```
// Practica 3. Ignacio y Joaquin.cpp : Este archivo contiene la función "main". La
ejecución del programa comienza y termina ahí.
//

#include <iostream>
#include <Windows.h> // Para los colores
#include <fstream> // Para los ficheros

using namespace std;

// Constantes
const int N_FILAS = 6;
const int N_COLUMNAS = 8;
const int MAX_JUGADORES = 100;

// Enumerado para casilla
enum Casilla {
    VACIO = 0,
    JUGADOR1 = 1,
    JUGADOR2 = 2
};

// Tablero
typedef struct {
    Casilla casillas[N_FILAS][N_COLUMNAS];
    int espaciosLibres;
} Tablero;

// Jugador
typedef struct {
    string nick;
    int ganadas;
    int perdidas;
} Jugador;

// DatosPartida
typedef struct {
    Jugador jugadores[100];
    int numeroJugadores;
} DatosPartida;

////////////////////////////// FUNCIONES ////////////////////////////////
//función que devuelve true si hay espacios libres en el tabrero y false en caso
//contrario
bool quedanHuecos(const Tablero t)
{
    return (t.espaciosLibres > 0);
}

//procedimiento que, dada una casilla, muestra el caracter correspondiente al
jugador
//1, jugador 2 (o máquina) o casilla vacía en el color correspondiente
void muestraCharColor(Casilla c) {
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    if (c == JUGADOR1) {
        SetConsoleTextAttribute(hConsole, 6);
        cout << char(4);
        SetConsoleTextAttribute(hConsole, 15);
    }
```

```
    else if (c == JUGADOR2) {
        SetConsoleTextAttribute(hConsole, 4);
        cout << char(4);
        SetConsoleTextAttribute(hConsole, 15);
    }
    else {
        cout << " ";
    }
}

//procedimiento que, dado un Tablero lo muestra en la consola en forma de tablero
//gráfico
void muestraTablero(const Tablero t)
{
    //Línea superior
    cout << char(218) << char(196) << char(196) << char(196); // ┌ ─ ─ ─
    for (int i = 1; i < N_COLUMNAS; i++) {
        cout << char(194) << char(196) << char(196) << char(196); //┬ ─ ─ ─
    }
    cout << char(191); // ┐
    cout << endl;

    //Filas
    //Fila N
    for (int j = 0; j < N_COLUMNAS; j++)
    {
        cout << char(179); // │
        cout << " ";
        muestraCharColor(t.casillas[N_FILAS - 1][j]);
        cout << " ";
    }
    cout << char(179); // │
    cout << endl;
    //Resto de filas
    for (int i = N_FILAS - 2; i >= 0; i--)
    {
        //Linea interior
        cout << char(195) << char(196) << char(196) << char(196); // ├ ─ ─ ─
        for (int j = 1; j < N_COLUMNAS; j++)
        {
            cout << char(197) << char(196) << char(196) << char(196); // ┼ ─ ─ ─
        }
        cout << char(180); // ┤
        cout << endl;

        //Fila "i"
        for (int j = 0; j < N_COLUMNAS; j++)
        {
            cout << char(179); // │
            cout << " ";
            muestraCharColor(t.casillas[i][j]);
            cout << " ";
        }
        cout << char(179); // │
        cout << endl;
    }

    //Línea Inferior
    cout << char(192) << char(196) << char(196) << char(196); // └ ─ ─ ─
```

```
    for (int i = 1; i < N_COLUMNAS; i++) {
        cout << char(193) << char(196) << char(196) << char(196); //┴ ─ ─ ─
    }
    cout << char(217); // ┘
    cout << endl;

    // Números
    for (int i = 0; i < N_COLUMNAS; i++) {
        cout << "  " << i + 1 << " ";
    }
    cout << endl;
}

//función que, dados un tablero y un número de columna, busca el hueco adecuado
para
//inertar la ficha y devuelve el valor de la fila de dicho hueco(EN ESTA FUNCIÓN NO
SE
//GUARDA LA FICHA EN EL TABLERO(!!))
int ponerFicha(const Tablero t, int columna)
{
    int filaVacia = N_FILAS;
    for (int fila = 0; fila < N_FILAS; fila++)
    {
        if (t.casillas[fila][columna] == VACIO)
        {
            filaVacia = fila;
            break;
        }
    }
    return filaVacia;
}

//función que devuelve true si una columna ya está llena y no caben más fichas;
false
//en caso contrario
bool columnaLlena(const Tablero t, int columna)
{
    return (t.casillas[N_FILAS - 1][columna] != VACIO);
}

//función que comprueba si un jugador es ganador
bool ganador(Tablero& t, int fila, int columna, int jugador)
{
    t.casillas[fila][columna] = (Casilla)jugador;
    t.espaciosLibres--;
    // Comprobación de la ficha, a ver si es ganadora
    bool gana = false;

    // Comprobación vertical. Fila a fila manteniendo columna
    // Vamos de arriba a abajo, buscando solamente todas las iguales que estén
seguidas desde la ficha introducida
    // El resto ya fueron comprobadas en iteraciones anteriores
    int fichasSeguidas = 0;
    for (int i = fila; i >= 0; i--)
    {
        if (t.casillas[i][columna] == jugador)
        {
            fichasSeguidas++;
        }
```

```
        else
        {
            break;
        }
        if (fichasSeguidas >= 4)
        {
            return true; // GANA! Tenemos cuatro seguidas en vertical desde la que
acabamos de poner hacia abajo
        }
    }

    // Comprobación horizontal. Columna a columna manteniendo la fila
    // Vamos a recorrer toda la fila de la ficha de izquierda a derecha buscando
bloques de fichas seguidas y contandolas
    bool contandoFichas = false; // Usaremos este booleano para saber si estamos
contando fichas del jugador
    fichasSeguidas = 0;
    for (int i = 0; i < N_COLUMNAS; i++)
    {
        if (t.casillas[fila][i] == jugador) // Encontramos una ficha del jugador
        {
            if (contandoFichas)
            {
                fichasSeguidas++;
                if (fichasSeguidas >= 4)
                {
                    return true; // Ganamos, 4 o más fichas seguidas en diagonal \!
                }
            }
            else
            {
                contandoFichas = true;
                fichasSeguidas = 1;
            }
        }
        else
        {
            contandoFichas = false;
            fichasSeguidas = 0;
        }
    }

    // Comprobación diagonal /
    // Vamos a recorrerla muy parecido a la comprobación horizontal, pero sumando
en filas y columnas a la vez.
    // Tenemos que inicializar la posición desde la que empezaremos a comprobar
para que sea el principio de la diagonal (abajo izquierda)
    contandoFichas = false;
    fichasSeguidas = 0;
    int filaInicial = fila - min(fila, columna);
    int columnaInicial = columna - min(fila, columna);
    for (int i = 0; i < min(N_FILAS, N_COLUMNAS); i++)
    {
        if ((filaInicial + i >= N_FILAS) || (columnaInicial + i >= N_COLUMNAS)) //
nos hemos salido de la matriz
        {
            break;
        }
        if (t.casillas[filaInicial + i][columnaInicial + i] == jugador) //
```

```
Encontramos una ficha del jugador
        {
            if (contandoFichas)
            {
                fichasSeguidas++;
                if (fichasSeguidas >= 4)
                {
                    return true; // Ganamos, 4 o más fichas seguidas en diagonal /!
                }
            }
            else
            {
                contandoFichas = true;
                fichasSeguidas = 1;
            }
        }
        else
        {
            contandoFichas = false;
            fichasSeguidas = 0;
        }
    }

    // Comprobación diagonal \
    // Vamos a recorrerla como la otra diagonal, pero ahora sumamos en columnas y
restamos en filas,
    // para ir de izquierda a derecha, de arriba a abajo
    // Tenemos que inicializar la posición desde la que empezaremos a comprobar
para que sea el principio de la diagonal (arriba izquierda).
    contandoFichas = false;
    fichasSeguidas = 0;
    filaInicial = min(N_FILAS - 1, fila + columna);
    columnaInicial = columna - min(N_FILAS - 1 - fila, columna);
    for (int i = 0; i < min(N_FILAS, N_COLUMNAS); i++)
    {
        if ((filaInicial - i < 0) || (columnaInicial + i >= N_COLUMNAS)) // nos
hemos salido de la matriz
        {
            break;
        }
        if (t.casillas[filaInicial - i][columnaInicial + i] == jugador) //
Encontramos una ficha del jugador
        {
            if (contandoFichas)
            {
                fichasSeguidas++;
                if (fichasSeguidas >= 4)
                {
                    return true; // Ganamos, 4 o más fichas seguidas en horizontal!
                }
            }
            else
            {
                contandoFichas = true;
                fichasSeguidas = 1;
            }
        }
        else
        {
```

```
            contandoFichas = false;
            fichasSeguidas = 0;
        }
    }

    // Si no hemos encontrado 4, devolvemos false, nadie gana todavía
    return false;
}

//procedimiento encargado de poner todas las casillas del tablero a "VACIO"
void inicializaTablero(Tablero& t)
{
    for (int i = 0; i < N_FILAS; i++)
    {
        for (int j = 0; j < N_COLUMNAS; j++)
        {
            t.casillas[i][j] = VACIO;
        }
    }
    t.espaciosLibres = N_FILAS * N_COLUMNAS;
}

//función encargada de realizar el turno de un jugador
bool turnoJugador(Tablero& t, int jugador)
{
    int columnaFicha;
    int filaFicha;
    bool columnaValida = false;
    while (!columnaValida)
    {
        cout << "Introduce ficha. COLUMNA: ";
        cin >> columnaFicha;
        columnaFicha--; // Corregimos la entrada de usuario a nuestros números de
matriz
        if (columnaFicha > N_COLUMNAS - 1 || columnaFicha < 0)
        {
            cout << "Numero de columna invalido." << endl;
            columnaValida = false;
        }
        else if (columnaLlena(t, columnaFicha))
        {
            cout << "Columna llena, introduce la ficha en una columna con hueco."
<< endl;
            columnaValida = false;
        }
        else
        {
            columnaValida = true;
        }
    }
    filaFicha = ponerFicha(t, columnaFicha);
    return ganador(t, filaFicha, columnaFicha, jugador);
}

//procedimiento encargado de la ejecución principal del juego
void juegoConecta4(Tablero t, Jugador& j, DatosPartida& listaJugadores)
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE); // Vamos a usar colores en
los cout
```

```
    bool gana1 = false;
    bool gana2 = false;
    while (!gana1 && !gana2 && quedanHuecos(t))
    {
        if (!gana1 && !gana2 && quedanHuecos(t))
        {
            muestraTablero(t);
            SetConsoleTextAttribute(hConsole, 6);
            cout << j.nick << "  "; // Mostramos el nombre del jugador
            SetConsoleTextAttribute(hConsole, 15);
            gana1 = turnoJugador(t, 1); // Turno del jugador 1
        }
        if (!gana1 && !gana2 && quedanHuecos(t))
        {
            muestraTablero(t);
            SetConsoleTextAttribute(hConsole, 4);
            cout << "JUGADOR 2  ";
            SetConsoleTextAttribute(hConsole, 15);
            gana2 = turnoJugador(t, 2); // Turno del jugador 2
        }
    }
    muestraTablero(t); //Mostramos el tablero por última vez
    // Fin del juego
    cout << "FIN DEL JUEGO" << endl;
    cout << "-------------" << endl;
    cout << "Resultado: ";
    if (gana1)
    {
        j.ganadas++; // Aumentamos las partidas ganadas del jugador
        SetConsoleTextAttribute(hConsole, 6);
        cout << "GANA " << j.nick << "!!!" << endl; //Ponemos el nombre del jugador
        SetConsoleTextAttribute(hConsole, 15);
    }
    else if (gana2)
    {
        j.perdidas++; // Aumentamos las partidas perdidas del jugador
        SetConsoleTextAttribute(hConsole, 4);
        cout << "GANA EL JUGADOR 2. " << j.nick << " PIERDE!!!" << endl;
        SetConsoleTextAttribute(hConsole, 15);
    }
    else
    {
        cout << "EMPATE!!!" << endl;
    }

}

// busca al jugador con el nick especificado y devuelve un entero indicando su
posición en la lista.
// Si no lo encuentra, devuelve - 1
int buscaJugador(DatosPartida d, string nick)
{
    for (int i = 0; i < d.numeroJugadores; i++)
    {
        if (d.jugadores[i].nick == nick)
        {
            return i;
        }
    }
```

```
    return -1;
}

// añade un nuevo jugador a la lista. Si no cabe, muestra un mensaje de error
void guardaJugadorNuevo(DatosPartida& d, Jugador j)
{
    if (d.numeroJugadores < MAX_JUGADORES)
    {
        d.jugadores[d.numeroJugadores] = j;
        d.numeroJugadores++;
    }
    else
    {
        cout << "ERROR: Lista llena, no se pudo añadir el jugador." << endl;
    }
}

// devuelve los datos del jugador con el nick especificado. Si no existe, se crea
un nuevo jugador
Jugador iniciarSesion(DatosPartida& d, string nick)
{
    Jugador j;
    int posicion = buscaJugador(d, nick);
    if (posicion == -1) // Jugador no encontrado, crearlo
    {
        j.ganadas = 0;
        j.perdidas = 0;
        j.nick = nick;
        guardaJugadorNuevo(d, j);
    }
    else // Jugador encontrado, devolverlo
    {
        j = d.jugadores[posicion];
    }
    return j;
}

// carga los datos de los jugadores del archivo "jugadores.txt"
void cargaDatos(DatosPartida& d)
{
    ifstream txt;
    txt.open("jugadores.txt");
    if (txt.is_open())
    {
        txt >> d.numeroJugadores;
        for (int i = 0; i < d.numeroJugadores; i++)
        {
            txt >> d.jugadores[i].nick;
            txt >> d.jugadores[i].ganadas;
            txt >> d.jugadores[i].perdidas;
        }
    }
}

// muestra la información del jugador con el nick especificado
void muestraInfo(DatosPartida d, string nick)
{
    int posicion = buscaJugador(d, nick);
    if (posicion == -1) // Jugador no encontrado
```

```
    {
        cout << "Jugador no encontrado." << endl;
    }
    else // Jugador encontrado, mostrarlo
    {
        cout << "NICK: " << d.jugadores[posicion].nick <<
            " GANADAS: " << d.jugadores[posicion].ganadas <<
            " PERDIDAS: " << d.jugadores[posicion].perdidas << endl;
    }
}

// muestra la información de todos los jugadores registrados
void infoJugadores(DatosPartida d)
{
    cout << "Informacion de todos los jugadores:" << endl;
    for (int i = 0; i < d.numeroJugadores; i++)
    {
        cout << "NICK: " << d.jugadores[i].nick <<
            " GANADAS: " << d.jugadores[i].ganadas <<
            " PERDIDAS: " << d.jugadores[i].perdidas << endl;
    }
}

// actualiza la lista de jugadores con los nuevos datos del jugador j
void actualizaJugador(DatosPartida& d, Jugador j)
{
    int posicion = buscaJugador(d, j.nick);
    if (posicion == -1) // Jugador no encontrado
    {
        cout << "Jugador no encontrado. No se pudo actualizar" << endl;
    }
    else // Jugador encontrado, añadir info actualizada
    {
        d.jugadores[posicion] = j;
    }

    // Fichero de texto
    ofstream txt;
    txt.open("jugadores.txt");
    if (txt.is_open())
    {
        txt << d.numeroJugadores << endl;
        for (int i = 0; i < d.numeroJugadores; i++)
        {
            txt << d.jugadores[i].nick << " " <<
                d.jugadores[i].ganadas << " " <<
                d.jugadores[i].perdidas << endl;
        }
    }
}

// muestra el menu y devuelve la seleccion de usuario
int menu()
{
    cout << "Menu: " << endl;
    cout << "1 - Jugar a conecta 4" << endl;
    cout << "2 - Ver informacion del jugador" << endl;
    cout << "3 - Ver informacion de todos los jugadores" << endl;
    cout << "4 - Buscar informacion de un jugador" << endl;
```

```
    cout << "0 - Salir" << endl;
    int seleccion;
    cin >> seleccion;
    return seleccion;
}

////////////////////////////// MAIN ////////////////////////////////
int main()
{
    // Cargar datos de partidas y jugadores
    DatosPartida datos;
    cargaDatos(datos);
    // Iniciar sesión
    cout << "Nick para iniciar sesion: ";
    string nombre;
    cin >> nombre;
    Jugador jugador = iniciarSesion(datos, nombre);

    Tablero tablero;
    inicializaTablero(tablero);

    bool continuar = true;
    while (continuar)
    {
        cout << endl;
        cout << "CONECTA 4!" << endl;
        cout << "----------" << endl;
        cout << "Jugador: " << jugador.nick << endl;
        int seleccion = menu();
        if (seleccion == 0) // SALIR
        {
            continuar = false;
        }
        else if (seleccion == 1) // JUGAR
        {
            juegoConecta4(tablero, jugador, datos); // Juego
            actualizaJugador(datos, jugador);  //Actualizar datos del jugador
        }
        else if (seleccion == 2) // VER MI INFORMACIÓN
        {
            muestraInfo(datos, jugador.nick);
        }
        else if (seleccion == 3) // VER INFORMACIÓN DE LOS JUGADORES REGISTRADOS
        {
            infoJugadores(datos);
        }
        else if (seleccion == 4) // BUSCAR INFORMACIÓN DE UN JUGADOR
        {
            cout << "Nick del jugador: ";
            string nick;
            cin >> nick;
            muestraInfo(datos, nick);
        }
    }
}
```