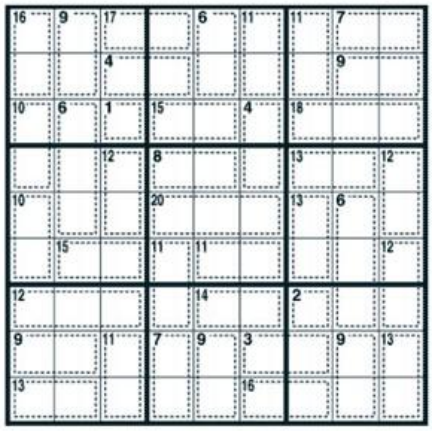
RETO II

SUDOKU KILLER



```
//  RETO 2
//  TRABAJO REALIZADO POR:
//  --> NOURA LACHHAB BOUHMADI
//  --> QUINTÍN MESA ROMERO
//

#ifndef _SUDOKU_H
#define _SUDOKU_H

#include <iostream>
#include <vector>
using namespace std;

// TDA SUDOKU KILLER

/**
 * @file sudoku_killer.h
 * @brief  Archivo de especificación del TDA SudokuKiller
 * @author Noura Lachhab Bouhmadi y Quintín Mesa Romero
 */

/**
 * @brief Tipo de dato dato
 *
 * Está constituído por tres campos: un entero para la fila en la que está un
 * determinado número, otro para la columna, y otro para el valor almacenado en dichas
 * coordenadas.
 *
 **/

struct dato
{
    int fila;
    int columna;
    int valor;

};

/**
 * @brief Tipo de dato celda
 *
 * Está constituído por tres campos: un entero para la suma de las casillas que la componen,
 * otro para el número de casillas, y un vector de objetos de tipo dato en el que se
 * almacenan los valores de dichas casillas junto con sus respectivas posiciones.
 *
 **/

struct celda
```

```
{
    int suma;
    int num_elementos;
    vector<dato> elem;


};

/**
 * @brief T.D.A SudokuKiler
 *
 * Cualquier sudoku killer que tomemos va a estar constituído por un tablero cuyas casillas
 * están agrupadas en una serie de celdas que llevan asociadas el valor de
 * la suma de los valores de las casillas que las constituyen. En "la capa de arriba", nosotros
 * consideramos el sudoku como una matríz de 9x9, con sus casillas distribuídas en celdas delimitadas
 * por una línea discontinua, pero, en "la capa de abajo", vemos el sudoku killer como un vector de
 * objetos de tipo celda en el cual, cada una de sus componentes se corresponde con cada una de las
 * celdas del sudoku, entendiéndose por celda a cada una de las áreas delimitadas por las líneas discontínuas
 * que se pueden apreciar en la imagen. Cada una de las celdas está constituída por los tres campos,
 * especificados anteriormente (la suma de las casillas, el número de casillas que la constituyen y
 * un vector de datos de tipo dato en el cual se almacenan las coordenadas de las casillas con sus respectivos valores).
 *
 **/


class SudokuKiller{

private:

    // Vector de celdas
    vector<celda> sudoku;
    //Número de celdas que hay en el sudoku
    int num_celdas;

public:

    /**
     * @brief Construtor con parámetros
     * @param numero_celdas Indica el número de celdas que va a tener el nuevo sudoku killer
     * @pre 1<= numero_celdas <= 81
     * @doc Genera un objeto SudokuKiller vacío (pone todas las casillas con el carácter " ").
     *  Para ello se recorre el vector de celdas hasta el número de celdas especificado y a
     * continuacion se recorre cada celda poniendo cada una de sus casillas inicializadas con
     * el carácter espacio en blanco " " (se le asigna a cada valor de la casilla atoi (" "))
     */
     SudokuKiller (int numero_celdas);


    /**
     * @brief Método que devuelve el valor de la suma de una determinada celda
     * @param c Número de celda en el vector
     * @pre 0 <= c < 81 (como máximo puede haber 81 celdas (si consideremos cada casilla como una celda))
     * @return Valor de la suma de la celda c
     * @post El objeto no se modifica
     *
     **/
    int getSuma(int c);

    /**
     * @brief Método que devuelve el número de casillas que tiene una determinada celda
     * @param c Número de celda en el vector.
     * @pre 0 <= c < num_celdas
     * @return Número de casillas que contiene la celda c
     * @post El objeto no se modifica
     * @post El número de casillas devuelto siempre va a ser mayor o igual que 1 y menor o igual que 81
     *
     **/
    int getSizeCelda(int c);

    /**
     * @brief Método que devuelve las coordenadas de las casillas de una determinada celda
     * @param c Número de celda de la cual se quiere saber las coordenadas
     * @pre 0 <= c < num_celdas
     * @return Un dato de tipo vector en el cual se almacenan datos de tipo dato, con valor atoi(" ")
     * @post El objeto no se modifica
     *
     **/
    vector<dato> Coordenadas(int c);
```

```
/**
 * @brief Método que devuelve la celda a la que pertenece una determinada casilla, dada su posición (f,c)
 * @param f fila de la casilla
 * @param c columna de la casilla
 * @pre 0 <= f,c < 9
 * @return Número de celda a la que pertence la casilla (f,c)
 * @post El objeto no se modifica
 * @post Se devuelve una única celda (una casilla pertenece a una sola celda)
 */
 int getCelda (int f, int c);

/**
 * @brief Método que devuelve un elemento en la posición indicada (fila, columna)
 * @param f fila en la que se encuentra el elemento a devolver
 * @param c columna en la que se encuentra el elemento a devolver
 * @pre 0<= f,c < 9
 * @return Elemento en la posición (f,c)
 * @post El objeto no se modifica.
 * @post Puede resultar que no haya ningún elemento a devolver, en cuyo caso se
 * devuelve el carácter " "
 */
 int GetElemento (int f, int c);

/**
 * @brief Método que devuelve el número de combinaciones posibles de números que pueden
 * escribirse en una determinada casilla del sudoku
 * @param f fila de la casilla
 * @param c columna de la casilla
 * @pre 0 <= f,c < 9
 * @return Devuelve las posibles combinaciones de números que pueden valer en la casilla indicada.
 * @post El objeto no se modifica
 */
int NumCombinaciones (int f, int c);


/**
 * @brief Método que nos da las posibles combinaciones de números que pueden escribirse
 * en una determinada casilla del
 * sudoku
 * @param f fila de la casilla
 * @param c columna de la casilla
 * @pre 0 <= f,c < 9
 * @return Devuelve un vector de enteros con las posibles combinaciones de números
 * que pueden valer en la casilla indicada.
 * @post El objeto no se modifica
 */
 vector<int> Combinacionees (int f, int c);

 /**
 * @brief Método que comprueba si una casilla está vacía
 * @param f fila de la casilla
 * @param c columna de la casilla
 * @pre 0 <= f,c < 9
 * @return Devuelve true si la casilla está vacía, false en caso contrario
 * @post El objeto no se modifica
 */
 bool CasillaVacia(int f, int c);

  /**
   * @brief Método que comprueba la validez de un elemento a introducir, dadas sus
   * coordenadas y su valor
   * @param dat Número a introducir en el sudoku, el cual se quiere comprobar
   * @pre 1 <= dat <= 9
   * @param f fila de la casilla del número a introducir
   * @param c columna de la casilla del número a introducir
   * @pre 0 <= f,c < 9
   * @return True en caso de que el elemento es válido, false en caso contrario
   * @post El objeto no se modifica
   * @doc Se calcula la celda a la que pertenece la casilla en la que se desea introducir
   * el elemento, se comprueba si la casilla correspondiente a las coordenadas indicadas
   * está vacía (método CasillaVacia) y a continuación, si está vacía, se comprueba si el
   * elemento es válido en la celda a la que pertenece, además de en la fila, columna y
   * submatriz de 3x3 en la que se encuentra.
   */
   bool ComprobacionDatoCorrecto (int f, int c, int dat);

   /**
```

```
        * @brief Método para la inserción de números en una determinada posición del tablero
        * @param num Número entero a introducir en el sudoku
        * @param f fila de la casilla del número a introducir
        * @param c columna de la casilla del número a introducir
        * @pre 0 <= f,c < 9
        * @pre El número no puede estar repetido ni en la fila, ni columna a la que pertenece
        * ni en el subtablero de 3x3 en el que se encuentra situado, ni en la celda a la que pertenece.
        * @post El objeto se moodifica introduciendo un elemento  nuevo al sudoku
        * @doc Se encuentra la celda a la que pertenece la casilla donde se quiere introducir el número.
        * A continuación ,se comprueba la validez del número a introducir. Para esto se hace uso
        * del método ComprobacionDatoCorrecto. Si el resultado de llamar a dicho método es true se
        * asigna el valor del elemento a introducir al valor de la casilla especificada. En caso
        * contrario, el elemento no se coloca y sigue valiendo " ".
        *
        */

        void SetNum (int f, int c, int num);

        /**
        * @brief Método que comprueba que la suma de las submatrices 3x3 sea 45
        * @param f1 fila donde comienza la submatriz
        * @param c1 columna donde comienza la submatriz
        * @param f2 fila donde acaba la submatriz
        * @param c2 columna donde acaba la submatriz
        * @pre 0<= f1,f2,c1,c2 < 9
        * @return True si la suma de las casillas de la subatriz indicada es 45
        * @post El objeto no se modifica
        * @doc Se recorren las celdas del vector sudoku, y a continuación se buscan
        * todos los elementos que se encuentren entre (f1,c1) y (f2,c2) y calculamos la suma de todos ellos.
        */
       bool ComprobacionSumaSubmatriz(int f1, int c1, int f2, int c2);

      /**
       * @brief Método que resuelve el sudoku killer por completo
       * @return Un dato de tipo SudokuKiller resuelto
       * @post El objeto original es modificado
       *
       */
      SudokuKiller Solucion();



};
#endif //_SUDOKU_H
```


NOURA LACHHAB BOUHMADI

QUINTÍN MESA ROMERO