Hoja 10

Importante: Los ficheros deben entregarse a través de web (por **DomJudge** y **Blackboard)**. Para cada ejercicio se entrega un **fichero** con el nombre:

<<nombre>><<apellidos>><<numE>>.c

donde<<nombre>> es el nombre del alumno, <<apellidos>> los apellidos, y <<numE>> es el número del ejercicio.

La fecha de entrega: consultar la página de la actividad en Blackboard

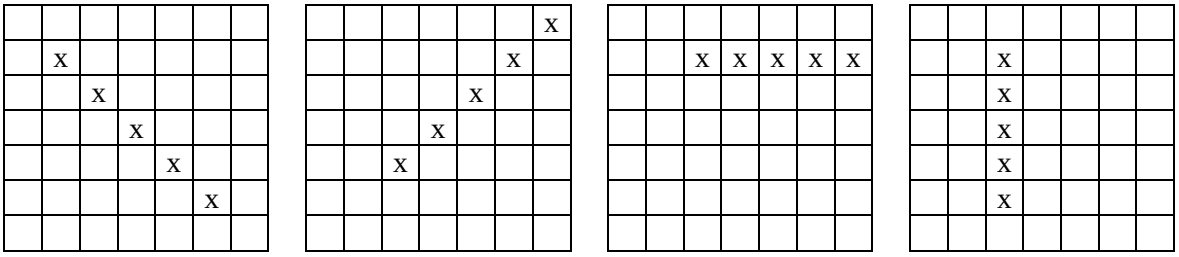
---

Observación: Ejercicio 44 no necesita una variable de tipo struct.

Ejercicio 44 (4 puntos): Se quiere desarrollar un programa que permita simular el juego de las K en Raya (similar a las 3 en Raya, pero con un tamaño variable del tablero y con la regla de K en Raya, con K determinado por el usuario). El tablero se deberá representar mediante una matriz de tamaño dinámico de NxN, en la que se deberá almacenar los siguientes caracteres:

- ‘ ’ si la casilla se encuentra vacía
- ‘X’ si la casilla está ocupada por el jugador X
- ‘O’ si la casilla está ocupada por el jugador O

El ganador del juego es el jugador que ocupe K casillas consecutivas en línea (ya sea horizontal, vertical o diagonal). Para simplificar la implementación, en el caso de diagonal, se considera sólo la diagonal principal y la diagonal secundaria de la matriz (ver los siguientes ejemplos para K=5).



El código de la función principal se encuentra en el fichero KEnRaya_main.c. Por tanto, se pide desarrollar las siguientes funciones para poder implementar el juego:

- **InicializarTablero**
  Recibe el tamaño del tablero como un entero N, reserva memoria para un tablero de NxN e inicializa el tablero de forma que todas las casillas estén vacías. Devuelve la referencia del tablero al programa principal.
- **ComprobarLleno**
  Recibe el tablero y su tamaño y devuelve un entero indicando si el tablero está lleno. El programa devuelve cero si el tablero tiene alguna casilla vacía y devuelve uno si no tiene ninguna casilla vacía y por tanto no se puede colocar ninguna ficha más.
- **ComprobarMovimiento**
  Recibe el tablero y su tamaño y la posición x e y. Comprueba si se puede colocar una ficha en la posición x e y del tablero. Una ficha solamente podrá colocarse si las

posiciones x e y se encuentran dentro del tablero y si la casilla está vacía. La función devuelve 1 si se puede colocar una ficha en la posición dada, y 0 en caso contrario.

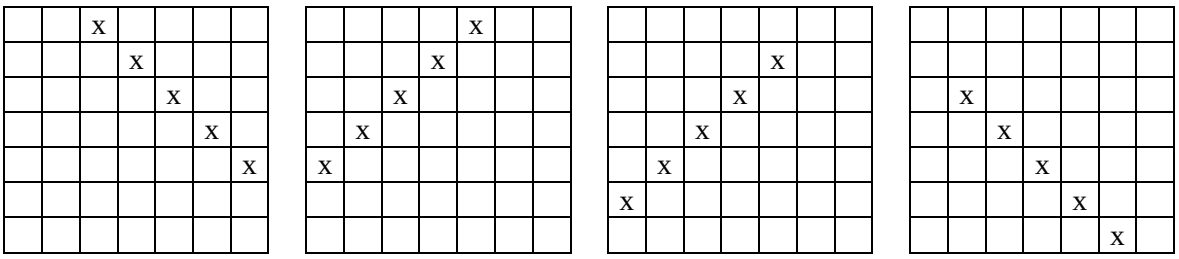
- **RealizarMovimiento**
  Recibe el tablero, la posición x e y, y el tipo del jugador ('X' o 'O') y coloca una ficha del tipo jugador indicado en la posición indicada por x e y.
- **ComprobarKenRaya**
  Recibe el tablero, su tamaño, el valor K y el tipo del jugador ('X' o 'O') y comprueba si el jugador indicado como parámetro ha conseguido hacer K en Raya. La función devuelve uno si se ha conseguido hacer K en Raya y cero en el caso contrario.
- **LiberarTablero**
  Recibe el tablero, su tamaño y libera toda la memoria reservada en la función InicializarTablero.

Nota: Todos los printf y scanf necesarios ya están incluidos en el código del programa principal. Las funciones implementadas por el alumno no deben imprimir o pedir datos.

Ejemplo de salida del programa:

```
Tamanio del tablero:
4
El valor K:
3
Jugador X elige posicion del
tablero
Fila:
0
Columna:
0

|X|| || || |
| || || || |
| || || || |
| || || || |

Jugador O elige posicion del
tablero
Fila:
0
Columna:
1

|X||O|| || |
| || || || |
| || || || |
| || || || |

Jugador X elige posicion del
tablero
Fila:
1
Columna:
1

|X||O|| || |
| ||X|| || |
| || || || |
| || || || |

Jugador O elige posicion del
tablero
Fila:
0
Columna:
2

|X||O||O|| |
| ||X|| || |
| || || || |
| || || || |

Jugador X elige posicion del
tablero
Fila:
2
Columna:
2

|X||O||O|| |
| ||X|| || |
| || ||X|| |
| || || || |

Ganador jugador X
```

Ejercicio opcional 45 (0,5 puntos Extra): Modificar la función ComprobarKenRaya del ejercicio anterior para que además de los casos anteriores también considere el jugador como el ganador si ocupa K casillas consecutivas en cualquier línea diagonal de la matriz (ver los siguientes ejemplos para K=5)



Ejercicio 46 (1 puntos): Escribir un programa para pedir una fecha y calcular el día siguiente de la fecha. El programa debe definir una estructura para guardar los tres componentes numéricos de una fecha. A continuación, para pedir la fecha, el programa llamará a una función pasando la estructura como parámetro. Esta función debe usar el formato "%d/%d/%d" en el scanf para pedir los tres números y guardarlos en los campos de la estructura recibida. Después, la función debe comprobar si la fecha es válida o no. Para simplificar, se supone que todos los meses son de 30 días y no hay años bisiestos. En caso de que la fecha introducida no sea válida la función debe dar un mensaje de "ERROR" y volver a pedir la fecha. Cuando termine esta función el programa principal llamará a otra función para calcular la fecha siguiente. Esta función recibe la estructura de la fecha actual y devuelve la fecha siguiente. Por ejemplo si la fecha es 30/12/2017 la función devolverá la fecha 01/01/2018 como una estructura. Al final, el programa principal escribirá la fecha leída y la fecha siguiente. Para escribir el resultado en formato de una fecha utilizar este formato con printf: `"%02d/%02d/%04d".`

```
Ejemplo:
30/12/2017
Hoy es: 30/12/2017
Manana es: 01/01/2018
```

Ejercicio 47 (5 puntos): Este ejercicio sirve para manejar una lista de registros de alumnos (máximo 100 alumnos). Los datos para cada alumno deberán almacenarse mediante una estructura que contenga los siguientes campos:

- **DNI**: Cadena de caracteres con un máximo de 9 caracteres (Para facilitar la lectura esta cadena no podrá contener espacios en blanco)
- **Nombre:** Cadena de caracteres con un máximo de 50 caracteres (Para facilitar la lectura esta cadena no podrá contener espacios en blanco)
- **PrimerApellido**: Cadena de caracteres con un máximo de 50 caracteres. (Para facilitar la lectura esta cadena no podrá contener espacios en blanco)
- **SegundoApellido**: Cadena de caracteres con un máximo de 50 caracteres. (Para facilitar la lectura esta cadena no podrá contener espacios en blanco)

- **FechaNacimiento**: Una estructura con los tres componentes de fecha tal como se implementó en el ejercicio 46.
- **NumAsignaturas**: Valor entero indicando el número de asignaturas que tiene matriculado el alumno
- **ListaAsignaturas**: Un array dinámico de estructuras de datos de asignaturas que tiene matriculado el alumno

Los datos de cada asignatura se almacenan mediante una estructura que contenga los siguientes campos:

- **NombreAsignatura:** Cadena de caracteres con un máximo de 50 caracteres (Para facilitar la lectura esta cadena no podrá contener espacios en blanco)
- **Creditos**: Valor entero indicando el número de créditos de la asignatura
- **Nota**: Valor float indicando la nota

El programa debe definir un array de máximo 100 estructuras de alumnos y llamar a la función **cargar_datos** para leer los datos y almacenarlos en el array. Esta función debe pedir los datos de los alumnos uno por uno y almacenarlos en la estructura de alumnos. Para pedir la fecha de nacimiento la función llamara a la función implementada en el ejercicio 46. Después de pedir los datos de cada alumno la función preguntará si el usuario quiere continuar añadiendo datos de otro alumno y en el caso de que sí, la función volverá a pedir datos.

Después de almacenar los datos, el programa pedirá el DNI de un alumno para mostrar sus datos, y calcular y mostrar su nota media. Para ello el programa debe buscar el DNI en la lista de alumnos y en caso de encontrarlo pasar la estructura del alumno a una función para calcular la nota media e imprimir toda la información.

A continuación el programa preguntará si el usuario quiere ver datos de otro alumno, y en el caso de que sí, volverá a pedir un DNI nuevo.

Nota: para almacenar la lista de asignaturas de cada alumno es necesario usar malloc con el tamaño indicado por el usuario (el número de asignaturas que tiene matriculadas el alumno). Al final el programa se deberá liberar toda la memoria reservada en el programa.

Nota 2: para escribir los datos de alumnos (nombre completo y fecha de nacimiento) utilizar el formato `"%s %s %s %02d/%02d/%02d\n"`. Para escribir datos de cada asignatura del alumno (nombre de la asignatura, créditos, nota) utilizar el formato `"%s %d %.2f\n".` Para escribir la nota media utilizar el formato `"Media: %.2f\n".`

Ejemplo de salida del programa:

```
92614859Y
Miguel Alvarez Martin
23/4/1999
2
Matematicas 3 3.6
```

```
Estadistica 3 5
Continuar (S/N)?
S
02753928A
Carmen Cortes Bustos
11/5/1998
1
Programacion 6 9
Continuar (S/N)?
S
00012345D
Antonio Bonito Iglesias
30/10/1998
3
Programacion 6 10
Matematicas 3 9.5
Estadistica 3 8
Continuar (S/N)?
N
DNI?
12345678Y
ERROR - DNI no encontrado.
Mas (S/N)?
S
00012345D
Antonio Bonito Iglesias 30/10/1998
Programacion 6 10.00
Matematicas 3 9.50
Estadistica 3 8.00
Media: 9.17
Mas (S/N)?
N
```

Ejercicio opcional 48 (0.5 puntos Extra): Se pide implementar una versión avanzada del juego ahorcado del ejercicio 43 de la hoja 9. En esta versión del juego, el jugador además de tener 5 vidas al inicio del juego, por cada letra acertada obtiene un punto y por cada 5 puntos obtiene una vida extra. El juego termina si el jugador pierde todas las vidas o no quedan palabras para adivinar. Para mantener los datos del jugador, se define una estructura con los siguientes campos:
-**nombre**: Cadena de caracteres con un máximo de 10 caracteres (Para facilitar la lectura esta cadena no podrá contener espacios en blanco)
-**vidas**: Valor entero indicando el número de vidas del jugador, inicializado por 5.
-**puntos**: Valor entero indicando la puntuación, inicializado por 0.

El programa al principio debe llamar a la función inicializar_jugador(). Esta función pedirá el nombre del usuario e inicializará la estructura del jugador y la devolverá al programa principal. A continuación el programa empieza el juego. Se pide modificar la función ahorcado para que reciba la estructura del jugador además de la palabra. Esta

función por cada letra acertada debe añadir un punto al jugador y por cada 5 puntos debe añadir una vida al jugador. La función además de imprimir el número de vidas debe imprimir la puntuación también con el formato `"Vidas y puntos %d %d\n"`.

El programa permite jugar varias partidas mientras que haya palabras para adivinar y vidas para jugar. Después de cada partida el programa principal imprimirá los datos del jugador almacenado en la estructura (nombre, vidas, puntos con el formato `">>> %s %d %d"`), y si todavía hay palabras sin adivinar y el usuario no ha perdido todas las vidas, el programa preguntará al usuario si quiere jugar otra partida.

Nota: Para simplificar el programa, la lista de palabras se define al principio del programa con esta línea del código y entonces no hay que reservar o liberar memoria con malloc o free:

```
char *lista[10] = {"coche","mesa","ahora", "gato", "hola", "uno", "una","chewbacca", "daenerys", "tyrion"};
```

Además en lugar de elegir las palabras de forma aleatoria, se elige la palabra de forma secuencial, empezando desde la primera palabra de la lista (“coche”) hasta el final (“tyrion”).

Ejemplo de salida del programa:

```
nombre?
Juan
Adivinar: -----
Vidas y puntos 5 0
Letra?
c
Has acertado!
Adivinar: c-c--
Vidas y puntos 5 1
Letra?
o
Has acertado!
Adivinar: coc--
Vidas y puntos 5 2
Letra?
w
Has fallado!
Adivinar: coc--
Vidas y puntos 4 2
Letra?
c
Adivinar: coc--
Vidas y puntos 4 2
Letra?
h
Has acertado!
Adivinar: coch-
Vidas y puntos 4 3
Letra?
p
Has fallado!
Adivinar: coch-
Vidas y puntos 3 3
Letra?
e
Has ganado!
La palabra: coche
>>> Juan 3 4
Otra Partida (S/N)?
S
Adivinar: ----
Vidas y puntos 3 4
Letra?
m
Has acertado!
Adivinar: m---
Vidas y puntos 4 5
Letra?
e
Has acertado!
Adivinar: me--
Vidas y puntos 4 6
Letra?
w
Has fallado!
Adivinar: me--
```

```
Vidas y puntos 3 6
Letra?
g
Has fallado!
Adivinar: me--
Vidas y puntos 2 6
Letra?
s
Has acertado!
Adivinar: mes-
Vidas y puntos 2 7
Letra?
p
Has fallado!
Adivinar: mes-
Vidas y puntos 1 7
Letra?
n
Has fallado!
Has perdido!
La palabra: mesa
>>> Juan 0 7
```