



Facultad de Ingeniería  
Universidad de Buenos Aires  
Departamento de Computación  
Algoritmos y Programación II (75.01)  
Catedra: Ing. Patricia Calvo

Alumno: Federico Pratto (96.223)

## Trabajo Práctico 1: CUATRO EN LÍNEA V1.0

# Manual de usuario

## Reglas del juego

- El objetivo es alinear cuatro fichas sobre un tablero formado por diez filas y diez columnas.
- Cada jugador dispone de 50 fichas (50 turnos).
- Ganará la partida el primero que consiga alinear cuatro fichas consecutivas de un mismo tipo en horizontal, vertical o diagonal.
- Si todas las columnas están llenas, pero nadie ha hecho una fila válida, hay empate.

## Desarrollo del juego

- Cada jugador deberá ingresar su nombre y presionar la tecla “**ENTER**” para comenzar.
- Por turnos, los jugadores deben introducir una ficha en la columna que prefieran (siempre que no esté completa) y ésta caerá a la posición más baja.
- Los turnos se sucederán hasta que uno de los jugadores consiga colocar cuatro de sus fichas en línea - horizontal, vertical o diagonal.
- ¡El primer jugador que consiga 4 en línea gana!

# Manual del programador

## Introducción

En este documento se describirán todas las unidades que conforman el proyecto y su funcionamiento, con el fin de que el lector pueda modificar a su gusto algunos de los valores y parámetros de las funciones expuestas a su gusto.

El proyecto está formado por tres archivos fuente:

- “**main.cpp**” – archivo principal donde se realizan las llamadas a las funciones fundamentales para el funcionamiento del programa.
- “**tablero.cpp**” – archivo donde se encuentran todas las funciones relacionadas con el manejo del tablero del juego (Ej.: revisar si el tablero está lleno, mostrar el tablero, guardar la última ficha ingresada, etc.).
- “**partida.cpp**” – archivo donde se encuentran todas las funciones relacionadas con el desarrollo de la partida (Ej.: Verificar si alguien gana, realizar un cambio de turno, presentar el juego, etc.)

*Nota: los archivos **tablero.cpp** y **partida.cpp** tienen además sus correspondientes header files donde están las declaraciones de cada función.*

## Bloque “**main.cpp**”

Importo las funciones principales de los bloques **tablero.cpp** y **partida.cpp** e inicio el ciclo **main**.

```
1
2 // Importo las funciones para el inicio del juego.
3 #include "tablero.h"
4 #include "partida.h"
5
6
7 // Inicio del bloque principal.
8 int main() {
9
```

Realizamos la presentación del programa, así como la carga de los nombres de los jugadores en las variables de tipo **string** **jugador1** y **jugador2** mediante la función **presentación** declarada en el bloque **partida.cpp**

```
10 // Presentación del juego y carga de jugadores.
11 string jugador1 = "", jugador2 = "";
```

```
12      presentacion(jugador1, jugador2);  
13
```

Declaro un *array* de 10 x 10 de tipo **char** llamado *tablero*, el cual contendrá (representadas mediante letras) las fichas que vayan ingresando los jugadores en la posición correspondiente.

Además, llamamos a la función **inicializarTablero** del bloque *tablero.cpp* para inicializar la variable recién creada con el valor '-' en todas sus posiciones.

Por último, declaro e inicializo la variable de tipo **bool** *sigueElJuego* la cual le indicara al programa si debe continuar ejecutándose hasta que el juego acabe.

```
14  
15      // Creo é inicializo el tablero y comiendo el juego  
16      char tablero[10][10];  
17      inicializarTablero(tablero);  
18      bool sigueElJuego = true;  
19
```

Generamos una nueva variable de tipo **string** llamada *jugadorActual*, así como una variable de tipo **char** llamada *fichaActual*, las cuales guardaran a lo largo de la partida, el nombre y la letra que represente la ficha del jugador actual.

Además, generamos un *array* de tipo **int** de dos elementos donde se guardará la posición del tablero en donde quedo guardada la última ficha ingresada.

Finalmente, llamamos a la función **mostrarTablero** del bloque *tablero.cpp* para que muestre el tablero vacío antes del turno del primer jugador.

```
20      // Indico quien es el jugador actual y cuál es su ficha.  
21      string jugadorActual= jugador1;  
22      char fichaActual = 'X';  
23      int posicionUltimaFicha[2];  
24  
25      mostrarTablero(tablero);  
26
```

Inicio un ciclo **do-while** el cual tiene como condición de corte que el juego termine (*sigueElJuego* == **false**).

Inicializo la variable *posicionUltimaFicha* con el valor '0' en todas sus posiciones al empezar un nuevo turno.

Finalmente, indico por pantalla quien es el jugador actual, cuál es su ficha.

```
27      do{  
28          posicionUltimaFicha[0] = 0;  
29          posicionUltimaFicha[1] = 0;  
30          cout << "Es el turno de " << jugadorActual << " (tu ficha es la  
          " << fichaActual << "' )" << endl;  
31
```

Mediante la función ***elegirColumna*** del bloque ***partida.cpp*** le solicito al usuario que elija el número de columna donde desea colocar su ficha y valido si su elección fue correcta (Esto es, el valor de la columna esta entre 1 y 10).

Si el numero de la columna elegido se encuentra dentro de los limites validos, la funcion ***elegirColumna*** corrobora ademas si dicha columna no esta llena. En caso de estarlo solicita que el usuario que ingrese una nueva columna, caso contrario retorna dicho valor por referencia en ***posicionUltimaFicha[1]***, y se llama a la función ***guardarUltimaFicha*** perteneciente al bloque ***tablero.cpp*** para que registre dicha ficha en la variable ***tablero***.

```
32      // El usuario ingresa una ficha.
33      elegirColumna(posicionUltimaFicha[1], tablero);
34
35      // Guardo la ubicacion donde fue ingresada la ultima ficha.
36      guardarUltimaFicha(posicionUltimaFicha, fichaActual, tablero);
37
```

Finalmente, muestro el tablero a los jugadores para que puedan ver el estado actual del mismo luego de terminado el turno, y llamo a la funcion ***terminoElJuego*** del bloque ***partida.cpp*** la cual en base a la posicion donde fue insertada la ultima ficha determina si existe cuatro en linea en alguna direccion, o si el tablero se encuentra lleno.

En caso de que el juego siga, se realiza un cambio de turno mediante la función ***cambioDeTurno***, cambiando los valores de las variables ***fichaActual*** y ***jugadorActual*** por los que correspondan, y se reinicia el ciclo ***do-while***.

```
38      // Muestro el tablero por pantalla.
39      mostrarTablero(tablero);
40
41      // Verifico si termino el juego.
42      sigueElJuego = !terminoElJuego(posicionUltimaFicha, tablero,
43                                     fichaActual, jugadorActual);
44
45      // Si el juego continua cambio de jugador.
46      cambioDeTurno(sigueElJuego, jugador1, jugador2,
47                   jugadorActual, fichaActual);
48
49      } while(sigueElJuego); // Fin del juego.
50      return 0;
51  } // Fin del bloque principal.
```

## Bloque “*tablero.cpp*”

Inicio el bloque incluyendo al *header file* donde están todas las declaraciones de las funciones aquí implementadas.

```
1
2 #include "tablero.h"
3
```

Implementamos la función **inicializarTablero** la cual toma la variable *tablero* que es un *array* de 10 x 10 y rellena cada uno de sus elementos con el valor ‘-’ el cual representa que no hay ninguna ficha guardada.

```
4
5 /*
6  * Esta función se encarga de inicializar con el valor '-'
7  * todas los casilleros del tablero
8  */
9 void inicializarTablero(char tablero[10][10]) {
10     for (int fila = 0; fila < 10; fila++) {
11         for (int columna = 0; columna < 10; columna++) {
12             tablero[fila][columna] = '-';
13         }
14     }
15 } // Fin de la función inicializarTablero.
16
```

Implementamos la función **columnaLlena** la cual recibe la variable *tablero* y una variable de tipo **int** llamada *columna* que almacena la columna elegida por el usuario y verifica si hay espacio en dicha columna del tablero para almacenar una ficha. En caso de que haya espacio, devuelve el valor **false**, caso contrario devuelve **true**.

```
17
18 /*
19  * Esta función revisa si hay espacio disponible para agregar
20  * fichas a una columna del tablero.
21  */
22 bool columnaLlena(int columna, char tablero[10][10]) {
23     if (tablero[9][columna] == '-') {
24         return false;
25     } else {
26         cout << "La columna elegida esta llena, selecciona otra." <<
27         endl;
28         return true;
29     }
30 } // Fin de la función columnaLlena.
```

Implementamos la función **guardarUltimaFicha** la cual recibe las variables *posicionUltimaFicha*, *fichaActual* y *tablero*. La primera almacena dentro la columna que fue elegida por el usuario y recibirá la fila donde será guardada la ficha a almacenar en el tablero, la variable *fichaActual* tiene el valor de la letra que representa la ficha del jugador actual.

Iniciamos un ciclo **while** que ira revisando fila a fila las fichas del tablero, en la columna elegida por el usuario, hasta encontrar una ficha que sea igual al valor vacío '-' (Debe haber al menos una fila vacía, pues esta función es llamada luego de utilizar la función **columnaLlena** para verificar que la columna elegida por el usuario tiene espacio para más fichas)

Una vez encontrado un sitio para almacenar la ficha, se guarda esta en esa posición en el tablero y en la variable *posicionUltimaFicha[0]* se registra el número de fila donde se colocó la ficha.

```
31
32 /*
33  * Esta función se encarga de guardar la última ficha ingresada en
34  * el tablero, buscando la fila que le corresponda en base a la
35  * columna elegida por el usuario.
36  */
37 void guardarUltimaFicha(int posicionUltimaFicha[2], char fichaActual,
38     char tablero[10][10]) {
39
40     int filaVacía = 0;
41     while (tablero[filaVacía][posicionUltimaFicha[1]] != '-') {
42         filaVacía++;
43     }
44
45     // Guardo la posición donde esta la ultima ficha ingresada y la
46     // coloco en el tablero.
47     posicionUltimaFicha[0] = filaVacía;
48     tablero[posicionUltimaFicha[0]][posicionUltimaFicha[1]] =
49     fichaActual;
50 }
```

Implementamos la función **mostrarTablero** la cual recibe la variable *tablero*. Luego de realizar una limpieza de pantalla mediante un ciclo **for**, se muestra por pantalla el tablero en forma de matriz mediante un doble ciclo **for** y finalmente se imprimen los números de columna debajo de cada fila matriz con un último ciclo **for** para facilitar la claridad del juego.

```
51
52 /*
53  * Esta función se encarga de mostrar el estado del tablero luego de
54  * cada jugada.
55  */
56 void mostrarTablero(char tablero[10][10]) {
57     // Limpio la pantalla.
58     for(int i = 0; i < 20; i++){
```

```
58         cout << endl;
59     }
60     // Muestro el tablero.
61     for (int fila = 9; fila >= 0; fila--) {
62         for (int columna = 0; columna < 10; columna++) {
63             cout << tablero[fila][columna] << " ";
64         }
65         cout << endl;
66     }
67     for(int i = 1; i <= 10; i++){
68         cout << i << " " << endl;
69     }
70     cout << endl << endl;
71 } // Fin de la función mostrarTablero.
72
```

Implementamos la función **tableroLleno** la cual recibe la variable *tablero* y verifica mediante un ciclo **do-while** si hay algún espacio vacío en la última fila de cada columna (Es decir, si hay alguna columna que aun pueda almacenar fichas).

Declaramos e inicializamos la variable *hayLugar* = **false** y en caso de que aun haya espacio en el tablero el valor de esta cambiara a verdadero durante el ciclo **do-while** y sino permanecerá como falso.

Finalmente, la función retorna el valor de la variable *hayLugar* negado (Ya que, si hay lugar, eso significa que el tablero no está lleno)

```
73
74 /*
75  * Esta función verifica si el tablero no se ha llenado.
76  */
77 bool tableroLleno(char tablero[10][10]){
78
79     int i = 0;
80     bool hayLugar = false;
81
82     // Verifico si alguna columna todavia tiene lugar para almacenar
83     fichas.
84     do{
85         hayLugar = (tablero[9][i] == '-');
86         i++;
87     }while(!hayLugar && i<10);
88
89     return !hayLugar;
90 } // Fin de la función tableroLleno.
```



Implementamos la función **limpiarHilera** la cual recibe la variable de tipo **char** *hilera* la cual es un *array* de 10 elementos y será quien almacene en futuras funciones la hilera del tablero que queremos revisar para ver si algún jugador logro realizar un cuatro en línea.

Nuestra función mediante un ciclo **for** se encargara de rellenar cada espacio del *array* *hilera* con el valor '-'.

```
91
92  /*
93   * Esta función limpia mi hilera de fichas a revisar.
94   */
95  void limpiarHilera(char hilera[10]){
96      // Relleno mi hilera con '-'
97      for(int i=0; i<10; i++){
98          hilera[i] = '-';
99      }
100 } // Fin de la función
101
```

Implementamos la función **copiarHileraVertical** la cual recibe la variable de tipo **int** *columnaUltimaFicha* que contiene, como indica su nombre, la columna del tablero donde se almaceno la última ficha ingresada. Recibe además la variable de *tablero* y por último la variable *hilera* la cual almacenara la hilera que vamos a copiar del tablero para analizar si un jugador logro un cuatro en línea.

Primero que nada, llamamos a la función **limpiarHilera**, la cual se encarga de borrar la “basura” que pueda haber en la variable *hilera*, y luego mediante un ciclo **for** rellenamos cada espacio de la hilera con los elementos de la columna del tablero donde fue ingresada la última ficha.

```
102
103  /*
104   * Esta función hace una copia de la columna donde fue colocada la ultima
105   * ficha del tablero
106   * a una hilera para poder verificar si hubo 4 en linea.
107   */
108 void copiarHileraVertical(int columnaUltimaFicha, char tablero[10][10],
109 char hilera[10]){
110     // limpio mi hilera.
111     limpiarHilera(hilera);
112     // relleno mi hilera con las fichas colocadas en el tablero.
113     for(int fila = 0 ; fila < 10 ; fila++){
114         hilera[fila] = tablero[fila][columnaUltimaFicha];
115     }
116 } // Fin de la función.
```

Implementamos la función ***copiarHileraHorizontal*** la cual recibe la variable de tipo ***int*** ***filaUltimaFicha*** que contiene, como indica su nombre, la fila del tablero donde se almaceno la última ficha ingresada. Recibe además la variable de ***tablero*** y por último la variable ***hilera*** la cual almacenara la hilera que vamos a copiar del tablero para analizar si un jugador logro un cuatro en línea.

Primero que nada, llamamos a la función ***limpiarHilera***, la cual se encarga de borrar la “basura” que pueda haber en la variable ***hilera***, y luego mediante un ciclo ***for*** rellenamos cada espacio de la hilera con los elementos de la fila del tablero donde fue ingresada la última ficha.

```
117
118  /*
119   * Esta función hace una copia de la fila donde fue colocada la ultima
120   * ficha del tablero
121   * a una hilera para poder verificar si hubo 4 en linea.
122   */
123 void copiarHileraHorizontal(int filaUltimaFicha, char tablero[10][10],
124 char hilera[10]){
125     // limpio mi hilera.
126     limpiarHilera(hilera);
127     // relleno mi hilera con las fichas colocadas en el tablero.
128     for(int columna = 0 ; columna < 10 ; columna++){
129         hilera[columna] = tablero[filaUltimaFicha][columna];
130     }
131 } // Fin de la función.
```

Implementamos la función ***copiarHileraDiagonalCreciente*** la cual recibe la variable de tipo ***int*** ***posicionUltimaFicha*** que contiene, como indica su nombre, la fila y la columna del tablero donde se almaceno la última ficha ingresada. Recibe además la variable de ***tablero*** y por último la variable ***hilera*** la cual almacenara la hilera que vamos a copiar del tablero para analizar si un jugador logro un cuatro en línea.

Primero que nada, llamamos a la función ***limpiarHilera***, la cual se encarga de borrar la “basura” que pueda haber en la variable ***hilera***. Después le asignamos por referencia otro nombre a los valores almacenados en la variable ***posicionUltimaFicha*** para brindar mayor claridad al código.

Finalmente, mediante un ciclo ***for*** (cuyos límites dependerán de los valores relativos de la fila y la columna donde se ingresó la última ficha) rellenamos cada espacio de la hilera con los elementos de la diagonal creciente del tablero en base a la posición donde fue ingresada la última ficha.

```
132
133  /*
134   * Esta función hace una copia de la diagonal creciente tomando como
135   * referencia donde fue
136   * colocada la ultima ficha del tablero a una hilera para poder verificar
137   * si hubo 4 en linea.
```

```
136  */
137  void copiarHileraDiagonalCreciente(int posicionUltimaFicha[2], char
138  tablero[10][10], char hilera[10]){
139      // limpio mi hilera.
140      limpiarHilera(hilera);
141
142      // Le cambio el nombre a mis variables para mas comodidad.
143      int &fila = posicionUltimaFicha[0];
144      int &columna = posicionUltimaFicha[1];
145
146      // relleno mi hilera con las fichas colocadas en el tablero.
147      if(fila <= columna){
148          for (int i = 0, j = (columna - fila); j <= 9; i++, j++) {
149              hilera[i] = tablero[i][j];
150          }
151      } else {
152          for (int i = (fila - columna), j = 0; i <= 9; i++, j++) {
153              hilera[j] = tablero[i][j];
154          }
155      } // Fin de la función.
156
```

Implementamos la función ***copiarHileraDiagonalDecreciente*** la cual recibe la variable de tipo ***int posicionUltimaFicha*** que contiene, como indica su nombre, la fila y la columna del tablero donde se almaceno la última ficha ingresada. Recibe además la variable de ***tablero*** y por último la variable ***hilera*** la cual almacenara la hilera que vamos a copiar del tablero para analizar si un jugador logro un cuatro en línea.

Primero que nada, llamamos a la función ***limpiarHilera***, la cual se encarga de borrar la “basura” que pueda haber en la variable ***hilera***. Después le asignamos por referencia otro nombre a los valores almacenados en la variable ***posicionUltimaFicha*** para brindar mayor claridad al código.

Finalmente, mediante un ciclo ***for*** (cuyos límites dependerán de los valores relativos de la fila y la columna donde se ingresó la última ficha) rellenos cada espacio de la hilera con los elementos de la diagonal decreciente del tablero en base a la posición donde fue ingresada la última ficha.

```
157
158  /*
159   * Esta funcion hace una copia de la diagonal decreciente tomando como
160   * referencia donde fue
161   * colocada la ultima ficha del tablero a una hilera para poder verificar
162   * si hubo 4 en linea.
163   */
164  void copiarHileraDiagonalDecreciente(int posicionUltimaFicha[2], char
165  tablero[10][10], char hilera[10]){
166      // limpio mi hilera.
167      limpiarHilera(hilera);
168
```

```
166 // Le cambio el nombre a mis variables para mas comodidad.
167 int &fila = posicionUltimaFicha[0];
168 int &columna = posicionUltimaFicha[1];
169
170 // relleno mi hilera con las fichas colocadas en el tablero.
171 if(fila + columna <= 9){
172     for (int i = (fila + columna), j = 0; i >= 0; i--, j++) {
173         hilera[j] = tablero[i][j];
174     }
175 } else {
176     for (int i = 9, j = (fila + columna - 9); j <= 9; i--, j++) {
177         hilera[9-i] = tablero[i][j];
178     }
179 }
180 } // Fin de la funcion.
181
```

## Bloque “*partida.cpp*”

Inicio el bloque incluyendo al *header file* donde están todas las declaraciones de las funciones aquí implementadas.

```
1
2 #include "partida.h"
3
```

Implementamos la función **presentacion** la cual recibe por referencia las variables de tipo **string** *jugador1* y *jugador 2* y luego de darle la bienvenida a los jugadores les solicita que ingresen sus nombres.

```
4
5 /*
6  * Esta función realiza la presentacion del programa y solicita los datos
7  * para empezar el juego.
8  */
9 void presentacion(string &jugador1, string &jugador2) {
10     // Presentacion del programa
11     cout << "Bienvenido al 4 en linea!" << endl << endl;
12
13     // Carga de jugadores.
14     cout << "Jugador 1 - Ingrese su nombre: ";
15     cin >> jugador1;
16
17     cout << "Jugador 2 - Ingrese su nombre: ";
18     cin >> jugador2;
19     cout << endl;
20
21 } // Fin de la función presentacion.
22
```

Implementamos la función ***elegirColumna*** la cual recibe por referencia la variable de tipo ***int*** ***columnaElegida*** y la variable ***tablero*** que es un ***array*** de 10x10.

Luego declaro e inicializo la variable ***seguir = true*** la cual me servirá para controlar un ciclo ***do-while*** con el cual primero le solicitaremos al usuario que elija una columna, si el valor elegido es válido (se encuentra entre 1 y 10) llamaremos a la función ***columnaLlena*** del bloque ***tablero.cpp*** la cual verificara que la columna elegida tenga lugar para albergar fichas y terminara la función. Caso contrario, se le solicitara al usuario que elija otra columna.

```
23
24  /*
25   * Esta función le solicita al jugador que elija donde ingresar su ficha
    y valida dicha elección.
26   */
27  void elegirColumna(int &columnaElegida, char tablero[10][10]) {
28      // Le pido al usuario que elija una columna y valido su decisión.
29      bool seguir = true;
30
31      do {
32          cout << "Ingresa una columna entre 1 y 10: ";
33          cin >> columnaElegida;
34          cout << endl;
35
36          // Si la columna es invalido le solicito al usuario que ingrese
            otra.
37          if (columnaElegida < 1 || columnaElegida > 10) {
38              cout << "Valor invalido, intenta de nuevo." << endl;
39              columnaElegida = 0;
40          } else {
41              // Si la columna es valida verifico si queda lugar en
                ella.
42              columnaElegida--;
43              seguir = columnaLlena(columnaElegida, tablero);
44          }
45      } while (seguir);
46  } // Fin de la función elegirColumna.
47
```

Implementamos la función ***hay4EnLinea*** la cual recibe la variable de tipo ***char*** ***hilera*** la cual es un ***array*** de 10 elementos donde esta almacenada la hilera del tablero que queremos verificar si tiene un cuatro en línea dentro y la variable de tipo ***char*** ***fichaActual*** la cual contiene la letra que simboliza la ficha del jugador actual.

Declaro la variable ***fichasEncontradas = 0*** y mediante un ciclo ***do-while*** buscaremos si existen cuatro fichas consecutivas iguales a la ***fichaActual*** en la variable ***hilera*** y por tanto hay un cuatro en línea. Este ciclo se detendrá si encuentra 4 fichas consecutivas iguales o termina de revisar la hilera.

Por último, la función retornara el resultado de evaluar si ***fichasEncontradas == 4***.

```

48
49  /*
50   * Esta función determina si un jugador gana logrando un 4 en línea.
51   */
52  bool hay4EnLinea(char hilera[10], char fichaActual) {
53
54      // Inicio mi contador de fichas.
55      int fichasEncontradas = 0;
56
57      // Cuento la cantidad de fichas consecutivas del jugador actual.
58      int i = 0;
59      do {
60          if (hilera[i] == fichaActual) {
61              fichasEncontradas++;
62          } else {
63              fichasEncontradas = 0;
64          }
65          i++;
66      } while (i < 10 && fichasEncontradas < 4);
67
68      // Retorno el resultado.
69      return (fichasEncontradas == 4);
70  } // Fin de la función hay4EnLinea.
71

```

Implementamos la función **ganoAlguien** la cual recibe la variable de tipo **int** *posicionUltimaFicha* la cual es un *array* de 2 elementos que contiene la posición donde fue colocada la última ficha ingresada. Además, recibe las variables *tablero* y *fichaActual*.

Esta función se encargará de determinar si alguien logra ganar el juego o este continua, llamando a las cuatro funciones **copiarHilera...** del bloque **tablero.cpp** para ir tomando las distintas hileras de fichas que rodean a la última ficha ingresa en el tablero (vertical, horizontal, diagonal creciente, diagonal decreciente) guardar dichas hileras en el *array* *hilera* y mediante varios **if** anidados donde se llamara a la función **hay4EnLinea**, verificar si alguna de estas hileras efectivamente tiene un cuatro en línea en su interior. Caso contrario, nadie habrá ganado, y por tanto continua el juego.

```

72
73  /*
74   * Esta función evalúa si alguien gana el juego.
75   */
76  bool ganoAlguien(int ultimaFicha[2], char tablero[10][10], char
77  fichaActual) {
78
79      // Creo un array hilera donde guardare la fila a revisar.
80      char hilera[10];
81
82      copiarHileraVertical(ultimaFicha[1], tablero, hilera);
83      if (!hay4EnLinea(hilera, fichaActual)) {
84          copiarHileraHorizontal(ultimaFicha[0], tablero, hilera);
85

```

```
84         if(!hay4EnLinea(hilera, fichaActual)){
85             copiarHileraDiagonalCreciente(ultimaFicha, tablero,
86             hilera);
87             if(!hay4EnLinea(hilera, fichaActual)){
88                 copiarHileraDiagonalDecreciente(ultimaFicha,
89                 tablero, hilera);
90                 if(!hay4EnLinea(hilera, fichaActual)){
91                     return false;
92                 }
93             }
94         }
95     } return true;
96 } // Fin de la funcion ganoAlguien;
```

Implementamos la función **terminoElJuego** la cual recibe las variables *ultimaFicha* (la cual es un *array* de 2 elementos que contiene la posición donde fue guardada la última ficha justamente), *tablero*, *fichaActual*, y *jugador*. Esta última variable contiene el nombre del jugador actual.

Esta función mediante el uso de **if** anidados llamara primero a la función **ganoAlguien** para determinar si alguien gana el juego y luego a la función **tableroLleno** del bloque **tablero.cpp** para determinar si ya no queda más lugar para continuar el juego y por lo tanto hubo un empate.

```
97
98 /*
99  * Esta función verifica si termino el juego.
100 */
101 bool terminoElJuego(int ultimaFicha[2], char tablero[10][10], char
102     fichaActual, string jugador){
103     if(!ganoAlguien(ultimaFicha, tablero, fichaActual)){
104         if(!tableroLleno(tablero)){
105             return false;
106         } else {
107             cout << "Empataron, ya no hay mas fichas por colocar." <<
108                 endl;
109         }
110     } else {
111         cout << ";Felicidades " << jugador << " ganaste!" << endl;
112     }
113     return true;
114 } // Fin de la función terminoElJuego.
```

Implementamos a la función **cambioDeTurno** la cual recibe la variable *sigueElJuego*, las variables que contienen los nombres de los jugadores y las variables que contienen la ficha y el nombre del jugador actual.

Esta función verificara si aún continua el juego (*sigueElJuego* == **true**) y en ese caso cambiara el nombre del *jugadorActual* y la *fichaActual* por los valores que correspondan según quien haya jugado su turno.

```
115
116 /*
117  * Esta función realiza un cambio de turno si aún continua el juego.
118  */
119 void cambioDeTurno(bool sigueElJuego, string jugador1, string jugador2,
120 string &jugadorActual, char &fichaActual) {
121     if (sigueElJuego) {
122         if (jugadorActual == jugador1) {
123             jugadorActual = jugador2;
124             fichaActual = 'O';
125         } else {
126             jugadorActual = jugador1;
127             fichaActual = 'X';
128         }
129     } // Fin de la función cambioDeTurno.
130
```

## Bloque “*tablero.h*”

Inicio el bloque incluyendo el componente *iostream* de la biblioteca estándar de C++ y simplificando el uso de *std::cout* y *std::endl* mediante **using**.

```
1
2 #ifndef TABLERO_H_
3 #define TABLERO_H_
4
5 #include <iostream>
6
7 using std::cout;
8 using std::endl;
9
```

Declaro todas las funciones utilizadas en el bloque **tablero.cpp**

```
10
11 /*
12  * Esta función se encarga de inicializar con el valor '-'
13  * todos los casilleros del tablero
14  */
```



```
15 void inicializarTablero(char tablero[10][10]);
16
17
18 /*
19  * Esta función revisa si hay espacio disponible para agregar
20  * fichas a una columna del tablero.
21  */
22 bool columnaLlena(int columna, char tablero[10][10]);
23
24
25 /*
26  * Esta función se encarga de guardar la ultima ficha ingresada en
27  * el tablero, buscando la fila que le corresponda en base a la
28  * columna elegida por el usuario.
29  */
30 void guardarUltimaFicha(int posicionUltimaFicha[2], char fichaActual,
31 char tablero[10][10]);
32
33 /*
34  * Esta función se encarga de mostrar el estado del tablero luego de cada
35  * jugada.
36  */
37 void mostrarTablero(char tablero[10][10]);
38
39 /*
40  * Esta función verifica si el tablero no se ha llenado.
41  */
42 bool tableroLleno(char tablero[10][10]);
43
44
45 /*
46  * Esta función limpia mi hilera de fichas a revisar.
47  */
48 void limpiarHilera(char hilera[10]);
49
50
51 /*
52  * Esta función hace una copia de la columna donde fue colocada la ultima
53  * ficha del tablero
54  * a una hilera para poder verificar si hubo 4 en linea.
55  */
56 void copiarHileraVertical(int columnaUltimaFicha, char tablero[10][10],
57 char hilera[10]);
58
59 /*
60  * Esta función hace una copia de la fila donde fue colocada la ultima
61  * ficha del tablero
62  * a una hilera para poder verificar si hubo 4 en linea.
```

```
61  */
62  void copiarHileraHorizontal(int filaUltimaFicha, char tablero[10][10],
63  char hilera[10]);
64
65  /*
66   * Esta función hace una copia de la diagonal creciente tomando como
67   * referencia donde fue
68   * colocada la ultima ficha del tablero a una hilera para poder verificar
69   * si hubo 4 en linea.
70   */
71
72  void copiarHileraDiagonalCreciente(int posicionUltimaFicha[2], char
73  tablero[10][10], char hilera[10]);
74
75  /*
76   * Esta funcion hace una copia de la diagonal decreciente tomando como
77   * referencia donde fue
78   * colocada la ultima ficha del tablero a una hilera para poder verificar
79   * si hubo 4 en linea.
80   */
81  void copiarHileraDiagonalDecreciente(int posicionUltimaFicha[2], char
82  tablero[10][10], char hilera[10]);
83
84  #endif /* TABLERO_H_ */
```

## Bloque “partida.h”

Inicio el bloque incluyendo el archivo `tablero.h` para poder acceder a todas sus funciones y bibliotecas importadas, incluyo además la biblioteca estándar `string` y simplificando el uso de `std::string` y `std::cin` mediante `using`.

```
1
2  #ifndef PARTIDA_H_
3  #define PARTIDA_H_
4
5  #include "tablero.h"
6  #include <string>
7
8  using std::string;
9  using std::cin;
10
```

Declaro todas las funciones utilizadas en el bloque **partida.cpp**

```
11
12  /*
13   * Esta función realiza la presentación del programa y solicita los datos
14   * para empezar el juego.
15   */
16  void presentacion(string &jugador1, string &jugador2);
17
18  /*
19   * Esta función le solicita al jugador que elija donde ingresar su ficha
20   * y valida dicha eleccion.
21   */
22  void elegirColumna(int &columnaElegida, char tablero[10][10]);
23
24  /*
25   * Esta funcion determina si un jugador gano logrando un 4 en linea.
26   */
27  bool hay4EnLinea(char hilera[10], char fichaActual);
28
29  /*
30   * Esta funcion evalua si alguien gano el juego.
31   */
32  bool ganoAlguien(int ultimaFicha[2], char tablero[10][10], char
33  fichaActual);
34
35  /*
36   * Esta funcion verifica si termino el juego.
37   */
38  bool terminoElJuego(int ultimaFicha[2], char tablero[10][10], char
39  fichaActual, string jugador);
40
41  /*
42   * Esta función realiza un cambio de turno si aún continua el juego.
43   */
44  void cambioDeTurno(bool sigueElJuego, string jugador1, string jugador2,
45  string &jugadorActual, char &fichaActual);
46
47  #endif /* PARTIDA_H_ */
48
```

# Informe del TP

## Introducción

El objetivo del siguiente documento es explicar las ideas, suposición y estrategias que se siguieron para poder resolver el enunciado del TP.

## Suposiciones

- Supuse que el usuario no va a cometer errores al ingresar tipos de datos. Por ejemplo, si se le pide que ingrese un número no va a ingresar letras o signos.
- Supuse que el usuario puede cometer errores al ingresar un numero entre un rango. Por ejemplo, si se le pide ingresar un número entre 1 y 10, y el usuario ingresa 11 se le pedirá un nuevo número.

## Archivos del programa

El programa está conformado por los siguientes archivos:

- “**main.cpp**” – archivo principal donde se realizan las llamadas a las funciones fundamentales para el funcionamiento del programa.
- “**tablero.cpp**” – archivo donde se encuentran todas las funciones relacionadas con el manejo del tablero del juego (Ej.: revisar si el tablero está lleno, mostrar el tablero, guardar la última ficha ingresada, etc.).
- “**tablero.h**” – archivo donde se encuentran todas las funciones relacionadas con el manejo del tablero del juego.
- “**partida.cpp**” – archivo donde se encuentran todas las funciones relacionadas con el desarrollo de la partida (Ej.: Verificar si alguien gana, realizar un cambio de turno, presentar el juego, etc.).
- “**partida.h**” – archivo con la declaracion de todas las funciones relaciones con el desarrollo de la partida.

## Desarrollo del juego

- 1- Todas las llamadas a las funciones principales para que el juego funcione ocurren desde el main y el juego está conformado por un ciclo do-while que se repetirá hasta que algún jugador haya ganado la partida o el tablero este lleno.
- 2- Primero que nada, se le presenta el programa al usuario, y se solicita el nombre de cada jugador, estos nombres serán guardados en dos variables de tipo string.

- 3- Se le asigna de forma automática una letra a modo de ficha a cada jugador (una 'X' al jugador 1 y un 'O' al jugador 2).
- 4- Se crea una variable llamada tablero que será un array de tipo char bidimensional de 10x10 elementos. Dicha variable será inicializada con el valor '-' en todos sus elementos para representar el vacío. Y será en esta variable donde se irán almacenando las letras correspondientes a la ficha de cada jugador a medida que transcurra el juego.
- 5- Una vez asignados los nombre y las fichas a cada jugador, se crea una variable jugadorActual y una variable fichaActual que guardaran los valores de quien esté jugando en ese momento.
- 6- Se inicia el ciclo do-while donde se solicita al jugador actual que elija una columna donde desea colocar se ficha. Si la elección del usuario es válida (la columna elegida esta entre 1 y 10), se verifica que haya lugar en dicha columna en la variable tablero (es decir, la última posición de dicha columna no este ocupada). Si alguna condición no se cumple, se le solicita al usuario que ingrese un nuevo número de columna.
- 7- Si se cumplen ambas condiciones, se llamará a una función que revise fila a fila desde la posición cero, la columna elegida por el usuario en la variable tablero, hasta encontrar una fila en dicha columna que no esté ocupada por ninguna ficha (tenga el valor '-'). Una vez encontrada dicha posición, se procede a guardar la ficha en el tablero y se registra la ubicación de esta en una variable de tipo char de 2 elementos llamada posicionUltimaFicha.
- 8- Una vez terminado el proceso de guardar la ficha, se muestra el estado del tablero por pantalla en forma de matriz con los números de columna debajo para brindar más claridad al juego.
- 9- Luego se revisa el tablero, tomando la posición de la última ficha ingresada como punto de referencia. Para esto, se crea una variable llamada hilera, la cual es un array de 10 elementos de tipo char, y se la inicializa en todos sus elementos con el valor '-'. Esta variable será donde se guarden las hileras de fichas circundantes a la última ficha ingresada y la que se usará para ver si algún jugador logro hacer un 4 en línea.
  - a- Se copia la columna del tablero donde fue guardada la última ficha en una variable llamada hilera y se envía a una función llamada hay4EnLinea que verifica si hay cuatro fichas del mismo jugador consecutivas. Si encuentra un 4 en línea, pasa al punto 10.

-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	O	-
-	-	-	-	-	-	-	O	X	-
-	-	-	-	-	-	X	X	O	-
-	-	-	-	-	X	O	X	O	-

Tablero

Ultima ficha ingresada = [7,2]



Se reemplazan los '-' de la hilera con las fichas provenientes del tablero

X	X	O	-	-	-	-	-	-	-
---	---	---	---	---	---	---	---	---	---

hilera

- b- Si no pudo hallar un 4 en línea, se limpia la variable hilera rellenándola con el valor '-' en todos sus elementos y se copia la fila del tablero donde fue guardada la última ficha en una variable llamada hilera y se envía a la función hay4EnLinea que verifica si hay cuatro fichas del mismo jugador consecutivas. Si encuentra un 4 en línea, pasa al punto 10.

-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	O	-
-	-	-	-	-	-	-	O	X	-
-	-	-	-	-	-	X	X	O	-
-	-	-	-	-	X	O	X	O	-

Tablero

Ultima ficha ingresada = [7,2]



Se reemplazan los '-' de la hilera con las fichas provenientes del tablero

-	-	-	-	-	-	-	O	X	X
---	---	---	---	---	---	---	---	---	---

hiler

- c- Si no pudo hallar un 4 en línea, se limpia la variable hilera rellenándola con el valor '-' en todos sus elementos y se copia la diagonal ascendente que se forma en el tablero, tomando la última ficha ingresada como punto de referencia, a la variable hilera, y enviara está a la función hay4EnLinea que verifica si hay cuatro fichas del mismo jugador consecutivas. Si encuentra un 4 en línea, pasa al punto 10.

-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	O	-	-
-	-	-	-	-	-	-	O	X	-
-	-	-	-	-	-	X	X	O	-
-	-	-	-	-	X	O	X	O	-

Tablero

Ultima ficha ingresada = [7,2]



Se reemplazan los '-' de la hilera con las fichas provenientes del tablero

X	X	O	O	-	-	-	-	-	-
---	---	---	---	---	---	---	---	---	---

hiler



# Cuestionario

1. ¿Qué es un Debug?

Debug, traducido al español como “depurar” es la acción de probar un programa en busca de errores, mediante un programa depurador (Debugger)

2. ¿Qué es un “Breakpoint”?

Un “Breakpoint” (punto de quiebre) es una marca que yo realizo en una instrucción del código fuente, en la cual quiero que el debugger se detenga cuando este analizando dicho código.

3. ¿Qué es “Step Into”, “Step Over” y “Step Out”?

Al detener el debugger en un breakpoint, puedo solicitarle al mismo que realice algunas de las siguientes acciones.

- i) Step Into: ejecuta la siguiente línea de código. Si esa línea es una llamada a otra función, el programa entrará en esa función.
- ii) Step Over: igual que la anterior, pero si la siguiente línea es una función, la ejecuta sin entrar en ella.
- iii) Step Out: sale de la función actual.