*--------------------------------------------------------------------------*/
/*
  pre : la lista no debe haber sido creada.
  post: lista queda creada y preparada para ser usada.

  lista : estructura de datos a ser creado.
*/
void crearListaEquipo(ListaEquipo &lista);

/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: Devuelve true si lista esta vacia, sino devuelve false.

  lista : lista sobre la cual se invoca la primitiva.
*/
bool listaVaciaEquipo(ListaEquipo &lista);
/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: devuelve la representacion de lo Siguiente al último Nodo de la lista,
        o sea el valor Null, que en esta implementacion representa el final de
        la lista.

  return representación del fin de la lista.
*/
PtrNodoListaEquipo finEquipo();
/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si
        esta vacia

  lista : lista sobre la cual se invoca la primitiva.
  return puntero al primer nodo.
*/
PtrNodoListaEquipo primeroEquipo(ListaEquipo &lista);
```

```
/*-------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si
        ptrNodo apuntaba a fin() o si lista esta vacia.

  lista : lista sobre la cual se invoca la primitiva.
  prtNodo : puntero al nodo a partir del cual se requiere el siguiente.
  return puntero al nodo siguiente.
*/
PtrNodoListaEquipo siguienteEquipo(ListaEquipo &lista, PtrNodoListaEquipo ptrNodo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
        ptrNodo es un puntero a un nodo de lista.
  post: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si
        ptrNodo apuntaba al primero o si lista esta vacia.

  lista : lista sobre la cual se invoca la primitiva.
  prtNodo : puntero al nodo a partir del cual se requiere el anterior.
  return puntero al nodo anterior.
*/
PtrNodoListaEquipo anteriorEquipo(ListaEquipo &lista,PtrNodoListaEquipo ptrNodo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si
        si lista esta vacia.

  lista : lista sobre la cual se invoca la primitiva.
  return puntero al último nodo.
*/
PtrNodoListaEquipo ultimoEquipo(ListaEquipo &lista);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: crea el nodo de la lista.

  dato : elemento a almacenar en el nodo.
  return puntero al nodo creado.
*/
PtrNodoListaEquipo crearNodoListaEquipo(Equipo equipo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: agrega un nodo nuevo al principio de la lista con el dato proporcionado
        y devuelve un puntero a ese elemento.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a adicionar al principio de la lista.
  return puntero al nodo adicionado.
*/
PtrNodoListaEquipo adicionarAlPrincipio(ListaEquipo &lista, Equipo equipo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: agrega un nodo despues del apuntado por ptrNodo con el dato
        proporcionado y devuelve un puntero apuntado al elemento insertado.
        Si la lista esta vacía agrega un nodo al principio de esta y devuelve
        un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta
        nada y devuelve fin().

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a adicionar.
  ptrNodo : puntero al nodo después del cual se quiere adicionar el dato.
  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.
```

```
*/
void eliminarNodoPrimero(ListaEquipo &lista);

/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: si la lista no esta vacia elimina su nodo ultimo,
        sino no realiza accion.

  lista : lista sobre la cual se invoca la primitiva.
*/
void eliminarNodoUltimo(ListaEquipo &lista);

/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: elimina todos los Nodos de la lista quedando destruida e inhabilitada
        para su uso.

  lista : lista sobre la cual se invoca la primitiva.
*/
void eliminarLista(ListaEquipo &lista);
/*-------------------------------------------------------------------------*/
/*
  pre : ninguna.
  post: compara ambos dato1 y dato2, devuelve
          mayor si dato1 es mayor que dato2,
          igual si dato1 es igual a dato2,
          menor si dato1 es menor que dato2.

  dato1 : dato a comparar.
  dato2 : dato a comparar.
  return resultado de comparar dato1 respecto de dato2.
*/
ResultadoComparacionEquipo compararDatoEquipo(Equipo equipo1, Equipo equipo2);
/*-------------------------------------------------------------------------*/
/*
  pre : lista fue creada con crearLista().
  post: si el dato se encuentra en la lista, devuelve el puntero al primer nodo
        que lo contiene. Si el dato no se encuentra en la lista devuelve fin().

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a localizar.
  return puntero al nodo localizado o fin().
*/
PtrNodoListaEquipo localizarDato(ListaEquipo &lista, Equipo equipo);
/*-------------------------------------------------------------------------*/
/*
  pre : la lista fue creada con crearLista().
  post : elimina el dato de la lista, si el mismo se encuentra.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a eliminar.
*/
void eliminarDato(ListaEquipo &lista, Equipo equipo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista fue creada con crearLista() y cargada con datos ordenados de
        menor a mayor respecto del sentido progresivo.
  post: agrega a la lista el dato manteniendo el orden pero con multiples
        valores iguales y devuelve un puntero al nodo insertado.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a insertar.
  return puntero al nodo insertado.
*/
```

```
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 dato1 y dato2, devuelve
278           mayor si dato1 es mayor que dato2,
279           igual si dato1 es igual a dato2,
280           menor si dato1 es menor que dato2.
281 
282   dato1 : dato a comparar.
283   dato2 : dato a comparar.
284   return resultado de comparar dato1 respecto de dato2.
285 */
286 ResultadoComparacionEquipo compararGolesEquipo(Equipo equipo1, Equipo equipo2) ;
287 /*************************************************************************/
288 #endif // _LISTAEQUIPO_H
```

Archivo ListaGrupo.h

```
#ifndef _LISTAGRUPO_H_
#define _LISTAGRUPO_H_
#include "Grupo.h"
#ifndef NULL
#define NULL      0
#endif
/*--------------------------------------------------------------------------*/
//                                ESTRUCTURAS
/*--------------------------------------------------------------------------*/
enum ResultadoComparacionGrupo{
  MAYOR_GRUPO,
  IGUAL_GRUPO,
  MENOR_GRUPO
};

struct NodoGrupo{
Grupo grupo;
NodoGrupo* siguiente;
};
typedef NodoGrupo* PtrNodoGrupo;

struct ListaGrupo{
PtrNodoGrupo primero;
};
/*--------------------------------------------------------------------------*/
//                                PRIMITIVAS
/*--------------------------------------------------------------------------*/
/*
  pre : la lista no debe haber sido creada.
  post: lista queda creada y preparada para ser usada.

  lista : estructura de datos a ser creado.
*/
void crearListaGrupo(ListaGrupo &lista);
/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: devuelve la representacion de lo Siguiente al último Nodo de la lista,
        o sea el valor Null, que en esta implementacion representa el final de
        la lista.

  return representación del fin de la lista.
*/
PtrNodoGrupo finGrupo();
/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: Devuelve true si lista esta vacia, sino devuelve false.

  lista : lista sobre la cual se invoca la primitiva.
*/
bool listaVaciaGrupo(ListaGrupo lista);
/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si
        esta vacia

  lista : lista sobre la cual se invoca la primitiva.
  return puntero al primer nodo.
*/
PtrNodoGrupo primeroListaGrupo(ListaGrupo &lista);
/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  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 /*
```

```
  pre : lista creada con crearLista().
  post: agrega un nodo al final de la lista con el dato proporcionado y devuelve
        un puntero al nodo insertado.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a adicionar al final de la lista.
  return puntero al nodo adicionado.
*/
PtrNodoGrupo adicionarFinal(ListaGrupo &lista,Grupo grupo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo
        y devuelve un puntero al nodo insertado. Si la lista esta vacia no
        inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo
        insertado sera el nuevo primero.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a adicionar.
  ptrNodo : puntero al nodo antes del cual se quiere adicionar el dato.
  return puntero al nodo adicionado.
*/
PtrNodoGrupo adicionarAntes(ListaGrupo &lista,Grupo grupo,PtrNodoGrupo ptrNodoGrupo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
  post: coloca el dato proporcionado en el nodo apuntado por ptrNodo.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a colocar.
  ptrNodo : puntero al nodo del cual se quiere colocar el dato.
*/
void colocarDato(ListaGrupo &lista,Grupo grupo, PtrNodoGrupo ptrNodoGrupo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
  post: devuelve el dato del nodo apuntado por ptrNodo.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento obtenido.
  ptrNodo : puntero al nodo del cual se quiere obtener el dato.
*/
void obtenerDato(ListaGrupo &lista, Grupo &grupo, PtrNodoGrupo ptrNodo);

/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: elimina el nodo apuntado por ptrNodo. No realiza accion si la lista
        esta vacia o si ptrNodo apunta a fin().

  lista : lista sobre la cual se invoca la primitiva.
  ptrNodo : puntero al nodo que se desea eliminar.
*/
void eliminarNodo(ListaGrupo &lista, PtrNodoGrupo ptrNodo);

/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: si la lista no esta vacia, elimina su nodo primero, sino no realiza
        accion alguna.

  lista : lista sobre la cual se invoca la primitiva.
*/
void eliminarNodoPrimero(ListaGrupo &lista);

/*-------------------------------------------------------------------------*/
```

```
/*
  pre : lista creada con crearLista().
  post: si la lista no esta vacia elimina su nodo ultimo,
        sino no realiza accion.

  lista : lista sobre la cual se invoca la primitiva.
*/
void eliminarNodoUltimo(ListaGrupo &lista);

/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: elimina todos los Nodos de la lista quedando destruida e inhabilitada
        para su uso.

  lista : lista sobre la cual se invoca la primitiva.
*/
void eliminarLista(ListaGrupo &lista);
/*-------------------------------------------------------------------------*/
/*
  pre : ninguna.
  post: compara ambos dato1 y dato2, devuelve
          mayor si dato1 es mayor que dato2,
          igual si dato1 es igual a dato2,
          menor si dato1 es menor que dato2.

  dato1 : dato a comparar.
  dato2 : dato a comparar.
  return resultado de comparar dato1 respecto de dato2.
*/
ResultadoComparacionGrupo compararDatoGrupo(Grupo grupo1, Grupo grupo2);
/*-------------------------------------------------------------------------*/
/*
  pre : lista fue creada con crearLista().
  post: si el dato se encuentra en la lista, devuelve el puntero al primer nodo
        que lo contiene. Si el dato no se encuentra en la lista devuelve fin().

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a localizar.
  return puntero al nodo localizado o fin().
*/
PtrNodoGrupo localizarDato(ListaGrupo &lista, Grupo grupo);
/*-------------------------------------------------------------------------*/
/*
  pre : la lista fue creada con crearLista().
  post : elimina el dato de la lista, si el mismo se encuentra.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a eliminar.
*/
void eliminarDato(ListaGrupo &lista, Grupo grupo);
/*-------------------------------------------------------------------------*/
/*************************************************************************/

#endif // _LISTAGRUPO_H
```

Archivo ListaJugadores.h

```
#ifndef __LISTAJUGADORES_H_
#define __LISTAJUGADORES_H_
#include "Jugador.h"
#ifndef NULL
#define NULL      0
#endif
/*-------------------------------------------------------------------------*/
//                              ESTRUCTURAS
/*-------------------------------------------------------------------------*/
enum ResultadoComparacionJugador{
  MAYOR_JUGADOR,
  IGUAL_JUGADOR,
  MENOR_JUGADOR
};


struct NodoListaJugador{
    Jugador jugador;
    NodoListaJugador* sgte;
};


typedef NodoListaJugador* PtrNodoListaJugador;

struct ListaJugador{
    PtrNodoListaJugador primero;
};
/*-------------------------------------------------------------------------*/
//                              PRIMITIVAS
/*-------------------------------------------------------------------------*/
/*
  pre : la lista no debe haber sido creada.
  post: lista queda creada y preparada para ser usada.

  lista : estructura de datos a ser creado.
*/
void crearListaJugador(ListaJugador &lista);

/*-------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: Devuelve true si lista esta vacia, sino devuelve false.

  lista : lista sobre la cual se invoca la primitiva.
*/
bool listaVaciaJugador(ListaJugador &lista);

/*-------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: devuelve la representacion de lo Siguiente al último Nodo de la lista,
        o sea el valor Null, que en esta implementacion representa el final de
        la lista.

  return representación del fin de la lista.
*/
PtrNodoListaJugador finJugador();

/*-------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si
        esta vacia

  lista : lista sobre la cual se invoca la primitiva.
  return puntero al primer nodo.
```

```
*/
PtrNodoListaJugador primeroJugador(ListaJugador &lista);

/*-------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si
        ptrNodo apuntaba a fin() o si lista esta vacia.

  lista : lista sobre la cual se invoca la primitiva.
  prtNodo : puntero al nodo a partir del cual se requiere el siguiente.
  return puntero al nodo siguiente.
*/
PtrNodoListaJugador siguienteJugador(ListaJugador &lista, PtrNodoListaJugador ptrNodo);

/*-------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
        ptrNodo es un puntero a un nodo de lista.
  post: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si
        ptrNodo apuntaba al primero o si lista esta vacia.

  lista : lista sobre la cual se invoca la primitiva.
  prtNodo : puntero al nodo a partir del cual se requiere el anterior.
  return puntero al nodo anterior.
*/
PtrNodoListaJugador anteriorJugador(ListaJugador &lista, PtrNodoListaJugador ptrNodo);

/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si
        si lista esta vacia.

  lista : lista sobre la cual se invoca la primitiva.
  return puntero al último nodo.
*/
PtrNodoListaJugador ultimoJugador(ListaJugador &lista);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: crea el nodo de la lista.

  dato : elemento a almacenar en el nodo.
  return puntero al nodo creado.
*/
PtrNodoListaJugador crearNodoListaJugador(Jugador jugador);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: agrega un nodo nuevo al principio de la lista con el dato proporcionado
        y devuelve un puntero a ese elemento.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a adicionar al principio de la lista.
  return puntero al nodo adicionado.
*/
PtrNodoListaJugador adicionarPrincipio(ListaJugador &lista, Jugador jugador);

/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: agrega un nodo despues del apuntado por ptrNodo con el dato
        proporcionado y devuelve un puntero apuntado al elemento insertado.
        Si la lista esta vacía agrega un nodo al principio de esta y devuelve
        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.
```

```
*/
void eliminarNodo(ListaJugador &lista, PtrNodoListaJugador ptrNodo);

/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: si la lista no esta vacia, elimina su nodo primero, sino no realiza
        accion alguna.

  lista : lista sobre la cual se invoca la primitiva.
*/
void eliminarNodoPrimero(ListaJugador &lista);

/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: si la lista no esta vacia elimina su nodo ultimo,
        sino no realiza accion.

  lista : lista sobre la cual se invoca la primitiva.
*/
void eliminarNodoUltimo(ListaJugador &lista);

/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: elimina todos los Nodos de la lista quedando destruida e inhabilitada
        para su uso.

  lista : lista sobre la cual se invoca la primitiva.
*/
void eliminarLista(ListaJugador &lista);
/*-------------------------------------------------------------------------*/
/*
  pre : ninguna.
  post: compara ambos dato1 y dato2, devuelve
          mayor si dato1 es mayor que dato2,
          igual si dato1 es igual a dato2,
          menor si dato1 es menor que dato2.

  dato1 : dato a comparar.
  dato2 : dato a comparar.
  return resultado de comparar dato1 respecto de dato2.
*/
ResultadoComparacionJugador compararDatoJugador(Jugador jugador1, Jugador jugador2);
/*-------------------------------------------------------------------------*/
/*
  pre : lista fue creada con crearLista().
  post: si el dato se encuentra en la lista, devuelve el puntero al primer nodo
        que lo contiene. Si el dato no se encuentra en la lista devuelve fin().

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a localizar.
  return puntero al nodo localizado o fin().
*/
PtrNodoListaJugador localizarDato(ListaJugador &lista, Jugador jugador);
/*-------------------------------------------------------------------------*/
/*
  pre : la lista fue creada con crearLista().
  post : elimina el dato de la lista, si el mismo se encuentra.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a eliminar.
*/
void eliminarDato(ListaJugador &lista, Jugador jugador);
/*-------------------------------------------------------------------------*/
```

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

Archivo ListaPartidos.h

```
#ifndef _LISTAPARTIDO_H_
#define _LISTAPARTIDO_H_
#include "Partido.h"
#ifndef NULL
#define NULL      0
#endif
/*--------------------------------------------------------------------------*/
//                                ESTRUCTURAS
/*--------------------------------------------------------------------------*/
enum ResultadoComparacionPartido{
  MAYOR_PARTIDO,
  IGUAL_PARTIDO,
  MENOR_PARTIDO
};
struct NodoListaPartido{
Partido partido;
NodoListaPartido* siguiente;

};

typedef NodoListaPartido* PtrNodoPartido;

struct ListaPartido{
PtrNodoPartido primero;
};
/*--------------------------------------------------------------------------*/
//                                PRIMITIVAS
/*--------------------------------------------------------------------------*/
/*
  pre : la lista no debe haber sido creada.
  post: lista queda creada y preparada para ser usada.

  lista : estructura de datos a ser creado.
*/
void crearListaPartido(ListaPartido &lista);
/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: Devuelve true si lista esta vacia, sino devuelve false.

  lista : lista sobre la cual se invoca la primitiva.
*/
bool listaVaciaPartido(ListaPartido &lista);
/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: devuelve la representacion de lo Siguiente al último Nodo de la lista,
        o sea el valor Null, que en esta implementacion representa el final de
        la lista.

  return representación del fin de la lista.
*/
PtrNodoPartido finListaPartido();
/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
  post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si
        esta vacia

  lista : lista sobre la cual se invoca la primitiva.
  return puntero al primer nodo.
*/
PtrNodoPartido primeroPartido(ListaPartido &lista);
/*--------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
```

```
  post: devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si
        ptrNodo apuntaba a fin() o si lista esta vacia.

  lista : lista sobre la cual se invoca la primitiva.
  prtNodo : puntero al nodo a partir del cual se requiere el siguiente.
  return puntero al nodo siguiente.
*/
PtrNodoPartido siguientePartido(ListaPartido &lista, PtrNodoPartido ptrNodo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista Creada con crearLista().
        ptrNodo es un puntero a un nodo de lista.
  post: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si
        ptrNodo apuntaba al primero o si lista esta vacia.

  lista : lista sobre la cual se invoca la primitiva.
  prtNodo : puntero al nodo a partir del cual se requiere el anterior.
  return puntero al nodo anterior.
*/
PtrNodoPartido anteriorPartido(ListaPartido &lista, PtrNodoPartido ptrNodo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si
        si lista esta vacia.

  lista : lista sobre la cual se invoca la primitiva.
  return puntero al último nodo.
*/
PtrNodoPartido ultimoPartido(ListaPartido &lista);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: crea el nodo de la lista.

  dato : elemento a almacenar en el nodo.
  return puntero al nodo creado.
*/
PtrNodoPartido crearNodoListaPartido(Partido partido);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: agrega un nodo nuevo al principio de la lista con el dato proporcionado
        y devuelve un puntero a ese elemento.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a adicionar al principio de la lista.
  return puntero al nodo adicionado.
*/
PtrNodoPartido adicionarPrincipio(ListaPartido &lista, Partido partido);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: agrega un nodo despues del apuntado por ptrNodo con el dato
        proporcionado y devuelve un puntero apuntado al elemento insertado.
        Si la lista esta vacía agrega un nodo al principio de esta y devuelve
        un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta
        nada y devuelve fin().

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a adicionar.
  ptrNodo : puntero al nodo después del cual se quiere adicionar el dato.
  return puntero al nodo adicionado.
*/
PtrNodoPartido adicionarDespues(ListaPartido &lista, Partido partido, PtrNodoPartido ptrNodo);
/*-------------------------------------------------------------------------*/
```

```
/*
  pre : lista creada con crearLista().
  post: agrega un nodo al final de la lista con el dato proporcionado y devuelve
        un puntero al nodo insertado.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a adicionar al final de la lista.
  return puntero al nodo adicionado.
*/
PtrNodoPartido adicionarFinal(ListaPartido &lista, Partido partido);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo
        y devuelve un puntero al nodo insertado. Si la lista esta vacia no
        inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo
        insertado sera el nuevo primero.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a adicionar.
  ptrNodo : puntero al nodo antes del cual se quiere adicionar el dato.
  return puntero al nodo adicionado.
*/
PtrNodoPartido adicionarAntes(ListaPartido &lista, Partido partido, PtrNodoPartido ptrNodo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
  post: coloca el dato proporcionado en el nodo apuntado por ptrNodo.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento a colocar.
  ptrNodo : puntero al nodo del cual se quiere colocar el dato.
*/
void colocarDato(ListaPartido &lista, Partido &partido, PtrNodoPartido ptrNodo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista(), no vacia. ptrNodo es distinto de fin().
  post: devuelve el dato del nodo apuntado por ptrNodo.

  lista : lista sobre la cual se invoca la primitiva.
  dato : elemento obtenido.
  ptrNodo : puntero al nodo del cual se quiere obtener el dato.
*/
void obtenerDato(ListaPartido &lista, Partido &partido, PtrNodoPartido ptrNodo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: elimina el nodo apuntado por ptrNodo. No realiza accion si la lista
        esta vacia o si ptrNodo apunta a fin().

  lista : lista sobre la cual se invoca la primitiva.
  ptrNodo : puntero al nodo que se desea eliminar.
*/
void eliminarNodo(ListaPartido &lista, PtrNodoPartido ptrNodo);
/*-------------------------------------------------------------------------*/
/*
  pre : lista creada con crearLista().
  post: si la lista no esta vacia, elimina su nodo primero, sino no realiza
        accion alguna.

  lista : lista sobre la cual se invoca la primitiva.
*/
void eliminarNodoPrimero(ListaPartido &lista);
/*-------------------------------------------------------------------------*/
/*
  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
```

Archivo Partido.h

```
#ifndef _PARTIDO_H
#define _PARTIDO_H
#ifndef NULL
#define NULL      0
#endif
/*-----------------------------------------------------------------------------*/
//                                ESTRUCTURAS
/*-----------------------------------------------------------------------------*/
typedef struct{
    int id;
    int idEquipoL;
    int idEquipoV;
    int golesL;
    int golesV;
}Partido;
/*-----------------------------------------------------------------------------*/
//                                PRIMITIVAS
/*-----------------------------------------------------------------------------*/
/*
  pre : el partido no debe haber sido creado.
  post: el partido queda creado y preparado para ser usado.

  partido : estructura de datos a ser creado.
*/
void crearPartido(Partido &partido);
/*-----------------------------------------------------------------------------*/
/*
  pre : el partido debe haber sido creado.
  post: se obtiene el id del partido creado pasado por parametro.

  partido : estructura de datos a ser usada.
  return: id obtenida del partido.
*/
int getId(Partido &partido);
/*-----------------------------------------------------------------------------*/
/*
  pre : el partido debe haber sido creado.
  post: se obtiene el id del equipoL del partido creado pasado por parametro.

  partido : estructura de datos a ser usada.
  return: id del equipoL obtenido del partido.
*/
int getIdEquipoL(Partido &partido);
/*-----------------------------------------------------------------------------*/
/*
  pre : el partido debe haber sido creado.
  post: se obtiene el id del equipoV del partido creado pasado por parametro.

  partido : estructura de datos a ser usada.
  return: id del equipoV obtenido del partido.
*/
int getIdEquipoV(Partido &partido);
/*-----------------------------------------------------------------------------*/
/*
  pre : el partido debe haber sido creado.
  post: se obtiene los golesL del partido creado pasado por parametro.

  partido : estructura de datos a ser usada.
  return: golesL obtenidos del partido.
*/
int getGolesL(Partido &partido);
/*-----------------------------------------------------------------------------*/
/*
  pre : el partido debe haber sido creado.
  post: se obtiene los golesV del partido creado pasado por parametro.

```

```
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
```

Archivo Equipo.h

```
#ifndef _EQUIPO_H
#define _EQUIPO_H
#ifndef NULL
#define NULL      0
#endif
#include "ListaJugadores.h"
#include <string>
using namespace std;
/*-------------------------------------------------------------------------*/
//                              ESTRUCTURAS
/*-------------------------------------------------------------------------*/
typedef struct {
int id;
string nombre;
int golesAFavor;
int golesEnContra;
int puntos;
ListaJugador listaJugadores;
}Equipo;
/*-------------------------------------------------------------------------*/
//                              PRIMITIVAS
/*-------------------------------------------------------------------------*/
/*
  pre : el equipo no debe haber sido creado.
  post: el equipo queda creado y preparado para ser usado.

  equipo : estructura de datos a ser creado.
*/
void crearEquipo(Equipo &equipo);
/*-------------------------------------------------------------------------*/
/*
  pre : el equipo debe haber sido creado.
  post: se obtiene el id del equipo creado pasado por parametro.

  equipo : estructura de datos a ser usada.
  return: id obtenida del equipo.
*/
int getId(Equipo equipo);
/*-------------------------------------------------------------------------*/
/*
  pre : el equipo debe haber sido creado.
  post: se setea el id del equipo creado pasado por parametro.

  equipo : estructura de datos a ser usada.
  id: valor a cargar en el equipo.
*/

void setId(Equipo &equipo, int id);
/*-------------------------------------------------------------------------*/
/*
  pre : el equipo debe haber sido creado.
  post: se obtiene el nombre del equipo creado pasado por parametro.

  equipo : estructura de datos a ser usada.
  return:nombre obtenido del equipo.
*/
string getNombre(Equipo equipo);
/*-------------------------------------------------------------------------*/
/*
  pre : el equipo debe haber sido creado.
  post: se setea el nombre del equipo creado pasado por parametro.

  equipo : estructura de datos a ser usada.
  nombre: valor a cargar en el equipo.
*/
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 getLista(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
```

Archivo Grupo.h

```
#ifndef _GRUPO_H
#define _GRUPO_H_
#ifndef NULL
#define NULL      0
#endif
#include <string>
using namespace std;
/*-------------------------------------------------------------------------*/
//                              ESTRUCTURAS
/*-------------------------------------------------------------------------*/
typedef struct{
char id;
string nombre;
int idEquipo1;
int idEquipo2;
int idEquipo3;
int idEquipo4;
}Grupo;
/*-------------------------------------------------------------------------*/
//                              PRIMITIVAS
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo no debe haber sido creado.
  post: el grupo queda creado y preparado para ser usado.

  grupo : estructura de datos a ser creado.
*/
void crearGrupo(Grupo &grupo);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se setea el id del grupo creado pasado por parametro.

  grupo : estructura de datos a ser usada.
  id: valor a cargar en el grupo.
*/
void setId(Grupo &grupo,char id);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se obtiene el id del grupo creado pasado por parametro.

  grupo : estructura de datos a ser usada.
  return: id obtenida del grupo.
*/
char getId(Grupo grupo);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se setea el nombre del grupo creado pasado por parametro.

  jugador : estructura de datos a ser usada.
  nombre: valor a cargar en el grupo.
*/
void setNombre(Grupo &grupo,string nombre);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se obtiene el nombre del grupo creado pasado por parametro.

  grupo : estructura de datos a ser usada.
  return: nombre obtenido del grupo.
*/
string getNombre(Grupo grupo);
/*-------------------------------------------------------------------------*/
/*
```

```
  pre : el grupo debe haber sido creado.
  post: se setea el id del equipo 1 del grupo creado pasado por parametro.

  grupo : estructura de datos a ser usada.
  id: valor del equipo 1 a cargar en el grupo.
*/
void setIdEquipo1(Grupo &grupo,int id);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se obtiene el id del equipo 1 del grupo creado pasado por parametro.

  grupo : estructura de datos a ser usada.
  return: id del equipo 1 obtenida del grupo.
*/
int getIdEquipo1(Grupo grupo);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se setea el id del equipo 2 del grupo creado pasado por parametro.

  grupo : estructura de datos a ser usada.
  id: valor del equipo 2 a cargar en el grupo.
*/
void setIdEquipo2(Grupo &grupo,int id);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se obtiene el id del equipo 2 del grupo creado pasado por parametro.

  grupo : estructura de datos a ser usada.
  return: id del equipo 2 obtenida del grupo.
*/
int getIdEquipo2(Grupo grupo);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se setea el id del equipo 3 del grupo creado pasado por parametro.

  grupo : estructura de datos a ser usada.
  id: valor del equipo 3 a cargar en el grupo.
*/
void setIdEquipo3(Grupo &grupo,int id);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se obtiene el id del equipo 3 del grupo creado pasado por parametro.

  grupo : estructura de datos a ser usada.
  return: id del equipo 3 obtenida del grupo.
*/
int getIdEquipo3(Grupo grupo);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se setea el id del equipo 4 del grupo creado pasado por parametro.

  grupo : estructura de datos a ser usada.
  id: valor del equipo 4 a cargar en el grupo.
*/
void setIdEquipo4(Grupo &grupo,int id);
/*-------------------------------------------------------------------------*/
/*
  pre : el grupo debe haber sido creado.
  post: se obtiene el id del equipo 4 del grupo creado pasado por parametro.

```

```
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
```

Archivo Jugador.h

```
#ifndef _JUGADOR_H_
#define _JUGADOR_H_
#ifndef NULL
#define NULL      0
#endif
#include <string>
using namespace std;
/*-------------------------------------------------------------------------------*/
//                                ESTRUCTURAS
/*-------------------------------------------------------------------------------*/
typedef struct{
int id;
string nombre;
int goles;
int idEquipo;
}Jugador;
/*-------------------------------------------------------------------------------*/
//                                PRIMITIVAS
/*-------------------------------------------------------------------------------*/
/*
  pre : el jugador no debe haber sido creado.
  post: el jugador queda creado y preparado para ser usado.

  jugador : estructura de datos a ser creado.
*/
void crearJugador(Jugador &jugador);
/*-------------------------------------------------------------------------------*/
/*
  pre : el jugador debe haber sido creado.
  post: se obtiene el id del jugador creado pasado por parametro.

  jugador : estructura de datos a ser usada.
  return: id obtenida del jugador.
*/
int getId(Jugador &jugador);
/*-------------------------------------------------------------------------------*/
/*
  pre : el jugador debe haber sido creado.
  post: se obtiene el nombre del jugador creado pasado por parametro.

  jugador : estructura de datos a ser usada.
  return: nombre obtenido del jugador.
*/
string getNombre(Jugador &jugador);
/*-------------------------------------------------------------------------------*/
/*
  pre : el jugador debe haber sido creado.
  post: se obtiene los goles del jugador creado pasado por parametro.

  jugador : estructura de datos a ser usada.
  return: goles obtenidos del jugador.
*/
int getGoles(Jugador &jugador);
/*-------------------------------------------------------------------------------*/
/*
  pre : el jugador debe haber sido creado.
  post: se setea el nombre del jugador creado pasado por parametro.

  jugador : estructura de datos a ser usada.
  nombre: valor a cargar en el jugador.
*/
void setNombre(Jugador &jugador,string nombre);
/*-------------------------------------------------------------------------------*/
/*
  pre : el jugador debe haber sido creado.
  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_
```