



# UNIVERSIDAD DE GRANADA

UNIVERSIDAD DE GRANADA

2ºC

GRADO EN INGENIERÍA INFORMÁTICA

---

## Reto 2

---

**Autores:** Blanca Abril González,  
Carlos Romero de la Puente

**Asignatura:** Estructura de Datos

# Índice

<b>1. Solución del Sistema</b>	<b>2</b>
1.1. Archivo matriz.h . . . . .	3
1.2. Archivo pieza.h . . . . .	6
1.3. Archivo tablero.h . . . . .	8
1.4. Archivo colaPieza.h . . . . .	11
1.5. Archivo imagen.h . . . . .	13
1.6. Archivo makefile . . . . .	15

## 1. Solución del Sistema

La estructura de carpetas la hemos de la siguiente forma:

```
Reto 2
├── doxyfile
├── makefile
├── include
│   ├── colaPieza.h
│   ├── imagen.h
│   ├── matriz.h
│   ├── pieza.h
│   └── tablero.h
├── src
│   ├── colaPieza.cpp
│   ├── imagen.cpp
│   ├── main.cpp
│   ├── pieza.cpp
│   └── tablero.cpp
```

Ahora vamos a ir mostrando el contenido.

## 1.1. Archivo matriz.h

```
#ifndef MATRIZ_H
#define MATRIZ_H
#include <iostream>

using namespace std;

/**
 * @brief Matriz
 *
 * Un template del tipo de datos abstracto @c Matriz, se utiliza
 * para las clases @c Pieza y @c Tablero
 *
 * @author Blanca Abril Gonzalez
 * @author Carlos Romero de la Puente
 * @date Octubre 2020
 */

template <class M>
class Matriz
{
private:
    M **datos;
    int filas, columnas;

public:

    /**
     * @brief Constructor sin parametros
     */

    Matriz();

    /**
     * @brief Constructor con dos parametros, las filas y las columnas
     * @param f Numero de filas que tiene la matriz
     * @param c Numero de columnas que tiene la matriz
     * @pre f > 0 && f != null
     * @pre c > 0 && c != null
     */
}
```

```
Matriz(int f, int c);

/**
 * @brief Destructor de la clase Matriz
 */

~Matriz();

/**
 * @brief Obtenemos el valor que almacena la matriz
 * @param fil Posicion de la fila
 * @param col Posicion de la columna
 * @pre fil >= 0 && fil <= filas
 * @pre col >= 0 && col <= columnas
 */

M getContenido (int fil , int col);

/**
 * @brief Obtenemos el numero de filas
 */

int getFila ();

/**
 * @brief Obtenemos el numero de columnas
 */

int getColumna ();

/**
 * @brief Insertamos un valor dentro de la matriz
 * @param fil Posicion de la fila
 * @param col Posicion de la columna
 * @param dato Dato que queremos introducir en la matriz
 * @pre fil >= 0 && fil <= filas
 * @pre col >= 0 && col <= columnas
 * @pre dato != null
 */
```

```
void insertarContenido (int fil , int col, const M &dato);

/**
 * @brief Borramos un valor dentro de la matriz
 * @param fil Posicion en la fila
 * @param col Posicion en la columna
 * @pre fil >= 0 && fil <= filas
 * @pre col >= 0 && col <= columnas
 */

void borrarContenido (int fil , int col);

};

#endif MATRIZ_H
```

## 1.2. Archivo pieza.h

```
#ifndef PIEZA_H
#define PIEZA_H
#include <iostream>
#include "matriz.h"

using namespace std;

/**
 * @brief Pieza
 *
 * Una instancia del tipo de dato @c Pieza , es un objeto
 * compuesto por una matriz y sus filas y columnas
 *
 * @author Blanca Abril Gonzalez
 * @author Carlos Romero de la Puente
 * @date Octubre 2020
 */

class Pieza
{
private:
    matriz<bool> Pieza;
    int filas , columnas;

public:

    /**
     * @brief Constructor sin parametros
     */
    Pieza ();

    /**
     * @brief Constructor con dos parametros, las filas y las columnas
     * @param fila Numero de filas que tiene la pieza
     * @param columna Numero de columnas que tiene la pieza
     * @pre fila > 0 && fila != null
     * @pre columna > 0 && columna != null
     */
}
```

```
Pieza(int fila , int columna);

/**
 * @brief Destructor de la clase Pieza
 */

~Pieza();

/**
 * @brief Rota la pieza
 * @param pieza Pieza que se quiere rotar
 * @param direccion en la que se quiere rotar (r = derecha ,
 * l = izquierda)
 * @pre pieza != null
 * @pre direccion != "r" || direccion != "l"
 */

bool rotarPieza(Pieza pieza , char direccion);

/**
 * @brief Consultor de las dimensiones de una pieza
 * @param pieza Pieza de la que queremos conocer su dimension
 * @pre pieza != null
 */

void consultarDimensiones(Pieza pieza);
```



### 1.3. Archivo tablero.h

```
#ifndef TABLERO_H
#define TABLERO_H
#include <iostream>
#include "matriz.h"
#include "pieza.h"

using namespace std;

/**
 * @brief Tablero
 *
 * Una instancia del tipo de dato @c Tablero, es un objeto
 * compuesto por una matriz y sus filas y columnas
 *
 * @author Blanca Abril Gonzalez
 * @author Carlos Romero de la Puente
 * @date Octubre 2020
 */

class Tablero
{
private:
    matriz<bool> Tablero;
    int filas, columnas;

public:

    /**
     * @brief Constructor sin parametros
     */
    Tablero();

    /**
     * @brief Constructor con dos parametros, las filas y las columnas
     * @param fila Numero de filas que tiene la pieza
     * @param columna Numero de columnas que tiene la pieza
     * @pre fila > 0 && fila != null
     * @pre columna > 0 && columna != null
     */
}
```

```
Tablero(int fila , int columna);

/**
 * @brief Destructor de la clase Pieza
 */

~Tablero();

/**
 * @brief Consultor que nos dice si esta libre esa posicion
 * @param fila La posicion en la fila
 * @param columna La posicion en la columna
 * @pre fila >= 0 && fila <= filas
 * @pre columna >= 0 && columna != columnas
 */

bool estaLibre(int fila , int columna);

/**
 * @brief Consulta por referencia las filas y las columnas
 * @param fila La cantidad de filas
 * @param columna La cantidad de columnas
 */

void consultarDimensiones(int & filas , int & columnas);

/**
 * @brief Consulta si una pieza puede encajar en una posicion dada
 * @param pieza Pieza que queremos encajar
 * @param fila La posicion en la fila
 * @param columna La posicion en la columna
 * @pre pieza != null
 * @pre fila >= 0 && fila <= filas
 * @pre columna >= 0 && columna != columnas
 */

bool encajaPieza(Pieza & pieza , int fila , int columna);

/**
 * @brief Aade una fila en una posicion dada
 * @param posFila Posicion donde queremos anadir la nueva fila
 * @pre posFila >= 0
 */
```

```
void aniadeFila(int posFila);

/**
 * @brief Elimina una linea
 * @param fila Posicion donde queremos eliminar la linea
 * @pre fila >= 0 && fila <= 0
 */

void eliminaLinea (int fila);

/**
 * @brief Comprueba que la linea esta completa
 * @param fila Posicion donde queremos comprobar que la linea esta
 * @param linsComple Numero de lineas completas
 * @pre fila >= 0 && fila <= 0
 */

bool hayLineaCompleta (int fila , int &linsComple);

};

#endif TABLERO_H
```

## 1.4. Archivo colaPieza.h

```
#ifndef COLA_PIEZA_H
#define COLA_PIEZA_H
#include <iostream>
#include "pieza.h"

using namespace std;

/**
 * @brief ColaPieza
 *
 * Una instancia del tipo de dato @c ColaPieza, es un objeto
 * compuesto por un vector cola, su tamaño inicial y el total de piezas
 * que almacena
 * @author Blanca Abril Gonzalez
 * @author Carlos Romero de la Puente
 * @date Octubre 2020
 */

class ColaPieza
{
private:
    int tamInicial = 4;
    Pieza cola[tamInicial];
    int totalPiezas = 0;

public:

    /**
     * @brief Constructor sin parametros
     */
    ColaPieza();

    /**
     * @brief Destructor de la clase Pieza
     */
    ~ColaPieza();

    /**que
     * @brief Coloca la pieza
     * @param pieza Pieza que se quiere colocar
     * @pre pieza != null
     */
}
```

```
*/

void colocaPieza(Pieza &pieza);

/**
 * @brief Obtiene la siguiente pieza de la cola
 */

Pieza obtenerSiguiente();

/**
 * @brief Devuelve la pieza que hay en la posicion indicada
 * @param posicion Posicion de donde queremos la pieza
 * @pre posicion >= totalPiezas && posicion <= totalPiezas
 */

Pieza consultaPieza(int posicion);

/**
 * @brief Añade una pieza al final de la cola
 * @param pieza Pieza que queremos añadir
 * @pre pieza != null
 */

void aniadePieza(Pieza &pieza);
}
```

## 1.5. Archivo imagen.h

```
#ifndef IMAGEN_H
#define IMAGEN_H
#include <iostream>
#include "tablero.h"
#include "colaPieza.h"

using namespace std;

/**
 * @brief Imagen
 *
 * Una instancia del tipo de dato @c Imagen, es un objeto
 * compuesto por un tablero, la cola de piezas y la puntuacion
 *
 * @author Blanca Abril Gonzalez
 * @author Carlos Romero de la Puente
 * @date Octubre 2020
 */

class Imagen
{
private:
    Tablero tablero;
    ColaPieza colaPieza;
    int puntuacion;

public:
    /**
     * @brief Constructor sin parametros
     */
    Imagen();

    /**
     * @brief Destructor de la clase Pieza
     */
    ~Imagen();

    /**
     * @brief Guarda el juego en un fichero externo
     * @param fichero Fichero en el que queremos guardar la partida
     */
}
```

```
bool guardaJuego(std::ostream fichero);

/**
 * @brief Carga el juego de un fichero
 * @param fichero Fichero en el que se aloja la partida
 */

bool cargaJuego(std::istream fichero);

/**
 * @brief Muestra la interfaz del juego
 */

void dibujaJuego();

/**
 * @brief Dibuja solo el tablero de la partida
 */

void dibujaTablero();

/**
 * @brief Dibuja solo la cola de piezas de la partida
 */

void dibujaCola();

/**
 * @brief Devuelve la puntuacion de la partida
 */

int getPuntuacion();

/**
 * @brief Dibuja los marcadores con la puntuacion de
 * la partida
 */

void dibujaMarcadores();
}
```

## 1.6. Archivo makefile

```
OBJ=obj
BIN=bin
SRC=src
INC=include

all: $(BIN)/reto2

# Ejecutables
$(BIN)/reto2: $(OBJ)/reto2.o $(OBJ)/colaPieza.o $(OBJ)/imagen.o
$(OBJ)/pieza.o $(OBJ)/tablero.o

        g++ -o $(BIN)/reto2 $(OBJ)/reto2.o $(OBJ)/colaPieza.o
        $(OBJ)/imagen.o $(OBJ)/pieza.o $(OBJ)/tablero.o

#Objetos

$(OBJ)/reto2.o: $(SRC)/reto2.cpp $(INC)/colaPieza.h
$(INC)/imagen.h $(INC)/pieza.h $(INC)/tablero.h
        g++ -c -o $(OBJ)/reto2.o -I$(INC) $(SRC)/reto2.cpp

$(OBJ)/pieza.o:$(SRC)/pieza.cpp $(INC)/pieza.h $(INC)/matriz.h
        g++ -c -o $(OBJ)/pieza.o -I$(INC) $(SRC)/pieza.cpp

$(OBJ)/colaPieza.o:$(SRC)/colaPieza.cpp $(INC)/colaPieza.h
$(INC)/pieza.h
        g++ -c -o $(OBJ)/colaPieza.o -I$(INC) $(SRC)/colaPieza.cpp

$(OBJ)/tablero.o:$(SRC)/tablero.cpp $(INC)/tablero.h
$(INC)/pieza.h $(INC)/matriz.h
        g++ -c -o $(OBJ)/tablero.o -I$(INC) $(SRC)/tablero.cpp

$(OBJ)/imagen.o:$(SRC)/imagen.cpp $(INC)/imagen.h
$(INC)/tablero.h $(INC)/colaPieza.h
        g++ -c -o $(OBJ)/imagen.o -I$(INC) $(SRC)/imagen.cpp

clean:
        rm $(OBJ)/* $(BIN)/*
```