

# MEMORIA PROYECTO FINAL MP



# UNIVERSIDAD DE GRANADA

Proyecto realizado por:

Jose Carlos Ibarrondo Maciá | Estudiante de Ingeniería informática en la UGR

# ÍNDICE

- Introducción
- Clase MatrizEnteros
- Clase Tablero
- Clase ConjuntoJugadores
- Clase Partida
- Main
- Tests

# INTRODUCCIÓN

Para afrontar esta práctica final, vamos a Modificar, Añadir y Solucionar Errores que surjan a partir de lo que nos piden introducir y editar de forma repetitiva hasta llegar a nuestra finalidad (el correcto funcionamiento del código). A este método lo vamos a llamar **“Proceso de Ajuste de Código”**.

Partiremos de las prácticas anteriores, las cuales se centraban en crear las diferentes clases por separado, juntándolas en un mismo proyecto.

## ¿Cómo vamos a llevar a cabo este método?

### 1º Identificación de los cambios a realizar

Identificamos aquello que vamos a introducir o modificar (métodos, funciones, variables, etc.). (lo llamamos “objetos” por referirnos a ellos de alguna forma)

### 2º Búsqueda de impactos en el código existente

Buscamos en el código aquellos métodos que usen, tengan como parámetro, modifiquen o consulten alguno de estos “objetos” a introducir. (el compilador reporta estas salidas a buscar)

### 3º Introducción de los cambios y actualización del código existente

Introducimos un “objeto” del paso 1º en la clase, y cambiamos todo aquello que hemos buscado en el paso 2º. Con esto evitamos olvidarnos el cambiar o actualizar alguna sentencia (puede que el compilador no nos lo reporte)

### 4º Repeticiones y seguimiento del proceso

Repetimos el paso 3º con todos hasta terminar.

### 5º Pruebas y verificación de los cambios

Hacemos pruebas por medio de test o en el propio main para comprobar que los cambios realizados han sido correctos y funcionan como deberían. Dichos test pueden ser manuales (por medio de entradas o escribir en el main) o programados en archivos txt y pasarlos como parámetros al programa principal.

## ¿Qué estructura vamos a llevar en este proyecto?

La estructura de esta memoria se va a basar en una breve introducción de lo pedido por cada Clase, y una serie de descripciones acerca de cómo realizarlo.

Una vez explicadas las clases, se pasará al main y por último algunos ejemplos de ejecución del programa, junto con algunos detalles extra.

# Clase MatrizEnteros

En un primer lugar, se nos pide **eliminar** una **variable constante** “max” y **cambiar** la **matriz estática** por una **dinámica**, **añadir** unos **métodos** de **reserva** y **liberación** de **memoria**, modificar un método **resize** e introducir dos **operadores** “=” y “<<”.

Al efectuar dichos cambios a nuestras clases ya definidas, el compilador nos reporta una serie de errores (de memoria, acceso a direcciones o declaraciones/uso de métodos), que se solucionan de la siguiente manera:

**1º\_Creamos** los métodos de **reserva de memoria dinámica** de la matriz de los datos privados de la clase, y **liberación de memoria dinámica** de la misma.

```
void MatrizEnteros::liberarMemoria(){
    for (int i = 0; i < filas; i++){
        delete[] m[i];
    }
    delete[] m;
    tamreservado = 0;
    filas = 0;
}

void MatrizEnteros::reservarMemoria(int f, int c){
    filas = f;
    columnas = c;
    m = new int * [filas];
    for(int i = 0; i < filas; i++){
        m[i] = new int [columnas];
    }
}
```

**2º\_**La **reserva de la memoria** se debe **llamar en el constructor**. Dicha función de reservar memoria que debe de estar definida previamente y **funcionar correctamente**.

```
MatrizEnteros::MatrizEnteros(int fil, int col, int def) {
    if((fil > 0) && (col > 0)){
        filas = fil;
        columnas = col;
        defecto = def;
        reservarMemoria(filas, columnas);
        modificarValorporDefecto(def);
        inicializarMatriz();
    }
}
```

**3º\_**La función de **putValue(fil,col)** reporta un **error de direccionamiento de memoria**, ya que, al trabajar con **memoria dinámica**, debemos **referirnos a las direcciones de memoria** y **no a los valores respectivos**. Por lo que **debemos trabajar con punteros**.

```
int & MatrizEnteros:: putValue(int fil, int col){
    int *a;
    if(((fil >= 0) && (fil < filas)) && ((col >= 0) && (col < columnas))){
        a = &m[fil][col];
    }
    return *a;
}
```

**4º\_**La función **getValue(fil,col)** la convertimos a **constante**, ya que lo que queremos es una referencia del valor de una dirección de memoria, **no queremos que dicho valor cambie**.

```
int MatrizEnteros::getValue(int fil, int col) const{
    if(((fil >= 0) && (fil < filas)) && ((col >= 0) && (col < columnas))){
        return m[fil][col];
    }
    return 1;
}
```

5º\_También, se nos pide crear **el constructor de copia**, al cual le pasamos por parámetro un objeto de la misma clase al que copiar. Se **copian las variables privadas**, se **reserva memoria del mismo tamaño** que el objeto a copiar y se **copian los valores de la matriz privada**.

```
MatrizEnteros::MatrizEnteros(const MatrizEnteros& orig) {
    filas = orig.filas;
    columnas = orig.columnas;
    reservarMemoria(filas, columnas);

    for(int i = 0; i < filas; i++){
        for(int j = 0; j < columnas; j++){
            putValue(i,j) = orig.getValue(i,j);
        }
    }
}
```

6º\_Para el **destructor**, llamamos a la **función de liberar memoria**, la cual se encargará de “destruir”.

```
MatrizEnteros::~MatrizEnteros() {
    liberarMemoria();
}
```

7º\_Para el **método resize**, creamos una **nueva matriz dinámica** y **reservamos memoria** para el tamaño pasado como parámetro, **copiamos los valores de la matriz inicial** a la nueva y **ajustamos las variables filas, columnas, tamreservado...** Por último, **liberamos memoria de la matriz inicial** y **asociamos la matriz inicial a la nueva** (será la definitiva) y **el puntero de la nueva matriz lo asignamos como nullptr** para evitar direccionar la misma memoria con varios punteros.

```
void MatrizEnteros::resize(int newtam){
    if((newtam > tamreservado) && (newtam > 0)){
        int ** aux = new int * [newtam];
        for(int i = 0; i < newtam; i++){
            aux[i] = new int [newtam];
        }

        for(int i = 0; i < newtam; i++){
            for(int j = 0; j < newtam; j++){
                aux[i][j] = getValue(i,j);
            }
        }

        liberarMemoria();
        tamreservado = newtam;
        filas = newtam;
        columnas = newtam;
        m = aux;
        aux = nullptr;
    }
}
```

8º\_El **operador “=”**, lo definiremos igual que el constructor de copia, pero nos debemos asegurar previamente que **el objeto de la derecha del igual** (o el pasado como argumento) **es diferente al objeto “\*this”**. Hacemos la copia de todos los datos y variables, y **devolvemos una referencia al objeto “\*this”**.

```
MatrizEnteros & MatrizEnteros::operator=(const MatrizEnteros &orig){
    if(this == &orig){
        return *this;
    }
    else{
        filas = orig.filas;
        columnas = orig.columnas;
        defecto = orig.defecto;

        reservarMemoria(filas, columnas);

        for(int i = 0; i < filas; i++){
            for(int j = 0; j < columnas; j++){
                putValue(i,j) = orig.getValue(i,j);
            }
        }

        return *this;
    }
}
```

9º\_Por último, para el **operador “<<”** usaremos la variable “flujo” pasada por parámetro del “ostream”, y pondremos la siguiente sentencia: **flujo << m.matriztostring()**. Esto lo que hará será almacenar en flujo la salida de la función matriztostring que devuelve un string de la matriz entera. Y **devolvemos dicha variable flujo**.

```
std::ostream & operator<< (std::ostream & flujo, const MatrizEnteros & m){
    flujo << m.matriztostring();
    return flujo;
}
```

# Clase ConjuntoJugadores

En este caso, **no** se nos pide **modificar** la **clase jugador** (contenida en esta clase ConjuntoJugadores), a su vez, nos **piden crear** una nueva **variable** “**competición**” de la clase MatrizEnteros (la cual almacenará las victorias de los jugadores con respecto a otros), el **método esquemaCompetición** y **apuntaPartida**.

Es posible que debamos modificar algunas partes del código que disponemos, así que hacemos una lectura del mismo y analizamos si debemos cambiar algo o no:

1º\_En el ConjuntoJugadores.h **declaramos** en el ámbito privado el **objeto competición**.

```
private:
    /**
     * @brief Ordena los Jugadores de un objeto ConjuntoJugadores en base a su id.
     */
    void ordenaporId();

    /**
     * @brief Aumenta el tamaño reservado del vector. La información del vector de
     * Jugadores se mantiene.
     * @pre newtam debe ser mayor que el tamaño reservado actualmente.
     * @param newtam Nuevo tamaño de espacio reservado.
     */
    void resize(int newtam);

    /* Vector de objetos Jugador */
    /*
    Jugador * vectorJugadores;
    /* Número de jugadores en el objeto */
    int numjugadores;
    /*Tamaño reservado en el vector*/
    int tamreservado;
    /*Objeto MatrizEnteros que se inicializa en el constructor
    de ConjuntoJugadores*/
    MatrizEnteros competicion;
};
```

2º\_En el **constructor** que recibe como parámetro k, **llamamos** a los **métodos** de **reserva de memoria** e **inicializarMatriz** de **competición** para poder usar dicho objeto matriz con **datos controlados** y no datos “basura”.

```
ConjuntoJugadores::ConjuntoJugadores(int k) {
    if(k > 0){
        tamreservado = k;
        numjugadores = 0;
        vectorJugadores = new Jugador[tamreservado];
        competicion.reservarMemoria(k,k);
        competicion.inicializarMatriz();
    }
}
```

3º\_En el **resize**, llamamos al **resize** del objeto **competición** por si fuese necesario.

```
void ConjuntoJugadores::resize(int newtam){
    if((newtam > tamreservado) && (newtam > 0)){
        Jugador * vectorAux = new Jugador[newtam];
        for(int i = 0; i < numjugadores; i++){
            vectorAux[i] = vectorJugadores[i];
        }

        delete[] vectorJugadores;
        vectorJugadores = vectorAux;
        tamreservado = newtam;

        competicion.resize(newtam);
    }
}
```

4º\_En el **constructor de copia**, debemos **hacer una copia del objeto competición** procedente del ConjuntoJugadores a copiar. (también debemos hacer un **resize** para **no desperdiciar memoria**).

```
ConjuntoJugadores::ConjuntoJugadores(const ConjuntoJugadores& orig){
    tamreservado = orig.tamreservado;
    numjugadores = orig.numjugadores;
    vectorJugadores = new Jugador[tamreservado];

    for(int i = 0; i < numjugadores; i++){
        vectorJugadores[i] = orig[i];
    }

    competicion.resize(orig.numjugadores);
    competicion = orig.competicion;
}
```

5º\_En el **constructor de parámetros** (n, \*nombres, \*apellidos), llamamos a **resize** de **competición**.

```
ConjuntoJugadores::ConjuntoJugadores(int n, string * nombres, string * apellidos){
    tamreservado = n;
    numjugadores = n;
    vectorJugadores = new Jugador[tamreservado];

    for(int i = 0; i < n; i++){
        vectorJugadores[i].putNombre() = nombres[i];
        vectorJugadores[i].putApellidos() = apellidos[i];
        vectorJugadores[i].setId() = i + 1;
    }
    competicion.resize(n);
}
```

6º\_Creamos el **método esquemaCompetición** el cual **devuelve un string** con la **matriz de competición**.

```
string ConjuntoJugadores::esquemaCompeticion(){
    string resultado;
    resultado = "MATRIZ JUGADOR X JUGADOR\n";
    /*cout << competicion.matriztoString();
    * no funcionaría porque el matriztoString es una función constante, deberíamos
    * arreglarlo para que funcionase, cambiando el que la función deje de ser constante
    */

    for(int i = 0; i < competicion.numfilas(); i++){
        //int jug = i+1;
        //resultado += "Jugador " + vectorJugadores[i].getNombre() + ": ";
        for(int j = 0; j < competicion.numcolumnas(); j++){
            if(i == j){
                resultado += "- ";
            }
            else{
                resultado += to_string(competicion.getValue(i,j)) + ' ';
            }
        }
        resultado += '\n';
    }

    return resultado;
}
```

7º\_Creamos el **método apuntaPartida**, el cual recibe por parámetros jug1 y jug2, **llamamos** a las funciones de **partidajugada** y **partidaganada** de forma respectiva (jug1 gana a jug2) e **incrementamos** el valor de **competición(jug1)(jug2)**.

```
void ConjuntoJugadores::apuntaPartida(int jug1, int jug2){
    competicion.putValue(jug1,jug2) = competicion.getValue(jug1,jug2) + 1;
    vectorJugadores[jug1].partidaJugada();
    vectorJugadores[jug2].partidaJugada();

    vectorJugadores[jug1].partidaGanada();
}
```



# Clase Tablero

Aquí nos piden introducir 2 operadores “=” y “<<” y un método “**ganador**”. Como no nos piden manejar memoria dinámica en esta clase, no tenemos que aplicar ni modificar nada, ya que la matriz “**t**”, ya lo hará porque ya está definido el control de la memoria dinámica en su clase MatrizEnteros.

1º\_Para este **método “ganador”**, debemos de tener muy claro como recorrer la matriz (horizontal, vertical, diagonal) y a su vez tener bien implementado el método `getValue` ya que vamos a necesitar ver los valores de la matriz.

```
//Buscamos en horizontal
for(int i = 0; i < tam && !continua; i++){
    for(int j = 0; j <= tam - Ntwin && !continua; j++){
        jug = t.getValue(i,j);
        if(jug != 0){
            gana = true;
            for(int k = 1; k < Ntwin; k++){
                if(jug != t.getValue(i,j+k)){
                    gana = false;
                }
            }
            if(gana){
                ganador = jug;
                continua = true;
            }
        }
    }
}

//Buscamos en vertical
for(int i = 0; i <= tam - Ntwin && !continua; i++){
    for(int j = 0; j < tam && !continua; j++){
        jug = t.getValue(i,j);
        if(jug != 0){
            gana = true;
            for(int k = 1; k < Ntwin; k++){
                if(jug != t.getValue(i+k,j)){
                    gana = false;
                }
            }
            if(gana){
                ganador = jug;
                continua = true;
            }
        }
    }
}

//Buscamos en diagonal up-left hasta down-right
for(int i = 0; i <= tam - Ntwin && !continua; i++){
    for(int j = 0; j <= tam - Ntwin && !continua; j++){
        jug = t.getValue(i,j);
        if(jug != 0){
            gana = true;
            for(int k = 1; k < Ntwin; k++){
                if(jug != t.getValue(i+k,j+k)){
                    gana = false;
                }
            }
            if(gana){
                ganador = jug;
                continua = true;
            }
        }
    }
}

//Buscamos en diagonal down-left hasta up-right
for(int i = tam - 1; i >= Ntwin - 1 && !continua; i++){
    for(int j = 0; j <= tam - Ntwin && !continua; j++){
        jug = t.getValue(i,j);
        if(jug != 0){
            gana = true;
            for(int k = 1; k < Ntwin; k++){
                if(jug != t.getValue(i-k,j+k)){
                    gana = false;
                }
            }
            if(gana){
                ganador = jug;
                continua = true;
            }
        }
    }
}
```

2º\_EL operador “=” comprobamos si son **iguales** los **objetos**, y si son **diferentes** asignamos la **igualación** de los **datos** y **devolvemos** la **referencia** al **objeto**.

```
Tablero & Tablero::operator=(const Tablero &orig){
    if(this == &orig){
        return *this;
    }
    else{
        tam = orig.tam;
        Ntwin = orig.Ntwin;
        t = orig.t;

        return *this;
    }
}
```

3º\_Para el **operador** “<<”, le pasamos a la variable **flujo** un **string** que devuelve la **función** de **tablerotostring()**. Para que esta función se pueda hacer sin problema, debemos **hacer constante** el método “**tablerotostring()**” el cual nos devuelve un string del tablero, esto lo hacemos constante ya que **no se debe modificar el contenido de tablero**. Devolvemos con un **return** la variable **flujo**.

```
std::ostream & operator << (std::ostream & flujo, const Tablero & tab){  
    flujo << tab.tablerotostring();  
    return flujo;  
}
```

**NOTA:**

En la función **introducirficha()**, le restamos 1 a la columna introducida para no hacer al usuario pensar en posiciones de un array (es decir, hacemos el cambio de columna 1 a columna 0 para evitar que el usuario piense en dirección lógica o real)

# Clase Partida

Para esta clase, nos piden crearla desde cero añadiendo variables:

```
Tablero tab,
ConjuntoJugadores jug,
int jug1, int jug2,
int turnoActual;
```

uno métodos tales como:

```
Partida(Tablero & newtab, ConjuntoJugadores & newjug),
Partida(const Partida & orig),
~Partida(),
void turno(),
void inicializaPartida(int newjug1, newjug2),
void save(string file),
void load(string file),
void realizaPartida(),
void muestraResultadosCompetición();
```

y unas funciones externas a la clase:

```
void NuevaPartida(string savefile, int numnjugadores, string * nombres, string
* apellidos, int jug1, int jug2, int N, int Ntwin),
void CargaPartida(string savefile, int jug1, int jug2, int N, int Ntwin).
```

1º\_Declaramos todas las **variables** que vamos a utilizar.

```
private:
//Objeto tablero que representa el tablero de conectaN en donde están jugando los jugadores
Tablero tab;
//Objeto que contiene información de los jugadores que participan en el campeonato
ConjuntoJugadores jug;
int jug1, jug2; // ids de los jugadores que están participando en la partida
int turnoActual; //id del jugador al que le toca introducir una ficha en el Tablero
//Columna a leer para introducir ficha
int columna;
};
```

2º\_Creamos el **constructor** de **parámetros** de la clase, el **constructor** de **copia** y el **destructor** (el cual estará vacío puesto que no trabajamos con memoria dinámica en esta clase) y el **método inicializapartida** ya que son valores que se les da al **crearse** el **objeto**:

```
Partida::Partida(Tablero& newtab, ConjuntoJugadores& newjug){
    tab = newtab;
    jug = newjug;
    jug1 = jug2 = turnoActual = 0;
    inicializaPartida(jug1,jug2);
}

Partida::Partida(const Partida & orig){
    tab = orig.tab;
    jug = orig.jug;
    jug1 = orig.jug1;
    jug2 = orig.jug2;
    turnoActual = orig.turnoActual;
    inicializaPartida(jug1,jug2);
}

Partida::~Partida(){
}

void Partida::inicializaPartida(int newjug1, int newjug2){
    jug1 = newjug1;
    jug2 = newjug2;
    turnoActual = jug1;
}
```

3º\_Definimos el **método turno()**, el cual va a interactuar con el usuario e introducir la ficha en el tablero:

```
void Partida::turno(){
    cout << endl << "Jugador con el id " << turnoActual
        << " escoge una columna para introducir la ficha: ";
    cout << endl << tab;
    cin >> columna;
    if(tab.introducirFicha(columna, turnoActual)){
        if(turnoActual == jug1){
            turnoActual = jug2;
        }
        else{
            turnoActual = jug1;
        }
    }
}
```

4º\_Creamos los **métodos load()** y **save()** que serán los **encargados** de **interactuar** con el archivo de **campeonato.txt**:

<pre>void Partida::save(string file){     ofstream fo;     fo.open(file);      if(fo.is_open()){          jug.save(file);          fo.close();     }     else{         cout &lt;&lt; "No se pudo abrir el archivo: " &lt;&lt; file &lt;&lt; endl;     } }</pre>	<pre>void Partida::load(string file){     ifstream fi;     fi.open("./data/" + file);      if(fi.is_open()){          jug.load(file);          fi.close();     }     else{         cout &lt;&lt; "No se pudo abrir el archivo: " &lt;&lt; file &lt;&lt; endl;     } }</pre>
---	---

5º\_El **método realizaPartida()** será el método **“principal”** ya que este se encargará de que **el juego se lleve a cabo** hasta encontrar un ganador, y **llama al método muestraResultadosCompetición()**.

<pre>void Partida::realizaPartida(){     while(tab.ganador() == 0){         turno();     }     if(tab.ganador() &gt; 0){         cout &lt;&lt; endl &lt;&lt; tab &lt;&lt; endl;         cout &lt;&lt; "El ganador es: " &lt;&lt; tab.ganador() &lt;&lt; endl;         if(tab.ganador() == jug1){             jug.apuntaPartida(jug1 -1, jug2 -1);         }         else{             jug.apuntaPartida(jug2 -1, jug1 -1);         }         muestraResultadosCompeticion();     }     else{         cout &lt;&lt; endl &lt;&lt; "no se suman victorias a nadie" &lt;&lt; endl;         muestraResultadosCompeticion();     } }</pre>	<pre>void Partida::muestraResultadosCompeticion(){     cout &lt;&lt; jug.esquemaCompeticion() &lt;&lt; endl &lt;&lt; jug.rankingJugadores(); }</pre>
---	--

6º Para los **métodos NuevaPartida()** y **CargaPartida()**, crearemos objetos **tablero**, **conjuntojugadores** y **partida**, en función de las necesidades del programa. **Inicializaremos** y haremos las **llamadas** a los **constructores** de estos objetos, a través de los **parámetros** que le pasemos a las funciones respectivamente. Dichos **parámetros** se **inicializarán** en el **main.cpp** en función de la lectura del **archivo de configuración**.

```
void NuevaPartida(string savefile, int numjugadores, string * nombres,
                 string * apellidos, int jug1, int jug2, int N, int Ntwin){

    ConjuntoJugadores jugadores(numjugadores, nombres, apellidos);

    Tablero tablero(N, Ntwin);

    Partida partida(tablero, jugadores);

    partida.inicializaPartida(jug1, jug2);

    partida.realizaPartida();

    partida.save(savefile);
}
```

```
void CargaPartida(string savefile, int jug1, int jug2, int N, int Ntwin){

    Tablero tablero(N, Ntwin);

    ConjuntoJugadores jug;

    jug.load(savefile);

    Partida partida(tablero, jug);

    partida.inicializaPartida(jug1, jug2);

    partida.realizaPartida();

    jug.save(savefile);

    partida.save(savefile);
}
```

# MAIN

Dado que **nuestro programa debe leer un archivo config.txt** que se le pasa por **argumento**, y **ejecutar la partida** con respecto a las **indicaciones** dadas en el mismo archivo config.txt, **definiremos nuestro main.cpp** para que **lea dicho archivo** y haga las **llamadas pertinentes a CargaPartida() o NuevaPartida()**.

Para ello, **leeremos línea por línea el config.txt**, y **guardaremos los datos relevantes** (nombres, apellidos, tamaño del tablero, número de jugadores, fichas para ganar, etc.), y **estableceremos la condición** de que si **“nuevapartida”** es **“si”** llame a **NuevaPartida()** o **“no”** y llame a **CargaPartida()**, pasando como **argumentos** los **datos leídos** previamente.

```
#include "Partida.h"
#include "ConjuntoJugadores.h"
#include "Tablero.h"
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main(int argc, char** argv) {
    string fichero = argv[1];
    string nuevapartida;
    int jug1 = 0, jug2 = 0;
    int tam = 0, ntowin = 0, numjugadores = 0;
    string * nombre, * apellido;
    string savefile;

    if (argc < 2) {
        cout << "Uso: " << argv[0] << " << <configuración.txt>" << endl;
        return 1;
    }

    //cout << "El nombre del fichero pasado como argumento es: " << fichero << endl;

    ifstream archivo(fichero); // Abre el archivo para lectura

    if (archivo.is_open()) {
        string linea;
        getline(archivo, linea);
        size_t pos = linea.find("=");

        //Leemos los datos del fichero config.txt
        while (getline(archivo, linea)) {
            size_t pos = linea.find("=");
            if (pos != string::npos && pos < linea.size()) {
                string clave = linea.substr(0, pos);
                string valor = linea.substr(pos + 1);

                if (clave == "NUEVAPARTIDA ") {
                    nuevapartida = valor;
                } else if (clave == "FICHEROCAMPEONATO ") {
                    savefile = valor;
                } else if (clave == "JUGADORESID ") {
                    jug1 = stoi(valor);
                    valor = linea.substr(pos + 3);
                    jug2 = stoi(valor);
                } else if (clave == "TAM ") {
                    tam = stoi(valor);
                } else if (clave == "NTOWIN ") {
                    ntowin = stoi(valor);
                } else if (clave == "JUGADORES ") {
                    numjugadores = stoi(valor);
                    nombre = new string[numjugadores];
                    apellido = new string[numjugadores];
                    for (int i = 0; i < numjugadores; i++) {
                        archivo >> nombre[i] >> apellido[i];
                        //cout << nombre[i] << ", " << apellido[i];
                    }
                }
            }
        }
    }
    archivo.close();
}
```

```
        archivo.close();
    } else {
        cout << "No se pudo abrir el archivo." << endl;
    }

    cout << "Haciendo la lectura del fichero de configuración:" << endl
        << "\tNuevapartida: " << nuevapartida << endl
        << "\tFichero con actualización de campeonato: ./data/" << fichero << endl
        << "\tJugador1: " << jug1 << endl
        << "\tJugador2: " << jug2 << endl
        << "\tTamaño del tablero: " << tam << endl
        << "\tFichas consecutivas para ganar: " << ntowin << endl
        << "\tJugadores en el campeonato: " << numjugadores << endl;

    if(nuevapartida == " si"){
        cout << endl << "Creando una nueva partida" << endl;

        NuevaPartida(savefile, numjugadores, nombre, apellido, jug1, jug2, tam, ntowin);
    }
    else if(nuevapartida == " no"){
        cout << endl << "Cargando la partida del fichero ./data/" << fichero << endl;
        CargaPartida(savefile, jug1, jug2, tam, ntowin);
    }
    else{
        cout << "Error al leer los datos del fichero " << fichero;
        cout << endl << "Revise que esté bien configurado";
    }

    delete [] apellido;
    delete [] nombre;
    apellido = nullptr;
    nombre = nullptr;

    return 0;
}
```

Dado que la **inicialización y/o reserva de datos** de los que leemos del archivo **no sabemos su tamaño**, trabajamos con **memoria dinámica**, por lo que, **liberamos la memoria** como hemos hecho a lo largo del proyecto en las diferentes clases.

# TESTS

Para finalizar el proyecto, realizaremos una serie de **ejecuciones modificando** el archivo **config.txt** para **comprobar que se realizan correctamente las diferentes ejecuciones de nuestro programa**. (Nota: tanto el config.txt como el campeonato1.txt los copiamos tal cual se nos indica en la guía del proyecto)

Ejecutamos dichas pruebas **compilando** todo el **proyecto** y haciendo la **llamada al ejecutable** y **pasando** como **argumento** el archivo **config.txt**. Para esto, debemos compilar el programa ejecutando:

**“g++ -I include main.cpp ./src/\*.cpp -o miprograma”**

(Esto está indicado en el archivo “EJECUTABLE.txt”)

Una vez aclarado esto, vamos con las ejecuciones:

```

main.cpp x  config.txt x  campeonato1.txt x
Source  History
1      4
2      1      Pedro  Ramírez      0      0
3      2      Pepe   Morente     0      0
4      3      Matilde López  1      1
5      4      Ana    Jiménez    0      1
6      0 0 0 0
7      0 0 0 0
8      0 0 0 1
9      0 0 0 0

Terminal - ...suario-vb: ~/Escritorio/final/final x  Output - final (Clean, Build)
usuario@usuario-vb:~/Escritorio/final/final$ ./miprograma config.t
Haciendo la lectura del fichero de configuración:
  Nuevapartida: no
  Fichero con actualización de campeonato: ./data/config.txt
  Jugador1: 3
  Jugador2: 4
  Tamaño del tablero: 7
  Fichas consecutivas para ganar: 4
  Jugadores en el campeonato: 4

Cargando la partida del fichero ./data/config.txt

MATRIZ JUGADOR X JUGADOR
- 0 0 0
0 - 0 0
0 0 - 1
0 0 0 -

RANKING DE JUGADORES
Matilde López 1.000000
Pedro Ramírez 0.000000
Pepe Morente 0.000000
Ana Jiménez 0.000000

Jugador con el id 3 escoge una columna para introducir la ficha:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

```

Aquí se observa la ejecución del programa y el archivo campeonato1.txt (antes)



Tras varias entradas del usuario (las hacemos “rápidas” para que la partida llegue al final y comprobar que ha funcionado correctamente), **comprobamos** que el **programa finaliza** la ejecución, **actualiza** la **matriz competición**, y si leemos el archivo **campeonato.txt**, comprobaremos que **ha sido actualizado**.

The screenshot shows a code editor with three tabs: `main.cpp`, `config.txt`, and `campeonato1.txt`. The `main.cpp` tab is active, displaying a C++ program. The program uses a 9x9 matrix to store scores for four players. The matrix is initialized with zeros, and then scores are entered for each player. The program then calculates the winner based on the scores and displays the results.

The terminal output shows the following text:

```

0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 3 4 0 0 0
0 0 3 4 0 0 0
0 0 3 4 0 0 0
3

0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 3 0 0 0 0
0 0 3 4 0 0 0
0 0 3 4 0 0 0
0 0 3 4 0 0 0

El ganador es el jugador con el id: 3
MATRIZ JUGADOR X JUGADOR
- 0 0 0
0 - 0 0
0 0 - 2
0 0 0 -

RANKING DE JUGADORES
Matilde López 1.000000
Pedro Ramírez 0.000000
Pepe Morente 0.000000
Ana Jiménez 0.000000
usuario@usuario-vb:~/Escritorio/final/final$

```

En este caso, se ha indicado que **no es una nueva partida**.

Veamos ahora cuando **si es una nueva partida**:

```

usuario@usuario-vb:~/Escritorio/final/final$ ./miprograma config.txt
Haciendo la lectura del fichero de configuración:
  Nuevapartida: si
  Fichero con actualización de campeonato: ../data/config.txt
  Jugador1: 3
  Jugador2: 4
  Tamaño del tablero: 7
  Fichas consecutivas para ganar: 4
  Jugadores en el campeonato: 4

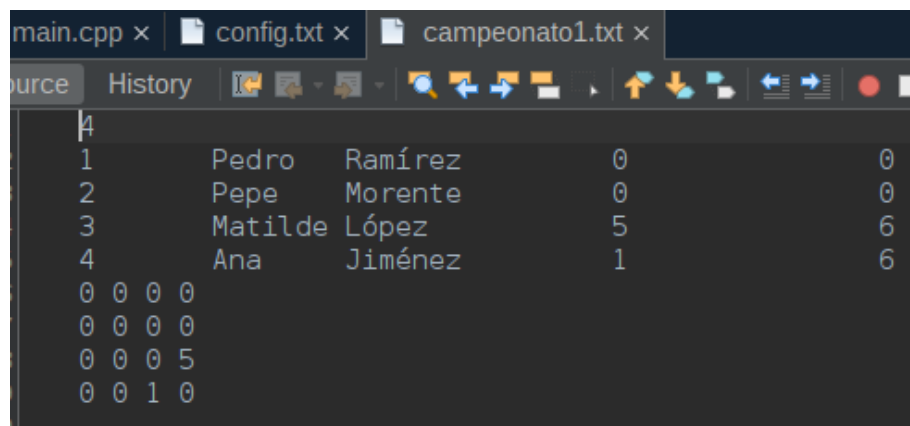
Creando una nueva partida

MATRIZ JUGADOR X JUGADOR
- 0 0 0
0 - 0 0
0 0 - 0
0 0 0 -

RANKING DE JUGADORES
Pedro Ramírez 0.000000
Pepe Morente 0.000000
Matilde López 0.000000
Ana Jiménez 0.000000

Jugador con el id 3 escoge una columna para introducir la ficha:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

```



The screenshot shows a code editor with three tabs: 'main.cpp', 'config.txt', and 'campeonato1.txt'. The 'campeonato1.txt' tab is active, displaying the following content:

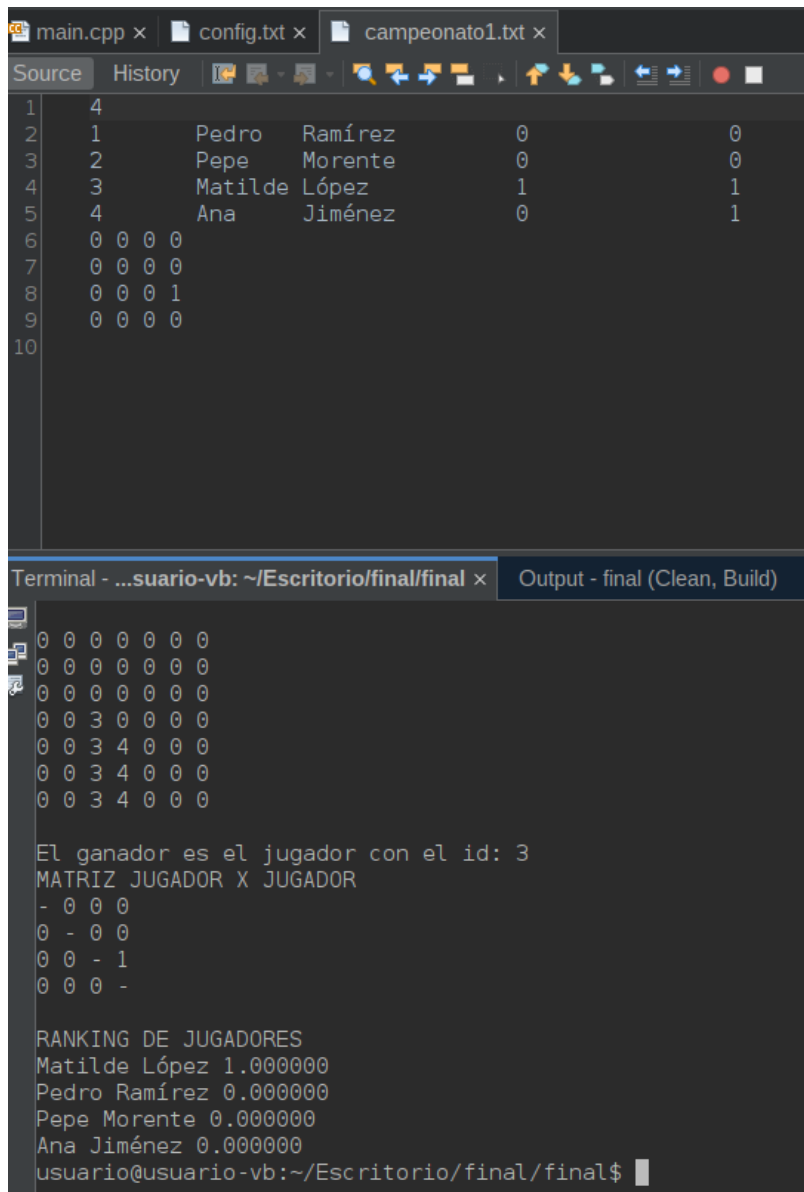
ID	Nombre	Puntaje	Victorias
1	Pedro Ramírez	0	0
2	Pepe Morente	0	0
3	Matilde López	5	6
4	Ana Jiménez	1	6

0	0	0	0
0	0	0	0
0	0	0	5
0	0	1	0

Aquí podemos ver el antes del archivo campeonato.txt.

De nuevo, tras varias entradas “rápidas” del usuario:



The screenshot shows a code editor with three tabs: `main.cpp`, `config.txt`, and `campeonato1.txt`. The `Source` tab is active, displaying a C++ program. The program defines a tournament with four players: Pedro Ramírez, Pepe Morente, Matilde López, and Ana Jiménez. It uses a matrix to track wins and a ranking system to determine the winner. The terminal output shows the results of the simulation.

```
1 4
2 1 Pedro Ramírez 0 0
3 2 Pepe Morente 0 0
4 3 Matilde López 1 1
5 4 Ana Jiménez 0 1
6 0 0 0 0
7 0 0 0 0
8 0 0 0 1
9 0 0 0 0
10
```

```
Terminal - ...suario-vb: ~/Escritorio/final/final x Output - final (Clean, Build)
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 3 0 0 0 0
0 0 3 4 0 0 0
0 0 3 4 0 0 0
0 0 3 4 0 0 0
0 0 3 4 0 0 0

El ganador es el jugador con el id: 3
MATRIZ JUGADOR X JUGADOR
- 0 0 0
0 - 0 0
0 0 - 1
0 0 0 -

RANKING DE JUGADORES
Matilde López 1.000000
Pedro Ramírez 0.000000
Pepe Morente 0.000000
Ana Jiménez 0.000000
usuario@usuario-vb:~/Escritorio/final/final$
```

Una vez **finaliza**, **comprobamos** que el **campeonato1.txt** se ha **actualizado** correctamente.