·Reto 2 TDA: Sudoku Killer **·Asignatura:Estructura de datos**

·Manuel Marín Rodríguez, Informática - ADE

En este documento pdf se verá la especificación del TDA Sudoku Killer que yo he realizado. Mi implementación estaría formada por tres clases: Casilla, Bloque y Sudoku Killer; las cuáles se ven a continuación.

·Casilla.h:

```
//
// Created by manumarin on 13/10/23.
//


#ifndef SUDOKU_KILLER_CASILLA_H
#define SUDOKU_KILLER_CASILLA_H

#endif //SUDOKU_KILLER_CASILLA_H


/**
* @file Casilla.h
* @author Manuel Marín Rodríguez (you@domain.com)
* @brief
* @version 0.1
* @date 2023-10-03
*
* @copyright Copyright (c) 2023
*
*/

Class casilla{
	private:
	int valor,id_bloque,num_opciones;
	bool * opciones;	// Tamaño podría cambiar
dependiendo del sudoku (18x18,nxn)
	// Ese vector
	public:

/**
* @brief Constructor por parámetros de la clase casilla.
*/
casilla(int id_bloque,num_opciones);
```

```
/**
* @brief Constructor por defecto de la clase casilla.
*/
       casilla();

/**
* @brief Destructor de la clase casilla.
*/
~casilla()

/*Se suponen constructor de
copia,allocate,deallocate,reallocate y resto de métodos
relacionados con memoria dinámica... Se suponen con objetivo
de prestar atención a la solución,
 no a su implementación.*/

/**
* @brief Fija el valor de la casilla cuando ya no queden
posibilidades, comprueba que unica_opcion() sea true.
* @throw  std::incorrect  si la casilla tiene más de una
opción
*/
       void set_casilla()

/**
* @pre id no puede ser 0 o negativo.
* @brief Fijar el id del bloque al que esta casilla pertenece.
* @param id : numero entero
*/
       void set_idbloque(int id)

/**
* @brief Obtiene el numero valor de la casilla.
* @return valor
*/
       int get_valorcas() const;

/**
* @brief Obtiene el id del bloque al que pertenece la casilla.
* @return id_bloque
*/
       int get_idbloque() const;

/**
```

```
* @pre Int n debe estar entre los números 1 y 9   -->
1=posicion 0 , 2 =posicion 1 ... n= posición n-1
* @brief Obtiene la posibilidad de numero que tenga una
casilla según el índice.
* @param n La casilla deseada
* @throw std::out_of_range Throws an std::out_of_range
exception si el número de casilla no es válida
* @return Referencia a dato booleano
*/
	const bool& operator[](int n) const;

/**
* @pre Int n debe estar entre los números 1 y 9   -->
1=posicion 0 , 2 =posicion 1 ... n= posición n-1
* @brief Obtiene la posibilidad de numero que tenga una
casilla según el índice.
* @param n La casilla deseada
* @throw std::out_of_range Throws an std::out_of_range
exception si el número de casilla no es válida
* @return Referencia a dato booleano
*/
	bool& operator[](int n);

/**
* @brief Elimina una opcioń de una casilla
 * @param index : número posición
 */
	void elim_opcion(int index);
/**
 * @prem @num_unico no puede ser menor que uno ni mayor que
num_opciones.
 * @brief Cuando se tenga claro que va a ser un número si o
si, se introduce ese nuúmero como @num_unico, y el resto de
opciones se
 * ponen automáticamente en false dentro del puntero
@opciones, y directamente se establece con set_casilla(int) el
valor de la casilla,
 * que no dará error con el booleano @unica_opcion() ya que
todas se habían puesto a false menos el que necesitamos.
 * @param num_unico
 */
	void definitiva_opcion(int num_unico)

/**
```

```
 * @brief Busca que haya posibilidad del index que hayamos
introducido.Busca no repetir mucho el operador corchete a lo
largo del código, pues la clase SudokuKiller también usa el
operador
 * @return  True: Si existe esa posibilidad
 *          False: No existe esa posibilidad
 * @param index : número posición
 */
       bool existe_posibilidad(int index);

/**
 * @brief Comprueba que nuestra casilla solo tenga una opción
disponible
 * @return True: solo hay casilla única
 *         False: si hay varias casillas
 */
       bool unica_opcion();

};
```

·Bloque.h:

```
//
// Created by manumarin on 13/10/23.
//

#include "Casilla.h"

#ifndef SUDOKU_KILLER_BLOQUE_H
#define SUDOKU_KILLER_BLOQUE_H

#endif //SUDOKU_KILLER_BLOQUE_H

/**
* @file Bloque.h
* @author Manuel Marín Rodríguez (you@domain.com)
* @brief
* @version 0.1
* @date 2023-10-03
*
* @copyright Copyright (c) 2023
*
*/



Class Bloque{
       private:
       int
num_valor_total,bloque_id,num_cas,num_cas_libres,tam_max,;
       casilla * bloque;
       public:
/**
* @brief Constructor por parámetros de la clase Bloque.
*/
Bloque(int numvaltotal, int id, int num_cas);

/**
*@brief Constructor por defecto de la clase Bloque.
*/
Bloque();
```

```
/**
* @brief Destructor de la clase Bloque.
*/
~Bloque()


/*Se suponen constructor de
copia,allocate,deallocate,reallocate y resto de métodos
relacionados con memoria dinámica... Se suponen con
objetivo de prestar atención a la solución,
 no a su implementación.*/

/**
* @pre El valor de id no puede ser negativo ni 0.
* @brief Fijar el valor de ID de un bloque
* @param id : número entero
*/
   void set_bloqueId(int id);

/**
* @pre El valor de num no puede ser negativo ni 0.
* @brief Fija el num_valor_total que tiene que tener el
bloques
* @param num : número entero
*/
   void set_num_valor_total(int num);

/**
* @pre  EL valor de n no puede ser negativo ni 0.
* @brief Fija el numero de casillas que tiene un bloque.
* @param n : número entero
*/
   void set_num_cas(int n);

/**
* @brief Devuelve el id de nuestro bloque.
* @return bloque_id
*/
   int get_bloqueId() const;
```

```
/**
* @brief Devuelve el numero del valor total del bloque
* @return num_valor_total
*/
   int get_numval_tot() const;

/**
* @brief Devuelve el número de casillas que conforman el
bloque
* @return num_cas
*/
   int get_numcas () const;

/**
* @pre  Usar siempre que se haya cambiado la casilla de un
bloque. @numvalor_total y @n_elements tienen que ser mayor
que cero.
*         @nelements no puede ser mayor que @tam_max del
private.
* @brief Actualiza el bloque cuando hayamos introducido el
valor de una casilla, es decir, si rellenamos una casilla
y es posible la opción escogida.
*        Se meterá el nuevo valor en el bloque si es
válido, y una vez este metido el @num_cas pasa a
*        ser @num_cas-1(ya que hemos ocupado una) y
@numvalor_total pasa a ser
(@numvalor_total-@valor_newcasilla)
* @param numvalor_total, n_elements
*/
   void Actualizar_Bloque(int valor_newcasilla);

/**
 * @brief Con el @num_cas y el @num_valor_total, calcula
el valor más pequeño que se pueda usarpara llegar a
valor_total en función del número
 * de casillas libres y el valor de las ya ocupadas:
 * @example Valor total 17, 3 casillas: [4] [0] [0] --->
Si los valores van del 1 al 9, el valor mínimo podrá ser
cuatro (ya que si la otra casilla es nueve se necesitaría
otro cuatro)
```

```
 * @example Valor total 13, 4 casillas: [3] [4] [3] [0]
---> En este caso, el valor mínimo sería tres ya que tiene
que sumar si o si trece.
 * @return @min_value
 */
   int min_valor();

/**
 * @brief  Con el @num_cas y el @num_valor_total, calcula
el valor más grande que se pueda usar para llegar a
valor_total en función del número de casillas libres y el
valor de las ya ocupadas:
 * @example Valor total 17, 3 casillas: [4] [0] [0] --->
Si los valores van del 1 al 9, el valor máximo podrá ser 9
(ya que si la otra casilla es cuatro se necesitaría otro
nueve)
 * @example Valor total 13, 4 casillas: [3] [4] [3] [0]
---> En este caso, el valor máximo sería tres ya que tiene
que sumar si o si trece.
 * @return min_value
 */
   int max_valor();

/**
*  @brief Nos dice si realmente todas las casillas de
nuestro bloque han sido rellenadas con números diferentes
y han cumplido el valor total:
*  @return @true si el bloque ha sido completado
cumpliendo las condiciones correspondientes.
*          @false si el bloque todavía está incompleto.
*/
   bool Comprueba_bloque();
/**
* @pre Int n debe ser menor que @num_cas y mayor que 0 -->
1=posicion 0 , 2 =posicion 1 ... n= posición n-1
* @brief Obtiene la casilla  de la posición que deseemos
en nuestro bloque.
* @param n La casilla deseada
* @throw std::out_of_range Throws an std::out_of_range
exception si el número de casilla no es válida.
* @return Referencia a casilla del bloque
```

```
*/
       const casilla& pos_cas(int n) const;

/**
* @pre Int n debe ser menor que @num_cas y mayor que 0 -->
1=posicion 0 , 2 =posicion 1 ... n= posición n-1
* @brief Obtiene la casilla de la posición que deseemos en
nuestro bloque.
* @param n La casilla deseada
* @throw std::out_of_range Throws an std::out_of_range
exception si el número de casilla no es válida.
* @return Referencia a casilla del bloque
*/
       casilla& pos_cas(int n);
};
```

·SudokuKiller.h:

```
//
// Created by manumarin on 13/10/23.
//

#include "Casilla.h"
#include "Bloque.h"

#ifndef SUDOKU_KILLER_SUDOKUKILLER_H
#define SUDOKU_KILLER_SUDOKUKILLER_H

#endif //SUDOKU_KILLER_SUDOKUKILLER_H

/**
* @file SudokuKiller.h
* @author Manuel Marín Rodríguez (you@domain.com)
* @brief
* @version 0.1
* @date 2023-10-03
*
* @copyright Copyright (c) 2023
*
*/


Class SudokuKiller{
       private:
       casillas ** sudoku;         // Esta matriz de
casillas nos permitirá ver que en ninguna columna de
casillas se repite un mismo número dos veces, que será la
primera condición.
       bloque * array_bloques;     //Nuestro array_bloques
iría ordenado por los id_bloques con sus correspondientes
casillas.
                                   // Esto nos sirve para
ver la segunda condición de que todos los bloques cumplen
su val_total

       int num_f,num_c,num_bloques;
```

```
        public:

/**
* @brief Constructor por parámetros de la clase
SudokuKiller.
*/
SudokuKiller(int num_f,int num_c,num_bloques);

/**
* @brief Constructor por parámetro: Documento de texto
(caracteres)
*/
Sudokukiller(const char fileName[]);

/**
* @brief Constructor por defecto de la clase SudokuKiller.
*/
        SudokuKiller();

/**
* @brief Destructor de la clase SudokuKiller.
*/
        ~SudokuKIller()


        /*Se suponen constructor de
copia,allocate,deallocate,reallocate y resto de métodos
relacionados con memoria dinámica... Se suponen con
objetivo de prestar atención a la solución,
     no a su implementación.*/


/**
* @brief Usamos todos los métodos anteriormente
implementados para rellenar una fila entera, que respete y
coicida con los datos del resto de filas y columnas.
* @param f fila que se quiere llenar y tachar.
* @throw std::out_of_range Throws an std::out_of_range
exception si @f es mayor que @num_fil o menor igual que
cero.
*/
```

```
        void tachar_fil(int f);  //Usaría get_num() de
casilla y operador ()

/**
* @brief Usamos todos los métodos anteriormente
implementados para rellenar una columna entera, que
respete y coicida con los datos del resto de filas y
columnas.
* @param c columna que se quiere llenar y tachar.
* @throw std::out_of_range Throws an std::out_of_range
exception si @c es mayor que @num_col o menor igual que
cero.
*/
        void tachar_col(int c);

/**
* @brief Usamos todos los métodos anteriormente
implementados para rellenar un bloque entero, que respete
y coicida con los datos del resto de filas y columnas.
* @param block_id que se quiere llenar y tachar.
* @note Este método se basará en un algoritmo que jugará
con Actualizar_bloque(int), y los valores min_valor() y
max_valor, para poder realizar estimaciones en el
algoritmo de como rellenar
* cada bloque sin saltarse ninguna de las condiciones de
nuestra matriz de casillas al mismo tiempo.
* @throw std::out_of_range Throws an std::out_of_range
exception si @num_block es mayor que @num_bloques o menor
igual que cero.
*/
        void tachar_bloq(block_id);   /*Esta función no se
ha implementado en su bloque porque necesita saber los
valores del resto de las casillas d ela matriz, la cual se
encuentra en esta
                                        clase. Aún así,
usará todos los métodos implementados en la clase @bloque,
dentro de cada uno de los bloques del array_bloques*/

/**
* @brief Se usará en resuelve_sudokukiller para comprobar
que hemos resuelto el sudoku. Si todos los bloques estan
```

```
completos cumpliendo sus calores totales, y todas las
filas y columnas
* también sin repetirse ningún valor en filas,columnas ni
bloques, se habrá resuelto.
* @return @true se ha resuelto el sudoku
*         @false no se ha resuelto el sudoku
*/
       bool resuelto?();

/**
* @pre @filename debe ser un archivo de texto, no se
aceptan archivos de otro tipo de formato como .JPEG por
ejemplo.
* @brief Se cargará un sudoku a resolver en función del
archivo de texto (caracteres) que le pasemos como
parámetro.
* @param filename
*/
       void load(const char fileName[]);
/**
* @brief Una vez se haya resuelto el sudoku, esta función
se encargará de guardarlo en @filename, dejando así la
solución del sudoku en un archivo de texto.
*
*
*/
       void save_file(const char fileName[]) const
/**
* @brief Saca el estado del sudoku por pantalla, para que
tengamos una representación gráfica de la situación actual
de nuestro sudoku. Si todas las casillas están vacías,
* imprimirá por pantalla un "SUDOKU VACÍO" en vez de nxn
casillas vacías, para que sea más eficiente y ahorrar
trabajo.
*/
       void imprime_sudoku()



/**
```

```
* @brief Resuelve nuestro sudoku de manera exacta y sin errores. Para ello usa los métodos void tachar_col(int),tachar_fil(int) y tachar_bloq(int), que a su vez usan todos los anteriores
* implementados, dándole así una solución correcta a nuestro sudoku de nxn. (Aunque en este caso hayamos puesto su tamaño 9x9 para simplificar su explicación)
*/
	void resuelve_sudokukiller();

/**
* @brief Obtiene una referencia al objeto casilla elegido en la matriz, servirá para usar los objetos casillas de la matriz.
* @param f La fila deseada
* @param c La columna deseada
* @throw std::out_of_range Throws an std::out_of_range exception si la columna o filas no son válidas
* @return Una referencia a la casilla que seleccionamos
*/
	const casilla& operator[](int f, int c) const;

/**
* @brief Obtiene una referencia al objeto casilla elegido en la matriz, servirá para usar los objetos casillas de la matriz.
* @param f La fila deseada
* @param c La columna deseada
* @throw std::out_of_range Throws an std::out_of_range exception si la columna o filas no son válidas
* @return Una referencia a la casilla que seleccionamos
*/
	casilla& operator[](int f, int c );

/**
* @pre Int n debe ser menor que @num_bloques y mayor que 0 --> 1=posicion 0 , 2 =posicion 1 ... n= posición n-1
* @brief Obtiene el bloque de la posición que deseemos en nuestro array de bloques.
* @param n El bloque deseado
```

```
* @throw std::out_of_range Throws an std::out_of_range
exception si el número bloque no es válido.
* @return Referencia al bloque del @array_bloques
*/
	const Bloque& pos_arraybloques(int n) const;

/**
* @pre Int n debe ser menor que @num_bloques y mayor que 0
--> 1=posicion 0 , 2 =posicion 1 ... n= posición n-1
* @brief Obtiene el bloque de la posición que deseemos en
nuestro array de bloques.
* @param n El bloque deseado
* @throw std::out_of_range Throws an std::out_of_range
exception si el número bloque no es válido.
* @return Referencia al bloque del @array_bloques
*/
	Bloque& pos_arraybloques(int n);
};
```

Aquí terminaría mi TDA de SudokuKiller, espero que haya cumplido con la expectativa y sea una idea válida.