#ifndef CLUB\_H\_INCLUDED

#define CLUB\_H\_INCLUDED

#include <string>

using namespace std;

typedef struct {

int idClub;

string nombre;

} Club;

void constructor(Club &club);

void destructor(Club &club);

void setIdClub(Club &club, int idClub);

int getIdClub(Club &club);

string getNombre(Club &club);

void setNombre(Club &club, string nombre);

#endif // CLUB\_H\_INCLUDED

#ifndef EQUIPO\_H\_INCLUDED

#define EQUIPO\_H\_INCLUDED

#include <string>

#include "ListaJugadores.h"

using namespace std;

typedef struct {

int idEquipo;

string nombreEquipo;

string nombreUsuario;

float monto;

ListaJ ARQ;

ListaJ DEF;

ListaJ MED;

ListaJ DEL;

} Equipo;

void constructor(Equipo &equipo);

void destructor(Equipo &equipo);

void setIdEquipo(Equipo &equipo, int idEquipo);

int getIdEquipo(Equipo &equipo);

string getNombreEquipo(Equipo &equipo);

void setNombreEquipo(Equipo &equipo, string nombreEquipo);

string getNombreUsuario(Equipo &equipo);

void setNombreUsuario(Equipo &equipo, string nombreUsuario);

float getMonto(Equipo &equipo);

void setMonto(Equipo &equipo, float monto);

ListaJ& getListaARQ(Equipo &equipo);

void setListaARQ(Equipo &equipo, ListaJ &ARQ);

ListaJ& getListaDEF(Equipo &equipo);

void setListaDEF(Equipo &equipo, ListaJ &DEF);

ListaJ& getListaMED(Equipo &equipo);

void setListaMED(Equipo &equipo, ListaJ &MED);

ListaJ& getListaDEL(Equipo &equipo);

void setListaDEL(Equipo &equipo, ListaJ &DEL);

#endif // EQUIPO\_H\_INCLUDED

#ifndef JUGADOR\_H\_INCLUDED

#define JUGADOR\_H\_INCLUDED

#include <string>

using namespace std;

enum Posicion{

INDEF,

ARQ,

DEF,

MED,

DEL

};

typedef struct {

int idJugador;

string nombre;

Posicion posicion;

float valor;

} Jugador;

void constructor(Jugador &jugador);

void destructor(Jugador &jugador);

void setIdJugador(Jugador &jugador, int idJugador);

int getIdJugador(Jugador &jugador);

string getNombre(Jugador &jugador);

void setNombre(Jugador &jugador, string nombre);

Posicion& getPosicion(Jugador &jugador);

void setPosicion(Jugador &jugador, Posicion posicion);

float getValor(Jugador &jugador);

void setValor(Jugador &jugador, float valor);

#endif // CLUB\_H\_INCLUDED

#ifndef PUNTOSFECHA\_H\_INCLUDED

#define PUNTOSFECHA\_H\_INCLUDED

typedef struct {

int idJugador;

int fecha;

int puntos;

} PuntosFecha;

void constructor(PuntosFecha &puntosFecha);

void destructor(PuntosFecha &puntosFecha);

void setIdJugador(PuntosFecha &puntosFecha, int idJugador);

int getIdJugador(PuntosFecha &puntosFecha);

void setFecha(PuntosFecha &puntosFecha, int fecha);

int getFecha(PuntosFecha &puntosFecha);

int getPuntos(PuntosFecha &puntosFecha);

#endif // EQUIPO\_H\_INCLUDED

#include "Club.h"

void constructor(Club &club){

club.idClub = 0;

club.nombre = "";

}

void destructor(Club &club){

}

int getIdClub(Club &club){

return club.idClub;

}

void setIdClub(Club &club, int idClub){

club.idClub = idClub;

}

string getNombre(Club &club){

return club.nombre;

}

void setNombre(Club &club, string nombre ){

club.nombre = nombre;

}

#include "Equipo.h"

#include <string>

#include "ListaEquipos.h"

using namespace std;

void constructor(Equipo &equipo){

equipo.idEquipo = 0;

equipo.nombreEquipo = "";

equipo.nombreUsuario = "";

equipo.monto = 0;

crearLista(equipo.ARQ);

crearLista(equipo.DEF);

crearLista(equipo.MED);

crearLista(equipo.DEL);

}

void destructor(Equipo &equipo){}

int getIdEquipo(Equipo &equipo){

return equipo.idEquipo;

}

void setIdEquipo(Equipo &equipo, int idEquipo){

equipo.idEquipo = idEquipo;

}

string getNombreEquipo(Equipo &equipo){

return equipo.nombreEquipo;

}

void setNombreEquipo(Equipo &equipo, string nombreEquipo){

equipo.nombreEquipo = nombreEquipo;

}

string getNombreUsuario(Equipo &equipo){

return equipo.nombreUsuario;

}

void setNombreUsuario(Equipo &equipo, string nombreUsuario){

equipo.nombreUsuario = nombreUsuario;

}

float getMonto(Equipo &equipo){

return equipo.monto;

}

void setMonto(Equipo &equipo, float monto){

equipo.monto = monto;

}

ListaJ& getListaARQ(Equipo &equipo){

return equipo.ARQ;

}

void setListaARQ(Equipo &equipo, ListaJ &ARQ){

equipo.ARQ = ARQ;

}

ListaJ& getListaDEF(Equipo &equipo){

return equipo.DEF;

}

void setListaDEF(Equipo &equipo, ListaJ &DEF){

equipo.DEF = DEF;

}

ListaJ& getListaMED(Equipo &equipo){

return equipo.MED;

}

void setListaMED(Equipo &equipo, ListaJ &MED){

equipo.MED = MED;

}

ListaJ& getListaDEL(Equipo &equipo){

return equipo.DEL;

}

void setListaDEL(Equipo &equipo, ListaJ &DEL){

equipo.DEL = DEL;

}

#include <string>

#include "Jugador.h"

void constructor(Jugador &jugador){

jugador.idJugador = 0;

jugador.posicion = INDEF;

jugador.nombre = "";

jugador.valor = 0;

}

void destructor(Jugador &jugador){}

void setIdJugador(Jugador &jugador, int idJugador){

jugador.idJugador = idJugador;

}

int getIdJugador(Jugador &jugador){

return jugador.idJugador;

}

string getNombre(Jugador &jugador){

return jugador.nombre;

}

void setNombre(Jugador &jugador, string nombre){

jugador.nombre = nombre;

}

Posicion& getPosicion(Jugador &jugador){

return jugador.posicion;

}

void setPosicion(Jugador &jugador, Posicion posicion){

jugador.posicion = posicion;

}

float getValor(Jugador &jugador){

return jugador.valor;

}

void setValor(Jugador &jugador, float valor){

jugador.valor = valor;

}

#include "PuntosFecha.h"

void constructor(PuntosFecha &puntosFecha){

int idJugador = 0;

int fecha = 0;

int puntos = -100;

}

void destructor(PuntosFecha &puntosFecha){}

void setIdJugador(PuntosFecha &puntosFecha, int idJugador){

puntosFecha.idJugador = idJugador;

}

int getIdJugador(PuntosFecha &puntosFecha){

return puntosFecha.idJugador;

}

void setFecha(PuntosFecha &puntosFecha, int fecha){

puntosFecha.fecha = fecha;

}

int getFecha(PuntosFecha &puntosFecha){

return puntosFecha.fecha;

}

void setPuntos(PuntosFecha &puntosFecha, int puntos){

puntosFecha.puntos = puntos;

}

int getPuntos(PuntosFecha &puntosFecha){

return puntosFecha.puntos;

}

**Programa principal:**

struct Configuracion{

float ValorMaximoXEquipo;

int maxArq;

int maxDef;

int maxMed;

int maxDel;

};

struct PuntosEquipo{

int idEquipo;

string nomEquipo;

string nomUsuario;

int fecha;

int puntos;

};

Configuracion cargarConfig(FILE \*pf);

void cargarClubes(ListaC &listaClubes, FILE \*pf, FILE \*errores);

void cargarJugadores(ListaJ &listaJugadores, ListaC &listaClubes, FILE \*pf, FILE \*errores);

void cargarEquipos(ListaE &listaEquipos, FILE \*pf, FILE \*errores);

void cargarJugadoresPorEquipo(ListaE &listaEquipos, ListaJ &listaJugadores, FILE \*pf, FILE \*errores, Configuracion config);

void cargarPuntosPorJugador(ListaE &listaEquipos, ListaJ &listaJugadores, Cola &colaPuntosFecha, FILE \*pf, FILE \*errores);

// METODOS DE PRUEBA

bool traerClubes(ListaC &listaClubes);

bool traerJugadores(ListaJ &listaJugadores);

bool traerEquipos(ListaE &listaEquipos);

bool traerJugadorePorEquipo(ListaE &listaEquipos);

// METODOS DE VALIDACION

bool validarJugador(Equipo &equipo, Configuracion config, FILE \*errores, Posicion posicion);

bool validarValorEquipo(Equipo &equipo, Configuracion config, FILE \*errores);

// METODO DE ORDENAMIENTO E IMPRESION DE ARCHIVOS

void imprimirOrdenado(PuntosEquipo puntosEquipo[], int ultimaFecha, ListaE &listaEquipos, ListaJ &listaJugadores, Cola &colaPuntosFecha);

int main()

{

setlocale(LC\_ALL, "");

ListaC listaClubes;

crearLista(listaClubes);

ListaJ listaJugadores;

crearLista(listaJugadores);

ListaE listaEquipos;

crearLista(listaEquipos);

Cola colaPuntosFecha;

crearCola(colaPuntosFecha);

Configuracion config;

FILE \*errores;

FILE \*pf;

config = cargarConfig(pf);

int opcion = 0;

while(opcion!=6){

system("cls");

cout << "Ingrese el archivo del que desea cargar los datos: " << endl;

cout << "1. Clubes.db" << endl;

cout << "2. Jugadores.db" << endl;

cout << "3. Equipos.db" << endl;

cout << "4. JugadoresPorEquipo.db" << endl;

cout << "5. PuntosPorJugador.db" << endl;

cout << "6. Salir" << endl;

cin >> opcion;

switch(opcion){

case 1:

cargarClubes(listaClubes, pf, errores); //TERMINADO

break;

case 2:

cargarJugadores(listaJugadores, listaClubes, pf, errores); //TERMINADO

break;

case 3:

cargarEquipos(listaEquipos, pf, errores); // TERMINADO

break;

case 4:

cargarJugadoresPorEquipo(listaEquipos, listaJugadores, pf, errores, config);

break;

case 5:

cargarPuntosPorJugador(listaEquipos, listaJugadores, colaPuntosFecha, pf, errores);

break;

case 6:

break;

case 11:

traerClubes(listaClubes);

break;

case 12:

traerJugadores(listaJugadores);

break;

case 13:

traerEquipos(listaEquipos);

break;

case 14:

traerJugadorePorEquipo(listaEquipos);

break;

}

system("PAUSE");

}

eliminarLista(listaClubes);

eliminarLista(listaJugadores);

eliminarLista(listaEquipos);

eliminarCola(colaPuntosFecha);

return 0;

}

**Procedimientos y funciones llamados por el programa principal:**

**Configuracion cargarConfig(FILE \*pf)**{

Configuracion aux;

if ((pf = fopen("configuracion.conf", "r")) == NULL){

cout << "Error al abrir el archivo de configuracion" << endl;

Sleep(2000);

exit(0);

}

fscanf(pf, "ValorMaximoXEquipo=%f\n", &aux.ValorMaximoXEquipo);

fscanf(pf, "ARQ=%d\n", &aux.maxArq);

fscanf(pf, "DEF=%d\n", &aux.maxDef);

fscanf(pf, "MED=%d\n", &aux.maxMed);

fscanf(pf, "DEL=%d\n", &aux.maxDel);

fclose(pf);

return aux;

}

**void cargarClubes(ListaC &listaClubes, FILE \*pf, FILE \*errores)**{

Club club;

constructor(club);

int xIdClub;

char xNombre[60];

if ((errores = fopen("errores.txt", "a")) == NULL){

cout << "Error al abrir el archivo errores.txt" << endl;

Sleep(2000);

exit(0);

}

if ((pf = fopen("Clubes.db", "r")) == NULL){

cout << "Error al abrir el archivo Clubes.db" << endl;

Sleep(2000);

exit(0);

}

while(!feof(pf)){

fscanf(pf, "%d;%[A-Z a-z 0-9 .' ä-ü á-ú]\n", &xIdClub, xNombre);

setIdClub(club, xIdClub);

setNombre(club, xNombre);

if(localizarDato(listaClubes, club) == NULL){

adicionarFinal(listaClubes, club);

}

else{

cout << "El club " << getIdClub(club) << " ya existe" << endl;

fprintf(errores, "E100-0 | El club %d ya existe.\n", getIdClub(club));

}

}

destructor(club);

fclose(pf);

fclose(errores);

cout << "Se cargaron los clubes correctamente" << endl;

}

**void cargarJugadores(ListaJ &listaJugadores, ListaC &listaClubes, FILE \*pf, FILE \*errores)**{

Jugador jugador;

constructor(jugador);

Club club;

constructor(club);

bool seCargoUno = false;

int xIdJugador, xIdClub;

float xValor;

char xNombre[110];

char xPosicion[3];

PtrNodoListaC ptr;

PtrNodoListaJ ptrJ = primero(listaJugadores);

if ((errores = fopen("errores.txt", "a")) == NULL){

cout << "Error al abrir el archivo errores.txt" << endl;

Sleep(2000);

exit(0);

}

if ((pf = fopen("Jugadores.db", "r")) == NULL){

cout << "Error al abrir el archivo Jugadores.db" << endl;

Sleep(2000);

exit(0);

}

while(!feof(pf)){

fscanf(pf, "%d;%[A-Z a-z 0-9 .' ä-ü á-ú Á-Ú];%d;%[A-Z a-z 0-9 .' ä-ü á-ú];%f\n", &xIdJugador, xNombre, &xIdClub, xPosicion, &xValor);

setIdJugador(jugador, xIdJugador);

setNombre(jugador, xNombre);

setValor(jugador, xValor);

if(strcmpi(xPosicion,"ARQ") == 0) setPosicion(jugador, ARQ);

if(strcmpi(xPosicion,"DEF") == 0) setPosicion(jugador, DEF);

if(strcmpi(xPosicion,"MED") == 0) setPosicion(jugador, MED);

if(strcmpi(xPosicion,"DEL") == 0) setPosicion(jugador, DEL);

setIdClub(club, xIdClub);

ptr = localizarDato(listaClubes, club);

ptrJ = localizarDato(listaJugadores, jugador); // validar si jugador esta duplicado

if (ptrJ == NULL){

if(ptr!=NULL){

adicionarFinal(listaJugadores, jugador);

seCargoUno = true;

}

else{

cout << "Error: No se cargo el jugador " << getIdJugador(jugador) << " en el club "<< getIdClub(club) << ". Revisar errores.txt" << endl;

fprintf(errores, "E200-0 | No existe el club %d para agregar al jugador %d\n", getIdClub(club), getIdJugador(jugador));

}

}else{ // jugador no existe

cout << "Error: No se cargo el jugador " << getIdJugador(jugador) << " en el club "<< getIdClub(club) << ". Revisar errores.txt" << endl;

fprintf(errores, "E201-0 | El jugador %d se encuentra duplicado\n", getIdJugador(jugador));

}

}

destructor(club);

destructor(jugador);

fclose(pf);

fclose(errores);

if(seCargoUno == true) cout << "Se cargaron los jugadores correctamente." << endl;

if(seCargoUno == false) cout << "No se cargo ningun jugador." << endl;

}

**void cargarEquipos(ListaE &listaEquipos, FILE \*pf, FILE \*errores)**{

Equipo equipo;

constructor(equipo);

int xIdEquipo;

char xNombreEquipo[60];

char xNombreUsuario[60];

if ((errores = fopen("errores.txt", "a")) == NULL){

cout << "Error al abrir el archivo errores.txt" << endl;

Sleep(2000);

exit(0);

}

if ((pf = fopen("Equipos.db", "r")) == NULL){

cout << "Error al abrir el archivo Equipos.db" << endl;

Sleep(2000);

exit(0);

}

while(!feof(pf)){

fscanf(pf, "%d;%[A-Z a-z 0-9];%[A-Z a-z 0-9 .' ä-ü á-ú]\n", &xIdEquipo, xNombreEquipo, xNombreUsuario);

setIdEquipo(equipo, xIdEquipo);

setNombreEquipo(equipo, xNombreEquipo);

setNombreUsuario(equipo, xNombreUsuario);

if(localizarDato(listaEquipos, equipo) == NULL){

adicionarFinal(listaEquipos, equipo);

}

else{

cout << "El equipo " << getIdEquipo(equipo) << " ya existe" << endl;

fprintf(errores, "E300-0 | El equipo %d ya existe\n", getIdEquipo(equipo));

}

}

destructor(equipo);

fclose(pf);

fclose(errores);

cout << "Se cargaron los equipos correctamente" << endl;

}

**void cargarJugadoresPorEquipo(ListaE &listaEquipos, ListaJ &listaJugadores, FILE \*pf, FILE \*errores, Configuracion config)**{

Equipo equipo;

constructor(equipo);

Jugador jugador;

constructor(jugador);

bool equipoValido=0, jugadorValido=0;

int xIdEquipo;

int xIdJugador;

PtrNodoListaE ptrE;

PtrNodoListaJ ptrJ;

if ((errores = fopen("errores.txt", "a")) == NULL){

cout << "Error al abrir el archivo errores.txt" << endl;

Sleep(2000);

exit(0);

}

if ((pf = fopen("JugadoresPorEquipo.db", "r")) == NULL){

cout << "Error al abrir el archivo JugadoresPorEquipo.db" << endl;

Sleep(2000);

exit(0);

}

while(!feof(pf)){

fscanf(pf, "%d;%d\n", &xIdEquipo, &xIdJugador);

setIdEquipo(equipo, xIdEquipo);

setIdJugador(jugador, xIdJugador);

ptrE = localizarDato(listaEquipos, equipo);

ptrJ = localizarDato(listaJugadores, jugador);

if(ptrE && ptrJ != NULL){

obtenerDato(listaEquipos, equipo, ptrE);

obtenerDato(listaJugadores, jugador, ptrJ);

//Seteando monto y validando que no se pase del maximo

setMonto(equipo, (getMonto(equipo))+getValor(jugador));

equipoValido = validarValorEquipo(equipo, config, errores);

if(equipoValido){

if(getPosicion(jugador) == ARQ){

if(localizarDato(getListaARQ(equipo), jugador) == NULL){ // validar si jugador ya esta cargado en el equipo

adicionarFinal(getListaARQ(equipo), jugador);

jugadorValido = validarJugador(equipo, config, errores, ARQ);

}

else{

cout << "ERROR: El jugador " << getIdJugador(jugador) << " ya esta cargado en el equipo: " << getIdEquipo(equipo) << endl;

fprintf(errores, "E400-1 | El jugador %d ya esta cargado en el equipo %d\n", getIdJugador(jugador), getIdEquipo(equipo));

}

}

if(getPosicion(jugador) == DEF){

if(localizarDato(getListaDEF(equipo), jugador) == NULL){ // validar si jugador ya esta cargado en el equipo

adicionarFinal(getListaDEF(equipo), jugador);

jugadorValido = validarJugador(equipo, config, errores, DEF);

}

else{

cout << "ERROR: El jugador " << getIdJugador(jugador) << " ya esta cargado en el equipo: " << getIdEquipo(equipo) << endl;

fprintf(errores, "E400-2 | El jugador %d ya esta cargado en el equipo %d\n", getIdJugador(jugador), getIdEquipo(equipo));

}

}

if(getPosicion(jugador) == MED){

if(localizarDato(getListaMED(equipo), jugador) == NULL){ // validar si jugador ya esta cargado en el equipo

adicionarFinal(getListaMED(equipo), jugador);

jugadorValido = validarJugador(equipo, config, errores, MED);

}

else{

cout << "ERROR: El jugador " << getIdJugador(jugador) << " ya esta cargado en el equipo: " << getIdEquipo(equipo) << endl;

fprintf(errores, "E400-3 | El jugador %d ya esta cargado en el equipo %d\n", getIdJugador(jugador), getIdEquipo(equipo));

}

}

if(getPosicion(jugador) == DEL){

if(localizarDato(getListaDEL(equipo), jugador) == NULL){ // validar si jugador ya esta cargado en el equipo

adicionarFinal(getListaDEL(equipo), jugador);

jugadorValido = validarJugador(equipo, config, errores, DEL);

}

else{

cout << "ERROR: El jugador " << getIdJugador(jugador) << " ya esta cargado en el equipo: " << getIdEquipo(equipo) << endl;

fprintf(errores, "E400-4 | El jugador %d ya esta cargado en el equipo %d\n", getIdJugador(jugador), getIdEquipo(equipo));

}

}

if(jugadorValido) colocarDato(listaEquipos, equipo, ptrE);

}

if(!equipoValido || !jugadorValido) eliminarDato(listaEquipos, equipo);

}

else { //errores.txt

if(ptrE == NULL){

printf("ERROR: El equipo con numero %d no existe. Revisar errores.txt\n", getIdEquipo(equipo));

fprintf(errores, "E403-0 | El equipo con numero %d no existe\n", getIdEquipo(equipo));

}

if(ptrJ == NULL){

printf("ERROR: El jugador con numero %d no existe. Revisar errores.txt\n", getIdJugador(jugador));

fprintf(errores, "E404-0 | El jugador con numero %d no existe. Revisar Jugadores.db\n", getIdJugador(jugador));

}

}

}

destructor(jugador);

destructor(equipo);

fclose(errores);

fclose(pf);

cout << "Se cargaron los jugadores por equipo correctamente" << endl;

}

**void cargarPuntosPorJugador(ListaE &listaEquipos, ListaJ &listaJugadores, Cola &colaPuntosFecha, FILE \*pf, FILE \*errores)**{

PuntosFecha puntosFecha;

constructor(puntosFecha);

int flagFecha = 0;

int i=0;

Equipo equipo;

constructor(equipo);

Jugador jugador;

constructor(jugador);

PuntosEquipo puntoEquipo;

PtrNodoListaE ptrLista;

ptrNodoCola ptrUltimo;

ptrNodoCola ptrCola;

PtrNodoListaJ ptrJugador = NULL;

string aux, aux2;

int fechaAux=1; //ACA SE DEBE PONER LA FECHA EN LA CUAL INICIA EL TORNEO

int xIdJugador, xFecha, xPuntos;

if ((pf = fopen("PuntosPorJugador.db", "r")) == NULL){

cout << "Error al abrir el archivo Equipos.db" << endl;

Sleep(2000);

exit(0);

}

while(!feof(pf)){ // Cargando cola para trabajar directamente de ella

fscanf(pf,"%d;%d;%d", &xIdJugador, &xFecha, &xPuntos);

setIdJugador(puntosFecha, xIdJugador);

setFecha(puntosFecha, xFecha);

setPuntos(puntosFecha, xPuntos);

encolar(colaPuntosFecha, puntosFecha);

}

ptrUltimo = ultimo(colaPuntosFecha);

puntosFecha = getDato(ptrUltimo);

int ultimaFecha = getFecha(puntosFecha);

PuntosEquipo puntosEquipo[(longitud(listaEquipos)\*ultimaFecha)]; //CREANDO ARRAY DE ESTRUCTURA

while(fechaAux<=ultimaFecha){ //WHILE GENERAL PARA IR RECORRIENDO TODAS LAS FECHAS

puntoEquipo.fecha = fechaAux; //ASIGNO LA FECHA DE LA CUAL SE CONSULTAN LOS PUNTAJES DE LOS EQUIPOS

/////////////SE CREA EL ARCHIVO DE LA FECHA//////////////////

/\*aux = static\_cast<std::ostringstream\*>(&(std::ostringstream() << fechaAux))->str();

aux2 = "PuntosFecha"+aux+".txt";

const char \* nombreArchivo = aux2.c\_str();

if ((pf = fopen(nombreArchivo,"a")) == NULL){

cout << "Error al abrir el archivo Fecha" << fechaAux << ".db" << endl;

Sleep(2000);

exit(0);

}\*/

////////////////////////////////////////////////////////////

ptrLista = primero(listaEquipos);

while(ptrLista!=NULL){

puntoEquipo.puntos = 0;

obtenerDato(listaEquipos, equipo, ptrLista); //TRAIGO AL EQUIPO Y LO METO EN LA INSTANCIA TDA EQUIPO

puntoEquipo.idEquipo = getIdEquipo(equipo); //ASIGNO EL ID DE EQUIPO DEL QUE ESTOY CONSULTANDO EL PUNTAJE

ptrCola = primero(colaPuntosFecha);

flagFecha=0;

while(ptrCola!=NULL){

puntosFecha = getDato(ptrCola); //METO EN LA INSTANCIA PUNTOSFECHA EL PRIMER ELEMENTO DE LA COLA

if(getFecha(puntosFecha) != fechaAux) flagFecha=1; //SI LA FECHA CAMBIO NO LA PROCESA

setIdJugador(jugador, getIdJugador(puntosFecha)); //SETEO EL ID PARA BUSCARLO EN LA LISTA

ptrJugador = localizarDato(listaJugadores, jugador); //LO BUSCO EN LA LISTA Y DEVUELVO EL PUNTERO

if(ptrJugador!=NULL && flagFecha==0){

obtenerDato(listaJugadores, jugador, ptrJugador); //OBTENGO EL JUGADOR Y LO GUARDO EN LA INSTANCIA TDA JUGADOR

ptrJugador=NULL; //SE SETEA EN NULL DE NUEVO PARA BUSCARLO EN LAS LISTAS DEL EQUIPO

if(getPosicion(jugador)==ARQ) ptrJugador=localizarDato(getListaARQ(equipo), jugador);//ES ARQUERO DE ESTE EQUIPO?

if(getPosicion(jugador)==DEF) ptrJugador=localizarDato(getListaDEF(equipo), jugador);//ES DEFENSOR DE ESTE EQUIPO?

if(getPosicion(jugador)==MED) ptrJugador=localizarDato(getListaMED(equipo), jugador);//ES MEDIOCAMPISTA DE ESTE EQUIPO?

if(getPosicion(jugador)==DEL) ptrJugador=localizarDato(getListaDEL(equipo), jugador);//ES DELANTERO DE ESTE EQUIPO?

if(ptrJugador!=NULL && flagFecha==0){

/\*cout << "Fecha: " << fechaAux << " Jugador: ";

cout << getIdJugador(jugador);

cout << " Equipo: ";

cout << puntoEquipo.idEquipo << endl;\*/

puntoEquipo.puntos += getPuntos(puntosFecha); //SUMO EL PUNTAJE DEL JUGADOR SELECCIONADO EN EL EQUIPO

}

}

ptrCola = siguiente(ptrCola); //SE VE EL SIGUIENTE ELEMENTO DE PUNTOSFECHA

flagFecha=0;

} //FIN WHILE PUNTOSFECHA

//IMPRIMIENDO EL ARCHIVO SIN METODO QUE BUGUEE TODO

//imprimirOrdenado(equipo, fechaAux, puntoEquipo, pf);

puntosEquipo[i].idEquipo = puntoEquipo.idEquipo;

puntosEquipo[i].nomEquipo = getNombreEquipo(equipo);

puntosEquipo[i].nomUsuario = getNombreUsuario(equipo);

puntosEquipo[i].fecha = puntoEquipo.fecha;

puntosEquipo[i].puntos = puntoEquipo.puntos;

i++;

ptrLista = siguiente(listaEquipos, ptrLista); //SE ANALIZAN LOS PUNTOS DEL SIGUIENTE EQUIPO

} //FIN WHILE EQUIPOS

fechaAux++;

fclose(pf);

} //FIN WHILE FECHA

imprimirOrdenado(puntosEquipo, ultimaFecha, listaEquipos, listaJugadores, colaPuntosFecha); //LLAMADA A PROCEDIMIENTO PARA ORDENAR E IMPRIMIR LO CAPTURADO

cout << "Se cargaron los puntos por jugador correctamente" << endl;

}

**void imprimirOrdenado(PuntosEquipo puntosEquipo[], int ultimaFecha, ListaE &listaEquipos, ListaJ &listaJugadores, Cola &colaPuntosFecha)**{ //SE IMPRIME Y ORDENA EL PUNTAJE DE CADA EQUIPO EN LA FECHA CORRESPONDIENTE

int i;

int j;

int cursor=0;

FILE \*pf;

FILE \*pf2;

FILE \*pf3;

PuntosEquipo puntosEquipoTotal[longitud(listaEquipos)];

PuntosEquipo auxiliar;

int fechaAux=1;

string aux,aux2;

/\*while(fechaAux<=ultimaFecha){

aux = static\_cast<std::ostringstream\*>(&(std::ostringstream() << fechaAux))->str();

aux2 = "PuntosFecha"+aux+".txt";

const char \* nombreArchivo = aux2.c\_str();

if ((pf = fopen(nombreArchivo,"a")) == NULL){

cout << "Error al abrir el archivo Fecha" << fechaAux << ".txt" << endl;

Sleep(2000);

exit(0);

}\*/

/////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////CARGAS Y ORDENAMIENTOS////////////////////////////////////

//SE ORDENA LA ESTRUCTURA DE ARRAY PARA LOS PUNTOS DE LA FECHA POR EQUIPO (YA CARGADA)

for(i=0 ; i<(longitud(listaEquipos)\*ultimaFecha) ; i++){

for(j=0 ; j<(longitud(listaEquipos)\*ultimaFecha)-1 ; j++){

if(puntosEquipo[j].puntos < puntosEquipo[j+1].puntos && puntosEquipo[j].fecha == puntosEquipo[j+1].fecha){

auxiliar = puntosEquipo[j];

puntosEquipo[j] = puntosEquipo[j+1];

puntosEquipo[j+1] = auxiliar;

}

}

}

//SE CARGA LA ESTRUCTURA DE ARRAY PARA LOS PUNTOS POR EQUIPO EN TODO EL TORNEO (LA FECHA SE IGNORA, =0)

for(i=0 ; i<(longitud(listaEquipos)) ; i++){

for(j=0 ; j<(longitud(listaEquipos)\*ultimaFecha) ; j++){

if(j==i){

puntosEquipoTotal[i].idEquipo = puntosEquipo[j].idEquipo;

puntosEquipoTotal[i].puntos = puntosEquipo[j].puntos;

puntosEquipoTotal[i].nomEquipo = puntosEquipo[j].nomEquipo;

puntosEquipoTotal[i].nomUsuario = puntosEquipo[j].nomUsuario;

puntosEquipoTotal[i].fecha = 0;

}

if(j>i && puntosEquipo[j].idEquipo == puntosEquipoTotal[i].idEquipo){

puntosEquipoTotal[i].puntos+=puntosEquipo[j].puntos; //Se suma el puntaje del resto de fechas

}

}

}

//SE ORDENA LA ESTRUCTURA DE ARRAY PARA LOS PUNTOS POR EQUIPO EN TODO EL TORNEO:

for(i=0 ; i<(longitud(listaEquipos)) ; i++){

for(j=0 ; j<(longitud(listaEquipos))-1 ; j++){

if(puntosEquipoTotal[j].puntos < puntosEquipoTotal[j+1].puntos && puntosEquipoTotal[j].fecha == puntosEquipoTotal[j+1].fecha){

auxiliar = puntosEquipoTotal[j];

puntosEquipoTotal[j] = puntosEquipoTotal[j+1];

puntosEquipoTotal[j+1] = auxiliar;

}

}

}

//SE AGRUPAN TODOS LOS JUGADORES CON SUS PUNTOS TOTALES (FECHA = 0 YA QUE NO IMPORTA)

Cola colaOrdenada;

crearCola(colaOrdenada);

PuntosFecha pfAux;

PuntosFecha pfMax;

PuntosFecha pfCursor;

ptrNodoCola cursorCola = primero(colaPuntosFecha);

ptrNodoCola auxCola = primero(colaPuntosFecha);

int encontrado = 0;

while(cursorCola!=NULL){

pfCursor=getDato(cursorCola);

while(auxCola !=NULL && encontrado==0){

pfAux = getDato(auxCola);

if(getIdJugador(pfCursor)==getIdJugador(pfAux) && cursorCola!=auxCola){

cursorCola->ptrdato.puntos += getPuntos(pfAux);

eliminarNodo(colaPuntosFecha, auxCola);

encontrado = 1;

}

auxCola=siguiente(auxCola);

}

encontrado=0;

auxCola=primero(colaPuntosFecha);

cursorCola=siguiente(cursorCola);

}

ptrNodoCola ptrPruebita = primero(colaPuntosFecha);

/\*while(ptrPruebita!=NULL){

pfAux = getDato(ptrPruebita);

cout << "Jugador: " << getIdJugador(pfAux) << " / Puntos: " << getPuntos(pfAux) << endl;;

ptrPruebita = siguiente(ptrPruebita);

}\*/

//SE ORDENA LA COLA DE JUGADORES POR PUNTAJE

setPuntos(pfMax, -100);

constructor(pfAux);

int flagcontrol;

ptrNodoCola ptrControl;

ptrNodoCola ptrCola = primero(colaPuntosFecha);

ptrNodoCola ptrMax = primero(colaPuntosFecha);

while(!colaVacia(colaPuntosFecha)){

while(ptrCola!=NULL){

pfAux = getDato(ptrCola);

if(getPuntos(pfAux)>getPuntos(pfMax)){

pfMax = pfAux;

ptrMax = ptrCola;

/\*while(ptrCola!=ptrMax){

ptrMax=siguiente(ptrMax);

}\*/

}

ptrCola=siguiente(ptrCola);

}

encolar(colaOrdenada, pfMax);

eliminarNodo(colaPuntosFecha, ptrMax);

ptrCola = primero(colaPuntosFecha);

ptrMax = primero(colaPuntosFecha);

setPuntos(pfMax, -100);

}

/\*ptrNodoCola ptrPruebita = primero(colaOrdenada);

while(ptrPruebita!=NULL){

pfAux = getDato(ptrPruebita);

cout << "Jugador: " << getIdJugador(pfAux) << " / Puntos: " << getPuntos(pfAux) << endl;;

ptrPruebita = siguiente(ptrPruebita);

}\*/

/////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////IMPRESIONES//////////////////////////////////////////

//SE IMPRIME EL ARCHIVO DE EQUIPOS EN TODO EL TORNEO ORDENADO POR PUNTOS

if ((pf2 = fopen("puntajesTotalesPorEquipo.txt","w")) == NULL){

cout << "Error al abrir el archivo puntajesTotalesPorEquipo.txt" << endl;

Sleep(2000);

exit(0);

}

for(i=0 ; i<longitud(listaEquipos) ; i++){

fprintf(pf2,"%d;%d;%s;%s;%d\n", i+1, puntosEquipoTotal[i].idEquipo, puntosEquipoTotal[i].nomEquipo.c\_str(), puntosEquipoTotal[i].nomUsuario.c\_str(), puntosEquipoTotal[i].puntos);

}

fclose(pf2);

//SE IMPRIMEN LOS ARCHIVOS DE FECHA PREVIAMENTE ORDENADOS POR PUNTOS

while (fechaAux <= ultimaFecha){

aux = static\_cast<std::ostringstream\*>(&(std::ostringstream() << fechaAux))->str();

aux2 = "puntajesFecha"+aux+".txt";

const char \* nombreArchivo = aux2.c\_str();

if ((pf = fopen(nombreArchivo,"a")) == NULL){

cout << "Error al abrir el archivo puntajesFecha" << fechaAux << ".txt" << endl;

Sleep(2000);

exit(0);

}

for(i=0 ; i<longitud(listaEquipos) ; i++){

fprintf(pf, "%d;%d;%s;%s;%d\n", i+1, puntosEquipo[cursor].idEquipo, puntosEquipo[cursor].nomEquipo.c\_str(), puntosEquipo[cursor].nomUsuario.c\_str(), puntosEquipo[cursor].puntos);

cursor++;

}

fechaAux++;

fclose(pf);

}

//SE IMPRIME EL RANKING DE JUGADORES ORDENADO POR PUNTOS

if ((pf3 = fopen("puntajesPorJugador.txt","w")) == NULL){

cout << "Error al abrir el archivo puntajesTotalesPorEquipo.txt" << endl;

Sleep(2000);

exit(0);

}

ptrNodoCola ptrCursor = primero(colaOrdenada);

PtrNodoListaJ control;

Jugador jugador;

constructor(jugador);

i=1;

while(ptrCursor!=NULL){

pfAux = getDato(ptrCursor);

setIdJugador(jugador, getIdJugador(pfAux));

control = localizarDato(listaJugadores, jugador);

if(control!=NULL){

fprintf(pf3, "%d;%d;%d\n", i, getIdJugador(pfAux), getPuntos(pfAux));

i++;

}

ptrCursor = siguiente(ptrCursor);

}

fclose(pf3);

eliminarCola(colaOrdenada);

//Destruyendo TDAs, listas y colas:

destructor(pfMax);

destructor(pfAux);

destructor(pfCursor);

destructor(jugador);

eliminarCola(colaOrdenada);

}

// COSAS AUXILIARES PARA MOSTRAR ------------------------------------------------------------------------------

**bool traerClubes(ListaC &listaClubes)**{

system("cls");

cout << "Clubes\n" << endl;

Club aux;

constructor(aux);

PtrNodoListaC ptr = primero(listaClubes);

while(ptr!=NULL){

obtenerDato(listaClubes, aux, ptr);

cout << "ID Club: " << getIdClub(aux) << " | Nombre: " << getNombre(aux) << endl;

ptr = siguiente(listaClubes, ptr);

}

destructor(aux);

return true;

}

**bool traerJugadores(ListaJ &listaJugadores)**{

Jugador aux;

constructor(aux);

PtrNodoListaJ ptrJ = primero(listaJugadores);

while(ptrJ!=NULL){

obtenerDato(listaJugadores, aux, ptrJ);

cout << "ID jugador: " << getIdJugador(aux) << " | Nombre: " << getNombre(aux) << " | Posicion: " << getPosicion(aux) << endl;

ptrJ = siguiente(listaJugadores, ptrJ);

}

destructor(aux);

return true;

}

**bool traerEquipos(ListaE &listaEquipos)**{

system("cls");

cout << "Equipos\n" << endl;

Equipo aux;

constructor(aux);

PtrNodoListaE ptr = primero(listaEquipos);

while(ptr!=NULL){

obtenerDato(listaEquipos, aux, ptr);

cout << "ID equipo: " << getIdEquipo(aux) << " | Nombre: " << getNombreEquipo(aux) << " | Usuario: " << getNombreUsuario(aux) << endl;

ptr = siguiente(listaEquipos, ptr);

}

destructor(aux);

return true;

}

**bool traerJugadorePorEquipo(ListaE &listaEquipos)**{

system("cls");

cout << "Jugadores por equipos\n" << endl;

Equipo auxE;

constructor(auxE);

PtrNodoListaE ptrE = primero(listaEquipos);

while(ptrE!=NULL){

obtenerDato(listaEquipos, auxE, ptrE);

cout << "ID equipo: " << getIdEquipo(auxE) << " | Nombre: " << getNombreEquipo(auxE) << " | Usuario: " << getNombreUsuario(auxE) <<"\n"<< endl;

cout << "Arqueros: " << endl;

traerJugadores(getListaARQ(auxE));

cout << "Defensores: " << endl;

traerJugadores(getListaDEF(auxE));

cout << "Mediocampistas: " << endl;

traerJugadores(getListaMED(auxE));

cout << "Delanteros: " << endl;

traerJugadores(getListaDEL(auxE));

cout << endl;

/\*

PtrNodoListaJ ptrJ = primero(getListaARQ(auxE));

while(ptrJ!=NULL){

obtenerDato(getListaARQ(auxE),auxJ, ptrJ);

cout << "ID jugador: " << getIdJugador(auxJ) << " | Nombre: " << getNombre(auxJ) << " | Posicion: " << getPosicion(auxJ) << endl;

ptrJ = siguiente(getListaARQ(auxE), ptrJ);

}

\*/

ptrE = siguiente(listaEquipos, ptrE);

}

//Destruyendo TDAs

destructor(auxE);

return true;

}

**bool validarJugador(Equipo &equipo, Configuracion config, FILE \*errores, Posicion posicion)**{

bool valido = 0;

PtrNodoListaJ ptrJ;

int numero = 0;

Jugador jugador;

constructor(jugador);

if(posicion==ARQ){

if(longitud(getListaARQ(equipo))>config.maxArq){

printf("ERROR: Se descalifica equipo %d por superar el maximo de arqueros. Revisar errores.txt\n", getIdEquipo(equipo));

fprintf(errores, "\nE401-1 | Se descalifica equipo %d por tener %d arqueros. (MAXIMO: %d)\n", getIdEquipo(equipo), longitud(getListaARQ(equipo)), config.maxArq);

ptrJ = primero(getListaARQ(equipo));

while (ptrJ != NULL){

numero ++;

obtenerDato(getListaARQ(equipo), jugador, ptrJ);

fprintf(errores, "\t| %d -> ID: %d | Nombre: %s | Posicion: %d\n", numero, getIdJugador(jugador), getNombre(jugador).c\_str(), getPosicion(jugador));

ptrJ = siguiente(getListaARQ(equipo), ptrJ);

}

fprintf(errores,"\n");

}

else valido=1;

}

if(posicion==DEF){

if(longitud(getListaDEF(equipo))>config.maxDef){

printf("ERROR: Se descalifica equipo %d por superar el maximo de defensores. Revisar errores.txt\n", getIdEquipo(equipo));

fprintf(errores, "\nE401-2 | Se descalifica equipo %d por tener %d defensores. (MAXIMO: %d)\n", getIdEquipo(equipo), longitud(getListaDEF(equipo)), config.maxDef);

ptrJ = primero(getListaDEF(equipo));

while (ptrJ != NULL){

numero ++;

obtenerDato(getListaDEF(equipo), jugador, ptrJ);

fprintf(errores, "\t| %d -> ID: %d | Nombre: %s | Posicion: %d\n", numero, getIdJugador(jugador), getNombre(jugador).c\_str(), getPosicion(jugador));

ptrJ = siguiente(getListaDEF(equipo), ptrJ);

}

fprintf(errores,"\n");

}

else valido=1;

}

if(posicion==MED){

if(longitud(getListaMED(equipo))>config.maxMed){

printf("ERROR: Se descalifica equipo %d por superar el maximo de mediocampistas. Revisar errores.txt\n", getIdEquipo(equipo));

fprintf(errores, "\nE402-3 | Se descalifica equipo %d por tener %d mediocampistas. (MAXIMO: %d)\n", getIdEquipo(equipo), longitud(getListaMED(equipo)), config.maxMed);

ptrJ = primero(getListaMED(equipo));

while (ptrJ != NULL){

numero ++;

obtenerDato(getListaMED(equipo), jugador, ptrJ);

fprintf(errores, "\t| %d -> ID: %d | Nombre: %s | Posicion: %d\n", numero, getIdJugador(jugador), getNombre(jugador).c\_str(), getPosicion(jugador));

ptrJ = siguiente(getListaMED(equipo), ptrJ);

}

fprintf(errores,"\n");

}

else valido=1;

}

if(posicion==DEL){

if(longitud(getListaDEL(equipo))>config.maxDel){

printf("ERROR: Se descalifica equipo %d por superar el maximo de delanteros. Revisar errores.txt\n", getIdEquipo(equipo));

fprintf(errores, "\nE402-4 | Se descalifica equipo %d por tener %d delanteros. (MAXIMO: %d)\n", getIdEquipo(equipo), longitud(getListaDEL(equipo)), config.maxDel);

ptrJ = primero(getListaDEL(equipo));

while (ptrJ != NULL){

numero ++;

obtenerDato(getListaDEL(equipo), jugador, ptrJ);

fprintf(errores, "\t| %d -> ID: %d | Nombre: %s | Posicion: %d\n", numero, getIdJugador(jugador), getNombre(jugador).c\_str(), getPosicion(jugador));

ptrJ = siguiente(getListaDEL(equipo), ptrJ);

}

fprintf(errores,"\n");

}

else valido=1;

}

return valido;

}

**bool validarValorEquipo(Equipo &equipo, Configuracion config, FILE \*errores)**{

bool valido = 0;

if(getMonto(equipo)>config.ValorMaximoXEquipo){

printf("ERROR: Se descalifica equipo %d por superar el valor maximo permitido. Revisar errores.txt\n", getIdEquipo(equipo));

fprintf(errores, "E405-0 | Se descalifica equipo %d por tener un valor de %.2f (MAXIMO: %.2f)\n", getIdEquipo(equipo), getMonto(equipo), config.ValorMaximoXEquipo);

}

else valido = 1;

return valido;

}