Backtracking

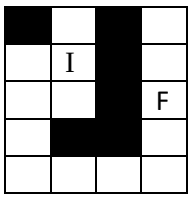
Práctica 1

- Para cada ejercicio debes entregar toda la carpeta de la solución de Visual Studio. Antes de comprimir la carpeta debes borrar los ficheros innecesarios. Para ello debes ir al menú Build, y elegir "**Clean Solution**" (limpiar solución). En el caso contrario se perderá puntos.
- La práctica se realizará por parejas, pero solo uno de los alumnos de cada grupo debe entregar la práctica en blackboard.
- Es necesario escribir en un comentario los nombres de los autores al principio del fichero que contiene el programa principal. En el caso contrario se perderá puntos.
- Si en algún ejercicio se ha indicado el orden o formato de entrada, debes respetar este orden/formato. En el caso contrario se perderá puntos.

---

1. (4 puntos) El objetivo de este ejercicio es obtener el camino a seguir para salir de un laberinto. El laberinto se representa por un tablero de NxM con algunas casillas marcadas como bloqueadas. El tablero también tiene una casilla de inicio del camino y otra casilla como el final del camino. Los movimientos permitidos son hacia arriba, abajo, izquierda y derecha.

   El programa debe pedir al usuario los datos del tablero. A continuación, el programa debe imprimir el tablero con casillas inicial, final y las bloqueadas marcado cada una con un símbolo diferente, y después, encontrar el camino y volver a imprimir el tablero con la solución ya marcada. Si el programa no encuentra una solución, debe indicarlo por un mensaje de texto.



   Es obligatorio que el programa pida al usuario los datos con el siguiente orden:
   - El número de filas del tablero, N.
   - El número de columnas del tablero, M.
   - Los índices (fila y columna) de las casillas bloqueadas (hasta que el usuario introduzca 0 y 0).
   - Los índices (fila y columna) del punto inicial.
   - Los índices (fila y columna) del punto final.

Nota 1: Para facilitar la tarea para el usuario, pedimos los índices con valores entre 1 hasta el tamaño de fila/columna, en lugar de 0 hasta tam-1. El programa debe tener los controles de error y volver a pedir los datos si no son correctos.

Por ejemplo, para el tablero de arriba las entradas serán:

Numero de filas? 5

Numero de Columnas? 4

Indices de la celda bloqueada (para salir, introducir 0 0) ? 1 1

Indices de la celda bloqueada (para salir, introducir 0 0) ? 1 3

Indices de la celda bloqueada (para salir, introducir 0 0) ? 2 3

Indices de la celda bloqueada (para salir, introducir 0 0) ? 3 3

Indices de la celda bloqueada (para salir, introducir 0 0) ? 4 2

Indices de la celda bloqueada (para salir, introducir 0 0) ? 4 3

Indices de la celda bloqueada (para salir, introducir 0 0) ? 0 0

Celda inicial? 2 2

Celda final? 3 4

2. (4 puntos, esta pregunta es del examen extraordinaria 2017/18) Desarrollar un algoritmo siguiendo el esquema de vuelta atrás o backtracking que permita resolver el juego del sudoku.

   El sudoku es un juego que está compuesto por una cuadrícula (matriz) de 9x9 casillas, dividida en 9 regiones de 3x3 casillas (las tres primeras filas y columnas forman una región, las tres primeras filas con las 3 segundas columnas forman otra, y así sucesivamente). Partiendo de números inicialmente dispuestos en algunas de las casillas, hay que completar las casillas vacías con dígitos del **1 al 9 sin que se repitan por fila, columna o región**. A continuación, se muestra una cuadricula de ejemplo:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

   Para codificar el algoritmo se propone que el alumno represente la cuadrícula del sudoku en una matriz de enteros de 9x9 (int sudoku [9][9]). Esta matriz contendrá los valores numéricos de cada celda o -1 si no se ha introducido ningún valor.

El algoritmo, siguiendo el esquema de vuelta atrás deberá completar la matriz con las casillas vacías respetando siempre las normas del juego. Por ejemplo, el sudoku anterior sería resuelto del siguiente modo:

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

A modo de ayuda, se propone (no es obligatorio) la siguiente función para comprobar si es posible anotar un valor en una posición del sudoku definida mediante su fila y columna.

```
bool esValido (int sudoku [9][9], int fila, int columna, int valor) {

   if (sudoku[fila][columna] != -1) return false;

   for (int i = 0; i < 9; i++) {
      if (sudoku[fila][i] == valor) return false;
   }

   for (int i = 0; i < 9; i++) {
      if (sudoku[i][columna] == valor) return false;
   }

   int despFila = (fila / 3) * 3;
   int despColumna = (columna / 3) * 3;

   for (int i = despFila; i < despFila + 3; i++) {
      for (int j = despColumna; j < despColumna + 3; j++) {
         if (sudoku[i][j] == valor) return false;
      }
   }

   return true;
}
```

El algoritmo de vuelta atrás final a desarrollar tendrá la siguiente **cabecera**:

```
void resolverSudoku (int sudoku [9][9]);
```

Dicho algoritmo **modificará** la matriz sudoku con los valores necesario para completar el juego.

NOTA: Suponer que el sudoku a resolver siempre tiene solución.

3. (2 puntos) Repetir el ejercicio anterior con la condición de que el sudoku pueda no tener solución. En este caso el programa debe indicarlo con un mensaje de texto.