Enunciado Trabajo Práctico

Rusia 2018

Algoritmos y Estructuras de Datos

Ing. Diego Azcurra – Ing. Damián Santos

UNLA

2018

Padrón	Nombre	Evaluación Individual
	Griselda Benítez Haugg	
	José Maximiliano Lucero	
	Nicolas Nahuel Trezza	
	Martín Olmos	
	José Víctor Ibáñez	
Evaluación Trabajo		

1. Índice General

2. Objetivo

Desarrollar una aplicación en c++ que permita administrar los partidos del mundial Rusia 2018.

3. Definiciones

3.1.Equipo

Un EQUIPO se define por:

- Id (entero): campo identificador del equipo.
- Nombre (cadena): nombre del equipo.
- Goles A Favor (entero): cantidad de goles convertidos por el equipo.
- Goles En Contra (entero): cantidad de goles recibidos por el equipo.
- Puntos (entero): Cantidad de puntos obtenidos por el equipo.

3.2.Jugador:

Un JUGADOR se define por:

- Id (entero): campo identificador del jugador.
- Nombre (cadena): nombre del jugador.
- Goles (entero): Cantidad de goles Convertidos (los goles en contra restan).

Los jugadores integran EQUIPOS.

3.3.Grupo.

Un GRUPO se define por:

- Id (char): campo identificador del grupo (de la A a la H).
- Nombre (cadena): nombre utilizado para el grupo (“Grupo “ + Id).
- IdEquipo1 (entero): id del equipo 1.
- IdEquipo2 (entero): id del equipo 2.
- IdEquipo3 (entero): id del equipo 3.
- IdEquipo4 (entero): id del equipo 4.

3.4.Partido

Un PARTIDO se define por:

- Id (entero): campo de identificación del partido.
- idEquipoL (entero): Id del equipo Local.
- idEquipoV (entero): Id del equipo Visitante.
- golesL (entero): goles a favor equipo Local.
- golesV (entero): goles a favor equipo Visitante.

4. Descripción

La FIFA ha decidido contratarnos para desarrollar el sistema que administrará los partidos del mundial de Rusia 2018.

El sistema deberá poder administrar los datos de los equipos, jugadores, grupos y partidos. Toda esta información estará registrada en archivos de texto y deberá ser cargada al inicio de la aplicación y almacenada luego del cierre de la misma.

La operación del sistema quedará en manos de un administrador que atenderá todas las solicitudes y registrará todas las actualizaciones.

Durante el mundial podrán jugarse varios partidos en paralelo. El operador deberá:

1) Registrar el inicio de un partido (identificado por id).
2) Registrar los goles ocurridos en cada uno (identificando equipo y jugador).
3) Registrar el fin de un partido (informando el id del mismo). En este momento los datos se reflejaran en los equipos (puntos y goles) y jugadores (goles). La información debe procesarse en el mismo orden en que ingresó.

El sistema deberá informar por pantalla un alerta cada 10 goles y hacer una mención especial para los múltiplos de 100.

Cuando un partido se procesa se deberán definir las llaves correspondientes en caso de ser posible.

Para la identificación de los partidos se tomará la información detallada en el sitio web de la FIFA:

https://resources.fifa.com/image/upload/2018-fifa-world-cup-russiatm-match-schedule-2667162-2667165.pdf?cloudid=yxmyaqmxyi7pvhd7your

Los grupos deberán diseñar alguna estrategia que les permita manejar la conformación de llaves.

5. Formato de archivos

5.1.Equipo

Archivo equipos.txt:

Id (entero); Nombre (cadena); Goles A Favor (entero); Goles En Contra (entero); Puntos (entero)

Ejemplo:

1;Argentina;10;3;9

…

5.2.Jugador:

Archivo jugadores.txt:

Id (entero); Nombre (cadena); Goles (entero); idEquipo (entero)

Ejemplo:

1;Marcos Rojo;10;1

…

5.3.Grupo.

Archivo grupos.txt:

Id (char); Nombre (cadena); IdEquipo1 (entero); IdEquipo2 (entero); IdEquipo3 (entero); IdEquipo4 (entero).

Ejemplo:

A;Grupo A;2;3;5;6

…

5.4.Partido

Archivo partidos.txt:

Id (entero); idEquipoL (entero); idEquipoV (entero); golesL (entero); golesV (entero).

Ejemplo:

1;14;12;0;0

…

En el caso de los partidos que aún no se hayan jugado, los goles o los ids de equipos se registrarán con el valor “-1”.

NOTA:

En todos los casos, el separador de campos debe ser “;”.

Como parte del trabajo, los alumnos deberán desarrollar un archivo de pruebas con datos válidos para ejecutar las validaciones.

6. Requerimientos

Se pide desarrollar una aplicación que permita:

1) Cargar la información de los equipos, jugadores, grupos y partidos desde los archivos equipos.txt, jugadores.txt, grupos.txt y partidos.txt (durante el inicio de la aplicación).
2) Guardar la información de los equipos, jugadores, grupos y partidos en los archivos equipos.txt, jugadores.txt, grupos.txt y partidos.txt (durante el cierre de la aplicación).
3) Presentar un menú con las siguientes opciones:
    a. Administrar equipos, jugadores, grupos y partidos (Alta, Baja y Modificación) debido a que pueden existir errores en los datos.
    b. Administrar Partidos
        i. Registrar el inicio de un partido (identificado por id).
        ii. Registrar los goles ocurridos en cada partido (identificando equipo y jugador).
        iii. Registrar el fin de un partido.
    c. Procesar reportes:
        i. Listado de goleadores. Ordenado en forma descendente por cantidad de goles. El reporte se deberá segmentar por cantidad de goles (por ejemplo, si el máximo de goles es 10, deberá figurar “10 goles” y todos los jugadores con 10 goles, después 9, y todos los que convirtieron 9 goles, etc).
        Se deberá poner un subtotal indicando la cantidad de jugadores en cada segmento y un total final con cantidad de jugadores y cantidad de goles.
        ii. Orden de equipos por grupo (con puntos y diferencia de gol). Es decir, Grupo A, equipos del Grupo A, Grupo B, equipos del Grupo B, etc. Además, se deben informar los goles parciales por grupos y el total de todos los grupos.
        iii. Listar el fixture con todos los resultados y los partidos pendientes ordenados por partido.
        iv. Deberá informar “el grupo de la muerte” que será el grupo en que se hayan efectuado mayor cantidad de goles totales.
        v. Porcentaje goles local o visitante.
    d. Salir de la aplicación (se deberán grabar los datos en los archivos).

Validaciones:

El sistema deberá contar con las validaciones necesarias para permitir su correcto funcionamiento. Por ejemplo:

- No deberán existir ids duplicados para cada una de las entidades administradas.

- Se deberá controlar la carga de los archivos. Si bien los archivos no van a tener errores de sintaxis, puede ocurrir que haya errores en las referencias entre las entidades, por ejemplo, un jugador que pertenezca a un equipo que no existe, un jugador en dos equipos, un equipo en dos grupos, un grupo sin todos los equipos, etc. El grupo deberá identificar y tratar la mayor cantidad de errores posibles.
  En caso de encontrar errores el sistema deberá imprimir por pantalla un informe con los errores encontrados en el momento en que se inicia la aplicación y antes de mostrar el menú principal.

7. Presentación

A continuación se detalla el cronograma de actividades para el trabajo práctico:

19/05/18: Presentación del enunciado.

26/05/18: **Primer Entrega:** "Estrategia de Resolución". Se presentará la estrategia de resolución propuesta por el equipo.

02/06/18: Consulta del trabajo práctico.

09/06/18: No hay actividades previstas para el TP.

16/06/18: Consulta del trabajo práctico.

23/06/18: **Segunda Entrega:** Presentación y Evaluación del TP.

06/07/18: **Recuperatorio del TP**.

Primer Entrega: "Estrategia de Resolución".

Para esta entrega deberá presentarse un documento impreso (en un folio o carpeta) conteniendo, como mínimo:

- Carátula de presentación con los datos de los integrantes del equipo (primer hoja de este documento).
- Indice de contenidos (en caso de ser necesario).
- Desarrollo de la estrategia de resolución detallando:
  - Estructuras a utilizar (arrays, pilas, listas, colas).
  - TDAs y sus relaciones (diagrama de interacción de todos los componentes).
  - Estrategia de resolución de operaciones.
- División del trabajo y cronograma: se deberá presentar la división de actividades dentro del grupo y un cronograma de trabajo a alto nivel.

Todas las hojas deben estar numeradas.

Segunda Entrega.

Para esta entrega deberá presentarse:

- Una copia impresa del enunciado del trabajo práctico (TODO este documento, incluyendo los anexos).
- Una copia impresa de la estrategia de resolución final del trabajo práctico (con los mismos requerimientos que en la "Primer Entrega".
- Una copia impresa de todos los archivos de prueba presentados por los alumnos.
- Una copia impresa de todos los archivos del proyecto (.h y .cpp). Poner como encabezado de cada hoja el nombre del archivo.
- Un CD conteniendo en formato digital todos los puntos anteriores y el proyecto completo.
- Se deberá incluir la primer entrega corregida.

La presentación deberá ser en un folio o carpeta, en forma prolija y debidamente identificada. Los CDs deberán contener el número de grupo y el nombre y los padrones o documentos de identidad de cada uno de los integrantes y deberán estar correctamente adjuntos al resto del trabajo práctico de forma tal que no puedan perderse. Además, deberá incluirse **todo** el proyecto desarrollado (**la carpeta completa** generada por el IDE, con los archivos del proyecto y el código fuente) incluyendo los archivos de pruebas.

Todas las hojas deben estar numeradas.

El incumplimiento de cualquiera de las normas de entrega implicará la desaprobación del trabajo práctico.

Metodología de evaluación:

La Evaluación de los trabajos prácticos contará con una etapa grupal y una individual.

- Grupal: Se realizará un conjunto de pruebas sobre el trabajo presentado por los alumnos en presencia de los mismos. Se deberá aprobar la totalidad de las pruebas. En caso de que una prueba falle, los alumnos podrán intentar corregir el código mientras dure la evaluación.
- Individual: Se realizará una evaluación individual oral o escrita para cada alumno. Los temas a evaluar podrán ser, por ejemplo: preguntas teóricas sobre el contenido de la materia, preguntas sobre el trabajo práctico, codificación de alguna primitiva o modificación del trabajo práctico, etc.

La nota final de la cursada se calculará en función de las notas obtenidas en forma grupal e individual. La nota grupal será el promedio entre la primer presentación y el recuperatorio (en caso de necesitarlo). Por este motivo, SOLO deberán presentarse

aquellos grupos que hayan concluido TODO el trabajo práctico ya que no se harán evaluaciones parciales.

8. Revisiones

Anexo – Correcciones

Prueba	Resultado	Comentario

Anexo – Correcciones (copia para los alumnos)

Prueba	Resultado	Comentario

9. Estructuras a utilizar

1- Se generará una cola de partidos ya que tiene un día y un horario, ya que juegan en ese orden.

2- Habrá una lista de grupos donde a su vez cada nodo tendrá una lista de equipos y cada equipo tendrá una lista de partidos y una lista de jugadores donde a su vez tendrá una lista de goles. Descripto en la Tabla 2.

3- Se construirán, en primera instancia, cinco TDAs primarios.
Equipos-Partidos-Grupos-Jugadores-Llaves
Cuyas relaciones están diagramadas en la Tabla 1.

4- A su vez generaremos otros TDAs secundarios, como funciones, administración, goles y PartidoEnCurso.

5- Las relaciones son las siguientes, el mundial tiene una lista de grupos, los grupos una lista de equipos, los equipos una lista de jugadores y una lista de goles. Además, el mundial tiene una lista de goles. Cada equipo posee una lista de partidos que deberá jugar. Los partidos se los puede pensar como una cola, como se dije en el ítem 1 puesto que hay que jugarlos en ese único orden.

6- Para manejar estas listas usaremos listas Void.

10. TDA y sus relaciones

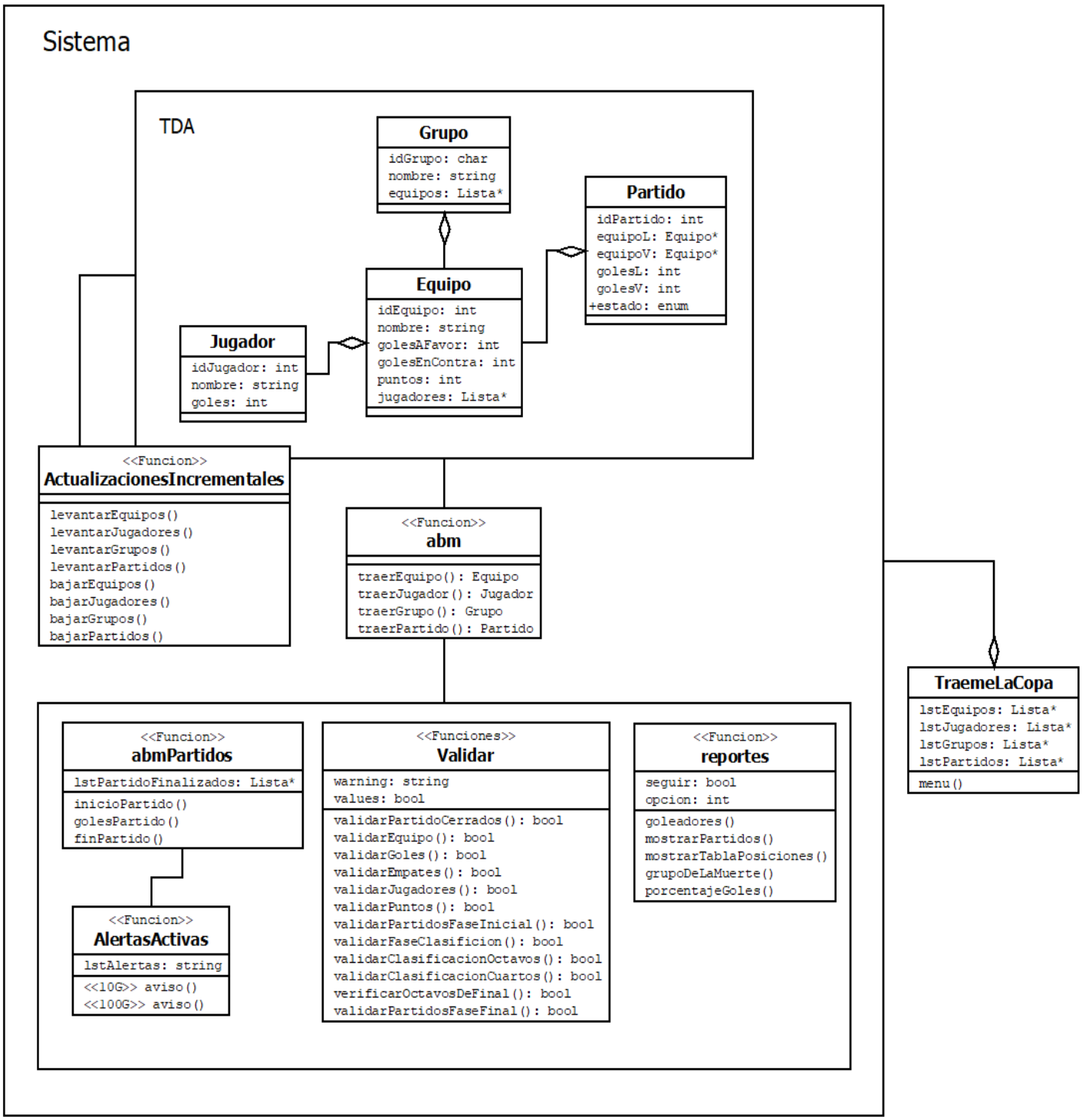

Tabla 1: Relación entre los TDAs principales.

Estructura Básicas:

```
typedef struct {
        int idEquipo;
        string nombre;
        int golesFavor;
        int golesContra;
        int puntos;
        Lista* jugadores;
}Equipo;
```

```
typedef struct {
        int idPartido;
        Equipo* equipoL;
        Equipo* equipoV;
        int golesL;
        int golesV;
        enum estado;
}Partido;
```

```
typedef struct {
        char idGrupo;
        string nombre;
        Lista* equipos;
}Grupo;
```

```
typedef struct {
        int idJugador;
        string nombre;
        int goles;
}Jugador;
```

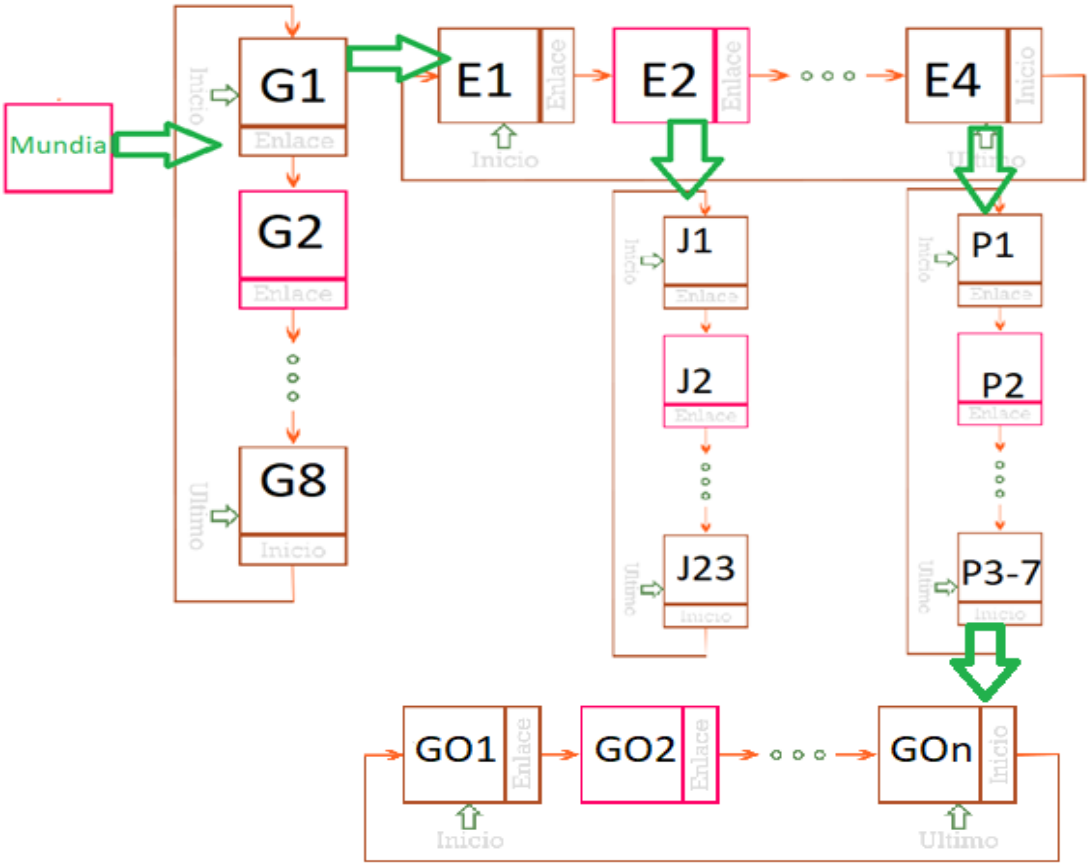

Tabla 2: Las listas del mundial

11. Estrategia de resolución

1) Un menú con las siguientes opciones:

- Va administrar los equipos, los jugadores, los grupos y los partidos con Alta, Baja y Modificación con las validaciones pertinentes, guardando la información con txt durante el cierre de la aplicación.

- Para administrar los partidos hay que registrar el inicio de un partido identificado por id, los goles ocurridos en cada partido, pero identificando equipo y jugador y por ultimo registrar el fin de un partido.

2) Procesar reportes:

- Tendrá un listado de goleadores ordenado en forma descendente por cantidad de goles con un subtotal y total final indicando la cantidad de jugadores y cantidad de goles.

3) Relaciones entre las estructuras:

- Al iniciar el mundial es necesario conocer los equipos que conforman los grupos, por esto es que se inicia la lista de grupos, desde el grupo A al H.

- Una vez que tengamos los grupos cargados, debemos cargarle los 4 equipos a estos 8 grupos, es decir agregarle una lista de equipos a cada nodo de la lista de grupos.

- Antes o después del paso anterior se agrega una lista de jugadores a cada elemento de la lista de equipos. Aquí se realiza la validación de que el jugador no exista en otros equipos, por ejemplo, suponiendo que su id es su dni.

- Con los grupos cargados, los equipos listos y los jugadores asignados se pasará a definir las llaves. Para definir las llaves debemos basarnos en el reglamento preestablecido, es decir el primero del A vs el segundo del B, etc.

- ¿Cómo realizar esta asignación entre los equipos? Para eso es necesario ordenar a los equipos en función de sus puntos. Para eso usaremos el campo puntos, que ira sumando de a 3 en las victorias y de a 1 en los empates. Una vez terminada la ronde de los grupos, es decir los 9 partidos por grupo se ordena por puntos a los equipos. En caso de igualdad de puntos se ordenará por diferencia de goles. Y en caso de empate ordenaremos según su enfrentamiento. En caso de haber empatado usaremos la función random.

- Ya en las llaves finales es mucho más simple definir los cruces. En caso de empate se define por penales. Los goles en tiempo extra los sumaremos como goles ordinarios y la victoria por penales también será como un gol extra.

- Los goles se irán guardando uno por uno, con su id auto incrementable para que sea más fácil encontrar los múltiplos de 10 y de 100.

12. Cronograma de trabajo grupal

Miércoles 23: Reunión completa del grupo para comprobar el entendimiento del ejercicio y para discutir formas de solucionar cada uno de los problemas planteados en el trabajo a resolver. Se tomaron nota de preguntas sobre el enunciado y futura codificación. Gráficos sobre la resolución y escritura provisoria de la primera entrega.

Viernes 1/Sábado 2: Primeras consultas con los docentes para sacarnos las dudas descubiertas en el primer encuentro. Entregar Primer Entrega: "Estrategia de Resolución".

Lunes 4: Segunda reunión grupal, donde se distribuirán los segmentos a resolver en primera instancia por cada uno. Creación de buena parte de los TDAs antes mencionados.
Miércoles 6: Se generarán los TDAs.

Viernes 8: Generación del Menú.

Lunes 11: Discusión grupal sobre el trabajo. Nueva distribución de actividades en el grupo. Recolección de dudas.

Sábado 16: Consulta de dudas.

Lunes 18: Reunión grupal, los integrantes plasmaremos con el resto del equipo los problemas individuales surgidos en la codificación y trataremos de resolverlos de forma grupal. Primer compilación "total" para comprobar correcto funcionamiento de las listas, pilas, colas y manejo de archivos. Distribución de los enunciados PUNTUALES a resolver.

Martes 19: Consulta de dudas sobre todo las que no pudieron ser resueltas de forma grupal.

Miércoles 20 /Jueves 21: Cada integrante llevara los enunciados que debía resolver, se hará una segunda compilación a posteriori de tratar de solucionar las trabas encontradas en la resolución personal.

Viernes 22: Último encuentro entre los integrantes, para unir los archivos, compilar, modificar y corregir.

Lunes 25: Se realización en forma grupal la mayor cantidad de los reportes pedidos en el tp.

Martes 26/Jueves 5: Reuniones vía Skype para solucionar problemas de compilación, y errores en los reportes.

13. Cronograma de división de tareas

- **José Victor Ibañez:** Generar el análisis y diseño, creación de las listas, listados de jugadores y la salida del menú.
- **Nicolas Nahuel Trezza:** Creación de TDA, creación de las listas, sistema de activo de alerta (goleadores) y porcentaje de goles local visitante.
- **Martin Olmos:** Creación de leer los archivos txt, el orden por grupo y el fixture.
- **José Maximiliano Lucero:** Creación del sistema manager de partido, crear archivos, el menú del grupo de la muerte y el porcentaje de goles local visitante.

  Griselda Benítez Haugg: Creación de diseño del menú principal, diseño de ABM y validar los ingresos de datos.

14. Archivos de prueba

Jugadores.txt

1;David Silva;0;6
2;Eden Hazard;0;25
3;Frank Fabra;0;31
4;Keylor Navas;0;19
5;Sergio Ramos;0;6
6;Achraf Hakimi;0;7
7;Cristiano Ronaldo;4;5
8;Toni Kroos;0;21
10;Lionel Messi;0;13
11;Neymar Jr;0;17
12;Aleksander Kolarov;1;20
13;Paulo Dybala;0;13
14;Luis Ovalle;0;26
15;Bryan Ruiz;0;19
17;Denis Cheryshev;3;1
18;Fiodor Smolov;0;1
19;James Rodriguez;0;31
20;Omar Hawsawi;0;2
21;Paolo Guerrero;0;11
22;Mohammed Marzouq;0;2
23;Ali karimi;0;8
24;Amr Gamal;0;3
25;Andre Gomes;0;5
26;Kaveh Rezaei;0;8
27;Paul Pogba;0;9
28;Tim Cahill;0;10
29;Luka Modric;1;15
30;Dimitri Petratos;0;10
31;Jefferson Farfan;0;11
32;Yussuf Poulsen;1;12
33;Luis Suarez;1;4
34;Andreas Cornelius;0;12
35;Gylfi Sigurasson;0;14
36;Birkir Bjarnason;0;14
37;Ivan Rakitic;0;15
38;Victor Moses;0;16
39;Alex Iwobi;0;16
40;Xherdan Shaqiri;0;18
44;Mohamed Salah;1;3
45;Sadio Mane;0;30
46;Seung-woo Lee;0;24
47;Moussa Sow;0;30
48;Sebastian Larsson;0;23
51;Dele Alli;0;28
54;Medhi Benatia;0;7
55;Marcelo;0;17
56;Emil Forsberg;0;23
60;Robert Lewandowski;0;29
66;Mats Hummels;0;21
70;Kevin De Bruyne;0;25
72;Shinji Okazaki;0;32
76;Harry Kane;2;28
77;Granit Xhaxa;0;18
79;Saber Khalifa;0;27
80;Javier Hernandez;0;22
85;Mohamed Ben Amor;0;27

87;Carlos Vela;0;22
88;Edinson Cavani;0;4
91;Kylian Mbappe;0;9
93;Nemanja Matic;0;20
95;Jakub Blaszczykowski;0;29
98;Shinji Kagawa;1;32
99;Son Heugn-min;0;24
100;Gabriel Torres;0;26
101;Yuri Gazinski;1;1
102;Artiom Dziuba;2;1
103;Jose Gimenez;1;4
104;Aziz Bouhaddouz;1;8
105;Aleksandr Golovin;1;1
106;Diego Costa;3;6
107;Antoine Griezmann;1;9
108;Mile Jedinak;2;10
109;Aziz Behich;1;9
110;Ahmed Fathy;1;1
111;Nacho Iglesias;1;6
112;Sergio Aguero;1;13
113;Oghenekaro Etebo;1;15
114;Hirving Lozano;1;22
115;Philippe Coutinho;1;17
116;Andreas Granqvist;1;23
117;Dries Mertens;1;25
118;Steven Zuber;1;18
119;Romelu Lukaku;2;25
120;Alfreo Finnbogason;1;14
121;Ferjani Sassi;1;27
122;Juan Fernando Quintero;1;31
123;Yuya Osako;1;32
124;Thiago Rangel Cionek;1;30
125;M Baye Niang;1;30
126;Grzegorz Krychowiak;1;29
127;Christian Eriksen;1;12

Archivo Equipo.txt
1;Rusia;8;1;6
2;Arabia Saudita;0;6;0
3;Egipto;1;4;0
4;Uruguay;2;0;6
5;Portugal;4;3;4
6;Espana;4;3;4
7;Marruecos;0;2;0
8;Iran;1;1;3
9;Francia;2;1;3
10;Australia;2;3;1
11;Peru;0;1;0
12;Dinamarca;2;1;4
13;Argentina;1;1;1
14;Islandia;1;1;1
15;Croacia;2;0;3
16;Nigeria;0;2;0
17;Brasil;1;1;1
18;Suiza;1;1;1
19;Costa Rica;0;1;0
20;Serbia;1;0;3
21;Alemania;0;1;0
22;Mexico;1;0;3
23;Suecia;1;0;3
24;Corea del sur;0;1;0
25;Belgica;3;0;3
26;Panama;0;3;0
27;Tunez;1;2;0
28;Inglaterra;2;1;3
29;Polonia;1;2;0

30;Senegal;2;1;3
31;Colombia;1;2;0
32;Japon;2;1;3

Archivo grupos.txt

A;Grupo A;1;3;2;4
B;Grupo B;5;6;7;8
C;Grupo C;9;10;11;12
D;Grupo D;13;14;15;16
E;Grupo E;17;18;19;20
F;Grupo F;21;22;23;24
G;Grupo G;25;26;27;28
H;Grupo H;29;30;31;32

Archivo Partidos.txt

1;1;2;5;0
2;3;4;0;1
3;5;6;3;3
4;7;8;0;1
5;9;10;2;1
6;11;12;0;1
7;13;14;1;1
8;15;16;2;0
9;17;18;1;1
10;19;20;0;1
11;21;22;0;1
12;23;24;1;0
13;25;26;3;0
14;27;28;1;2
15;29;30;1;2
16;31;32;1;2
17;1;3;3;1
18;4;2;1;0
19;5;7;1;0
20;8;6;0;1
21;9;11;-1;-1
22;12;10;1;1
23;13;15;-1;-1
24;16;14;-1;-1
25;17;19;-1;-1
26;20;18;-1;-1
27;21;23;-1;-1
28;24;22;-1;-1
29;25;27;-1;-1
30;28;26;-1;-1
31;29;31;-1;-1
32;32;30;-1;-1
33;4;1;-1;-1
34;2;3;-1;-1
35;8;5;-1;-1
36;6;7;-1;-1
37;12;9;-1;-1
38;10;11;-1;-1
39;16;13;-1;-1
40;14;15;-1;-1
41;20;17;-1;-1
42;18;19;-1;-1
43;24;21;-1;-1
44;22;23;-1;-1
45;28;25;-1;-1
46;26;27;-1;-1
47;32;29;-1;-1
48;30;31;-1;-1
49;0;0;-1;-1
50;0;0;-1;-1
51;0;0;-1;-1
52;0;0;-1;-1
53;0;0;-1;-1
54;0;0;-1;-1
55;0;0;-1;-1
56;0;0;-1;-1
57;0;0;-1;-1
58;0;0;-1;-1
59;0;0;-1;-1
60;0;0;-1;-1
61;0;0;-1;-1
62;0;0;-1;-1
63;0;0;-1;-1
64;0;0;-1;-1

15. Programa

Equipo.h

```
#ifndef _EQUIPO_H_
#define _EQUIPO_H_
/**
   Axiomas:
   * golesAFavor se cuenta con los goles a favor de todos los
   partidos jugados
   * golesEnContra se cuenta con los goles en contra de todos los
   partidos jugados
   * puntos se cuenta por los partidos ganados solo de la Primera Fase
*/
typedef struct {
   int id;
   string nombre;
   int golesAFavor;
   int golesEnContra;
   int puntos;
   Lista *jugadores;
} Equipo;


/*************************************************************************
**/
/* Definicion de Primitivas */
/*-------------------------*/

/*
  pre : el equipo no debe haber sido creada.
  post: el equipo queda creada y preparada para ser usada con valores 0.
*/
void crear(Equipo &equipo);

/*-------------------------------------------------------------------------*/
/*
  pre: el equipo debe haberse creado y no destruido
  post: deja la instacia de la lista para ser usado. Los parametros se aplica al tda (equipo).
*/
void crear(Equipo &equipo, int id, string nombre, int golesAFavor, int golesEnContra, int
puntos);

/*-------------------------------------------------------------------------*/
/*
  pre: el equipo debe haberse creado y no destruido.
```

```
 post: destruye la instancia del tda y ya no podrá ser usado.
*/
void destruir(Equipo &equipo);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: asigna el valor de id a la instancia de equipo.
*/
void setId(Equipo &equipo, int id);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: devuelve el id.
*/
int getId(Equipo &equipo);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: asigna el valor de nombre a la instancia de equipo.
*/
void setNombre(Equipo &equipo, string nombre);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: devuelve el nombre.
*/
string getNombre(Equipo &equipo);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: asigna el valor de goles a favor a la instancia de equipo.
*/
void setGolesAFavor(Equipo &equipo, int golesAFavor);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: devuelve el goles a favor.
*/
int getGolesAFavor(Equipo &equipo);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
```

```
 post: asigna el valor de goles en contra a la instancia de equipo.
*/
void setGolesEnContra(Equipo &equipo, int golesEnContra);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: devuelve el goles en contra.
*/
int getGolesEnContra(Equipo &equipo);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: asigna el valor de puntos a la instancia de equipo.
*/
void setPuntos(Equipo &equipo, int puntos);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: devuelve el puntos.
*/
int getPuntos(Equipo &equipo);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: asigna el valor de jugadores a la instancia de equipo.
*/
void setJugadores(Equipo &equipo, Lista *jugadores);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: asigna el valor de jugadores.
*/
void setJugadores(Equipo &equipo, Jugador *jugador);

/*-------------------------------------------------------------------------*/
/*
 pre: el equipo debe haberse creado y no destruido.
 post: devuelve los jugadores.
*/
Lista* getJugadores(Equipo &equipo);

/*-------------------------------------------------------------------------*/
/*
 pre: la instancia del tda(equipo,e) debe haberse creado y no destruido.
```

```
  post: devuelve Verdadero o Falso si son iguales los equipos.
*/
bool equals(Equipo &equipo, Equipo e);

/*-------------------------------------------------------------------------*/
/*
  pre: la instancia del tda(equipo) debe haberse creado y no destruido.
  post: devuelve los datos del equipo.
*/
string toString(Equipo &equipo);


#endif // _EQUIPO_H_
```

Equipo.cpp

```
#include <iostream>
#include <sstream>
using namespace std;

#include "Lista.h"
#include "Jugador.h"
#include "Equipo.h"

ResultadoComparacion compararIdE(PtrDato ptrDato1, PtrDato ptrDato2) {
   if (getId(*(Equipo*)ptrDato1) > getId(*(Equipo*)ptrDato2))
      return MAYOR;

   else if (getId(*(Equipo*)ptrDato1) < getId(*(Equipo*)ptrDato2))
      return MENOR;

   else
      return IGUAL;
}

/*************************************************************************
********/
/* Implementación de Primitivas */
/*-----------------------------*/

/*-------------------------------------------------------------------------*/
void crear(Equipo &equipo) {
   equipo.id = 0;
   equipo.nombre = "";
   equipo.golesAFavor = 0;
   equipo.golesEnContra = 0;
   equipo.puntos = 0;
   equipo.jugadores = new Lista;
   crearLista(*equipo.jugadores, compararIdE);
}

/*-------------------------------------------------------------------------*/
void crear(Equipo &equipo, int id, string nombre, int golesAFavor, int golesEnContra,
int puntos) {
   equipo.id = id;
   equipo.nombre = nombre;
   equipo.golesAFavor = golesAFavor;
   equipo.golesEnContra = golesEnContra;
```

```
   equipo.puntos = puntos;
   equipo.jugadores = new Lista;
   crearLista(*equipo.jugadores, compararIdE);
}

/*-------------------------------------------------------------------------*/
void destruir(Equipo &equipo) {
   equipo.id = 0;
   equipo.nombre = "";
   equipo.golesAFavor = 0;
   equipo.golesEnContra = 0;
   equipo.puntos = 0;
   eliminarLista(*equipo.jugadores);
}

/*-------------------------------------------------------------------------*/
void setId(Equipo &equipo, int id) {
   equipo.id = id;
}

/*-------------------------------------------------------------------------*/
int getId(Equipo &equipo) {
   return equipo.id;
}

/*-------------------------------------------------------------------------*/
void setNombre(Equipo &equipo, string nombre) {
   equipo.nombre = nombre;
}

/*-------------------------------------------------------------------------*/
string getNombre(Equipo &equipo) {
   return equipo.nombre;
}

/*-------------------------------------------------------------------------*/
void setGolesAFavor(Equipo &equipo, int golesAFavor) {
   equipo.golesAFavor = golesAFavor;
}

/*-------------------------------------------------------------------------*/
int getGolesAFavor(Equipo &equipo) {
   return equipo.golesAFavor;
}
```

```
/*-------------------------------------------------------------------------*/
void setGolesEnContra(Equipo &equipo, int golesEnContra) {
   equipo.golesEnContra = golesEnContra;
}

/*-------------------------------------------------------------------------*/
int getGolesEnContra(Equipo &equipo) {
   return equipo.golesEnContra;
}

/*-------------------------------------------------------------------------*/
void setPuntos(Equipo &equipo, int puntos) {
   equipo.puntos = puntos;
}

/*-------------------------------------------------------------------------*/
int getPuntos(Equipo &equipo) {
   return equipo.puntos;
}

/*-------------------------------------------------------------------------*/
void setJugadores(Equipo &equipo, Lista *jugadores) {
   equipo.jugadores = jugadores;
}

/*-------------------------------------------------------------------------*/
void setJugadores(Equipo &equipo, Jugador *jugador) {
   adicionarFinal(*equipo.jugadores, jugador);
}

/*-------------------------------------------------------------------------*/
Lista* getJugadores(Equipo &equipo) {
   return equipo.jugadores;
}

/*-------------------------------------------------------------------------*/
bool equals(Equipo &equipo, Equipo e) {
   return equipo.id == e.id;
}

/*-------------------------------------------------------------------------*/
string toString(Equipo &equipo) {
   string dato="NULL\n";
   if(getId(equipo)>0){
      dato = "Id: ";
```

```
    ostringstream convert;
    convert<<getId(equipo);
    dato+=convert.str()+"\n";
    dato+="Nombre: "+getNombre(equipo)+"\n";
    ostringstream convert2;
    convert2<<getGolesAFavor(equipo);
    dato+="Goles a favor: "+convert2.str()+"\n";
    ostringstream convert3;
    convert3<<getGolesEnContra(equipo);
    dato+="Goles en contra: "+convert3.str()+"\n";
    ostringstream convert4;
    convert4<<getPuntos(equipo);
    dato+="Puntos: "+convert4.str()+"\n";
    PtrNodoLista cursor = primero(*equipo.jugadores);
    while (cursor != fin()) {
      dato+=getNombre(*(Jugador*)cursor->ptrDato)+" || ";
      cursor = siguiente(*equipo.jugadores, cursor);
    }
  }
  return dato;
}
```

Grupo.h

```
#ifndef _GRUPO_H_
#define _GRUPO_H_

typedef struct {
   char id;
   string nombre;
   Lista *equipos;
} Grupo;

/*************************************************************************
*********/
/* Definicion de Primitivas */
/*-------------------------*/

/*
  pre : el grupo no debe haber sido creada.
  post: el grupo queda creada y preparada para ser usada con valores 0.
*/
void crear(Grupo &grupo);

/*-------------------------------------------------------------------------*/
/*
  pre: el grupo debe haberse creado y no destruido
  post: deja la instacia de la lista para ser usado. Los parametros se aplica al tda
(grupo).
*/
void crear(Grupo &grupo, char id, string nombre, Lista *equipos);

/*-------------------------------------------------------------------------*/
/*
  pre: el grupo debe haberse creado y no destruido.
  post: destruye la instancia del tda y ya no podrá ser usado.
*/
void destruir(Grupo &grupo);

/*-------------------------------------------------------------------------*/
/*
  pre: el grupo debe haberse creado y no destruido.
  post: asigna el valor de id a la instancia de grupo.
*/
```

```
void setId(Grupo &grupo, char id);

/*-------------------------------------------------------------------------*/
/*
  pre: el grupo debe haberse creado y no destruido.
  post: devuelve el id.
*/
char getId(Grupo &grupo);

/*-------------------------------------------------------------------------*/
/*
  pre: el grupo debe haberse creado y no destruido.
  post: asigna el valor de nombre a la instancia de grupo.
*/
void setNombre(Grupo &grupo, string nombre);

/*-------------------------------------------------------------------------*/
/*
  pre: el grupo debe haberse creado y no destruido.
  post: devuelve el nombre.
 */
string getNombre(Grupo &grupo);

/*-------------------------------------------------------------------------*/
/*
  pre: el grupo debe haberse creado y no destruido.
  post: asigna el valor de equipos a la instancia de grupo.
*/
void setEquipos(Grupo &grupo, Lista *equipos);

/*-------------------------------------------------------------------------*/
/*
  pre: el grupo debe haberse creado y no destruido.
  post: devuelve los equipos.
*/
Lista* getEquipos(Grupo &grupo);

/*-------------------------------------------------------------------------*/
/*
  pre: la instancia del tda(grupo,g) debe haberse creado y no destruido.
  post: devuelve Verdadero o Falso si son iguales los grupos.
*/
bool equals(Grupo &grupo, Grupo g);

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

```
/*
  pre: la instancia del tda(grupo) debe haberse creado y no destruido.
  post: devuelve los datos del grupo.
*/
string toString(Grupo &grupo);

/*-------------------------------------------------------------------------*/
/*
  pre: la instancia del tda(grupo) debe haberse creado y no destruido.
  post: devuelve los goles totales del grupo.
*/
int golesPorGrupo(Grupo &grupo);

#endif // _GRUPO_H_
```

Grupo.cpp

```
#include <iostream>
#include <sstream>
using namespace std;

#include "Lista.h"
#include "Jugador.h"
#include "Equipo.h"
#include "Grupo.h"

ResultadoComparacion compararPuntos(PtrDato ptrDato1, PtrDato ptrDato2) {
   if (getPuntos(*(Equipo*)ptrDato1) < getPuntos(*(Equipo*)ptrDato2))
      return MAYOR;
   else if (getPuntos(*(Equipo*)ptrDato1) > getPuntos(*(Equipo*)ptrDato2))
      return MENOR;
   else {
      if (getGolesAFavor(*(Equipo*)ptrDato1) -  getGolesEnContra(*(Equipo*)ptrDato1)
         < getGolesAFavor(*(Equipo*)ptrDato2) -
getGolesEnContra(*(Equipo*)ptrDato2))
         return MAYOR;
      else if (getGolesAFavor(*(Equipo*)ptrDato1) -
getGolesEnContra(*(Equipo*)ptrDato1)
         > getGolesAFavor(*(Equipo*)ptrDato2) -
getGolesEnContra(*(Equipo*)ptrDato2))
         return MENOR;
      else {
         if (getGolesAFavor(*(Equipo*)ptrDato1) < getGolesAFavor(*(Equipo*)ptrDato2))
            return MAYOR;

         else if (getGolesAFavor(*(Equipo*)ptrDato1) >
getGolesAFavor(*(Equipo*)ptrDato2))
            return MENOR;

         else
            return IGUAL;
      }
   }
}

/*****************************************************************
********/
/* Implementación de Primitivas */
/*-----------------------------*/
```

```
/*-------------------------------------------------------------------------*/
void crear(Grupo &grupo) {
   grupo.id = '0';
   grupo.nombre = "";
   grupo.equipos = new Lista;
   crearLista(*grupo.equipos, compararPuntos);
}

/*-------------------------------------------------------------------------*/
void crear(Grupo &grupo, char id, string nombre, Lista *equipos) {
   grupo.id = id;
   grupo.nombre = nombre;
   grupo.equipos = equipos;
   grupo.equipos->compara = compararPuntos;
}

/*-------------------------------------------------------------------------*/
void destruir(Grupo &grupo) {
   grupo.id = '0';
   grupo.nombre = "";
   eliminarLista(*grupo.equipos);
}

/*-------------------------------------------------------------------------*/
void setId(Grupo &grupo, char id) {
   grupo.id = id;
}

/*-------------------------------------------------------------------------*/
char getId(Grupo &grupo) {
   return grupo.id;
}

/*-------------------------------------------------------------------------*/
void setNombre(Grupo &grupo, string nombre) {
   grupo.nombre = nombre;
}

/*-------------------------------------------------------------------------*/
string getNombre(Grupo &grupo) {
   return grupo.nombre;
}

/*-------------------------------------------------------------------------*/
void setEquipos(Grupo &grupo, Lista *equipos) {
```

```
   grupo.equipos = equipos;
}

/*----------------------------------------------------------------------*/
Lista* getEquipos(Grupo &grupo) {
   return grupo.equipos;
}

/*----------------------------------------------------------------------*/
bool equals(Grupo &grupo, Grupo g) {
   return grupo.id == g.id;
}

/*----------------------------------------------------------------------*/
string toString(Grupo &grupo) {
   string dato="NULL\n";
   if(getId(grupo)>0){
      ostringstream convertId;
      convertId << getId(grupo);
      dato="Id: "+convertId.str()+" ";
      dato+="Nombre: "+getNombre(grupo)+"\n";
      PtrNodoLista cursor = primero(*grupo.equipos);
      while (cursor != fin()) {
         dato+=getNombre(*(Equipo*)cursor->ptrDato);
         cursor = siguiente(*grupo.equipos, cursor);
         if(cursor != fin())dato+=" - ";
      }
   }
   return dato;
}

/*----------------------------------------------------------------------*/
int golesPorGrupo(Grupo &grupo) {
   PtrNodoLista cursor = primero(*grupo.equipos);
   int golesDelGrupo = 0;

   while (cursor != fin()) {
      golesDelGrupo += getGolesAFavor(*(Equipo*)cursor->ptrDato);
      cursor = siguiente(*grupo.equipos, cursor);
   }

   return golesDelGrupo;
}
```

Jugador.h

```
#ifndef _JUGADOR_H_
#define _JUGADOR_H_
/**
   Axiomas:
   * Los goles del jugador se validan con los GolesAFavor
   del Equipo
*/
typedef struct {
   int id;
   string nombre;
   int goles;
} Jugador;

/************************************************************************
********/
/* Definicion de Primitivas */
/*-------------------------*/

/*
  pre : el jugador no debe haber sido creada.
  post: el jugador queda creada y preparada para ser usada con valores 0.
*/
void crear(Jugador &jugador);

/*-------------------------------------------------------------------------*/
/*
  pre: el jugador debe haberse creado y no destruido
  post: deja la instacia de la lista para ser usado. Los parametros se aplica al tda
(jugador).
*/
void crear(Jugador &jugador, int id, string nombre, int goles);

/*-------------------------------------------------------------------------*/
/*
  pre: el jugador debe haberse creado y no destruido.
  post: destruye la instancia del tda y ya no podrá ser usado.
*/
void destruir(Jugador &jugador);

/*-------------------------------------------------------------------------*/
/*
  pre: el jugador debe haberse creado y no destruido.
```

```
  post: asigna el valor de id a la instancia de jugador.
*/
void setId(Jugador &jugador, int id);

/*-------------------------------------------------------------------------*/
/*
  pre: el jugador debe haberse creado y no destruido.
  post: devuelve el id.
*/
int getId(Jugador &jugador);

/*-------------------------------------------------------------------------*/
/*
  pre: el jugador debe haberse creado y no destruido.
  post: asigna el valor de nombre a la instancia de jugador.
*/
void setNombre(Jugador &jugador, string nombre);

/*-------------------------------------------------------------------------*/
/*
  pre: el jugador debe haberse creado y no destruido.
  post: devuelve el nombre.
*/
string getNombre(Jugador &jugador);

/*-------------------------------------------------------------------------*/
/*
  pre: el jugador debe haberse creado y no destruido.
  post: asigna el valor de goles a la instancia de jugador.
*/
void setGoles(Jugador &jugador, int goles);

/*-------------------------------------------------------------------------*/
/*
  pre: el jugador debe haberse creado y no destruido.
  post: devuelve los goles.
*/
int getGoles(Jugador &jugador);

/*-------------------------------------------------------------------------*/
/*
  pre: la instancia del tda(jugador,j) debe haberse creado y no destruido.
  post: devuelve Verdadero o Falso si son iguales los jugador.
*/
bool equals(Jugador &jugador, Jugador j);
```

```
/*-------------------------------------------------------------------------*/
/*
  pre: la instancia del tda(jugador) debe haberse creado y no destruido.
  post: devuelve los datos del jugador.
*/
string toString(Jugador &jugador);

#endif // _JUGADOR_H_
```

Jugador.cpp

```
#include <iostream>
#include <sstream>
using namespace std;

#include "Jugador.h"

/*************************************************************************
********/
/* Implementación de Primitivas */
/*-----------------------------*/

/*-------------------------------------------------------------------------*/
void crear(Jugador &jugador) {
   jugador.id = 0;
   jugador.nombre = "";
   jugador.goles = 0;
}

/*-------------------------------------------------------------------------*/
void crear(Jugador &jugador, int id, string nombre, int goles) {
   jugador.id = id;
   jugador.nombre = nombre;
   jugador.goles = goles;
}

/*-------------------------------------------------------------------------*/
void destruir(Jugador &jugador) {
   jugador.id = 0;
   jugador.nombre = "";
   jugador.goles = 0;
}

/*-------------------------------------------------------------------------*/
void setId(Jugador &jugador, int id) {
   jugador.id = id;
}

/*-------------------------------------------------------------------------*/
int getId(Jugador &jugador) {
   return jugador.id;
}
```

```
/*-------------------------------------------------------------------------*/
void setNombre(Jugador &jugador, string nombre) {
   jugador.nombre = nombre;
}

/*-------------------------------------------------------------------------*/
string getNombre(Jugador &jugador) {
   return jugador.nombre;
}

/*-------------------------------------------------------------------------*/
void setGoles(Jugador &jugador, int goles) {
   jugador.goles = goles;
}

/*-------------------------------------------------------------------------*/
int getGoles(Jugador &jugador) {
   return jugador.goles;
}

/*-------------------------------------------------------------------------*/
bool equals(Jugador &jugador, Jugador j) {
   return jugador.id == j.id;
}

/*-------------------------------------------------------------------------*/
string toString(Jugador &jugador) {
   string dato="NULL\n";
   if(getId(jugador)>0){
      dato="Id: ";
      ostringstream convert;
      convert<<getId(jugador);
      dato+=convert.str()+"\n";
      dato+="Nombre: "+getNombre(jugador)+"\n";
      ostringstream convert2 ;
      convert2<<getGoles(jugador);
      dato+="Goles: "+convert2.str()+"\n";
   }
   return dato;
}
```

Partido.h

```
#ifndef _PARTIDO_H_
#define _PARTIDO_H_
/**
   Axiomas:
   * Cuando el partido tiene sus dos equipos se considera
   las victorias anteriores correspondientes aceptadas.
   * En el caso de los partidos que aún no se hayan jugado,
   los goles se registrarán con el valor “-1”.

*/
enum Estado {SIN_COMENZAR, EN_JUEGO, FINALIZADO};

typedef struct {
   int id;
   Equipo *equipoL;
   Equipo *equipoV;
   int golesL;
   int golesV;
   Estado estado;
} Partido;

/*****************************************************************************
********/
/* Definicion de Primitivas */
/*-------------------------*/

/*
  pre : el partido no debe haber sido creada.
  post: el partido queda creada y preparada para ser usada con valores 0.
*/
void crear(Partido &partido);

/*-----------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido
  post: deja la instacia de la lista para ser usado. Los parametros se aplica al tda
(partido).
*/
void crear(Partido &partido, int id, Equipo *equipoL, Equipo *equipoV, int golesL, int
golesV);
```

```
/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: destruye la instancia del tda y ya no podrá ser usado.
*/
void destruir(Partido &partido);


/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: asigna el valor de id a la instancia de partido.
*/
void setId(Partido &partido, int id);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: devuelve el id.
*/
int getId(Partido &partido);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: asigna el valor de equipo local a la instancia de partido.
*/
void setEquipoL(Partido &partido, Equipo *equipoL);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: devuelve el equipo local.
*/
Equipo* getEquipoL(Partido &partido);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: asigna el valor de equipo visitante a la instancia de partido.
*/
void setEquipoV(Partido &partido, Equipo *equipoV);

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

```
  pre: el partido debe haberse creado y no destruido.
  post: devuelve el equipo visitante.
*/
Equipo* getEquipoV(Partido &partido);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: asigna el valor de goles local a la instancia de partido.
*/
void setGolesL(Partido &partido, int golesL);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: devuelve los goles local.
*/
int getGolesL(Partido &partido);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: asigna el valor de goles visitante a la instancia de partido.
*/
void setGolesV(Partido &partido, int golesV);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: devuelve los goles visitante.
*/
int getGolesV(Partido &partido);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: asigna el valor de estado a la instancia de partido.
*/
void setEstado(Partido &partido, Estado estado);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: devuelve el estado.
*/
```

```
Estado getEstado(Partido &partido);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: devuelve Verdadero o Falso si son iguales los partidos.
*/
bool equals(Partido &partido, Partido p);

/*-------------------------------------------------------------------------*/
/*
  pre: el partido debe haberse creado y no destruido.
  post: devuelve los datos del partido.
*/
string toString(Partido &partido);


#endif // _PARTIDO_H_
```

Partido.cpp

```
#include <iostream>
#include <sstream>
using namespace std;

#include "Lista.h"
#include "Jugador.h"
#include "Equipo.h"
#include "Partido.h"

/*************************************************************************
********/
/* Implementación de Primitivas */
/*-----------------------------*/

/*-------------------------------------------------------------------------*/
void crear(Partido &partido) {
   partido.id = 0;
   crear(*partido.equipoL);
   crear(*partido.equipoV);
   partido.golesL = 0;
   partido.golesV = 0;
   partido.estado = SIN_COMENZAR;
}

/*-------------------------------------------------------------------------*/
void crear(Partido &partido, int id, Equipo *equipoL, Equipo *equipoV, int golesL, int
golesV) {
   partido.id = id;
   partido.equipoL = equipoL;
   partido.equipoV = equipoV;
   partido.golesL = golesL;
   partido.golesV = golesV;
   partido.estado = SIN_COMENZAR;
}

/*-------------------------------------------------------------------------*/
void destruir(Partido &partido) {
   partido.id = 0;
   destruir(*partido.equipoL);
   destruir(*partido.equipoV);
   partido.golesL = 0;
   partido.golesV = 0;
```

```
   partido.estado = FINALIZADO;
}

/*-------------------------------------------------------------------------*/
void setId(Partido &partido, int id) {
   partido.id = id;
}

/*-------------------------------------------------------------------------*/
int getId(Partido &partido) {
   return partido.id;
}

/*-------------------------------------------------------------------------*/
void setEquipoL(Partido &partido, Equipo *equipoL) {
   partido.equipoL = equipoL;
}

/*-------------------------------------------------------------------------*/
Equipo* getEquipoL(Partido &partido) {
   return partido.equipoL;
}

/*-------------------------------------------------------------------------*/
void setEquipoV(Partido &partido, Equipo *equipoV) {
   partido.equipoV = equipoV;
}

/*-------------------------------------------------------------------------*/
Equipo* getEquipoV(Partido &partido) {
   return partido.equipoV;
}

/*-------------------------------------------------------------------------*/
void setGolesL(Partido &partido, int golesL) {
   partido.golesL = golesL;
}

/*-------------------------------------------------------------------------*/
int getGolesL(Partido &partido) {
   return partido.golesL;
}

/*-------------------------------------------------------------------------*/
void setGolesV(Partido &partido, int golesV) {
```

```
   partido.golesV = golesV;
}

/*-------------------------------------------------------------------------*/
int getGolesV(Partido &partido) {
   return partido.golesV;
}

/*-------------------------------------------------------------------------*/
void setEstado(Partido &partido, Estado estado) {
   partido.estado = estado;
}

/*-------------------------------------------------------------------------*/
Estado getEstado(Partido &partido) {
   return partido.estado;
}

/*-------------------------------------------------------------------------*/
bool equals(Partido &partido, Partido p) {
   return partido.id == p.id;
}

/*-------------------------------------------------------------------------*/
string toString(Partido &partido) {
   string dato="NULL\n";
   if(getId(partido)>0){
      ostringstream convert;
      convert << getId(partido);
      dato="Id: "+convert.str()+"\n";
      dato+=getNombre(*partido.equipoL)+" ";
      ostringstream convert2;
      convert2 << getGolesL(partido);
      dato+=convert2.str()+"\n";
      dato+=getNombre(*partido.equipoV)+" ";
      ostringstream convert3;
      convert3 << getGolesV(partido);
      dato+=convert3.str()+"\n";
   }
   return dato;
}
```

Main.cpp

```
#include <iostream>
using namespace std;
#include <stdlib.h>
#include "Lista.h"
#include "Jugador.h"
#include "Equipo.h"
#include "Grupo.h"
#include "Partido.h"
#include "Sistema.h"

void admPartidos(Sistema &sistema) {
   bool seguir = true;
   int opcion;
   system("cls");
   while(seguir) {
      setearFases(sistema);
      cout << "1. Registrar inicio de partido." << endl;
      cout << "2. Registrar goles de partido." << endl;
      cout << "3. Registrar fin de partido." << endl;
      cout << "4. Mostrar partidos en curso." << endl;
      cout << "5. Volver." << endl;
      cout << "Opcion: ";
      cin >> opcion;

      switch(opcion) {
         case 1: inicioPartido(sistema); break;
         case 2: golesPartido(sistema); break;
         case 3: finPartido(sistema); break;
         case 4: mostrarPartidosEnCurso(sistema); break;
         case 5: seguir = false; break;
         default: cout<<"opcion incorrecta"<<endl; break;
      }
      system("pause");
      system("cls");
   }
}

void reportes(Sistema &sistema) {
   bool seguir = true;
   int opcion;
   system("cls");
   while(seguir) {
      cout << "1. Mostrar goleadores." << endl;
```

```
        cout << "2. Mostrar fixture." << endl;
        cout << "3. Mostrar grupos." << endl;
        cout << "4. Grupo de la muerte." << endl;
        cout << "5. Porcentaje de goles locales y visitantes." << endl;
        cout << "6. Volver." << endl;
        cout << "Opcion: ";
        cin >> opcion;

        switch(opcion) {
            case 1: goleadores(sistema); break;
            case 2: mostrarPartidos(sistema); break;
            case 3: mostrarTablaPosiciones(sistema); break;
            case 4: grupoDeLaMuerte(sistema); break;
            case 5: porcentajeGoles(sistema); break;
            case 6: seguir = false; break;
            default: cout<<"opcion incorrecta"<<endl; break;
        }
        system("pause");
        system("cls");
    }
}

void menu(Sistema &sistema) {
    bool seguir = true;
    int opcion;

    while(seguir) {
        cout << "1. Mostrar equipos." << endl;
        cout << "2. Administrar partidos." << endl;
        cout << "3. Procesar reportes." << endl;
        cout << "4. Salir." << endl;
        cout << "Opcion: ";
        cin >> opcion;

        switch(opcion) {
            case 1: mostrarEquipos(sistema); system("pause"); break;
            case 2: admPartidos(sistema); break;
            case 3: reportes(sistema); break;
            case 4: seguir = false; break;
            default: cout<<"opcion incorrecta"<<endl; break;
        }
        system("cls");
    }
}
```

```
int main() {
   Sistema *sistema = new Sistema;
   crear(*sistema);

   levantarEquipos(*sistema);
   levantarJugadores(*sistema);
   levantarGrupos(*sistema);
   levantarPartidos(*sistema);

   if(validar(*sistema)){
   menu(*sistema);
   bajarEquipos(*sistema);
   bajarJugadores(*sistema);
   bajarGrupos(*sistema);
   bajarPartidos(*sistema);
   }
   destruir(*sistema);
   delete sistema;

   return 0;
}
```

Sistema.h

```
#ifndef _SISTEMA_H_
#define _SISTEMA_H_
/**
   Axiomas:
   * No se podrá usar el programa hasta que los datos ingresados
   estén validados correctamente.
   * Se considera de mayor prioridad los TDA Partido
   para las "validaciones".
   * Se debe completar la fase de partidos en estado finalizados
   para que se validen la siguiente etapa.
   * Se debe finalizar el partido en curso en la opción
   Administrar partidos, para una correcta carga de puntos para los Equipos.
   * Los golesAFavor del Equipo se validan con la suma de
   goles de sus jugadores
*/
typedef struct {
   Lista *equipos;
   Lista *jugadores;
   Lista *grupos;
   Lista *partidos;
} Sistema;

void crear(Sistema &sistema);
void destruir(Sistema &sistema);
void levantarEquipos(Sistema &sistema);
void levantarJugadores(Sistema &sistema);
void levantarGrupos(Sistema &sistema);
void levantarPartidos(Sistema &sistema);
void mostrarEquipos(Sistema &sistema);
void mostrarJugadores(Sistema &sistema);
void mostrarGrupos(Sistema &sistema);
void mostrarPartidos(Sistema &sistema);
void goleadores(Sistema &sistema);
void bajarEquipos(Sistema &sistema);
void bajarJugadores(Sistema &sistema);
void bajarGrupos(Sistema &sistema);
void bajarPartidos(Sistema &sistema);
Equipo* traerEquipoPorJugador(Sistema &sistema, int id);
Equipo* traerEquipo(Sistema &sistema, int id);
Jugador* traerJugador(Sistema &sistema, int id);
Grupo* traerGrupo(Sistema &sistema, char id);
Partido* traerPartido(Sistema &sistema, int id);
```

```
void inicioPartido(Sistema &sistema);
void golesPartido(Sistema &sistema);
void finPartido(Sistema &sistema);
void mostrarPartidosEnCurso(Sistema &sistema);
void porcentajeGoles(Sistema &sistema);
void grupoDeLaMuerte(Sistema &sistema);
bool validar(Sistema& sistema);
void validarEquipo(Lista* equipos,string& warning);
void validarGoles(Lista* equipos,Lista* partidos,string& warning);
void validarEmpates(Lista* partidos,string& warning);
void validarJugadores(Lista* equipos,string& warning);
void validarPuntos(Lista* equipos,Lista* partidos,string& warning);
void validarPartidosFaseInicial(Lista* grupos,Lista* partidos,string& warning);
void validarFaseClasificion(Sistema& sistema,string& warning);
void validarClasificacionOctavos(Sistema& s,string& warning);
void validarClasificacionCuartos(Sistema& s,string& warning);
void verificarOctavosDeFinal(Sistema& s,string& warning);
void validarPartidosFaseFinal(Lista* partidos,Sistema& sistema,string& warning);
void validarPartidoCerrados(Sistema& s,string& warning);
bool verificarGrupo(Lista* equipos,Equipo* equipo);
bool verificarContinuidadEquipo(Lista* partidos,PtrNodoLista cursor,Equipo* equipo);
bool verificarPartido(Sistema& sistema,Equipo* equipo,int id);
bool verificarPartidos(Lista* partidos,int i,int f);
bool verificarVictoriaDifGoles(Equipo& e,Lista* equipos,Lista* partidos);
bool verificarVictoriaPuntos(Equipo& e,Lista* equipos);
bool localizar(Equipo* equipo,Sistema& s,int i,int f);
bool isPartidoCreado(Partido* p);
bool isPartidoJugado(Partido* p);
bool isEmpate(Partido* p);
void mostrarTablaPosiciones(Sistema &sistema);
int diferenciaDeGoles(Lista* partidos,Equipo& e,int i,int f);
int traerGolesAFavor(Lista* partidos,Equipo& e,int i,int f);
int traerGolesEnContra(Lista* partidos,Equipo& e,int i,int f);
void ordenarEquiposDeGrupos(Lista& grupos);
Equipo* traerGanador(Partido* p);
Equipo* traerPerdedor(Partido* p);
Equipo** traerGanadores(Lista* equipos,Lista* partidos);
void setearFases(Sistema &sistema);

#endif // _SISTEMA_H_
```

Sistema.CPP

```
#include <fstream>
#include <cstring>
#include <cstdlib>
#include <windows.h>
#include <stdexcept>
#include <sstream>
#include <iostream>
using namespace std;
#include "Lista.h"
#include "Jugador.h"
#include "Equipo.h"
#include "Grupo.h"
#include "Partido.h"
#include "Sistema.h"

ResultadoComparacion compararGoles(PtrDato ptrDato1, PtrDato ptrDato2) {
   if (getGoles(*(Jugador*)ptrDato1) < getGoles(*(Jugador*)ptrDato2))
      return MAYOR;

   else if (getGoles(*(Jugador*)ptrDato1) > getGoles(*(Jugador*)ptrDato2))
      return MENOR;

   else
      return IGUAL;
}

/*-------------------------------------------------------------------------*/
ResultadoComparacion compararEquipo(PtrDato ptrDato1, PtrDato ptrDato2) {
   if (getId(*(Equipo*)ptrDato1) > getId(*(Equipo*)ptrDato2))
      return MAYOR;

   else if (getId(*(Equipo*)ptrDato1) < getId(*(Equipo*)ptrDato2))
      return MENOR;

   else
      return IGUAL;
}

/*-------------------------------------------------------------------------*/
ResultadoComparacion compararJugador(PtrDato ptrDato1, PtrDato ptrDato2) {
   if (getId(*(Jugador*)ptrDato1) > getId(*(Jugador*)ptrDato2))
      return MAYOR;
```

```
   else if (getId(*(Jugador*)ptrDato1) < getId(*(Jugador*)ptrDato2))
      return MENOR;

   else
      return IGUAL;
}

/*-------------------------------------------------------------------------*/
ResultadoComparacion compararGrupo(PtrDato ptrDato1, PtrDato ptrDato2) {
   if (getId(*(Grupo*)ptrDato1) > getId(*(Grupo*)ptrDato2))
      return MAYOR;

   else if (getId(*(Grupo*)ptrDato1) < getId(*(Grupo*)ptrDato2))
      return MENOR;

   else
      return IGUAL;
}

/*-------------------------------------------------------------------------*/
ResultadoComparacion compararPartido(PtrDato ptrDato1, PtrDato ptrDato2) {
   if (getId(*(Partido*)ptrDato1) > getId(*(Partido*)ptrDato2))
      return MAYOR;

   else if (getId(*(Partido*)ptrDato1) < getId(*(Partido*)ptrDato2))
      return MENOR;

   else
      return IGUAL;
}

/*-------------------------------------------------------------------------*/
void crear(Sistema &sistema) {
   sistema.equipos = new Lista;
   crearLista(*sistema.equipos, compararEquipo);
   sistema.jugadores = new Lista;
   crearLista(*sistema.jugadores, compararJugador);
   sistema.grupos = new Lista;
   crearLista(*sistema.grupos, compararGrupo);
   sistema.partidos = new Lista;
   crearLista(*sistema.partidos, compararPartido);
}

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

```
void destruir(Sistema &sistema) {
   eliminarLista(*sistema.equipos);
   eliminarLista(*sistema.jugadores);
   eliminarLista(*sistema.grupos);
   eliminarLista(*sistema.partidos);
}

/*-------------------------------------------------------------------------*/
void levantarEquipos(Sistema &sistema) {
   ifstream archivo;

   archivo.open("archivos/equipos.txt", ios::in);

   if(archivo.fail()) {
      cout << "Error al abrir el archivo" << endl;
      exit(1);
   }

   string id, nombre, golesAFavor, golesEnContra, puntos;

   bool flag=false;
   int vectorValidar[32];
   int i = 0;
   string letras = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

   while (!archivo.eof()) {
      getline(archivo, id, ';');
      getline(archivo, nombre, ';');
      getline(archivo, golesAFavor, ';');
      getline(archivo, golesEnContra, ';');
      getline(archivo, puntos);

      Equipo *e = new Equipo;

      /**Valido si el id de equipo tiene una letra, si es asi el programa no funciona */
      bool errorLetra = false;
      if(atoi(id.c_str())==atoi(letras.c_str())){
         cout<<"-----------------------------------------------"<<endl;
         cout << "Error el equipo en el id tiene una letra, deberia tener numero"<< endl;
         cout << "No se puede cargar el equipo:" << nombre<< endl;
         //si entro aca hago el id es positivo para que no entre al otro error
         errorLetra = true;
         flag=true;
      }
```

```
    /**Valido si el id de equipo no tiene una letra y tiene un id negativo, si es asi el
programa no funciona */
    if (!errorLetra){
    if(atoi(id.c_str())<=0){
        cout<<"-----------------------------------------------"<<endl;
        cout << "Error el equipo tiene el id negativo"<< endl;
        cout << "No se puede cargar el equipo:" << nombre<< endl;
        flag=true;
      }
    }

    /**Valido si el id de equipo tiene mas de 32 equipos, si es asi el programa no
funciona */
    if(atoi(id.c_str())>32) {
      cout<<"-----------------------------------------------"<<endl;
      cout << "Error el id del equipo no debe ser mayor de 32"<< endl;
      flag=true;
    }

    /**Valido si el id de equipo esta repetido, si es asi el programa no funciona */
    for (int indice=0; indice<i; indice++){
      if (vectorValidar[indice]==atoi(id.c_str())){
        cout<<"-----------------------------------------------"<<endl;
        cout << "Error el id:  " << atoi(id.c_str()) << " esta repetido"<<endl;
        cout << "No se puede cargar el equipo:" << nombre<< endl;
        flag=true;
      }
    }

    /**Valido si los goles a favor o goles en contra no este en un valor negativo, si es
asi el programa no funciona */
    if(atoi(golesAFavor.c_str())<0 || atoi(golesEnContra.c_str())<0){
      cout<<"-----------------------------------------------"<<endl;
      cout<<"No se puede ingresar los goles con un valor negativo"<<endl;
      cout<<"Revise los goles de "<<nombre<<endl;
      flag=true;
    }

    /**Valido si los puntos no sean menor que cero o mayor que 9, si es asi el
programa no funciona */
    if(atoi(puntos.c_str())>9 || atoi(puntos.c_str())<0 || atoi(puntos.c_str())==8){
      cout<<"-----------------------------------------------"<<endl;
      cout<<"Los puntos ingresados no son validos"<<endl;
      cout<<"Revise los puntos de "<<nombre<<endl;
      flag=true;
```

```
      }

      vectorValidar[i] = atoi(id.c_str());
      i = i +1;
      crear(*e, atoi(id.c_str()), nombre, atoi(golesAFavor.c_str()),
atoi(golesEnContra.c_str()), atoi(puntos.c_str()));
      adicionarFinal(*sistema.equipos, e);
   }

   if(flag==true){
      exit(1);
   }

   reordenar(*sistema.equipos);

   archivo.close();
}

/*-------------------------------------------------------------------------*/
void levantarJugadores(Sistema &sistema) {
   ifstream archivo;

   archivo.open("archivos/jugadores.txt", ios::in);

   if(archivo.fail()) {
      cout << "Error al abrir el archivo" << endl;
      exit(1);
   }

   string id, nombre, goles, equipo;

   int vectorValidar[100];
   int i = 0;
   bool flag=false;
   bool aux=false;
   string letras = "abcdefghijklmnopqrstuvwxyzABECDEFGHIJKLMNOPQRSTVWXYZ";

   while (!archivo.eof()) {
      getline(archivo, id, ';');
      getline(archivo, nombre, ';');
      getline(archivo, goles, ';');
      getline(archivo, equipo);

      Jugador *j = new Jugador;
```

```
      /**Valido si el id de jugador tiene letra, si es asi el programa no funciona */
      bool errorLetra = false;
      if(atoi(id.c_str())==atoi(letras.c_str())){
         cout<<"-----------------------------------------------"<<endl;
         cout << "Error un jugador en el id tiene una letra, deberia tener numero"<<
endl;
         cout << "No se puede cargar el jugador:" << nombre<< endl;
         errorLetra = true;
         flag=true;
      }

      /**Valido si el id de jugador tiene un valor negativo, si es asi el programa no
funciona */
      if (!errorLetra){
      if(atoi(id.c_str())<=0){
            cout<<"-----------------------------------------------"<<endl;
            cout << "Error el jugador tiene el id negativo"<< endl;
            cout << "No se puede cargar el jugador:" << nombre<< endl;
            flag=true;
          }
      }

      /**Valido si el jugador tiene un equipo con un valor negativo, si es asi el programa
no funciona */
      if(atoi(equipo.c_str())<=0){
         cout<<"-----------------------------------------------"<<endl;
         cout << "Error en el jugador por un id negativo del equipo"<< endl;
         cout << "No se puede cargar el jugador:" << nombre<< "  por error de id de
equipo "<< endl;
         flag=true;
      }

      aux=false;
      PtrNodoLista cursor=primero(*sistema.equipos);
      while(cursor!=fin()){
         if(atoi(equipo.c_str())==getId(*(Equipo*)cursor->ptrDato)){
            aux=true;
         }
         cursor=siguiente(*sistema.equipos,cursor);
      }

      /**Valido si el jugador no tiene equipo con id, si es asi el programa no funciona */
      if(!aux){
         cout<<"-----------------------------------------------"<<endl;
         cout<<"Error en la carga del jugador "<<nombre<<endl;
```

```
        cout<<"No existe el equipo con id: "<<atoi(equipo.c_str())<<endl;
        flag=true;
    }

    /**Valido si el id de jugador esta repetido, si es asi el programa no funciona */
    for (int indice=0; indice<i; indice++){
        if (vectorValidar[indice]==atoi(id.c_str())){
            cout<<"-----------------------------------------------"<<endl;
            cout << "Error el id:  " << atoi(id.c_str()) << " esta repetido"<<endl;
            cout << "No se puede cargar el jugador:" << nombre<< endl;
            flag=true;
        }
    }

    if(atoi(goles.c_str())>getGolesAFavor(*traerEquipo(sistema,atoi(equipo.c_str())))){
        cout<<"-----------------------------------------------"<<endl;
        cout<<"La cantidad de goles marcados por "<< id <<" "<<nombre<<endl;
        cout<<"Es mayor a la cantidad de goles de su equipo"<<endl;
        flag=true;
    }

    int sumgol=0;

    cursor=primero(*getJugadores(*traerEquipo(sistema,atoi(equipo.c_str()))));
    while(cursor!=fin()){
        sumgol=sumgol+getGoles(*(Jugador*)cursor->ptrDato);

cursor=siguiente(*getJugadores(*traerEquipo(sistema,atoi(equipo.c_str()))),cursor);
    }


if((sumgol+atoi(goles.c_str()))>getGolesAFavor(*traerEquipo(sistema,atoi(equipo.c_str
()))){
        cout<<"-----------------------------------------------"<<endl;
        cout<<"Con la cantidad de goles marcados por "<<nombre<<endl;
        cout<<"Se supera la cantidad de goles a favor de su Seleccion"<<endl;
        flag=true;
    }

    vectorValidar[i] = atoi(id.c_str());
    i = i +1;

    crear(*j, atoi(id.c_str()), nombre, atoi(goles.c_str()));

    setJugadores(*traerEquipo(sistema, atoi(equipo.c_str())), j);
```

```
      adicionarFinal(*sistema.jugadores, j);
   }

   if(flag==true){
      exit(1);
   }

   reordenar(*sistema.jugadores);

   archivo.close();
}

/*-------------------------------------------------------------------------*/
void levantarGrupos(Sistema &sistema) {
   ifstream archivo;

   archivo.open("archivos/grupos.txt", ios::in);

   if(archivo.fail()) {
      cout << "Error al abrir el archivo" << endl;
      exit(1);
   }

   string id, nombre, e1, e2, e3, e4;
   bool flag=false,encontrado=false,encontrado1=false,encontrado2=false;
   bool encontrado3=false;
   string letras = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTVWXYZ";

   while (!archivo.eof()) {
      getline(archivo, id, ';');
      getline(archivo, nombre, ';');
      getline(archivo, e1, ';');
      getline(archivo, e2, ';');
      getline(archivo, e3, ';');
      getline(archivo, e4);

      /** Validacion para los id*/
      if(id[0]<'A' || id[0]>'H'){
         cout<<"-----------------------------------------------"<<endl;
         cout<<"Se ingreso el id de un grupo de forma incorrecta"<<endl;
         flag=true;
      }

      /** Validacion para letra en algun id de equipo*/
```

```
        bool errorLetra = false;
        if(atoi(e1.c_str())==atoi(letras.c_str()) || atoi(e2.c_str())==atoi(letras.c_str()) ||
atoi(e3.c_str())==atoi(letras.c_str()) || atoi(e4.c_str())==atoi(letras.c_str())){
            cout<<"-----------------------------------------------"<<endl;
            cout << "Error, en el grupo "<<id<<endl;
            cout << "Se ingreso un equipo con id en letras" <<endl;
            errorLetra = true;
            flag=true;
        }

        /** Validacion id de equipo negativo o 0 y todos los grupos esten completos*/
        if(atoi(e2.c_str())<=0 || atoi(e3.c_str())<=0 || atoi(e4.c_str())<=0 ||
atoi(e1.c_str())<=0){
            if(e1=="" || e2=="" || e3=="" || e4==""){
                cout<<"-----------------------------------------------"<<endl;
                cout<<"Al grupo "<<id<<" le falta algun equipo"<<endl;
                flag=true;
            }else{
                if(!errorLetra){
                    cout<<"-----------------------------------------------"<<endl;
                    cout<<"Hay un id negativo o con id 0 en el grupo "<<id<<endl;
                    flag=true;
                }
            }
        }

        /** Validacion para que no haya dos ids iguales en el mismo grupo */
        if(atoi(e1.c_str())==atoi(e2.c_str()) || atoi(e1.c_str())== atoi(e3.c_str()) ||
atoi(e1.c_str())==atoi(e4.c_str())){
            cout<<"-----------------------------------------------"<<endl;
            cout<<"El grupo: "<<id<<" tiene dos equipos con el id: "<<atoi(e1.c_str())<<endl;
            flag=true;
        }
        if(atoi(e2.c_str())==atoi(e3.c_str()) || atoi(e2.c_str())==atoi(e4.c_str())){
            cout<<"-----------------------------------------------"<<endl;
            cout<<"El grupo: "<<id<<" tiene dos equipos con el id: "<<atoi(e2.c_str())<<endl;
            flag=true;
        }
        if(atoi(e3.c_str())==atoi(e4.c_str())){
            cout<<"-----------------------------------------------"<<endl;
            cout<<"El grupo: "<<id<<" tiene dos equipos con el id: "<<atoi(e3.c_str())<<endl;
            flag=true;
        }

        /** Validacion de que no haya dos equipos con el mismo id en distintos grupos*/
```

```
    PtrNodoLista cursor = primero(*sistema.grupos);
    while(cursor!=fin()){
      PtrNodoLista cursor2 = primero(*(getEquipos(*(Grupo*)cursor->ptrDato)));
      if(id[0]==getId(*(Grupo*)cursor->ptrDato)){
        if(id[0]!=0){
          cout<<"-----------------------------------------------"<<endl;
          cout<<"hay dos o mas grupos con el id: "<<id<<endl;
          flag=true;
        }
      }
      while(cursor2!=fin()){
        if(atoi(e1.c_str())==getId(*(Equipo*)cursor2->ptrDato) && atoi(e1.c_str())>0){
          cout<<"-----------------------------------------------"<<endl;
          cout<<"el equipo con id: "<<atoi(e1.c_str())<<" esta cargado en dos grupos
distintos"<<endl;
          flag=true;
        }
        if(atoi(e2.c_str())==getId(*(Equipo*)cursor2->ptrDato) && atoi(e2.c_str())>0){
          cout<<"-----------------------------------------------"<<endl;
          cout<<"el equipo con id: "<<atoi(e2.c_str())<<" esta cargado en dos grupos
distintos"<<endl;
          flag=true;
        }
        if(atoi(e3.c_str())==getId(*(Equipo*)cursor2->ptrDato) && atoi(e3.c_str())>0){
          cout<<"-----------------------------------------------"<<endl;
          cout<<"el equipo con id: "<<atoi(e3.c_str())<<" esta cargado en dos grupos
distintos"<<endl;
          flag=true;
        }
        if(atoi(e4.c_str())==getId(*(Equipo*)cursor2->ptrDato) && atoi(e4.c_str())>0){
          cout<<"-----------------------------------------------"<<endl;
          cout<<"el equipo con id: "<<atoi(e4.c_str())<<" esta cargado en dos grupos
distintos"<<endl;
          flag=true;
        }
        cursor2=siguiente(*(getEquipos(*(Grupo*)cursor->ptrDato)),cursor2);
      }
      cursor=siguiente(*sistema.grupos,cursor);
    }

    /** Validacion para que equipos ingresados existan*/
    cursor=primero(*sistema.equipos);
    encontrado1=false;encontrado2=false;encontrado3=false;
    encontrado=false;
    while(cursor!=fin()){
```

```
        if(atoi(e2.c_str())==getId(*(Equipo*)cursor->ptrDato)){
          encontrado=true;
        }
        if(atoi(e1.c_str())==getId(*(Equipo*)cursor->ptrDato)){
          encontrado1=true;
        }
        if(atoi(e3.c_str())==getId(*(Equipo*)cursor->ptrDato)){
          encontrado2=true;
        }
        if(atoi(e4.c_str())==getId(*(Equipo*)cursor->ptrDato)){
          encontrado3=true;
        }
        cursor=siguiente(*sistema.equipos,cursor);
      }

      if(!encontrado || !encontrado2 || !encontrado1 || !encontrado3){
        if(!errorLetra){
          cout<<"-----------------------------------------------"<<endl;
          cout<<"En el grupo "<<id<<endl;
          cout<<"Se ingreso un id de equipo que no existe"<<endl;
          flag=true;
        }
      }

      Lista *equipos = new Lista;
      crearLista(*equipos, compararEquipo);
      adicionarFinal(*equipos, traerEquipo(sistema, atoi(e1.c_str())));
      adicionarFinal(*equipos, traerEquipo(sistema, atoi(e2.c_str())));
      adicionarFinal(*equipos, traerEquipo(sistema, atoi(e3.c_str())));
      adicionarFinal(*equipos, traerEquipo(sistema, atoi(e4.c_str())));

      Grupo *g = new Grupo;
      crear(*g, id[0], nombre, equipos);
      adicionarFinal(*sistema.grupos, g);
    }

    if(flag==true){
      exit(1);
    }

    reordenar(*sistema.grupos);

    archivo.close();
}
```

```
/*-------------------------------------------------------------------------*/
void levantarPartidos(Sistema &sistema) {
   ifstream archivo;

   archivo.open("archivos/partidos.txt", ios::in);

   if(archivo.fail()) {
      cout << "Error al abrir el archivo" << endl;
      exit(1);
   }

   string id, e1, e2, golL, golV;

   int vectorValidar[64];
   int i = 0;
   bool flag=false;

   while (!archivo.eof()) {
      getline(archivo, id, ';');
      getline(archivo, e1, ';');
      getline(archivo, e2, ';');
      getline(archivo, golL, ';');
      getline(archivo, golV);

      /**Valido si el id de partido si tiene un valor negativo, si es asi el programa no
funciona */
      if(atoi(id.c_str())<=0){
         cout<<"-----------------------------------------------"<<endl;
         cout << "Error el partido tiene el id negativo"<< endl;
         flag=true;
      }

      /**Valido si el id de partido esta repetido, si es asi el programa no funciona */
      for (int indice=0; indice<i; indice++){
         if (vectorValidar[indice]==atoi(id.c_str())){
            cout<<"-----------------------------------------------"<<endl;
            cout << "Error el id:  " << atoi(id.c_str()) << " esta repetido"<<endl;
            flag=true;
         }
      }

      vectorValidar[i] = atoi(id.c_str());
      i = i +1;

      Partido *p = new Partido;
```

```
      crear(*p, atoi(id.c_str()),traerEquipo(sistema,
atoi(e1.c_str())),traerEquipo(sistema, atoi(e2.c_str())),
          atoi(golL.c_str()),atoi(golV.c_str()));

      if(isPartidoJugado(p))setEstado(*p,FINALIZADO);

      adicionarFinal(*sistema.partidos, p);
   }

   if(flag==true){
      exit(1);
   }

   reordenar(*sistema.partidos);

   archivo.close();
}

/*-------------------------------------------------------------------------*/
void mostrarEquipos(Sistema &sistema) {
   PtrNodoLista cursor = primero(*sistema.equipos);

   while(cursor != fin()) {
      cout << toString(*(Equipo*)cursor->ptrDato)<< endl;
      cout << endl;
      cursor = siguiente(*sistema.equipos, cursor);
   }
}

/*-------------------------------------------------------------------------*/
void mostrarJugadores(Sistema &sistema) {
   PtrNodoLista cursor = primero(*sistema.jugadores);

   while(cursor != fin()) {
      cout << toString(*(Jugador*)cursor->ptrDato);
      cout << endl;
      cursor = siguiente(*sistema.jugadores, cursor);
   }
}

/*-------------------------------------------------------------------------*/
void mostrarGrupos(Sistema &sistema) {
   PtrNodoLista cursor = primero(*sistema.grupos);

   int golesTotales = 0;
```

```
   while(cursor != fin()) {
      cout << toString(*(Grupo*)cursor->ptrDato);
      golesTotales += golesPorGrupo(*(Grupo*)cursor->ptrDato);
      cout << endl;
      cursor = siguiente(*sistema.grupos, cursor);
   }
   cout << "Total de goles: " << golesTotales << endl;
}

/*-------------------------------------------------------------------------*/
void mostrarPartidos(Sistema &sistema) {
   PtrNodoLista cursor = primero(*sistema.partidos);

   while(cursor != fin()) {
      if (getGolesL(*(Partido*)cursor->ptrDato) != -1) {
         cout << toString(*(Partido*)cursor->ptrDato)<< endl;
         cout << endl;
      }
      if (!isPartidoJugado((Partido*)cursor->ptrDato)) {

         if(isPartidoCreado((Partido*)cursor->ptrDato)){
            cout <<"Id: "<<getId(*(Partido*)cursor->ptrDato)<< endl;
            cout <<getNombre(*getEquipoL(*(Partido*)cursor->ptrDato))<< endl;
            cout <<getNombre(*getEquipoV(*(Partido*)cursor->ptrDato))<< endl;
         }else{
            cout << "Id: "<<getId(*(Partido*)cursor->ptrDato)<<"\nNo fue creado"<<
endl;
         }

         cout << endl;
      }
      cursor = siguiente(*sistema.partidos, cursor);
   }
}

/*-------------------------------------------------------------------------*/
void goleadores(Sistema &sistema) {
   sistema.jugadores->compara = compararGoles;
   reordenar(*sistema.jugadores);
   PtrNodoLista cursor = primero(*sistema.jugadores);
   int goles = getGoles(*(Jugador*)cursor->ptrDato);
   int jugadoresPorGol = 0;
   int totalDeJugadores = 0;
   int totalDeGoles = 0;
```

```
    if (goles != 0) {
        cout << "Jugadores con " << goles << " goles\n" << endl;
        while (cursor != fin()) {
            if (goles > getGoles(*(Jugador*)cursor->ptrDato)) {
                cout << "Cantidad de jugadores: " << jugadoresPorGol << "\n" << endl;
                jugadoresPorGol = 0;
                goles = getGoles(*(Jugador*)cursor->ptrDato);
                if (goles == 0) {
                    cout << "\n\nTotal de jugadores: " << totalDeJugadores << endl;
                    cout << "Total de goles: " << totalDeGoles << endl;
                    break;
                }
                cout << "\nJugadores con " << goles << " goles\n" << endl;
            }
            totalDeJugadores++;
            totalDeGoles += getGoles(*(Jugador*)cursor->ptrDato);
            jugadoresPorGol++;
            cout << "\t-" << getNombre(*(Jugador*)cursor->ptrDato) << endl;
            cursor = siguiente(*sistema.jugadores, cursor);
            cout << endl;
        }
    }
    sistema.jugadores->compara = compararJugador;
    reordenar(*sistema.jugadores);
}

/*-------------------------------------------------------------------------*/
void bajarEquipos(Sistema &sistema) {
    ofstream archivo;

    archivo.open("archivos/equipos.txt", ios::out);

    if(archivo.fail()) {
        cout << "Error al abrir el archivo" << endl;
        exit(1);
    }

    PtrNodoLista cursor = primero(*sistema.equipos);
    while (siguiente(*sistema.equipos, cursor) != fin()) {
        archivo << getId(*(Equipo*)cursor->ptrDato) << ";" <<
getNombre(*(Equipo*)cursor->ptrDato) << ";" <<
        getGolesAFavor(*(Equipo*)cursor->ptrDato) << ";" <<
getGolesEnContra(*(Equipo*)cursor->ptrDato) << ";" <<
        getPuntos(*(Equipo*)cursor->ptrDato) << endl;
        cursor = siguiente(*sistema.equipos, cursor);
```

```
    }

    archivo << getId(*(Equipo*)cursor->ptrDato) << ";" << getNombre(*(Equipo*)cursor-
>ptrDato) << ";" <<
    getGolesAFavor(*(Equipo*)cursor->ptrDato) << ";" <<
getGolesEnContra(*(Equipo*)cursor->ptrDato) << ";" <<
    getPuntos(*(Equipo*)cursor->ptrDato);

    archivo.close();
}

/*-------------------------------------------------------------------------*/
void bajarJugadores(Sistema &sistema) {
    ofstream archivo;

    archivo.open("archivos/jugadores.txt", ios::out);

    if(archivo.fail()) {
        cout << "Error al abrir el archivo" << endl;
        exit(1);
    }

    PtrNodoLista cursor = primero(*sistema.jugadores);
    while (siguiente(*sistema.jugadores, cursor) != fin()) {
        archivo << getId(*(Jugador*)cursor->ptrDato) << ";" <<
getNombre(*(Jugador*)cursor->ptrDato) << ";" <<
        getGoles(*(Jugador*)cursor->ptrDato) << ";" <<
getId(*traerEquipoPorJugador(sistema, getId(*(Jugador*)cursor->ptrDato))) << endl;
        cursor = siguiente(*sistema.jugadores, cursor);
    }

    archivo << getId(*(Jugador*)cursor->ptrDato) << ";" <<
getNombre(*(Jugador*)cursor->ptrDato) << ";" <<
    getGoles(*(Jugador*)cursor->ptrDato) << ";" <<
getId(*traerEquipoPorJugador(sistema, getId(*(Jugador*)cursor->ptrDato)));

    archivo.close();
}

/*-------------------------------------------------------------------------*/
void bajarGrupos(Sistema &sistema) {
    ofstream archivo;

    archivo.open("archivos/grupos.txt", ios::out);
```

```
   if(archivo.fail()) {
      cout << "Error al abrir el archivo" << endl;
      exit(1);
   }

   PtrNodoLista cursor = primero(*sistema.grupos);
   while (siguiente(*sistema.grupos, cursor) != fin()) {
      archivo << getId(*(Grupo*)cursor->ptrDato) << ";" <<
getNombre(*(Grupo*)cursor->ptrDato);

      Lista *e = getEquipos(*(Grupo*)cursor->ptrDato);
      PtrNodoLista c = primero(*e);
      while (c != fin()) {
         archivo << ";" << getId(*(Equipo*)c->ptrDato);
         c = siguiente(*e, c);
      }
      archivo << endl;

      cursor = siguiente(*sistema.grupos, cursor);
   }

   archivo << getId(*(Grupo*)cursor->ptrDato) << ";" << getNombre(*(Grupo*)cursor-
>ptrDato);

   Lista *e = getEquipos(*(Grupo*)cursor->ptrDato);
   PtrNodoLista c = primero(*e);
   while (c != fin()) {
      archivo << ";" << getId(*(Equipo*)c->ptrDato);
      c = siguiente(*e, c);
   }

   archivo.close();
}

/*-------------------------------------------------------------------------*/
void bajarPartidos(Sistema &sistema) {
   ofstream archivo;

   archivo.open("archivos/partidos.txt", ios::out);

   if(archivo.fail()) {
      cout << "Error al abrir el archivo" << endl;
      exit(1);
   }
```

```
   PtrNodoLista cursor = primero(*sistema.partidos);
   while (siguiente(*sistema.partidos, cursor) != fin()) {
      archivo << getId(*(Partido*)cursor->ptrDato) << ";" <<
getId(*getEquipoL(*(Partido*)cursor->ptrDato)) << ";" <<
      getId(*getEquipoV(*(Partido*)cursor->ptrDato)) << ";" <<
getGolesL(*(Partido*)cursor->ptrDato) << ";" <<
      getGolesV(*(Partido*)cursor->ptrDato) << endl;
      cursor = siguiente(*sistema.partidos, cursor);
   }

   archivo << getId(*(Partido*)cursor->ptrDato) << ";" <<
getId(*getEquipoL(*(Partido*)cursor->ptrDato)) << ";" <<
   getId(*getEquipoV(*(Partido*)cursor->ptrDato)) << ";" <<
getGolesL(*(Partido*)cursor->ptrDato) << ";" <<
   getGolesV(*(Partido*)cursor->ptrDato);

   archivo.close();
}

/*-------------------------------------------------------------------------*/
Equipo* traerEquipoPorJugador(Sistema &sistema, int id) {
   PtrNodoLista cursor = primero(*sistema.equipos);
   Equipo *e = new Equipo;
   bool encontrado = false;

   while (cursor != fin() && !encontrado) {
      Lista *l = getJugadores(*(Equipo*)cursor->ptrDato);
      PtrNodoLista c = primero(*l);
      while (c != fin() && !encontrado) {
         if (getId(*(Jugador*)c->ptrDato) == id) {
            e = ((Equipo*)cursor->ptrDato);
            encontrado = true;
         }
         c = siguiente(*l, c);
      }

      cursor = siguiente(*sistema.equipos, cursor);
   }

   return e;
}

/*-------------------------------------------------------------------------*/
Equipo* traerEquipo(Sistema &sistema, int id) {
   PtrNodoLista cursor = primero(*sistema.equipos);
```

```
   Equipo *e = new Equipo;
   crear(*e);
   bool encontrado = false;

   while (cursor != fin() && !encontrado) {
      if (getId(*(Equipo*)cursor->ptrDato) == id) {
         e = (Equipo*)cursor->ptrDato;
         encontrado = true;
      }
      cursor = siguiente(*sistema.equipos, cursor);
   }

   return e;
}

/*-------------------------------------------------------------------------*/

bool esOctavos(Sistema &sistema){
   bool bandera=false;
   PtrNodoLista cursor = primero(*sistema.partidos);

   while (cursor != fin()) {
      if(getId(*(Partido*)cursor->ptrDato)<=48){
         if ((getGolesL(*(Partido*)cursor->ptrDato) == -
1)&&(getGolesV(*(Partido*)cursor->ptrDato) == -1)) {
            bandera = true;
         }
      }
      cursor = siguiente(*sistema.partidos, cursor);
   }
   return bandera;
}

/*-------------------------------------------------------------------------*/
bool esCuartos(Sistema &sistema){
   bool bandera=false;
   PtrNodoLista cursor = primero(*sistema.partidos);

   while (cursor != fin()) {
      if(getId(*(Partido*)cursor->ptrDato)<=56){
         if ((getGolesL(*(Partido*)cursor->ptrDato) == -
1)&&(getGolesV(*(Partido*)cursor->ptrDato) == -1)) {
            bandera = true;
         }
      }
```

```
      cursor = siguiente(*sistema.partidos, cursor);
   }
   return bandera;
}

/*-------------------------------------------------------------------------*/
bool esSemis(Sistema &sistema){
   bool bandera=false;
   PtrNodoLista cursor = primero(*sistema.partidos);

   while (cursor != fin()) {
      if(getId(*(Partido*)cursor->ptrDato)<=60){
         if ((getGolesL(*(Partido*)cursor->ptrDato) == -
1)&&(getGolesV(*(Partido*)cursor->ptrDato) == -1)) {
            bandera = true;
         }
      }
      cursor = siguiente(*sistema.partidos, cursor);
   }
   return bandera;
}

/*-------------------------------------------------------------------------*/
bool es3erY4to(Sistema &sistema){
   bool bandera=false;
   PtrNodoLista cursor = primero(*sistema.partidos);

   while (cursor != fin()) {
      if(getId(*(Partido*)cursor->ptrDato)<=62){
         if ((getGolesL(*(Partido*)cursor->ptrDato) == -
1)&&(getGolesV(*(Partido*)cursor->ptrDato) == -1)) {
            bandera = true;
         }
      }
      cursor = siguiente(*sistema.partidos, cursor);
   }
   return bandera;
}

/*-------------------------------------------------------------------------*/
bool esFinal(Sistema &sistema){
   bool bandera=false;
   PtrNodoLista cursor = primero(*sistema.partidos);

   while (cursor != fin()) {
```

```
      if(getId(*(Partido*)cursor->ptrDato)<=63){
         if ((getGolesL(*(Partido*)cursor->ptrDato) == -
1)&&(getGolesV(*(Partido*)cursor->ptrDato) == -1)) {
            bandera = true;
         }
      }
      cursor = siguiente(*sistema.partidos, cursor);
   }
   return bandera;
}

/*-------------------------------------------------------------------------*/
bool validar(Sistema& sistema){
   string warning="";
   boolean values=true;
   try{
      validarPartidoCerrados(sistema,warning);
      validarEquipo(sistema.equipos,warning);
      validarGoles(sistema.equipos,sistema.partidos,warning);
      validarEmpates(sistema.partidos,warning);
      validarJugadores(sistema.equipos,warning);
      validarPuntos(sistema.equipos,sistema.partidos,warning);
      validarPartidosFaseInicial(sistema.grupos,sistema.partidos,warning);
      validarFaseClasificion(sistema,warning);
      validarClasificacionOctavos(sistema,warning);
      validarClasificacionCuartos(sistema,warning);
      verificarOctavosDeFinal(sistema,warning);
      validarPartidosFaseFinal(sistema.partidos,sistema,warning);
      if(warning!="") throw invalid_argument(warning);
      ordenarEquiposDeGrupos(*sistema.grupos);
   }catch(invalid_argument& e){
      cout << e.what() << endl;
      values=false;
      system("pause");
   }
   system("cls");
   return values;
}
/*-------------------------------------------------------------------------*/
/* Se encontró solución ordenando los grupos */
void ordenarEquiposDeGrupos(Lista& grupos){
   PtrNodoLista cursor = primero(grupos);
   while(cursor != fin() && !listaVacia(grupos)) {
      reordenar(*getEquipos(*(Grupo*)cursor->ptrDato));
      cursor = siguiente(grupos, cursor);
```

```
   }
}
/*-------------------------------------------------------------------------*/
/* Valida los Jugadores de cada Equipo que no se encuentre en otro Equipo */
void validarEquipo(Lista* equipos,string& warning){
   PtrNodoLista cursorE = primero(*equipos);
   while(cursorE != fin() && !listaVacia(*equipos)){
      PtrNodoLista cursorEAux = primero(*equipos);
      while(cursorEAux != fin()){
         if(!equals(*(Equipo*)cursorE->ptrDato,*(Equipo*)cursorEAux->ptrDato)){

            PtrNodoLista cursorJ = primero(*getJugadores(*(Equipo*)cursorE->ptrDato));
            while(cursorJ != fin() && !listaVacia(*getJugadores(*(Equipo*)cursorE-
>ptrDato))){
               PtrNodoLista cursorJAux = primero(*getJugadores(*(Equipo*)cursorEAux-
>ptrDato));
               while(cursorJAux != fin() &&
!listaVacia(*getJugadores(*(Equipo*)cursorEAux->ptrDato))){
                  if(equals(*(Jugador*)cursorJ->ptrDato,*(Jugador*)cursorJAux->ptrDato)){
                     warning+="Jugador en otro Equipo -> "+getNombre(*(Equipo*)cursorE-
>ptrDato)+"\n";
                  }
                  cursorJAux=siguiente(*getJugadores(*(Equipo*)cursorE-
>ptrDato),cursorJAux);
               }
               cursorJ=siguiente(*getJugadores(*(Equipo*)cursorE->ptrDato),cursorJ);
            }

         }
         cursorEAux=siguiente(*equipos,cursorEAux);
      }
      cursorE = siguiente(*equipos,cursorE);
   }
}

/*-------------------------------------------------------------------------*/
/* Valida los goles de cada Equipo con respecto a los partidos jugados */
void validarGoles(Lista* equipos,Lista* partidos,string& warning){
   int sumaGAFavor=0,sumaGEnContra=0;
   PtrNodoLista cursorE = primero(*equipos);
   while(cursorE != fin() && !listaVacia(*equipos)){

      PtrNodoLista cursorP = primero(*partidos);
      while(cursorP != fin() && !listaVacia(*partidos)){
         if(isPartidoJugado((Partido*)cursorP->ptrDato)){
```

```
            if(equals(*getEquipoL(*(Partido*)cursorP->ptrDato),*(Equipo*)cursorE-
>ptrDato)){
               sumaGAFavor+=getGolesL(*(Partido*)cursorP->ptrDato);
               sumaGEnContra+=getGolesV(*(Partido*)cursorP->ptrDato);
            }else if(equals(*getEquipoV(*(Partido*)cursorP->ptrDato),*(Equipo*)cursorE-
>ptrDato)){
               sumaGAFavor+=getGolesV(*(Partido*)cursorP->ptrDato);
               sumaGEnContra+=getGolesL(*(Partido*)cursorP->ptrDato);
            }
         }
         cursorP = siguiente(*partidos,cursorP);
      }
      ostringstream convert1,convert2,convert3,convert4;
      convert1 << getGolesAFavor(*(Equipo*)cursorE->ptrDato);
      convert2 << sumaGAFavor;
      convert3 << getGolesEnContra(*(Equipo*)cursorE->ptrDato);
      convert4 << sumaGEnContra;
      if(convert1.str()!=convert2.str()){
         warning+="Los golesAFavor para Equipo -> "+getNombre(*(Equipo*)cursorE-
>ptrDato)
         +" son incorrectos. goles ["+convert1.str()+"] de ["+convert2.str()+"] goles
encontrados\n";
      }
      if(convert3.str()!=convert4.str()){
         warning+="Los GolesEnContra para Equipo -> "+getNombre(*(Equipo*)cursorE-
>ptrDato)
         +" son incorrectos. goles ["+convert3.str()+"] de ["+convert4.str()+"] goles
encontrados\n";
      }

      sumaGAFavor=0;sumaGEnContra=0;
      cursorE = siguiente(*equipos,cursorE);
   }
}

/*-------------------------------------------------------------------------*/
/* Valida Ronda de eliminatorias que no halla empates */
void validarEmpates(Lista* partidos,string& warning){
   PtrNodoLista cursorP = primero(*partidos);
   while(cursorP != fin() && !listaVacia(*partidos)){
      if(isPartidoJugado((Partido*)cursorP->ptrDato) && getId(*(Partido*)cursorP-
>ptrDato)>48 &&
         isPartidoCreado((Partido*)cursorP->ptrDato)){
         if(isEmpate((Partido*)cursorP->ptrDato)){
            ostringstream convert;
```

```
            convert << getId(*(Partido*)cursorP->ptrDato);
            warning+="El Partido id["+ convert.str()+"] esta en empate\n";
          }
        }
        cursorP = siguiente(*partidos,cursorP);
    }
}

/*-------------------------------------------------------------------------*/
/* Valida los goles de los jugadores con respecto al Equipo (GolesAFavor) */
void validarJugadores(Lista* equipos,string& warning){
    int sumG=0;
    PtrNodoLista cursor = primero(*equipos);
    while(cursor!=fin() && !listaVacia(*equipos)){

        PtrNodoLista cursorJ=primero(*getJugadores(*(Equipo*)cursor->ptrDato));
        while (cursorJ!=fin() && !listaVacia(*getJugadores(*(Equipo*)cursor->ptrDato))){
            sumG+=getGoles(*(Jugador*)cursorJ->ptrDato);
            cursorJ=siguiente(*getJugadores(*(Equipo*)cursor->ptrDato),cursorJ);
        }
        ostringstream convert,convert2,convert3;
        convert<<sumG;
        convert2<<getGolesAFavor(*(Equipo*)cursor->ptrDato);
        convert3<<getId(*(Equipo*)cursor->ptrDato);
        if(convert.str()!=convert2.str()){
            warning+="Los Goleadores del Equipo ["+convert3.str()+"]
["+getNombre(*(Equipo*)cursor->ptrDato)
                +"] su suma es incorrecta, GolesAFavor["+convert2.str()+"] goleadores:
"+convert.str()+" goles \n";
        }

        sumG=0;
        cursor=siguiente(*equipos,cursor);
    }
}

/*-------------------------------------------------------------------------*/
/* Valida los puntos de cada Equipo */
void validarPuntos(Lista* equipos,Lista* partidos,string& warning){

    PtrNodoLista cursorE=primero(*equipos);
    int puntos=0;
    while(cursorE!=fin() && !listaVacia(*equipos)){

        PtrNodoLista cursorP=primero(*partidos);
```

```
      while(cursorP!=fin() && !listaVacia(*partidos)){

         if(getId(*(Partido*)cursorP->ptrDato)<=48 &&
isPartidoJugado((Partido*)cursorP->ptrDato)){
            if(equals(*(Equipo*)cursorE->ptrDato,*getEquipoL(*(Partido*)cursorP-
>ptrDato))){
               if (getGolesL(*(Partido*)cursorP->ptrDato) > getGolesV(*(Partido*)cursorP-
>ptrDato)) {
                  puntos+=3;
               }else if(isEmpate((Partido*)cursorP->ptrDato)) {
                  puntos+=1;
               }
            }else if(equals(*(Equipo*)cursorE->ptrDato,*getEquipoV(*(Partido*)cursorP-
>ptrDato))){
               if (getGolesL(*(Partido*)cursorP->ptrDato) < getGolesV(*(Partido*)cursorP-
>ptrDato)) {
                  puntos+=3;
               }else if(isEmpate((Partido*)cursorP->ptrDato)) {
                  puntos+=1;
               }
            }
         }
         cursorP=siguiente(*partidos,cursorP);
      }
      ostringstream convert,convert2,convert3;
      convert << getId(*(Equipo*)cursorE->ptrDato);
      convert2 << puntos;
      convert3 << getPuntos(*(Equipo*)cursorE->ptrDato);
      if(puntos!=getPuntos(*(Equipo*)cursorE->ptrDato)){
         warning+="Puntos del Equipo id["+convert.str()+"]
"+getNombre(*(Equipo*)cursorE->ptrDato)
         +" son incorrectos: ["+convert3.str()+ "] se calculo -> ["+convert2.str()+"]\n";
      }
      puntos=0;
      cursorE=siguiente(*equipos,cursorE);
   }

}

/*-------------------------------------------------------------------------*/
/* Valida 1ra Ronda si el Equipo completa el cuadrangular simple de partidos a jugar */
void validarPartidosFaseInicial(Lista* grupos,Lista* partidos,string& warning){
   PtrNodoLista cursorG=primero(*grupos);
   int contP=0;
   while(cursorG!=fin() && !listaVacia(*grupos)){
```

```
    PtrNodoLista cursorE=primero(*getEquipos(*(Grupo*)cursorG->ptrDato));
    while(cursorE!=fin() && !listaVacia(*getEquipos(*(Grupo*)cursorG->ptrDato))){

      PtrNodoLista cursorP=primero(*partidos);
      while(cursorP!=fin() && !listaVacia(*partidos)){

        if(getId(*(Partido*)cursorP->ptrDato)<=48){
          // verifico si es del grupo su rival
          ostringstream convert;
          convert << getId(*(Partido*)cursorP->ptrDato);
          if(equals(*(Equipo*)cursorE->ptrDato,*getEquipoL(*(Partido*)cursorP-
>ptrDato))){
            if(!verificarGrupo(getEquipos(*(Grupo*)cursorG-
>ptrDato),getEquipoV(*(Partido*)cursorP->ptrDato))){
              warning+="El Partido id ["+convert.str()+"] Equipo Local
["+getNombre(*getEquipoL(*(Partido*)cursorP->ptrDato))
              +"] .su rival no es del grupo ["+getNombre(*(Grupo*)cursorG-
>ptrDato)+"] No se puede concretar partido\n";
            }
            contP++;
          }else if(equals(*(Equipo*)cursorE-
>ptrDato,*getEquipoV(*(Partido*)cursorP->ptrDato))){
            if(!verificarGrupo(getEquipos(*(Grupo*)cursorG-
>ptrDato),getEquipoL(*(Partido*)cursorP->ptrDato))){
              warning+="El Partido id ["+convert.str()+"] Equipo Visitante
["+getNombre(*getEquipoV(*(Partido*)cursorP->ptrDato))
              +"] .su rival no es del grupo ["+getNombre(*(Grupo*)cursorG-
>ptrDato)+"] No se puede concretar partido\n";
            }
            contP++;
          }
        }
        cursorP=siguiente(*partidos,cursorP);
      }
      if(contP!=3){
        ostringstream convert,convert2;
        convert << getId(*(Equipo*)cursorE->ptrDato);
        convert2 << contP;
        warning+="El Equipo id ["+convert.str()+"] ["+getNombre(*(Equipo*)cursorE-
>ptrDato)
        +"] no completa el cuadrangular simple, se calcula -> ["+convert2.str()
        +"] partidos que jugara\n";
      }
      contP=0;
      cursorE=siguiente(*getEquipos(*(Grupo*)cursorG->ptrDato),cursorE);
```

```
      }
      cursorG=siguiente(*grupos,cursorG);
   }
}

/*-------------------------------------------------------------------------*/
/* Valida 1ra Ronda si el Equipo pasa a octavos correctamente */
void validarFaseClasificion(Sistema& sistema,string& warning){
   ordenarEquiposDeGrupos(*sistema.grupos);
   if(verificarPartidos(sistema.partidos,1,48)){
      PtrNodoLista cursor = primero(*sistema.grupos);
      while(cursor != fin() && !listaVacia(*sistema.grupos)) {
         /*si todos los partidos de la fase uno fue jugados*/
         if(!localizar(traerGanadores(getEquipos(*(Grupo*)cursor-
>ptrDato),sistema.partidos)[0],sistema,49,57)){
            ostringstream convert;
            convert << getId(*traerGanadores(getEquipos(*(Grupo*)cursor-
>ptrDato),sistema.partidos)[0]);
            warning+="El Equipo id ["+convert.str()+"]
["+getNombre(*traerGanadores(getEquipos(*(Grupo*)cursor-
>ptrDato),sistema.partidos)[0])
                  +"] Gano la fase Inicial y no se encuentra en 8vos\n";
         }
         if(!localizar(traerGanadores(getEquipos(*(Grupo*)cursor-
>ptrDato),sistema.partidos)[1],sistema,49,57)) {
            ostringstream convert;
            convert << getId(*traerGanadores(getEquipos(*(Grupo*)cursor-
>ptrDato),sistema.partidos)[1]);
            warning+="El Equipo id ["+convert.str()+"]
["+getNombre(*traerGanadores(getEquipos(*(Grupo*)cursor-
>ptrDato),sistema.partidos)[1])
                  +"] Gano la fase Inicial y no se encuentra en 8vos\n";
         }

      cursor = siguiente(*sistema.grupos, cursor);
      }
   }
}
/*se compara la ronda de partidos si estan finalizados*/
bool verificarPartidos(Lista* partidos,int i,int f){
   bool values=true;
   PtrNodoLista cursor = primero(*partidos);
   while(cursor != fin() && !listaVacia(*partidos) &&
       getId(*(Partido*)cursor->ptrDato)>=i &&
       getId(*(Partido*)cursor->ptrDato)<=f && values==true) {
```

```
      if(!isPartidoJugado((Partido*)cursor->ptrDato))values=false;
      cursor = siguiente(*partidos, cursor);
   }
   return values;
}
Equipo** traerGanadores(Lista* equipos,Lista* partidos){
   Equipo* e[2];
   e[0]=new Equipo;e[1]=new Equipo;
   crear(*e[0]);crear(*e[1]);
   int cont=2;bool opc2=false;
   while(cont!=0){
      PtrNodoLista cursor = primero(*equipos);
      while(cursor != fin() && !listaVacia(*equipos)) {
         if(opc2==false){ //de cada 4 devuelvo 2 equipos
            if(verificarVictoriaPuntos(*(Equipo*)cursor->ptrDato,equipos)){
            e[0]=(Equipo*)cursor->ptrDato;
            cont--;}
            cursor = siguiente(*equipos, cursor);
            if(verificarVictoriaPuntos(*(Equipo*)cursor->ptrDato,equipos)){
            e[1]=(Equipo*)cursor->ptrDato;
            cont--;}
         }else{
            if(verificarVictoriaDifGoles(*(Equipo*)cursor->ptrDato,equipos,partidos) &&
getId(*e[0])==0){
            e[0]=(Equipo*)cursor->ptrDato;
            cont--;}
            cursor = siguiente(*equipos, cursor);
            if(verificarVictoriaDifGoles(*(Equipo*)cursor->ptrDato,equipos,partidos) &&
getId(*e[1])==0){
            e[1]=(Equipo*)cursor->ptrDato;
            cont--;}
            cont=0;
         }
         cursor = siguiente(*equipos, cursor);
      }
      if(cont!=0){
         opc2=true;
      }
   }
   return e;
}
bool verificarVictoriaPuntos(Equipo& e,Lista* equipos){
   int cont=0;
   bool values=false;
   PtrNodoLista cursor = primero(*equipos);
```

```
   while(cursor != fin() && !listaVacia(*equipos)) {
      if(!equals(e,*(Equipo*)cursor->ptrDato)){
         if(getPuntos(e)>getPuntos(*(Equipo*)cursor->ptrDato))cont++;
      }
      cursor = siguiente(*equipos, cursor);
   }
   if(cont>=2)values=true;
   return values;
}
bool verificarVictoriaDifGoles(Equipo& e,Lista* equipos,Lista* partidos){
   int cont=0;
   bool values=false;
   PtrNodoLista cursor = primero(*equipos);
   while(cursor != fin() && !listaVacia(*equipos)) {
      if(!equals(e,*(Equipo*)cursor->ptrDato)){

if(diferenciaDeGoles(partidos,e,1,48)>diferenciaDeGoles(partidos,*(Equipo*)cursor-
>ptrDato,1,48))cont++;
      }
      cursor = siguiente(*equipos, cursor);
   }
   if(cont>=1)values=true;
   return values;
}
bool localizar(Equipo* equipo,Sistema& s,int i,int f){
   bool values=false;
   for(int j=i;j<f && values==false;j++){
      if(verificarPartido(s,equipo,j))values=true;
   }
   return values ;
}
/*-------------------------------------------------------------------------*/
/* Valida 8vos si los equipos no se repiten */
void verificarOctavosDeFinal(Sistema& s,string& warning){
   bool values=false;
   int cont=0;
   PtrNodoLista cursorP=primero(*s.partidos);
   while(cursorP!=fin() && !listaVacia(*s.partidos) ){
      if(getId(*(Partido*)cursorP->ptrDato)>48 &&
        getId(*(Partido*)cursorP->ptrDato)<57 &&
        isPartidoCreado((Partido*)cursorP->ptrDato)){
         for(int i=49;i<=56;i++){
            if(verificarPartido(s,getEquipoL(*(Partido*)cursorP->ptrDato),i))cont++;
         }
         if(cont!=1){
```

```
            warning+="Error en Octavos de final: Equipo Local
["+getNombre(*getEquipoL(*(Partido*)cursorP->ptrDato))+"] repetido\n";
          }
          cont=0;
          for(int i=49;i<=56;i++){
            if(verificarPartido(s,getEquipoV(*(Partido*)cursorP->ptrDato),i))cont++;
          }
          if(cont!=1){
            warning+="Error en Octavos de final: Equipo Visitante
["+getNombre(*getEquipoV(*(Partido*)cursorP->ptrDato))+"] repetido\n";
          }
        }
        cont=0;
        cursorP=siguiente(*s.partidos,cursorP);
    }
}
/*-------------------------------------------------------------------------*/
/* Valida 8vos de Final si el Equipo que gano, paso a 4tos */
void validarClasificacionOctavos(Sistema& s,string& warning){

  bool values=false;
  if(!verificarPartidos(s.partidos,49,56)){
    for(int i=49;i<57;i++){
      Equipo* e=traerGanador(traerPartido(s,i));
      for(int j=57;j<=60;j++){
        if(equals(*e,*getEquipoL(*traerPartido(s,j))) ||
equals(*e,*getEquipoV(*traerPartido(s,j))))values=true;;
      }
      if(!values){
        ostringstream convert;
        convert << getId(*e);
        warning+="El Equipo id ["+convert.str()+"] ["+getNombre(*e)
            +"] Gano la fase de 8vos y no se encuentra en 4tos\n";
      }
      values=false;
    }
  }
}
/* Valida 4tos de Final si el Equipo que gano, paso a SemiFinal */
void validarClasificacionCuartos(Sistema& s,string& warning){
  ordenarEquiposDeGrupos(*s.grupos);
  bool values=false;
  if(!verificarPartidos(s.partidos,57,60)){
    for(int i=57;i<61;i++){
      Equipo* e=traerGanador(traerPartido(s,i));
```

```
      for(int j=61;j<=62;j++){
         if(equals(*e,*getEquipoL(*traerPartido(s,j))) ||
equals(*e,*getEquipoV(*traerPartido(s,j))))values=true;
      }
      if(!values){
         ostringstream convert;
         convert << getId(*e);
         warning+="El Equipo id ["+convert.str()+"] ["+getNombre(*e)
               +"] Gano la fase de 4tos y no se encuentra en SemiFinal\n";
      }
      values=false;
   }
  }
}

/*-------------------------------------------------------------------------*/
/* Valida 2da Ronda si el Equipo que juega es correcto */
void validarPartidosFaseFinal(Lista* partidos,Sistema& sistema,string& warning){
   PtrNodoLista cursorP=primero(*partidos);
   while(cursorP!=fin() && !listaVacia(*partidos)){

      if(getId(*(Partido*)cursorP->ptrDato)>48 && isPartidoCreado((Partido*)cursorP-
>ptrDato)){
         /* para partido del 49 al 62*/
         if(getId(*(Partido*)cursorP->ptrDato)<63){

if(verificarContinuidadEquipo(partidos,siguiente(*partidos,cursorP),traerPerdedor((Par
tido*)cursorP->ptrDato))){
               ostringstream convert,convert2;
               convert << getId(*traerPerdedor((Partido*)cursorP->ptrDato));
               convert2 << getId(*(Partido*)cursorP->ptrDato);
               warning+="El Equipo id ["+convert.str()+"]
["+getNombre(*traerPerdedor((Partido*)cursorP->ptrDato))
               +"] Perdio fase Eliminatoria. se encontro en el partido id
["+convert2.str()+"]\n";
            }
         }else if(getId(*(Partido*)cursorP->ptrDato)==63){
            /* para 3er y 4to puesto*/
            if(!verificarPartido(sistema,traerPerdedor(traerPartido(sistema, 61)),63) ){
               ostringstream convert;
               convert << getId(*traerPerdedor(traerPartido(sistema, 61)));
               warning+="El Partido para 3er y 4to puesto id [63] tiene Equipo erroneo,
siendo id ["+convert.str()+"] ["
               +getNombre(*traerPerdedor(traerPartido(sistema, 61)))+"] posible
correcto\n";
```

```
            }
            if(!verificarPartido(sistema,traerPerdedor(traerPartido(sistema, 62)),63) ){
               ostringstream convert;
               convert << getId(*traerPerdedor(traerPartido(sistema, 62)));
               warning+="El Partido para 3er y 4to puesto id [63] tiene Equipo erroneo,
siendo id ["+convert.str()+"] ["
               +getNombre(*traerPerdedor(traerPartido(sistema, 62)))+"] posible
correcto\n";
            }
         }else{
            /* para 1er y 2do puesto*/
            if(!verificarPartido(sistema,traerGanador(traerPartido(sistema, 61)),64) ){
               ostringstream convert;
               convert << getId(*traerGanador(traerPartido(sistema, 61)));
               warning+="El Partido para 1er y 2do puesto id [64] tiene Equipo erroneo,
siendo id ["+convert.str()+"] ["
               +getNombre(*traerGanador(traerPartido(sistema, 61)))+"] posible
correcto\n";
            }
            if(!verificarPartido(sistema,traerGanador(traerPartido(sistema, 62)),64) ){
               ostringstream convert;
               convert << getId(*traerGanador(traerPartido(sistema, 62)));
               warning+="El Partido para 1er y 2do puesto id [64] tiene Equipo erroneo,
siendo id ["+convert.str()+"] ["
               +getNombre(*traerGanador(traerPartido(sistema, 62)))+"] posible
correcto\n";
            }
         }
      }
      cursorP=siguiente(*partidos,cursorP);
   }
}

/*-------------------------------------------------------------------------*/
/* Verifico partidos imcompletos */
void validarPartidoCerrados(Sistema& s,string& warning){
   bool values=true;
   Equipo* e=new Equipo;
   crear(*e);
   for(int j=1;j<=64;j++){
      Partido *p=traerPartido(s,j);
      if(!isPartidoCreado(p) && !isPartidoJugado(p)
        && (!equals(*getEquipoL(*p),*e) || !equals(*getEquipoV(*p),*e))){
         ostringstream convert;
         convert << getId(*traerPartido(s,j));
```

```
            warning+="El Partido id ["+convert.str()+"] incompleto\n";
        }
    }
}
/*----------------------------------------------------------------------------*/
bool verificarPartido(Sistema& sistema,Equipo* equipo,int id){
    bool values=false;
    if(equals(*getEquipoL(*traerPartido(sistema, id)),*equipo)
      ||equals(*getEquipoV(*traerPartido(sistema, id)),*equipo))values=true;
    return values;
}
/*----------------------------------------------------------------------------*/
bool verificarContinuidadEquipo(Lista* partidos,PtrNodoLista cursor,Equipo* equipo){
    bool values=false;
    while(cursor!=fin() && !values && getId(*(Partido*)cursor->ptrDato)<63){
        if(equals(*getEquipoL(*(Partido*)cursor->ptrDato),*equipo) ||
          equals(*getEquipoV(*(Partido*)cursor->ptrDato),*equipo)){values=true;}
        cursor=siguiente(*partidos,cursor);
    }
    return values;
}
/*----------------------------------------------------------------------------*/
bool verificarGrupo(Lista* equipos,Equipo* equipo){
    bool values=false;
    PtrNodoLista cursor=primero(*equipos);
    while(cursor!=fin() && !listaVacia(*equipos)){
        if(equals(*(Equipo*)cursor->ptrDato,*equipo))values=true;
        cursor=siguiente(*equipos,cursor);
    }
    return values;
}
/*----------------------------------------------------------------------------*/
bool isPartidoJugado(Partido* p){
    return getGolesL(*p)!=-1 || getGolesV(*p)!=-1;
}
/*----------------------------------------------------------------------------*/
bool isPartidoCreado(Partido* p){
    return getId(*getEquipoL(*p))!=0 && getId(*getEquipoV(*p))!=0;
}
/*----------------------------------------------------------------------------*/
bool isEmpate(Partido* p){
    return getGolesL(*p)==getGolesV(*p);
}
/*----------------------------------------------------------------------------*/
Equipo* traerGanador(Partido* p){
```

```
   if (getGolesL(*p) > getGolesV(*p))
      return getEquipoL(*p);
   else
      return getEquipoV(*p);
}
/*-------------------------------------------------------------------------*/
Equipo* traerPerdedor(Partido* p){
   if (getGolesL(*p) < getGolesV(*p))
      return getEquipoL(*p);
   else
      return getEquipoV(*p);
}

/*-------------------------------------------------------------------------*/
// Tabla de valores por Equipos < >
void mostrarTablaPosiciones(Sistema &sistema) {
   PtrNodoLista cursor = primero(*sistema.grupos);

   int golesTotales = 0;
   while(cursor != fin() && !listaVacia(*sistema.grupos)) {

      if(getId(*(Grupo*)cursor->ptrDato)>0){
         cout << getNombre(*(Grupo*)cursor->ptrDato) << endl;
         cout << "Pos" << "\t||" << "Dif" << " \t|| " << "Pun" << " \t|| " << "Equipo" <<
endl;
         cout << "----------------------------------------" << endl;
         reordenar(*getEquipos(*(Grupo*)cursor->ptrDato));
         PtrNodoLista cursorE = primero(*getEquipos(*(Grupo*)cursor->ptrDato));
         int i = 1;
         while (cursorE != fin() && !listaVacia(*getEquipos(*(Grupo*)cursor->ptrDato))) {
            cout << i << " \t||";
            cout << getGolesAFavor(*(Equipo*)cursorE->ptrDato) -
getGolesEnContra(*(Equipo*)cursorE->ptrDato) << " \t|| ";
            cout << getPuntos(*(Equipo*)cursorE->ptrDato) << " \t|| ";
            cout << getNombre(*(Equipo*)cursorE->ptrDato) << endl;
            i++;
            cursorE = siguiente(*getEquipos(*(Grupo*)cursor->ptrDato), cursorE);
         }
         cout << "Goles del grupo: " << golesPorGrupo(*(Grupo*)cursor->ptrDato) <<
endl;
         cout << endl;
      }
      golesTotales += golesPorGrupo(*(Grupo*)cursor->ptrDato);
      cout << endl;
      cursor = siguiente(*sistema.grupos, cursor);
```

```
    }
    cout << "Total de goles: " << golesTotales << endl;
}
/*-------------------------------------------------------------------------*/
int diferenciaDeGoles(Lista* partidos,Equipo& e,int i,int f){
    return traerGolesAFavor(partidos,e,i,f) - traerGolesEnContra(partidos,e,i,f);
}
/*-------------------------------------------------------------------------*/
int traerGoles(Partido* p,Equipo& e){
    int goles=0;
    if(equals(*getEquipoL(*p),e))
        goles=getGolesL(*p);
    else
        goles=getGolesV(*p);
    return goles;
}
/*-------------------------------------------------------------------------*/
int traerGolesAFavor(Lista* partidos,Equipo& e,int i,int f){
    PtrNodoLista cursorP=primero(*partidos);
    int cont=0;
    while(cursorP!=fin() && !listaVacia(*partidos) && getId(*(Partido*)cursorP-
>ptrDato)>=i && getId(*(Partido*)cursorP->ptrDato)<=f){

        if(isPartidoJugado((Partido*)cursorP->ptrDato) ){
            if(equals(*getEquipoL(*(Partido*)cursorP->ptrDato),e)
                ||equals(*getEquipoV(*(Partido*)cursorP-
>ptrDato),e))cont+=traerGoles((Partido*)cursorP->ptrDato,e);
        }

        cursorP=siguiente(*partidos,cursorP);
    }
    return cont;
}
/*-------------------------------------------------------------------------*/
int traerGolesEnContra(Lista* partidos,Equipo& e,int i,int f){
    PtrNodoLista cursorP=primero(*partidos);
    int cont=0;
    while(cursorP!=fin() && !listaVacia(*partidos) && getId(*(Partido*)cursorP-
>ptrDato)>=i && getId(*(Partido*)cursorP->ptrDato)<=f){

        if(isPartidoJugado((Partido*)cursorP->ptrDato) ){
            if(equals(*getEquipoL(*(Partido*)cursorP->ptrDato),e)){
                cont+=traerGoles((Partido*)cursorP-
>ptrDato,*getEquipoV(*(Partido*)cursorP->ptrDato));
            }else if(equals(*getEquipoV(*(Partido*)cursorP->ptrDato),e)){
```

```
            cont+=traerGoles((Partido*)cursorP-
>ptrDato,*getEquipoL(*(Partido*)cursorP->ptrDato));
          }
        }

        cursorP=siguiente(*partidos,cursorP);
    }
    return cont;
}

/*-------------------------------------------------------------------------*/
void setearFases(Sistema &sistema){

    if(!esOctavos(sistema)){
        Equipo *ePrimero = new Equipo;
        Equipo *eSegundo = new Equipo;
        PtrNodoLista cursor1 = primero(*sistema.grupos);
        Grupo *g = new Grupo;
        while (cursor1 != fin()) {
            Partido *p;
            Lista *l = getEquipos(*(Grupo*)cursor1->ptrDato);
            g = (Grupo*)cursor1->ptrDato;
            PtrNodoLista cursor2 = primero(*l);

            Equipo *e = new Equipo;
            while (cursor2 != fin()) {
                //primero (*l)->ptrDato
                ePrimero = (Equipo*)primero (*l)->ptrDato;
                eSegundo = (Equipo*)siguiente(*l,primero (*l))->ptrDato;

                e = (Equipo*)cursor2->ptrDato;

                cursor2 = siguiente(*l, cursor2);
            }
            switch(getId(*(Grupo*)g)){
                case 'A':
                    p = traerPartido(sistema, 49);
                    setEquipoL(*p,ePrimero);
                    p = traerPartido(sistema, 51);
                    setEquipoV(*p,eSegundo);
                    break;
                case 'B':
                    p = traerPartido(sistema, 51);
                    setEquipoL(*p,ePrimero);
                    p = traerPartido(sistema, 49);
```

```
            setEquipoV(*p,eSegundo);
            break;
        case 'C':
            p = traerPartido(sistema, 50);
            setEquipoL(*p,ePrimero);
            p = traerPartido(sistema, 52);
            setEquipoV(*p,eSegundo);
            break;
        case 'D':
            p = traerPartido(sistema, 52);
            setEquipoL(*p,ePrimero);
            p = traerPartido(sistema, 50);
            setEquipoV(*p,eSegundo);
            break;
        case 'E':
            p = traerPartido(sistema, 53);
            setEquipoL(*p,ePrimero);
            p = traerPartido(sistema, 55);
            setEquipoV(*p,eSegundo);
            break;
        case 'F':
            p = traerPartido(sistema, 55);
            setEquipoL(*p,ePrimero);
            p = traerPartido(sistema, 53);
            setEquipoV(*p,eSegundo);
            break;
        case 'G':
            p = traerPartido(sistema, 54);
            setEquipoL(*p,ePrimero);
            p = traerPartido(sistema, 56);
            setEquipoV(*p,eSegundo);
            break;
        case 'H':
            p = traerPartido(sistema, 56);
            setEquipoL(*p,ePrimero);
            p = traerPartido(sistema, 54);
            setEquipoV(*p,eSegundo);
            break;

    }
    cursor1 = siguiente(*sistema.grupos, cursor1);
}

if(!esCuartos(sistema)){
```

```
        ///primer partido de cuartos
        if(getGolesL(*traerPartido(sistema, 49)) > getGolesV(*traerPartido(sistema,
49))){
           setEquipoL(*traerPartido(sistema, 57),getEquipoL(*traerPartido(sistema,
49)));
        }else{
           setEquipoL(*traerPartido(sistema, 57),getEquipoV(*traerPartido(sistema,
49)));
        }
        if(getGolesL(*traerPartido(sistema, 50)) > getGolesV(*traerPartido(sistema,
50))){
           setEquipoV(*traerPartido(sistema, 57),getEquipoL(*traerPartido(sistema,
50)));
        }else{
           setEquipoV(*traerPartido(sistema, 57),getEquipoV(*traerPartido(sistema,
50)));
        }

        ///segundo partido de cuartos
        if(getGolesL(*traerPartido(sistema, 51)) > getGolesV(*traerPartido(sistema,
51))){
           setEquipoL(*traerPartido(sistema, 58),getEquipoL(*traerPartido(sistema,
51)));
        }else{
           setEquipoL(*traerPartido(sistema, 58),getEquipoV(*traerPartido(sistema,
51)));
        }
        if(getGolesL(*traerPartido(sistema, 52)) > getGolesV(*traerPartido(sistema,
52))){
           setEquipoV(*traerPartido(sistema, 58),getEquipoL(*traerPartido(sistema,
52)));
        }else{
           setEquipoV(*traerPartido(sistema, 58),getEquipoV(*traerPartido(sistema,
52)));
        }

        ///tercer partido de cuartos
        if(getGolesL(*traerPartido(sistema, 53)) > getGolesV(*traerPartido(sistema,
53))){
           setEquipoL(*traerPartido(sistema, 59),getEquipoL(*traerPartido(sistema,
53)));
        }else{
           setEquipoL(*traerPartido(sistema, 59),getEquipoV(*traerPartido(sistema,
53)));
        }
```

```
        if(getGolesL(*traerPartido(sistema, 54)) > getGolesV(*traerPartido(sistema,
54))){
            setEquipoV(*traerPartido(sistema, 59),getEquipoL(*traerPartido(sistema,
54)));
        }else{
            setEquipoV(*traerPartido(sistema, 59),getEquipoV(*traerPartido(sistema,
54)));
        }

        ///cuarto partido de cuartos
        if(getGolesL(*traerPartido(sistema, 55)) > getGolesV(*traerPartido(sistema,
55))){
            setEquipoL(*traerPartido(sistema, 60),getEquipoL(*traerPartido(sistema,
55)));
        }else{
            setEquipoL(*traerPartido(sistema, 60),getEquipoV(*traerPartido(sistema,
55)));
        }
        if(getGolesL(*traerPartido(sistema, 56)) > getGolesV(*traerPartido(sistema,
56))){
            setEquipoV(*traerPartido(sistema, 60),getEquipoL(*traerPartido(sistema,
56)));
        }else{
            setEquipoV(*traerPartido(sistema, 60),getEquipoV(*traerPartido(sistema,
56)));
        }
    }
  }
  if(!esSemis(sistema)){
    ///primer partido semis
    if(getGolesL(*traerPartido(sistema, 57)) > getGolesV(*traerPartido(sistema, 57))){
      setEquipoL(*traerPartido(sistema, 61),getEquipoL(*traerPartido(sistema, 57)));
    }else{
      setEquipoL(*traerPartido(sistema, 61),getEquipoV(*traerPartido(sistema, 57)));
    }
    if(getGolesL(*traerPartido(sistema, 58)) > getGolesV(*traerPartido(sistema, 58))){
      setEquipoV(*traerPartido(sistema, 61),getEquipoL(*traerPartido(sistema, 58)));
    }else{
      setEquipoV(*traerPartido(sistema, 61),getEquipoV(*traerPartido(sistema, 58)));
    }

    ///segundo partido de semis
    if(getGolesL(*traerPartido(sistema, 59)) > getGolesV(*traerPartido(sistema, 59))){
      setEquipoL(*traerPartido(sistema, 62),getEquipoL(*traerPartido(sistema, 59)));
    }else{
```

```
        setEquipoL(*traerPartido(sistema, 62),getEquipoV(*traerPartido(sistema, 59)));
      }
      if(getGolesL(*traerPartido(sistema, 60)) > getGolesV(*traerPartido(sistema, 60))){
        setEquipoV(*traerPartido(sistema, 62),getEquipoL(*traerPartido(sistema, 60)));
      }else{
        setEquipoV(*traerPartido(sistema, 62),getEquipoV(*traerPartido(sistema, 60)));
      }
    }
    if(!es3erY4to(sistema)){
      ///3er y cuarto puesto
      if(getGolesL(*traerPartido(sistema, 61)) < getGolesV(*traerPartido(sistema, 61))){
        setEquipoL(*traerPartido(sistema, 63),getEquipoL(*traerPartido(sistema, 61)));
      }else{
        setEquipoL(*traerPartido(sistema, 63),getEquipoV(*traerPartido(sistema, 61)));
      }
      if(getGolesL(*traerPartido(sistema, 62)) < getGolesV(*traerPartido(sistema, 62))){
        setEquipoV(*traerPartido(sistema, 63),getEquipoL(*traerPartido(sistema, 62)));
      }else{
        setEquipoV(*traerPartido(sistema, 63),getEquipoV(*traerPartido(sistema, 62)));
      }
    }
    if(!esFinal(sistema)){
      ///FINAL!!!!!!!!!!!!!!!!!!
      if(getGolesL(*traerPartido(sistema, 61)) > getGolesV(*traerPartido(sistema, 61))){
        setEquipoL(*traerPartido(sistema, 64),getEquipoL(*traerPartido(sistema, 61)));
      }else{
        setEquipoL(*traerPartido(sistema, 64),getEquipoV(*traerPartido(sistema, 61)));
      }
      if(getGolesL(*traerPartido(sistema, 62)) > getGolesV(*traerPartido(sistema, 62))){
        setEquipoV(*traerPartido(sistema, 64),getEquipoL(*traerPartido(sistema, 62)));
      }else{
        setEquipoV(*traerPartido(sistema, 64),getEquipoV(*traerPartido(sistema, 62)));
      }

    }

}

/*-------------------------------------------------------------------------*/
Jugador* traerJugador(Sistema &sistema, int id) {
    PtrNodoLista cursor = primero(*sistema.jugadores);
    Jugador *j = new Jugador;
    bool encontrado = false;

    while (cursor != fin() && !encontrado) {
```

```
      if (getId(*(Jugador*)cursor->ptrDato) == id) {
         j = (Jugador*)cursor->ptrDato;
         encontrado = true;
      }
      cursor = siguiente(*sistema.jugadores, cursor);
   }

   return j;
}

/*-------------------------------------------------------------------------*/
Grupo* traerGrupo(Sistema &sistema, char id) {
   PtrNodoLista cursor = primero(*sistema.grupos);
   Grupo *g = new Grupo;
   bool encontrado = false;

   while (cursor != fin() && !encontrado) {
      if (getId(*(Grupo*)cursor->ptrDato) == id) {
         g = (Grupo*)cursor->ptrDato;
         encontrado = true;
      }
      cursor = siguiente(*sistema.grupos, cursor);
   }

   return g;
}

/*-------------------------------------------------------------------------*/
Partido* traerPartido(Sistema &sistema, int id) {
   PtrNodoLista cursor = primero(*sistema.partidos);
   Partido *p = new Partido;
   bool encontrado = false;

   while (cursor != fin() && !encontrado) {
      if (getId(*(Partido*)cursor->ptrDato) == id) {
         p = (Partido*)cursor->ptrDato;
         encontrado = true;
      }
      cursor = siguiente(*sistema.partidos, cursor);
   }

   return p;
}

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

```
void inicioPartido(Sistema &sistema) {
   int id;
   bool encontrado = false;
   cout << "Ingrese id del partido: ";
   cin >> id;

   if((id<=0)||(id>=65)) cout<<"id incorrecto"<<endl;

   Partido *p = traerPartido(sistema, id);

   if (id > 0 && id < 65) {
      //comprobacion de octavos de final
      if((id>=49) && (id<=56)){
         cout<<"partido de octavos:"<<endl;
         encontrado = esOctavos(sistema);

      }
      //comprobacion de cuartos de final
      if((id>=57) && (id<=60)){
         cout<<"partido de cuartos:"<<endl;
         encontrado = esCuartos(sistema);

      }
      //comprobacion de semifinal
      if((id>=61) && (id<=62)){
         cout<<"partido de semifinal:"<<endl;
         encontrado = esSemis(sistema);

      }
      //comprobacion de tercer y cuarto puesto
      if(id==63){
         cout<<"partido de tercer y cuarto puesto"<<endl;
         encontrado = es3erY4to(sistema);

      }
      //comprobacion de la final
      if(id==64){
         cout<<"partido FINAL"<<endl;
         encontrado = esFinal(sistema);

      }

      if(encontrado){
            cout<<"no se puede iniciar el partido hasta no completar la fase
anterior"<<endl;
```

```
      }
      else{
         if (getGolesL(*p) == -1) {
            setEstado(*p, EN_JUEGO);
            setGolesL(*p, 0);
            setGolesV(*p, 0);
            cout << "Partido en juego\n" << endl;
         }
         else if (getEstado(*p) == EN_JUEGO) {
            cout << "El partido ya se esta jugando" << endl;
         }
         else {
            cout << "El partido ya se jugo" << endl;
         }

         cout << toString(*p)<<endl;
         cout <<endl;
      }
   }
}

/*-------------------------------------------------------------------------*/
void golesPartido(Sistema &sistema) {
   int id = 0;

   cout << "Ingrese id del partido: ";
   cin >> id;

   Partido *p = traerPartido(sistema, id);

   if (getEstado(*p) == EN_JUEGO) {
      cout << getId(*getEquipoL(*p)) << ": " << getNombre(*getEquipoL(*p)) << endl;
      cout << getId(*getEquipoV(*p))<< ": " << getNombre(*getEquipoV(*p)) << endl;

      id = 0;
      while (getId(*getEquipoL(*p)) != id && getId(*getEquipoV(*p)) != id) {
         cout << "\nIngrese id del equipo: ";
         cin >> id;
      }

      Equipo *e = traerEquipo(sistema, id);
      PtrNodoLista cursor = primero(*e->jugadores);

      while (cursor != fin()) {
```

```
        cout << getId(*(Jugador*)cursor->ptrDato) << ": " <<
getNombre(*(Jugador*)cursor->ptrDato) << endl;
        cursor = siguiente(*e->jugadores, cursor);
    }

    id = 0;
    while (traerEquipoPorJugador(sistema, id) != e) {
        cout << "\nIngrese id del jugador: ";
        cin >> id;
    }

    Jugador *j = traerJugador(sistema, id);

    setGoles(*j, getGoles(*j) + 1);
    setGolesAFavor(*e, getGolesAFavor(*e) + 1);

    if (equals(*e, *getEquipoL(*p))){
        setGolesL(*p, getGolesL(*p) + 1);
        setGolesEnContra(*getEquipoV(*p), getGolesEnContra(*getEquipoV(*p)) + 1);
    }
    else {
        setGolesV(*p, getGolesV(*p) + 1);
        setGolesEnContra(*getEquipoL(*p), getGolesEnContra(*getEquipoL(*p)) + 1);
    }

    string frase = "\nGOOOOOOOOOOOOOOOOL de " + getNombre(*e) + ": " +
getNombre(*j);
    int i = 0;

    while(frase[i] != '\0') {
        cout << frase[i];
        i++;
        Sleep(90);
    }
    cout << endl;

    /**Mensaje de goles*/
    int sumgol=0;
    cursor=primero(*sistema.equipos);
    while(cursor!=fin()){
        sumgol=sumgol+getGolesAFavor(*(Equipo*)cursor->ptrDato);
        cursor=siguiente(*sistema.equipos,cursor);
    }
    if(sumgol<100 && sumgol%10==0){
        cout<<"ATENCION!!!"<<endl;
```

```
        cout<<"Se han alcanzado los "<<sumgol<<" goles en el mundial."<<endl;
      }
      if(sumgol>=100 && sumgol%100==0){
        cout<<"ATENCION!!!"<<endl;
        cout<<"Se han alcanzado los "<<sumgol<<" goles en el mundial."<<endl;
      }

    }
    else if (getEstado(*p) == SIN_COMENZAR){
      cout << "El partido no se esta jugando" << endl;
    }
    else {
      cout << "El partido ya se jugo" << endl;
    }
}

/*-------------------------------------------------------------------------*/
void finPartido(Sistema &sistema) {
    int id;

    cout << "Ingrese id del partido: ";
    cin >> id;

    Partido *p = traerPartido(sistema, id);

    if (getEstado(*p) == EN_JUEGO) {
      setEstado(*p, FINALIZADO);

      if (getId(*p) <= 48) {
        if (getGolesL(*p) > getGolesV(*p)) {
          setPuntos(*getEquipoL(*p), getPuntos(*getEquipoL(*p)) + 3);
        }
        else if (getGolesL(*p) == getGolesV(*p)) {
          setPuntos(*getEquipoL(*p), getPuntos(*getEquipoL(*p)) + 1);
          setPuntos(*getEquipoV(*p), getPuntos(*getEquipoV(*p)) + 1);
        }
        else {
          setPuntos(*getEquipoV(*p), getPuntos(*getEquipoV(*p)) + 3);
        }
      }

      cout << "Partido finalizado\n" << endl;
      cout << toString(*p) << endl;
    }
    else {
```

```
      cout << "El partido no estaba en juego" << endl;
   }
}

/*-------------------------------------------------------------------------*/
void mostrarPartidosEnCurso(Sistema &sistema) {
   PtrNodoLista cursor = primero(*sistema.partidos);

   while(cursor != fin()) {
      if (getEstado(*(Partido*)cursor->ptrDato) == EN_JUEGO) {
         cout << toString(*(Partido*)cursor->ptrDato)<< endl;
      }
      cursor = siguiente(*sistema.partidos, cursor);
   }
}

/*-------------------------------------------------------------------------*/
void porcentajeGoles(Sistema &sistema) {
   PtrNodoLista cursor = primero(*sistema.partidos);
   int golesL = 0;
   int golesV = 0;

   while(cursor != fin()) {
      if (getGolesL(*(Partido*)cursor->ptrDato) > -1 && getGolesV(*(Partido*)cursor-
>ptrDato) > -1) {
         golesL += getGolesL(*(Partido*)cursor->ptrDato);
         golesV += getGolesV(*(Partido*)cursor->ptrDato);
      }
      cursor = siguiente(*sistema.partidos, cursor);
   }
   if (golesL > 0 || golesV > 0) {
      cout << "Porcentaje de goles Locales: " << golesL * 100 / (golesL + golesV) << endl;
      cout << "Goles Locales: " << golesL << endl;
      cout << "\nPorcentaje de goles Visitantes: " << golesV * 100 / (golesL + golesV) <<
endl;
      cout << "Goles Visitantes: " << golesV << endl;
   }
   else
      cout << "Todavia no hubo goles" << endl;
}

/*-------------------------------------------------------------------------*/
void grupoDeLaMuerte(Sistema &sistema) {
   PtrNodoLista cursor = primero(*sistema.grupos);
   char id = 'A';
```

```
  while(cursor != fin()) {
    if (golesPorGrupo(*traerGrupo(sistema, id)) < golesPorGrupo(*(Grupo*)cursor-
>ptrDato))
        id = getId(*(Grupo*)cursor->ptrDato);
    cursor = siguiente(*sistema.grupos, cursor);
  }
  cout << "Grupo de la muerte: " << getNombre(*traerGrupo(sistema, id)) << endl;
}
```

Lista.h

```
/* TDA Lista
 * Implementación Simplemente Enlazada
 * Archivo : Lista.h
 * Versión : 1.1
 */

#ifndef __LISTA_H__
#define __LISTA_H__

#ifndef NULL
#define NULL    0
#endif

/*********************************************************************
********/
/* Definiciones de Tipos de Datos */
/*-------------------------------*/

/* tipo enumerado para realizar comparaciones */
enum ResultadoComparacion {
  MAYOR,
  IGUAL,
  MENOR
};
/* Tipo de Informacion que esta contenida en los Nodos de la
   Lista, identificada como Dato. */
typedef void* PtrDato;

/* Tipo de Estructura de los Nodos de la Lista. */
struct NodoLista {
    PtrDato ptrDato; // dato almacenado
    NodoLista* sgte; // puntero al siguiente
};

typedef ResultadoComparacion (*PFComparacion)(PtrDato , PtrDato);

/* Tipo de Puntero a los Nodos de la Lista, el cual se usa para recorrer
   la Lista y acceder a sus Datos. */
typedef NodoLista* PtrNodoLista;


/* Tipo de Estructura de la Lista */
struct Lista{
```

```
   PtrNodoLista primero;     // puntero al primer nodo de la lista
   PFComparacion compara;
};


/*************************************************************************
********/
/* Definicion de 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 crearLista(Lista &lista, PFComparacion compara );

/*-------------------------------------------------------------------------*/
/*
  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 listaVacia(Lista &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.
*/
PtrNodoLista fin();

/*-------------------------------------------------------------------------*/
/*
  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.
*/
PtrNodoLista primero(Lista &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.
*/
PtrNodoLista siguiente(Lista &lista, PtrNodoLista 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.
*/
PtrNodoLista anterior(Lista &lista, PtrNodoLista 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.
*/
PtrNodoLista ultimo(Lista &lista);

/*-------------------------------------------------------------------------*/
/*
 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.
*/
PtrNodoLista adicionarPrincipio(Lista &lista, PtrDato ptrDato);

/*-------------------------------------------------------------------------*/
/*
  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.
*/
PtrNodoLista adicionarDespues(Lista &lista, PtrDato ptrDato, PtrNodoLista 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.
*/
PtrNodoLista adicionarFinal(Lista &lista, PtrDato ptrDato);

/*-------------------------------------------------------------------------*/
/*
  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.
*/
PtrNodoLista adicionarAntes(Lista &lista, PtrDato ptrDato, PtrNodoLista 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(Lista &lista, PtrNodoLista 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(Lista &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(Lista &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(Lista &lista);
```

```
/*************************************************************************
********/
/* Definición de Operaciones Adicionales */
/*--------------------------------------*/

/*
 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().
*/
PtrNodoLista localizarDato(Lista &lista , PtrDato ptrDato);

/*-------------------------------------------------------------------------*/
/*
 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.
*/
PtrNodoLista insertarDato(Lista &lista, PtrDato ptrDato);

/*-------------------------------------------------------------------------*/
/*
 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(Lista &lista, PtrDato ptrDato);

/*-------------------------------------------------------------------------*/
/*
 pre : la lista fue creada con crearLista().
 post : reordena la lista.
```

```
  lista : lista sobre la cual se invoca la primitiva.
*/
void reordenar(Lista &lista);

/*-------------------------------------------------------------------------*/
/*
  pre : la lista fue creada con crearLista().
  post : devuelve la cantidad de datos que tiene la lista.

  lista : lista sobre la cual se invoca la primitiva.
*/
int longitud(Lista &lista);

#endif
```

Lista.cpp

```
/* TDA Lista
 * Implementación Simplemente Enlazada
 * Archivo : Lista.cpp
 * Versión : 1.1
 */


#include "Lista.h"

/************************************************************************
********/
/* Definición de Tipos de Datos para manejo interno */
/*----------------------------------------------*/

/************************************************************************
********/
/* Implementación de Primitivas */
/*-----------------------------*/

void crearLista(Lista &lista, PFComparacion compara) {
  lista.primero = fin();
  lista.compara = compara;
}

/*-------------------------------------------------------------------------*/
bool listaVacia(Lista &lista) {

  return (primero(lista) == fin());
}

/*-------------------------------------------------------------------------*/
PtrNodoLista fin() {
  return NULL;
}

/*-------------------------------------------------------------------------*/
PtrNodoLista primero(Lista &lista) {
  return lista.primero;
}

/*-------------------------------------------------------------------------*/
PtrNodoLista siguiente(Lista &lista, PtrNodoLista ptrNodo) {
```

```
 /* verifica si la lista está vacia o si ptrNodo es el último */
 if ((! listaVacia(lista)) && (ptrNodo != fin()))
   return ptrNodo->sgte;
 else
   return fin();
}

/*-------------------------------------------------------------------------*/
PtrNodoLista anterior(Lista &lista, PtrNodoLista ptrNodo) {

 PtrNodoLista ptrPrevio = fin();
 PtrNodoLista ptrCursor = primero(lista);

 while (( ptrCursor != fin()) && (ptrCursor != ptrNodo)) {
   ptrPrevio = ptrCursor;
   ptrCursor = siguiente(lista,ptrCursor);
 }
 return ptrPrevio;
}

/*-------------------------------------------------------------------------*/
PtrNodoLista ultimo(Lista &lista) {

 /* el último nodo de la lista es el anterior al fin() */
 return anterior(lista,fin());
}

/*-------------------------------------------------------------------------*/
PtrNodoLista crearNodoLista(PtrDato ptrDato) {

 /* reserva memoria para el nodo y luego completa sus datos */
 PtrNodoLista ptrAux = new NodoLista;

 ptrAux->ptrDato = ptrDato;
 ptrAux->sgte = fin();

 return ptrAux;
}

/*-------------------------------------------------------------------------*/
PtrNodoLista adicionarPrincipio(Lista &lista, PtrDato ptrDato) {

 /* crea el nodo */
 PtrNodoLista ptrNuevoNodo = crearNodoLista(ptrDato);
```

```
 /* lo incorpora al principio de la lista */
 ptrNuevoNodo->sgte = lista.primero;
 lista.primero = ptrNuevoNodo;

 return ptrNuevoNodo;
}

/*-------------------------------------------------------------------------*/
PtrNodoLista adicionarDespues(Lista &lista, PtrDato dato, PtrNodoLista ptrNodo) {

 PtrNodoLista ptrNuevoNodo = fin();

 /* si la lista está vacia se adiciona la principio */
 if (listaVacia(lista))
  ptrNuevoNodo = adicionarPrincipio(lista,dato);

 else {
  if (ptrNodo != fin()) {

   /* crea el nodo y lo intercala en la lista */
   ptrNuevoNodo = crearNodoLista(dato);

   ptrNuevoNodo->sgte = ptrNodo->sgte;
   ptrNodo->sgte = ptrNuevoNodo;
  }
 }
 return ptrNuevoNodo;
}

/*-------------------------------------------------------------------------*/
PtrNodoLista adicionarFinal(Lista &lista, PtrDato dato) {

 /* adiciona el dato después del último nodo de la lista */
 return adicionarDespues(lista,dato,ultimo(lista));
}

/*-------------------------------------------------------------------------*/
PtrNodoLista adicionarAntes(Lista &lista, PtrDato dato, PtrNodoLista ptrNodo) {

 PtrNodoLista ptrNuevoNodo = fin();

 if (! listaVacia(lista)) {
  if (ptrNodo != primero(lista))
   ptrNuevoNodo = adicionarDespues(lista,dato,anterior(lista,ptrNodo));
```

```
   else
     ptrNuevoNodo = adicionarPrincipio(lista,dato);
   }
   return ptrNuevoNodo;
}

/*-------------------------------------------------------------------------*/
void eliminarNodo(Lista &lista, PtrNodoLista ptrNodo) {

  PtrNodoLista ptrPrevio;

  /* verifica que la lista no esté vacia y que nodo no sea fin*/
  if ((! listaVacia(lista)) && (ptrNodo != fin())) {

    if (ptrNodo == primero(lista))
      lista.primero = siguiente(lista,primero(lista));

    else {
      ptrPrevio = anterior( lista , ptrNodo );
      ptrPrevio->sgte = ptrNodo->sgte;
    }
    // Si el dato es un TDA, acá habría que llamar al destructor.

    delete ptrNodo;
  }
}

/*-------------------------------------------------------------------------*/
void eliminarNodoPrimero(Lista &lista) {

  if (! listaVacia(lista))
    eliminarNodo(lista,primero(lista));
}

/*-------------------------------------------------------------------------*/
void eliminarNodoUltimo(Lista &lista) {

  if (! listaVacia(lista))
    eliminarNodo(lista,ultimo(lista));
}

/*-------------------------------------------------------------------------*/
void eliminarLista(Lista &lista) {

  /* retira uno a uno los nodos de la lista */
```

```
  while (! listaVacia(lista))
    eliminarNodo(lista,primero(lista));
}

/*-------------------------------------------------------------------------*/
PtrNodoLista localizarDato(Lista &lista, PtrDato ptrDato) {

   bool encontrado = false;
   PtrNodoLista ptrCursor = primero(lista);

  /* recorre los nodos hasta llegar al último o hasta
     encontrar el nodo buscado */
  while ((ptrCursor != fin()) && (! encontrado)) {

    if (lista.compara(ptrCursor->ptrDato,ptrDato) == IGUAL)
      encontrado = true;
    else
      ptrCursor = siguiente(lista,ptrCursor);

  }

  /* si no lo encontró devuelve fin */
  if (! encontrado)
    ptrCursor = fin();

  return ptrCursor;
}

/*-------------------------------------------------------------------------*/
void eliminarDato(Lista &lista, PtrDato ptrDato) {

  /* localiza el dato y luego lo elimina */
  PtrNodoLista ptrNodo = localizarDato(lista,ptrDato);
  if (ptrNodo != fin())
    eliminarNodo(lista,ptrNodo);
}

/*-------------------------------------------------------------------------*/
PtrNodoLista insertarDato(Lista &lista, PtrDato ptrDato) {

  PtrNodoLista ptrPrevio = primero(lista);
  PtrNodoLista ptrCursor = primero(lista);
  PtrNodoLista ptrNuevoNodo;
  bool ubicado = false;
```

```
  /* recorre la lista buscando el lugar de la inserción */
  while ((ptrCursor != fin()) && (! ubicado)) {

    if (lista.compara(ptrCursor->ptrDato,ptrDato) == MAYOR)
      ubicado = true;

    else {
      ptrPrevio = ptrCursor;
      ptrCursor = siguiente(lista,ptrCursor);
    }
  }

  if (ptrCursor == primero(lista))
    ptrNuevoNodo = adicionarPrincipio(lista,ptrDato);
  else
    ptrNuevoNodo = adicionarDespues(lista,ptrDato,ptrPrevio);

  return ptrNuevoNodo;
}

/*-------------------------------------------------------------------------*/

void reordenar(Lista &lista) {

  Lista temp = lista;
  PtrNodoLista ptrCursor = primero(temp);
  crearLista(lista, lista.compara);
  while ( ptrCursor != fin() ) {
       insertarDato( lista, ptrCursor->ptrDato );
       eliminarNodo( temp, ptrCursor );
       ptrCursor = primero(temp);
  }
  eliminarLista( temp );
}


/*-------------------------------------------------------------------------*/

int longitud(Lista &lista){
  PtrNodoLista ptrCursor = primero(lista);
  int longitud = 0;
  while ( ptrCursor != fin() ) {
       longitud++;
       ptrCursor = siguiente( lista, ptrCursor);
  }
```

```
 return longitud;
}

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

Cronograma de división de tareas

Tareas hasta la segunda entrega:

La mayor cantidad del tp fue elaborado de forma totalmente grupal dado a la cercanía de nuestras viviendas, horarios compatibles de trabajo y afinidad con el uso del Skype. Las únicas tareas que se individualizaron en la segunda parte del tp, fueron:

- **José Victor Ibañez:** Generar el análisis y diseño, validaciones, reportes, txt con datos oficiales de la fifa.
- **Nicolas Nahuel Trezza:** Creación de TDA, listas, menú y reportes.
- **Martin Olmos:** Inicio de partidos, llaves, reportes.
- **José Maximiliano Lucero:** Reporte de 10 a 100 goles, validaciónes, carga de txt, reportes.
- **Griselda Benítez Haugg:** Generar el análisis y diseño, validaciones.