PRÁCTICA FINAL OBLIGATORIA Y

PRÁCTICA FINAL OPTATIVA

DE LA ASIGNATURA

INFORMÁTICA

PARA LAS TITULACIONES DE GRADO EN

INGENIERÍA MECÁNICA (UPCT)

INGENIERÍA DE ORGANIZACIÓN INDUSTRIAL (CUD-AGA)

CURSO 2011-2012

Resolución del cuadro SUDOKU

Pedro María Alcover Garau
Pedro José García Laencina

INTRODUCCIÓN

Un Sudoku es un tipo especial de Cuadro Latino. Es posible encontrar referencia bibliográfica sobre los Cuadros Latinos desde hace unos 700 años en la literatura árabe. Más recientemente fue el famoso matemático Euler, en el siglo XVIII, quien redescubrió estos rompecabezas numéricos y quien hizo uso de ellos en sus investigaciones sobre objetos combinatorios.

Los Sudokus adquirieron gran popularidad en Japón en 1986. Allí este Cuadro numérico se conocía con el nombre de *Suuji wa dokushin*, que podría traducirse como "*los números deben estar solos*". En 1997 se acuñó en actual nombre, *Sudoku*, como abreviación del nombre original japonés. En el año 2004 alcanzaron su fama mundial, de la que siguen gozando en la actualidad.

Existen muchas y diferentes presentaciones del Sudoku, como el Sudoku Killer, el Sudoku Samurai, el Sudoku Flor o el Sudoku Mariposa. Nosotros vamos a centrar nuestro trabajo en el Cuadrado Sudoku de 81 celdas dispuestas en una matriz cuadrada o retícula de 9×9 celdas, y subdividida en submatrices de 3×3, que se llaman Cajas. Así pues, una Matriz Sudoku tiene 9 filas, 9 columnas y 9 cajas. Y se presenta de forma que algunas de sus celdas tienen asignado un número entre 1 y 9. El reto de resolver un Sudoku consiste en completar el cuadro asignando a las celdas vacías un valor entre 1 y 9 de forma que no se encuentren dos números iguales dentro de una Fila, o dentro de una Columna o dentro de una Caja. En la Figura 1 se puede ver un Sudoku planteado y su solución final.

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

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

Figura 1:
Sudoku planteado y resuelto.

TEORÍA MATEMÁTICA DEL SUDOKU

Definición: Un **SUDOKU** es una matriz cuadrada de 9×9 subdividida en nueve submatrices de 3×3, también llamadas **CAJAS**, en cuyas celdas se encuentran números del conjunto $A = \{1, 2, \dots, 9\}$, de tal forma que cada fila, cada columna y cada caja contiene uno y sólo uno de los elementos del conjunto A.

Definición: Un SUDOKU INICIAL se obtiene al eliminar m números en las celdas de un Sudoku. El número m es el **ORDEN DEL SUDOKU INICIAL** $(1 \leq m \leq 81)$. Decimos que un **SUDOKU** inicial está **BIEN DEFINIDO** si y sólo si existe una única forma de llenar las celdas vacías para alcanzar un Sudoku.

No parece que se halle en la literatura científica un método simple que determine cuando un Sudoku está bien definido sin necesidad de resolverlo (al menos quien esto escribe no lo ha encontrado). En el trabajo que se plantea en esta práctica no se va a pedir que se generen nuevos Sudokus: éstos se darán como entrada, y se pueden tomar de cualquier periódico: ¿existe alguna prensa que no ofrezca su dosis diaria de Sudoku a sus lectores? Nuestra aplicación deberá dar entrada un Sudoku Inicial y su objetivo se centrará en resolverlo.

MÉTODOS DE RESOLUCIÓN DE UN SUDOKU

Hay dos operaciones básicas, que se deben realizar una y otra vez, y que concentran los esfuerzos de búsqueda de los valores que deben insertarse en cada una de las celdas vacías de un Sudoku. Una de estas operaciones debe aplicarse sobre cada Fila, cada Columna o sobre cada Caja; la otra sobre cada celda, una a una.

(1) Probar, para cada elemento del conjunto $A = \{1, 2, \ldots, 9\}$, si hay una única posición o celda a la que se pueda asignar ese valor en cada Fila, en cada Columna o en cada Caja.

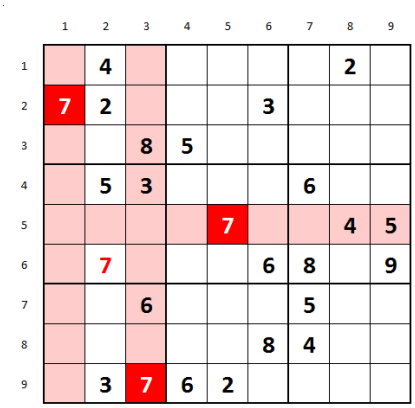


Figura 2:
Inserción de un valor en una celda porque no existe otra celda en una fila, o columna, o caja a la que pueda asignarse ese número. En el caso de la figura, el valor 7 sólo tiene una celda posible en la fila (6) y columna (2), que es donde ha quedado asignado.

Por ejemplo, en la Figura 2 podemos ver un Cuadro Sudoku al que hemos podido asignar el valor 7 a la celda de la fila (6) y columna (2), ubicada en la caja (4). Y eso porque entre todas las posiciones libres de esa columna (posiciones de las filas 3, 5, 6, 7 y 8) únicamente en la posición correspondiente de la fila (6) era posible la inserción de este número: en la fila (3) no

cabía porque ya había un 7 en la caja (1); en la fila (5) no cabía porque ya había un 7 en esa fila; en las filas (7) y (8) tampoco cabía porque ya hay un 7 en la caja (7). Otro modo de verlo: en la caja (4) únicamente en la celda de la fila (6) y columna (2) cabía el 7: en las otras, como se puede ver en las sombreadas, o estaban ya ocupadas o ya había, o en la fila o en la columna correspondiente, un 7 colocado.

(2) Probar, para cada celda libre del Sudoku, cuántos valores se podrían insertar en ella. Si se tiene una celda donde sólo es posible insertar un valor de A, entonces podremos ya insertar ese valor en esa celda.

Por ejemplo, en la Figura 3 podemos ver un Sudoku en el que es posible insertar el valor 6 en la celda de la fila (7) y columna (4). Y es que en esa posición no se puede insertar el 1, ni el 2, ni el 3, ni el 4, ni el 7, ni el 9 porque ya están en su misma caja; y no se puede insertar el valor 5 porque ya está en su misma fila (7), ni puede insertarse el valor 8 porque ya lo encontramos en su misma columna (4). Entonces en esa posición el único valor posible que cabe es el 6.

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

Figura 3:
Inserción de un valor en una celda porque no existe otro valor posible en ella.
En el caso de la figura, sólo el valor 6 puede ser insertado en la celda de fila (7) y columna (4).

Cada vez que se logra insertar, por uno de esos dos procedimientos, un nuevo valor en el Cuadro Sudoku hay que volver a comenzar todos los rastreos. Porque ese nuevo valor insertado cambia las condiciones de contorno de las demás celdas aún vacías, y puede forzar a que en otra celda de la misma caja, o misma columna, o misma fila, se pueda insertar otro valor que hasta hace un momento no tenía posición única o no era valor único posible en otra celda. Por ejemplo, en el Sudoku de la Figura 3, ahora que ya ha quedado insertado el valor 6 en la celda de fila (7) y columna (4), también podemos insertar el valor 5 en la celda de fila (9) y columna (5): porque en la caja (8) sólo quedan dos celdas libres, y ese valor 5, pendiente de ser insertado, no puede estar en la celda de fila (7) y columna (6) puesto que ya hay un 5 en esa fila. Y al mismo tiempo en la celda de fila (7) y columna (6) podemos insertar el 8 por ser el único valor posible en esa celda puesto que los demás valores ya se encuentran insertados en la correspondiente Caja. Y al haber insertado el valor 5 en la celda de fila (9) y columna (5), ahora podemos insertar otro 5 en la celda de fila (8) y columna (7) en la caja (9): porque en las otras dos filas de esa caja ya hay un 5, y de las dos columnas con celda libre en esa fila y caja sólo en la (7) no tenemos ya un 5 insertado. Y así podríamos seguir, celda a celda, hasta completar el cuadro entero.

Pero en ocasiones nos encontramos con que, de esa forma aquí presentada, no se puede rellenar por completo el cuadro Sudoku: porque se puede llegar a la situación en la que no hay un valor único posible, sino varios, en ninguna de las celdas libres…; y porque tampoco hay una posición única válida en ninguna fila, o columna, o caja, para un determinado valor.

Y entonces se debe tomar otro camino para resolver el Sudoku: buscar la solución a base de ensayo y error. Una vez se han rastreado todas las celdas y no se encuentra ninguna donde sólo pueda insertarse un valor; y una vez hemos rastreado todos los números y no se encuentra uno de ellos que sólo tenga una posible ubicación en una fila, o en una columna, o en una caja… entonces no queda más camino que probar suerte. Y entonces, de celda vacía en celda vacía, hay que insertar alguno de sus posibles valores (inicialmente ahora ya siempre más de uno) y probar, después de cada nueva inserción, si el cuadro Sudoku avanza o se atasca; y si se atasca hay que andar marcha atrás y probar con otros valores, y luego con otros, y luego con otros,… hasta que se da, por fin, con la combinación válida. No es trivial…

PRÁCTICA A DESARROLLAR:

Implementación de un programa
que resuelva un cuadro Sudoku

PARTE **OBLIGATORIA**:
Insertar valores por rastreo

PARTE **OPTATIVA**:
Insertar valores por fuerza bruta

PRIMERA PARTE: PARTE OBLIGATORIA.

(Todo el código que se ofrece en este texto de presentación de la práctica del cuadro Sudoku es orientativo. No se pretende que el alumno lo tome como solución definitiva ni, ¡mucho menos!, solución única. Y no se ofrece el código de estas líneas para que el alumno las copie en su implementación: sirven, sobre todo, como ayuda y orientación para otras partes del código que se le pide al alumno que diseñe.)

Se le pide que implemente un programa que realice las siguientes operaciones:

(1) **Admita como entrada un cuadro sudoku sin completar**. Esta entrada estará formada por una cadena de texto de longitud 81 (es decir, 82 caracteres: 81 de los dígitos más el carácter de final de cadena, ASCII 0: '\0'). Esos caracteres recogerán la entrada inicial, indicando con el carácter '0' las celdas que están sin rellenar. Los nueve primeros caracteres formarán la primera fila del sudoku; los siguientes nueve formarán la segunda fila; los siguientes, la tercera fila; y así sucesivamente.

El programa deberá validar la cadena de entrada: es decir, comprobar que todos los caracteres son dígitos y que no se han introducido datos repetidos en una misma fila, o columna, o caja.

Quizá sea útil implementar una función que verifique esa entrada. Se sugiere una posible implementación de esa función encargada de verificar la validez del Sudoku en su entrada inicial. Su prototipo sería el siguiente:

```
/*
 * ----------------------------------------------------------------
 * FUNCIÓN verificarDatosSudokuInicial()
 * ----------------------------------------------------------------
 * FUNCIÓN QUE VERIFICA LOS DATOS INICIALES DE UN SUDOKU.
 * VERIFICA QUE LOS DATOS HASTA EL MOMENTO INTRODUCIDOS CUMPLEN QUE:
 * 1. No hay dos dígitos iguales en una fila.
 * 2. No hay dos dígitos iniciales en una columna.
 * 3. No hay dos dígitos iguales en una caja.
 *
 * PARÁMETROS DE LA FUNCIÓN
 * 1. La variable array donde se tendrán los valores del sudoku.
 *
 * VALOR DE RETORNO
 * 1. Valor 1: si la entrada ha sido procesada correctamente.
 * 2. Valor 0: si la entrada se ha considerada errónea.
 *
 * ----------------------------------------------------------------
 * */
short verificarDatosSudokuInicial
(
   short s[_FILAS][_COLUMNAS]      // Cuadro Sudoku.
);
```

Y su código podría ser algo así:

```
short verificarDatosSudokuInicial(short s[_FILAS][_COLUMNAS])
{
   short f, c, k;
// Verificar todas las filas... ------------------------------------
   for(f = 0 ; f < _FILAS ; f++)
   {
      for(c = 0 ; c < _COLUMNAS ; c++)
      {
         if(s[f][c])
         {
            for(k = c + 1 ; k < _COLUMNAS ; k++)
            {
               if(s[f][c] == s[f][k])
               {
                  printf("ERROR fila %hd\n", f);
                  return 0;
               }
            }
         }
      }
   }
// Verificar todas las columnas... ---------------------------------
   for(c = 0 ; c < _COLUMNAS ; c++)
   {
      for(f = 0 ; f < _FILAS ; f++)
      {
         if(s[f][c])
         {
            for(k = f + 1 ; k < _FILAS ; k++)
            {
               if(s[f][c] == s[k][c])
               {
                  printf("ERROR columna %hd\n", c);
                  return 0;
               }
            }
         }
      }
   }
// Verificar todas las cajas... ------------------------------------
   for(f = 0 ; f < _FILAS ; f += _CAJAS)
   {
      for(c = 0 ; c < _COLUMNAS ; c += _CAJAS)
      {
         short ff, cc;
         for(ff = 0 ; ff < _FILAS / _CAJAS ; ff++)
         {
            for(cc = 0 ; cc < _COLUMNAS / _CAJAS ; cc++)
            {
               if(s[f + ff][c + cc])
               {
                  for(k = ff * _CAJAS + cc + 1 ; k < _FILAS; k++)
                  {
                     if(s[f + ff][c + cc] ==
                        s[f + k / _CAJAS][c + k % _CAJAS])
```

```
                    {
                        printf("\nERROR caja %hd - %hd\n", f, c);
                        for(k = 0 ; k < _FILAS ; k++)
                        {
                            if(k % _CAJAS == 0)
                            {
                                printf("\n");
                            }
                            printf("%4hd",
                            s[f + k / _CAJAS][c + k % _CAJAS]);
                        }
                        return 0;
                    }
                }
            }
        }
    }
  }
  return 1;
}
```

Donde la macro _FILAS ha quedado definida con el valor 9; donde la macro _COLUMNAS ha quedado definido también con el valor 9; y donde la macro _CAJAS ha quedado definido con el valor 3. Y donde la variable s es la matriz que contiene el cuadro Sudoku que ha tomado sus valores a partir de la cadena de texto de entrada. La función que toma esa cadena de 81 dígitos y asigna a partir de ella los valores del cuadro Sudoku podría ser una como la que aquí ahora se declara:

```
*
 * ---------------------------------------------------------------
 * FUNCIÓN verificarYCargarEntrada()
 * ---------------------------------------------------------------
 * FUNCIÓN QUE TOMA LA ENTRADA DEL SUDOKU COMO CADENA DE TEXTO
 * Y CARGA LA INFORMACIÓN EN LA MATRIZ s.
 * (Considera el 0 como celda vacía.)
 *
 * La función verifica que:
 * 1. Todos los caracteres de la entrada son dígitos del 0 al 9.
 * 2. La entrada tiene 81 dígitos.
 *
 * PARÁMETROS DE LA FUNCIÓN
 * 1. La cadena de entrada.
 * 2. La variable array donde se tendrán los valores del sudoku.
 *
 * VALOR DE RETORNO
 * 1. Valor 1: si la entrada ha sido procesada correctamente.
 * 2. Valor 0: si la entrada se ha considerada errónea.
 *
 * ---------------------------------------------------------------
 * */
```

```
short verificarYCargarEntrada
(
   char input[] ,
   short s[_FILAS][_COLUMNAS]
);
```

Y su código podría ser algo así como:

```
short verificarYCargarEntrada(char input[] , short s[_FILAS][_COLUMNAS])
{
   short i;
   if(strlen(input) != _CELDAS) return 0;
   for(i = 0 ; i < _CELDAS ; i++)
   {
          if(!isdigit(input[i]))
          {
                return 0;
          }
          s[i / _COLUMNAS][i % _COLUMNAS] = (short)(input[i] - '0');
   }
   return 1;
}
```

El modo de introducir el cadena queda a elección del alumno: o bien se introduce como una variable a la que se asigna el valor de la entrada en una línea de código; o bien toma la cadena de un archivo de texto; o bien se introduce la cadena desde teclado como resultado de una función de entrada. En el último apartado de este documento de presentación de la práctica final se ofrecen varios ejemplos de entrada de Sudoku; también se ofrecen las soluciones, para que se pueda verificar la correcta implementación de la aplicación.

Después de haber introducido la cadena de entrada, el programa deberá mostrar por pantalla una representación del cuadro Sudoku que recoja esos valores en su celda correspondiente. No trabajamos con bibliotecas gráficas, así que habrá que pelearse un poco con la ventana de ejecución y hacer gala de buen uso de la función `printf()`.

Por ejemplo, una entrada podría ser:

```
#define _CELDAS 81

char sudokuInput[_CELDAS + 1] =     "800000600"  "029670010"
                                    "000014050"  "600391502"
                                    "051000900"  "902006000"
                                    "060430000"  "090087160"
                                    "007000003";
```

(Como verá en el ejemplo del último apartado, se puede dar la entrada como una única cadena de 81 dígitos; en este caso de ahora, introducimos la cadena del Sudoku como concatenación de distintas cadenas separadas por espacios, no por comas.)

Esta entrada es correcta, y por tanto el programa deberá aceptarla y mostrar por pantalla un cuadro igual al mostrado en la Figura 4 u otro parecido que usted diseñe.

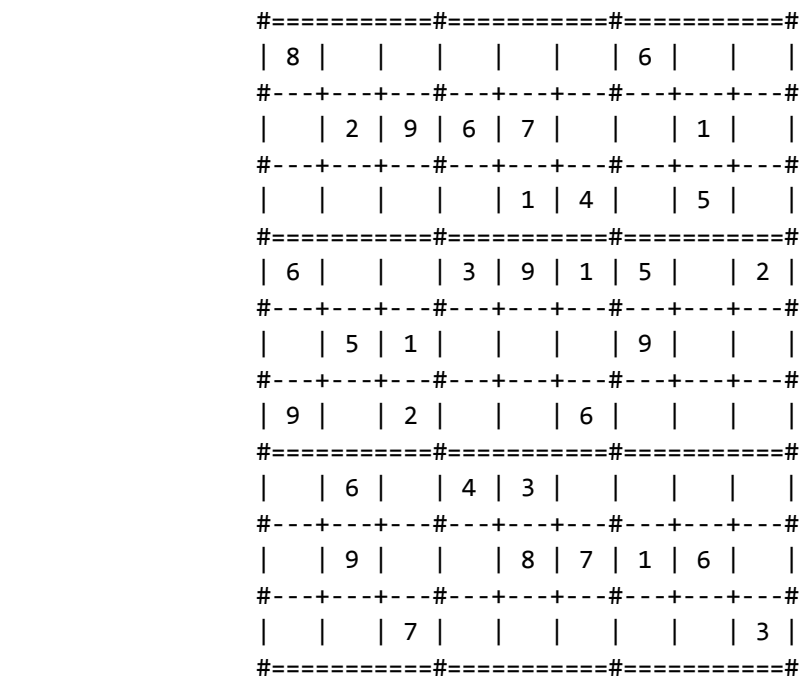


Figura 4:
Ejemplo de cómo se podría mostrar en cuadro del Sudoku
en una ventana de ejecución de comando.

Un posible prototipo de la función encargada de mostrar el Sudoku por pantalla podría ser el siguiente:

```
/*
 * ----------------------------------------------------------------
 * FUNCIÓN imprimirSudoku()
 * ----------------------------------------------------------------
 * FUNCIÓN QUE MUESTRA POR PANTALLA EL SUDOKU EN SU ESTADO ACTUAL.
 * SI TIENES VALORES 0 NO LOS IMPRIME.
 *
 * PARÁMETROS DE LA FUNCIÓN
 * 1. La variable array donde se tendrán los valores del sudoku.
 * 2. La fila y columna de la celda considerada como actual:
*
 * VALOR DE RETORNO
```

```
 * La función no devuelve nada.
 * ----------------------------------------------------------------
 * */
void imprimirSudoku
(
   short s[_FILAS][_COLUMNAS]
);
```

Y una posible definición de esta función podría ser algo como lo que a continuación se muestra:

```
void imprimirSudoku(short s[_FILAS][_COLUMNAS])
{
   short f, c;
   printf("\n\n\t#===========#===========#===========#\n\t|");
   for(f = 0 ; f < _FILAS ; f++)
   {
         if(f == _CAJAS || f == 2 * _CAJAS)
         {
               printf("#===========#===========#===========#\n\t|");
         }
         else if(f != 0)
         {
               printf("#---+---+---#---+---+---#---+---+---#\n\t|");
         }
         for(c = 0 ; c < _COLUMNAS ; c++)
         {
               if(s[f][c])
               {
                     printf(" %hd |", s[f][c]);
               }
               else
               {
                     printf("   |");
               }
         }
         printf("\n\t");
   }
   printf("#===========#===========#===========#\n");
   return;
}
```

Donde de nuevo _CAJAS es otro valor literal definido mediante una directiva define y es igual a 3 y donde _FILAS y _COLUMNAS habrán quedado definidos con sendas directivas define y cada una de ellas será equivalente al valor literal 9.

(2) **Deberá implementar dos funciones que se encarguen de realizar las dos operaciones básicas de rellenado del Sudoku**: una que determine si un determinado valor cabe en una sola posición dentro de una fila, o de una columna o de una caja; y otra que determine si en una determinada celda ocurre que solamente hay un valor posible que se le pueda asignar.

Estas dos funciones pueden tener, por ejemplo, el siguiente prototipo:

```
/*
 * ----------------------------------------------------------------
 * FUNCIÓN rellenarSiUnicaPosicion()
 * ----------------------------------------------------------------
 * FUNCIÓN QUE COMPLETA CELDAS DEL SUDOKU SI ENCUENTRA UN VALOR
 * (1 AL 9) CON UNA POSICIÓN ÚNICA VÁLIDA EN UNA FILA, O EN UNA
 * COLUMNA, O EN UNA CAJA.
 *
 * PARÁMETROS DE ENTRADA
 * 1. La variable array donde se tendrán los valores del sudoku.
 *
 * VALOR DE RETORNO
 * 1. Valor 1: si ha logrado insertar un nuevo valor.
 * 2. Valor 0: si NO ha logrado insertar en nuevo valor:
 *    no hay posiciones únicas para un valor.
 *
 * ----------------------------------------------------------------
 *
 */
short rellenarSiUnicaPosicion(short s[_FILAS][_COLUMNAS]);


/*
 * ----------------------------------------------------------------
 * FUNCIÓN rellenarSiUnicoValor()
 * ----------------------------------------------------------------
 * FUNCIÓN QUE COMPLETA CELDAS DEL SUDOKU SI ENCUENTRA UNA POSICION
 * DONDE SÓLO SEA POSIBLE INSERTAR UN VALOR (1 AL 9).
 *
 * PARÁMETROS DE ENTRADA
 * 1. La variable array donde se tendrán los valores del sudoku.
 *
 * VALOR DE RETORNO
 * 1. Valor 1: si ha logrado insertar un nuevo valor.
 * 2. Valor 0: si NO ha logrado insertar en nuevo valor:
 *    no hay posiciones con un único valor posible.
 *
 * ----------------------------------------------------------------
 * */
short rellenarSiUnicoValor(short s[_FILAS][_COLUMNAS]);
```

La primera función rastrea si en alguna fila, o en alguna columna o en alguna caja hay una única posición para algunos de los nueve valores posibles a insertar; si encuentra esa posición para ese valor lo inserta y devuelve el valor 1 (un valor True); si no lo encuentra simplemente devuelve el valor 0 (el valor False). La segunda función rastrea si en alguna celda ocurre que únicamente es posible insertar uno de los nueve valores; si la encuentra, entonces inserta ese valor en esa celda y devuelve el valor 1 (un valor True); en caso contrario devuelve el valor 0 (el valor False).

Deberá repetir la operación de rastreo valor por valor, y celda por celda, ejecutando estas dos funciones indicadas; y esto, hasta lograr completar todas las celdas del sudoku, o hasta que se

llegue a una situación en la que no es posible insertar más valores: en ese último caso, habría que proceder al relleno del resto de celdas del sudoku por técnica de fuerza bruta: tarea, ésta, que constituye el código de la parte optativa de la presente práctica.

Más adelante se le propone una función que ejecuta estas dos que ahora han quedado presentadas y realiza así la primera asignación de valores al cuadro Sudoku planteado: la función `rellenarSudoku()`.

Posiblemente le ayudaría disponer de otras funciones auxiliares menores que simplifiquen notablemente la implementación final de las dos funciones sugeridas en (2). Por ejemplo, las siguientes con los siguientes prototipos:

```
short candidatoCabeEnFila
(short s[_FILAS][_COLUMNAS] , short f , short candidato);

short candidatoCabeEnColumna
(short s[_FILAS][_COLUMNAS] , short c , short candidato);

short candidatoCabeEnCaja
(short s[_FILAS][_COLUMNAS] , short f , short c , short candidato);

short candidatoUnicoEnPosicion
(short s[_FILAS][_COLUMNAS], short f, short c, short candidato)
```

Estas funciones deberían definirse de la siguiente manera: La primera debería devolver un valor verdadero (distinto de 0) si efectivamente en la fila recibida como segundo parámetro (**short** f) sí se puede insertar el valor recibido como tercer parámetro (**short** candidato). La segunda debería devolver un valor verdadero (distinto de 0) si efectivamente en la columna recibida como segundo parámetro (**short** c) sí se puede insertar el valor recibido como tercer parámetro (**short** candidato). En la tercera debería devolver un valor verdadero (distinto de 0) si efectivamente en la caja donde está la celda marcada por la fila recibida como segundo parámetro (**short** f) y columna recibida como tercer parámetro (**short** c) sí se puede insertar el valor recibido como cuarto parámetro (**short** candidato). Y en la cuarta debería devolver un valor verdadero (distinto de 0) si efectivamente se cumple que en la celda indicada por la fila y columna recibidas como segundo y tercer parámetro (**short** f, **short** c) solamente el valor recibido como cuarto parámetro (**short** candidato) tiene cabida en esa celda.

Se muestra la implementación de dos de estas funciones, para que sirva de ayuda a la implementación de las otras dos.

```
short candidatoCabeEnFila
(short s[_FILAS][_COLUMNAS], short f , short candidato)
{
// Devuelve 1 si el candidato cabe en la fila f.
// Devuelve 0 si el candidato no cabe en la fila f.
   short c;     // Variable que recorrerá todas las columnas
   for(c = 0 ; c < _COLUMNAS ; c++)
   {
```

```
         if(s[f][c] == candidato)
         { // Si encuentra el valor candidato en la fila...
               return 0;    // No cabe: ya está el valor en la fila
         }
   }
   return 1;    // Sí cabe el valor en esa fila: aún no está ubicado.
}


short candidatoUnicoEnPosicion
(short s[_FILAS][_COLUMNAS], short f, short c, short candidato)
{
   short otroCandidato;
   for(   otroCandidato = 1 ;
         otroCandidato <= _VALORES ;
         otroCandidato++)
   {
         if(    otroCandidato != candidato                          &&
               candidatoCabeEnFila(s, f, otroCandidato)            &&
               candidatoCabeEnColumna(s, c, otroCandidato)         &&
               candidatoCabeEnCaja(s, f, c, otroCandidato))
         {// Es decir: otro candidato (distinto del recibido
          // como cuarto parámetro) cabe en la fila y en la
          // columna y en la caja indicada por los parámetros
          // segundo y tercero.
               return 0;
         }
   }
   return 1;
}
```

Donde el literal _VALORES puede definirse con un directiva de preprocesador:

```
#define _VALORES 9
```

Con estas cuatro funciones puede ya implementar al completo la segunda función, llamada `rellenarSiUnicoValor()`. Para la primera `rellenarSiUnicaPosicion()` le vendría bien disponer, además, de otras tres funciones que ayudasen a simplificar su implementación. Estas funciones podrían tener el siguiente prototipo:

```
short ubicacionUnicaEnFila
(short s[_FILAS][_COLUMNAS] , short f , short candidato);

short ubicacionUnicaEnColumna
(short s[_FILAS][_COLUMNAS] , short c , short candidato);

short ubicacionUnicaEnCaja
(short s[_FILAS][_COLUMNAS] , short f , short c , short candidato);
```

Donde la primera de ellas rastrea si en la fila indicada como segundo parámetro (**short** f) hay una única columna donde se puede insertar el valor recibido como tercer parámetro (**short** candidato): si es así, entonces la función inserta ese valor en esa posición única y

devuelve un valor True (distinto de 0). La segunda rastrea si en la columna indicada como segundo parámetro (**short** c) hay una única fila donde se puede insertar el valor recibido como tercer parámetro (**short** candidato): si es así, entonces la función inserta ese valor en esa posición única y devuelve un valor True (distinto de 0). Y la tercera rastrea si en la caja donde se encuentra la posición marcada por la fila y columna recibidas como segundo y tercer parámetro (**short** f, **short** c) hay una única celda donde se puede insertar el valor recibido como cuarto parámetro (**short** candidato): si es así, entonces la función inserta ese valor en esa posición única y devuelve un valor True (distinto de 0).

A modo de ejemplo, mostramos una posible implementación de una de estas funciones:

```
short ubicacionUnicaEnFila
(short s[_FILAS][_COLUMNAS], short f, short candidato)
{
   short c;     // Variable para las distintas columnas
   short pu;    // Cuenta el número de posiciones posibles.
   short pc;    // Almacena el valor de la última columna donde
                // puede ubicarse el valor candidato.
   for(pu = 0 , c = 0 ; c < _COLUMNAS ; c++)
   {
         if(    s[f][c] == 0                                  &&
                candidatoCabeEnColumna(s, c, candidato)      &&
                candidatoCabeEnCaja(s, f, c, candidato))
         {
         // Es decir: si la celda s[f][c] está vacía...
         // y el valor candidato cabe en esa Columna...
         // y el valor candidato cabe en esa Caja...
                pu++;        // Una nueva posición donde cabe candidato.
                pc = c;      // Columna donde cabe candidato.
         }
   }
   if(pu == 1)
   {     // Es decir, si solo hay una posición válida...
         s[f][pc] = candidato;     // Asignamos candidato a posición.
         return 1;                 // Devolvemos True
   }
   return 0;    // No se ha podido insertar valor: devolvemos False.
}
```

Y así, las funciones principales indicadas en este punto podrían tener una implementación sencilla. Por ejemplo, mostramos aquí cómo podría ser una de ellas:

```
short rellenarSiUnicaPosicion(short s[_FILAS][_COLUMNAS])
{
// Devuelve 1 si se han realizado cambios en el sudoku.
// Devuelve 0 si no se han realizado cambios.
   short candidato;
   short f, c;
   short noSeHanRealizadoCambios;

   for(noSeHanRealizadoCambios = 1 , candidato = 1 ;
```

```
       candidato <= _VALORES;
       candidato++)
   {
      for(f = 0 ; f < _FILAS ; f++)
      {
         for(c = 0 ; c < _COLUMNAS ; c++)
         {
// Si la posición ya tiene un valor, entonces no se le puede asignar otro.
// Debemos verificar que el candidato cabe en esa celda:
// 1. Cabe en esa fila.
// 2. Cabe en esa columna.
// 3. Cabe en esa misma caja.
            if(s[f][c] == 0                                    &&
               candidatoCabeEnFila(s, f, candidato)            &&
               candidatoCabeEnColumna(s, c, candidato)         &&
               candidatoCabeEnCaja(s, f, c, candidato))
            {
// Llegados aquí, el candidato cabe en la posición f, c.
// Siguiente cuestión:
//          ¿Es la posición f, c la única donde cabe candidato?
// 1. Miramos si es la única ubicación posible de la fila.
// Si lo es, inserta el valor y devuelve 1.
// Si no es la ubicación única devuelve un 0.
               noSeHanRealizadoCambios =
               !ubicacionUnicaEnFila(s, f, candidato);
// 2. Miramos si es la única ubicación posible de la columna.
// Si lo es, inserta el valor y devuelve 1.
// Si no es la ubicación única devuelve un 0.
               if(noSeHanRealizadoCambios)
               {
                  noSeHanRealizadoCambios =
                  !ubicacionUnicaEnColumna(s, c, candidato);
               }
// 3. Miramos si es la única ubicación posible de la caja.
// Si lo es, inserta el valor y devuelve 1.
// Si no es la ubicación única devuelve un 0.
               if(noSeHanRealizadoCambios)
               {
                  noSeHanRealizadoCambios =
                  !ubicacionUnicaEnCaja(s, f, c, candidato);
               }
               if(!noSeHanRealizadoCambios)
               {
                  return 1;  // Valor True: inserción realizada.
               }
            }
         }
      }
   }
   return 0;                // Valor False; inserción NO realizada.
}
```

Desde luego, esta es una posible implementación de la función. No es estrictamente necesaria su implementación, ni ésta que aquí se sugiere es la única implementación posible: caben otras muchas vías de llegar a una correcta solución.

Quizá se deberán declarar e implementar otras funciones más, que auxilien y hagan más sencilla la implementación final de la aplicación. Ésa es una decisión que compete al programador.

(3) Con todas estas funciones implementadas, **una función encargada de rellenar el sudoku** podría ser una como la que definimos a continuación. Primera mostramos su declaración: luego, su definición.

```
/*
 * ------------------------------------------------------------------
 * FUNCIÓN rellenarSudoku()
 * ------------------------------------------------------------------
 * FUNCIÓN CENTRAL EN LA TAREA DE RESOLVER EL SUDOKU.
 * INVOCA LAS TRES FUNCIONES CREADAS PARA SOLVENTAR EL PROBLEMA:
 * 1. rellenarSiUnicaPosicion()
 * 2. rellenarSiUnicoValor()
 * 3. completarASaco()
 *
 * PARÁMETROS DE LA FUNCIÓN
 * 1. La variable array donde se tendrán los valores del sudoku.
 *
 * VALOR DE RETORNO
 * La función NO devuelve valor alguno.
 * Cuando termina corresponderá a la función principal verificar que
 * los valores finales de la matriz del sudoku son correctos.
 *
 * ------------------------------------------------------------------
 * */
void rellenarSudoku
(
   short s[_FILAS][_COLUMNAS]
);
```

Como ve, en la presentación de la función, se hace referencia a una tercera función de relleno de valores en las celdas del cuadro Sudoku: la función `completarASaco()`. Esta función es precisamente la única tarea que debe resolver quien se enfrente a la parte optativa de esta práctica. Ya se ha dicho que no es trivial: ahora hay que decir que tampoco es excesivamente compleja.

Ahora mostramos una posible definición de esta función. En ella invocamos, efectivamente, a la función `completarASaco()`. Quien no se enfrente a la práctica final optativa deberá omitir de esta función las líneas marcadas en color rojo.

```
void rellenarSudoku (short s[_FILAS][_COLUMNAS])
{
   short cambio;      // Funciona como variable chivato o flag.
/* Primero rellenamos lo que se pueda.
 * Existen dos técnicas:
 * 1. Dado un número, averiguar si en un fila, o en una columna,
```

```
 *    o en un caja sólo cabe en una posición.
 * 2. Dada una posición, averiguar si en ella sólo es posible
 *    ubicar un dígito.
*/
   do
   {
         cambio = 0;
// Vayamos a la primera opción...
         while(rellenarSiUnicaPosicion(s))
         {
               cambio = 1;
         }
// Vayamos a la segunda opción...
         while(rellenarSiUnicoValor(s))
         {
               cambio = 1;
         }
   }
   while(cambio);
// Si no ha logrado terminar, procederemos a la fuerza bruta...
   if(!verificar81Datos(s))
   {
         completarASaco(s);
   }
   return;
}
```

Donde la función `verificar81Datos()` devuelve un valor verdadero si todas las celdas del Sudoku han sido rellenadas; y un valor falso (el 0) si quedan celdas sin rellenar. En caso de que las dos funciones sugeridas no logren terminar el Sudoku habrá que proceder a un ataque por fuerza bruta (función `completarASaco()`), probando a insertar valores hasta lograr una combinación final válida. Pero, de nuevo se repite esto, lograr implementar la aplicación hasta ese nivel será tarea de la parte optativa de la aplicación que el alumno debe desarrollar.

Una posible declaración de esta función podría ser:

```
/*
 * ------------------------------------------------------------------
 * FUNCIÓN verificar81Datos()
 * ------------------------------------------------------------------
 * FUNCIÓN QUE VERIFICA QUE TODAS LAS CELDAS DEL SUDOKU
 * TIENEN UN VALOR ASIGNADO.
 *
 * Esta función es invocada desde la función
 * rellenarSudoku() y no es parte de la interfaz de la aplicación.
 *
 * PARÁMETROS DE LA FUNCIÓN
 * 1. La variable array donde se tendrán los valores del sudoku.
 *
 * VALOR DE RETORNO
 * 1. Valor 1: si todos los valores de las celdas son distintos de 0.
 * 2. Valor 0: si algún valor de celda es igual a cero.
 *
```

```
 * ----------------------------------------------------------------
 * */
short verificar81Datos
(
   short s[_FILAS][_COLUMNAS]
);
```

Y unas posible implementación de la función `verificar81Datos()` podría ser la siguiente:

```
short verificar81Datos(short s[_FILAS][_COLUMNAS])
{
   short f, c;
   for(f = 0 ; f < _FILAS ; f++)
         for(c = 0 ; c < _COLUMNAS ; c++)
               if(s[f][c] == 0)
               {
                     return 0;
               }
   return 1;
}
```

Queda claro que toda la guía indicada en esta memoria es meramente orientativa. No es necesario implementar la aplicación de la misma forma que se sugiere, ni se pretende decir que esta vía sea "la mejor". Además, seguramente será útil implementar algunas funciones más, que sigan simplificando el código final de la aplicación.

Con todas las indicaciones recogidas hasta el momento en este documento, una posible función `main()` que arranque la aplicación podría ser la siguiente:

```
int main(void)
{
      short sudoku[_FILAS][_COLUMNAS];

// Cargar sudoku en la variable matriz "sudoku".
      if(verificarYCargarEntrada(sudokuInput, sudoku))
      {// El nombre de la cadena introducida en este caso en
       // sudokuInput02, que está declarada como variable global.
            imprimirSudoku(sudoku);
            if(!verificarDatosSudokuInicial(sudoku))
            {
                  printf("\n\n\nSUDOKU INICIAL INCORRECTO\n");
            }
            else
            {
                  printf("\n\n\nSUDOKU INICIAL CORRECTO\n");
```

```
// Solucionar el sudoku ...
                    rellenarSudoku(sudoku);
                    imprimirSudoku(sudoku);
                    if(verificarDatosSudokuCompleto(sudoku))
                    {
                          printf("\n\n\nSUDOKU FINAL CORRECTO\n");
                    }
                    else
                    {
                          printf("\n\n\nSUDOKU FINAL INCORRECTO\n");
                    }
             }
      }
      else
      {
             printf("El Sudoku de entrada no se puede cargar...\n");
      }

      return 0;
}
```

Donde la función `verificarDatosSudokuCompleto()` se parece a la otra invocada previamente, `verificarDatosSudokuInicial()`, y mostrada en páginas anteriores en esta memoria, pero donde ahora debe verificar que todas las celdas tienen un valor asignado y que siempre se cumple que no hay repeticiones ni en filas, ni en columnas, ni en cajas.

SEGUNDA PARTE. PARTE OPTATIVA.

Queda pendiente implementar la parte del código que realiza el ataque por fuerza bruta. Esta parte exige una función corta, pero no sencilla. Se supone que el cuadro Sudoku ha quedado rellenado hasta donde han podido las dos funciones anteriores; y que se ha llegado a un punto en el que no hay ninguna celda del cuadro donde sólo sea posible insertar un único valor, ni hay un valor con posición única en ninguna fila, o columna, o caja, del Sudoku. Hay que proceder a rellenar el cuadro probando valores posibles e intentar alcanzar una solución correcta. Si quedan, por ejemplo, 24 celdas por rellenar, y cada una de ellas tiene, al menos, 2 valores posibles, tenemos inicialmente 2^{24} posibilidades o combinaciones a probar. Evidentemente no hay que llegar a tantas pruebas, porque cada vez que se inserta un valor en una celda se modifican las posibilidades en el resto de celdas del cuadro. Cada vez que se llegue a una inserción de valores que conduzca a que en una celda no se puede insertar ningún valor habremos de desandar el camino andado y comenzar otro con otros valores.

Para esta parte del código no se facilita ayuda alguna. Hay muchas y diferentes formas de afrontar el problema. Cada alumno que quiera buscar una solución deberá enfrentarse al problema sin ayuda documental previa. Evidentemente, **cada alumno que lo desee puede acudir al profesor de teoría o a los de prácticas para recibir ayuda y orientación**.

ALGUNOS ENTRADAS PARA SUDOKUS DE EJEMPLO

E1

```
char sudokuInput01[_CELDAS + 1] =
"800000600029670010000014050600391502051000900902006000060430000090087160007000003";
```

Que ofrece el siguiente Sudoku:

```
#===========#===========#===========#
| 8 |   |   |   |   |   | 6 |   |   |
#---+---+---#---+---+---#---+---+---#
|   | 2 | 9 | 6 | 7 |   |   | 1 |   |
#---+---+---#---+---+---#---+---+---#
|   |   |   |   |   | 1 | 4 |   | 5 |
#===========#===========#===========#
| 6 |   |   | 3 | 9 | 1 | 5 |   | 2 |
#---+---+---#---+---+---#---+---+---#
|   | 5 | 1 |   |   |   | 9 |   |   |
#---+---+---#---+---+---#---+---+---#
| 9 |   | 2 |   |   | 6 |   |   |   |
#===========#===========#===========#
|   | 6 |   | 4 | 3 |   |   |   |   |
#---+---+---#---+---+---#---+---+---#
|   | 9 |   |   | 8 | 7 | 1 | 6 |   |
#---+---+---#---+---+---#---+---+---#
|   |   | 7 |   |   |   |   |   | 3 |
#===========#===========#===========#
```

Y que, mediante el procedimiento de búsqueda de valores definido en la primera parte de la práctica se llega a los siguientes valores:

```
#===========#===========#===========#
| 8 | 1 |   |   |   |   | 6 |   |   |
#---+---+---#---+---+---#---+---+---#
|   | 2 | 9 | 6 | 7 |   |   | 1 |   |
#---+---+---#---+---+---#---+---+---#
|   |   | 6 |   | 1 | 4 |   | 5 |   |
#===========#===========#===========#
| 6 |   |   | 3 | 9 | 1 | 5 |   | 2 |
#---+---+---#---+---+---#---+---+---#
|   | 5 | 1 |   |   |   | 9 |   | 6 |
#---+---+---#---+---+---#---+---+---#
| 9 |   | 2 |   |   | 6 |   |   | 1 |
#===========#===========#===========#
| 1 | 6 |   | 4 | 3 |   |   |   |   |
#---+---+---#---+---+---#---+---+---#
|   | 9 |   |   | 8 | 7 | 1 | 6 |   |
#---+---+---#---+---+---#---+---+---#
|   |   | 7 | 1 | 6 |   |   |   | 3 |
#===========#===========#===========#
```

El resto del cuadro se deberá rellenar por fuerza bruta, hasta llegar a la siguiente solución:

```
#===========#===========#===========#
| 8 | 1 | 4 | 5 | 2 | 3 | 6 | 7 | 9 |
#---+---+---#---+---+---#---+---+---#
| 5 | 2 | 9 | 6 | 7 | 8 | 3 | 1 | 4 |
#---+---+---#---+---+---#---+---+---#
| 7 | 3 | 6 | 9 | 1 | 4 | 2 | 5 | 8 |
#===========#===========#===========#
| 6 | 7 | 8 | 3 | 9 | 1 | 5 | 4 | 2 |
#---+---+---#---+---+---#---+---+---#
| 3 | 5 | 1 | 7 | 4 | 2 | 9 | 8 | 6 |
#---+---+---#---+---+---#---+---+---#
| 9 | 4 | 2 | 8 | 5 | 6 | 7 | 3 | 1 |
#===========#===========#===========#
| 1 | 6 | 5 | 4 | 3 | 9 | 8 | 2 | 7 |
#---+---+---#---+---+---#---+---+---#
| 4 | 9 | 3 | 2 | 8 | 7 | 1 | 6 | 5 |
#---+---+---#---+---+---#---+---+---#
| 2 | 8 | 7 | 1 | 6 | 5 | 4 | 9 | 3 |
#===========#===========#===========#
```

E2

```
char sudokuInput02[_CELDAS + 1] =
"008000245050708003400020870040801300100070009009265400086030050900604032320000600";
```

Que ofrece el siguiente Sudoku:

```
#===========#===========#===========#
|   |   | 8 |   |   |   | 2 | 4 | 5 |
#---+---+---#---+---+---#---+---+---#
|   | 5 |   | 7 |   | 8 |   |   | 3 |
#---+---+---#---+---+---#---+---+---#
| 4 |   |   |   | 2 |   | 8 | 7 |   |
#===========#===========#===========#
|   | 4 |   | 8 |   | 1 | 3 |   |   |
#---+---+---#---+---+---#---+---+---#
| 1 |   |   |   | 7 |   |   |   | 9 |
#---+---+---#---+---+---#---+---+---#
|   |   |   | 9 | 2 | 6 | 5 | 4 |   |   |
#===========#===========#===========#
|   | 8 | 6 |   | 3 |   |   | 5 |   |
#---+---+---#---+---+---#---+---+---#
| 9 |   |   | 6 |   | 4 |   | 3 | 2 |
#---+---+---#---+---+---#---+---+---#
| 3 | 2 |   |   |   |   | 6 |   |   |
#===========#===========#===========#
```

Y que, mediante el procedimiento de búsqueda de valores definido en la primera parte de la práctica se llega a la solución final del Sudoku, con los siguientes valores:

```
#===========#===========#===========#
| 6 | 7 | 8 | 3 | 1 | 9 | 2 | 4 | 5 |
#---+---+---#---+---+---#---+---+---#
| 2 | 5 | 1 | 7 | 4 | 8 | 9 | 6 | 3 |
#---+---+---#---+---+---#---+---+---#
| 4 | 9 | 3 | 5 | 2 | 6 | 8 | 7 | 1 |
#===========#===========#===========#
| 5 | 4 | 7 | 8 | 9 | 1 | 3 | 2 | 6 |
#---+---+---#---+---+---#---+---+---#
| 1 | 6 | 2 | 4 | 7 | 3 | 5 | 8 | 9 |
#---+---+---#---+---+---#---+---+---#
| 8 | 3 | 9 | 2 | 6 | 5 | 4 | 1 | 7 |
#===========#===========#===========#
| 7 | 8 | 6 | 9 | 3 | 2 | 1 | 5 | 4 |
#---+---+---#---+---+---#---+---+---#
| 9 | 1 | 5 | 6 | 8 | 4 | 7 | 3 | 2 |
#---+---+---#---+---+---#---+---+---#
| 3 | 2 | 4 | 1 | 5 | 7 | 6 | 9 | 8 |
#===========#===========#===========#
```

E3

```
char sudokuInput03[_CELDAS + 1] =
"000483000604000082000000000000830000010029076008001000500002300000175000000000107";
```

Que ofrece el siguiente Sudoku:

```
#===========#===========#===========#
|   |   |   | 4 | 8 | 3 |   |   |   |
#---+---+---#---+---+---#---+---+---#
| 6 |   | 4 |   |   |   |   | 8 | 2 |
#---+---+---#---+---+---#---+---+---#
|   |   |   |   |   |   |   |   |   |
#===========#===========#===========#
|   |   |   | 8 | 3 |   |   |   |   |
#---+---+---#---+---+---#---+---+---#
|   | 1 |   |   | 2 | 9 |   | 7 | 6 |
#---+---+---#---+---+---#---+---+---#
|   |   | 8 |   |   | 1 |   |   |   |
#===========#===========#===========#
| 5 |   |   |   |   | 2 | 3 |   |   |
#---+---+---#---+---+---#---+---+---#
|   |   |   | 1 | 7 | 5 |   |   |   |
#---+---+---#---+---+---#---+---+---#
|   |   |   |   |   |   | 1 |   | 7 |
#===========#===========#===========#
```

Y que, mediante el procedimiento de búsqueda de valores definido en la primera parte de la práctica se llega a la solución final del Sudoku, con los siguientes valores:

```
#===========#===========#===========#
| 1 | 2 | 5 | 4 | 8 | 3 | 7 | 6 | 9 |
#---+---+---#---+---+---#---+---+---#
| 6 | 3 | 4 | 9 | 1 | 7 | 5 | 8 | 2 |
#---+---+---#---+---+---#---+---+---#
| 8 | 9 | 7 | 2 | 5 | 6 | 4 | 3 | 1 |
#===========#===========#===========#
| 7 | 6 | 2 | 8 | 3 | 4 | 9 | 1 | 5 |
#---+---+---#---+---+---#---+---+---#
| 4 | 1 | 3 | 5 | 2 | 9 | 8 | 7 | 6 |
#---+---+---#---+---+---#---+---+---#
| 9 | 5 | 8 | 7 | 6 | 1 | 2 | 4 | 3 |
#===========#===========#===========#
| 5 | 7 | 1 | 6 | 4 | 2 | 3 | 9 | 8 |
#---+---+---#---+---+---#---+---+---#
| 3 | 8 | 9 | 1 | 7 | 5 | 6 | 2 | 4 |
#---+---+---#---+---+---#---+---+---#
| 2 | 4 | 6 | 3 | 9 | 8 | 1 | 5 | 7 |
#===========#===========#===========#
```

E4

```
char sudokuInput04[_CELDAS + 1] =
"000040060075000090900001000004008736300004000721900500000200001030000850010060000";
```

Que ofrece el siguiente Sudoku:

```
#===========#===========#===========#
|   |   |   |   | 4 |   |   | 6 |   |
#---+---+---#---+---+---#---+---+---#
|   | 7 | 5 |   |   |   |   | 9 |   |
#---+---+---#---+---+---#---+---+---#
| 9 |   |   |   |   | 1 |   |   |   |
#===========#===========#===========#
|   |   | 4 |   |   |   | 8 | 7 | 3 | 6 |
#---+---+---#---+---+---#---+---+---#
| 3 |   |   |   |   | 4 |   |   |   |
#---+---+---#---+---+---#---+---+---#
| 7 | 2 | 1 | 9 |   |   | 5 |   |   |
#===========#===========#===========#
|   |   |   | 2 |   |   |   |   | 1 |
#---+---+---#---+---+---#---+---+---#
|   | 3 |   |   |   |   | 8 | 5 |   |
#---+---+---#---+---+---#---+---+---#
|   | 1 |   |   | 6 |   |   |   |   |
#===========#===========#===========#
```

Y que, mediante el procedimiento de búsqueda de valores definido en la primera parte de la práctica se llega a la solución final del Sudoku, con los siguientes valores:

```
#===========#===========#===========#
| 2 | 8 | 3 | 7 | 4 | 9 | 1 | 6 | 5 |
#---+---+---#---+---+---#---+---+---#
| 1 | 7 | 5 | 6 | 8 | 2 | 4 | 9 | 3 |
#---+---+---#---+---+---#---+---+---#
| 9 | 4 | 6 | 3 | 5 | 1 | 2 | 7 | 8 |
#===========#===========#===========#
| 5 | 9 | 4 | 1 | 2 | 8 | 7 | 3 | 6 |
#---+---+---#---+---+---#---+---+---#
| 3 | 6 | 8 | 5 | 7 | 4 | 9 | 1 | 2 |
#---+---+---#---+---+---#---+---+---#
| 7 | 2 | 1 | 9 | 3 | 6 | 5 | 8 | 4 |
#===========#===========#===========#
| 8 | 5 | 7 | 2 | 9 | 3 | 6 | 4 | 1 |
#---+---+---#---+---+---#---+---+---#
| 6 | 3 | 2 | 4 | 1 | 7 | 8 | 5 | 9 |
#---+---+---#---+---+---#---+---+---#
| 4 | 1 | 9 | 8 | 6 | 5 | 3 | 2 | 7 |
#===========#===========#===========#
```

E5

```
char sudokuInput05[_CELDAS + 1] =
"2007008001070206438003012900406000826002070090000905060060720000725040000100300000";
```

Que ofrece el siguiente Sudoku:

```
#===========#===========#===========#
| 2 |   |   | 7 |   |   | 8 |   |   |
#---+---+---#---+---+---#---+---+---#
| 1 |   | 7 |   | 2 |   | 6 | 4 | 3 |
#---+---+---#---+---+---#---+---+---#
| 8 |   |   | 3 |   | 1 | 2 | 9 |   |
#===========#===========#===========#
|   | 4 |   | 6 |   |   |   | 8 | 2 |
#---+---+---#---+---+---#---+---+---#
| 6 |   |   | 2 |   | 7 |   |   | 9 |
#---+---+---#---+---+---#---+---+---#
|   |   |   |   | 9 |   | 5 |   | 6 |
#===========#===========#===========#
|   |   | 6 |   | 7 | 2 |   |   |   |
#---+---+---#---+---+---#---+---+---#
|   | 7 | 2 | 5 |   | 4 |   |   |   |
#---+---+---#---+---+---#---+---+---#
|   | 1 |   |   | 3 |   |   |   |   |
#===========#===========#===========#
```

Y que, mediante el procedimiento de búsqueda de valores definido en la primera parte de la práctica se llega a la solución final del Sudoku, con los siguientes valores:

```
#===========#===========#===========#
| 2 | 6 | 3 | 7 | 4 | 9 | 8 | 1 | 5 |
#---+---+---#---+---+---#---+---+---#
| 1 | 9 | 7 | 8 | 2 | 5 | 6 | 4 | 3 |
#---+---+---#---+---+---#---+---+---#
| 8 | 5 | 4 | 3 | 6 | 1 | 2 | 9 | 7 |
#===========#===========#===========#
| 7 | 4 | 9 | 6 | 5 | 3 | 1 | 8 | 2 |
#---+---+---#---+---+---#---+---+---#
| 6 | 8 | 5 | 2 | 1 | 7 | 4 | 3 | 9 |
#---+---+---#---+---+---#---+---+---#
| 3 | 2 | 1 | 4 | 9 | 8 | 5 | 7 | 6 |
#===========#===========#===========#
| 4 | 3 | 6 | 1 | 7 | 2 | 9 | 5 | 8 |
#---+---+---#---+---+---#---+---+---#
| 9 | 7 | 2 | 5 | 8 | 4 | 3 | 6 | 1 |
#---+---+---#---+---+---#---+---+---#
| 5 | 1 | 8 | 9 | 3 | 6 | 7 | 2 | 4 |
#===========#===========#===========#
```

E6

```
char sudokuInput06[_CELDAS + 1] =
"000000900000007300054000007600000000300000002000480100020000084000096000000014095";
```

Que ofrece el siguiente Sudoku:

```
#===========#===========#===========#
|   |   |   |   |   |   | 9 |   |   |
#---+---+---#---+---+---#---+---+---#
|   |   |   |   |   | 7 | 3 |   |   |
#---+---+---#---+---+---#---+---+---#
|   | 5 | 4 |   |   |   |   |   | 7 |
#===========#===========#===========#
| 6 |   |   |   |   |   |   |   |   |
#---+---+---#---+---+---#---+---+---#
| 3 |   |   |   |   |   |   |   | 2 |
#---+---+---#---+---+---#---+---+---#
|   |   |   | 4 | 8 |   | 1 |   |   |
#===========#===========#===========#
|   | 2 |   |   |   |   |   | 8 | 4 |
#---+---+---#---+---+---#---+---+---#
|   |   |   |   | 9 | 6 |   |   |   |
#---+---+---#---+---+---#---+---+---#
|   |   |   |   | 1 | 4 |   | 9 | 5 |
#===========#===========#===========#
```

Y que, mediante el procedimiento de búsqueda de valores definido en la primera parte de la práctica no es posible insertar ningún nuevo valor. El cuadro se deberá rellenar, pues, enteramente por fuerza bruta, hasta llegar a la siguiente solución:

```
#===========#===========#===========#
| 2 | 3 | 7 | 1 | 4 | 8 | 9 | 5 | 6 |
#---+---+---#---+---+---#---+---+---#
| 8 | 9 | 6 | 5 | 2 | 7 | 3 | 4 | 1 |
#---+---+---#---+---+---#---+---+---#
| 1 | 5 | 4 | 6 | 3 | 9 | 8 | 2 | 7 |
#===========#===========#===========#
| 6 | 1 | 9 | 7 | 5 | 2 | 4 | 3 | 8 |
#---+---+---#---+---+---#---+---+---#
| 3 | 4 | 8 | 9 | 6 | 1 | 5 | 7 | 2 |
#---+---+---#---+---+---#---+---+---#
| 5 | 7 | 2 | 4 | 8 | 3 | 1 | 6 | 9 |
#===========#===========#===========#
| 9 | 2 | 1 | 3 | 7 | 5 | 6 | 8 | 4 |
#---+---+---#---+---+---#---+---+---#
| 4 | 8 | 5 | 2 | 9 | 6 | 7 | 1 | 3 |
#---+---+---#---+---+---#---+---+---#
| 7 | 6 | 3 | 8 | 1 | 4 | 2 | 9 | 5 |
#===========#===========#===========#
```

E7

```
char sudokuInput07[_CELDAS + 1] =
"900080002070100000005003000000008100000020806800600049400060090030005000001700060";
```

Que ofrece el siguiente Sudoku:

```
#===========#===========#===========#
| 9 |   |   |   | 8 |   |   |   | 2 |
#---+---+---#---+---+---#---+---+---#
|   | 7 |   | 1 |   |   |   |   |   |
#---+---+---#---+---+---#---+---+---#
|   |   | 5 |   |   | 3 |   |   |   |
#===========#===========#===========#
|   |   |   |   |   |   | 8 | 1 |   |
#---+---+---#---+---+---#---+---+---#
|   |   |   |   |   | 2 |   | 8 |   | 6 |
#---+---+---#---+---+---#---+---+---#
| 8 |   |   | 6 |   |   |   | 4 | 9 |
#===========#===========#===========#
| 4 |   |   |   | 6 |   |   | 9 |   |
#---+---+---#---+---+---#---+---+---#
|   | 3 |   |   |   | 5 |   |   |   |
#---+---+---#---+---+---#---+---+---#
|   |   | 1 | 7 |   |   |   | 6 |   |
#===========#===========#===========#
```

Y que, mediante el procedimiento de búsqueda de valores definido en la primera parte de la práctica no es posible insertar ningún nuevo valor. El cuadro se deberá rellenar, pues, enteramente por fuerza bruta, hasta llegar a la siguiente solución:

```
#===========#===========#===========#
| 9 | 4 | 3 | 5 | 8 | 6 | 7 | 1 | 2 |
#---+---+---#---+---+---#---+---+---#
| 6 | 7 | 8 | 1 | 4 | 2 | 9 | 3 | 5 |
#---+---+---#---+---+---#---+---+---#
| 1 | 2 | 5 | 9 | 7 | 3 | 6 | 8 | 4 |
#===========#===========#===========#
| 3 | 6 | 9 | 4 | 5 | 8 | 1 | 2 | 7 |
#---+---+---#---+---+---#---+---+---#
| 7 | 1 | 4 | 3 | 2 | 9 | 8 | 5 | 6 |
#---+---+---#---+---+---#---+---+---#
| 8 | 5 | 2 | 6 | 1 | 7 | 3 | 4 | 9 |
#===========#===========#===========#
| 4 | 8 | 7 | 2 | 6 | 1 | 5 | 9 | 3 |
#---+---+---#---+---+---#---+---+---#
| 2 | 3 | 6 | 8 | 9 | 5 | 4 | 7 | 1 |
#---+---+---#---+---+---#---+---+---#
| 5 | 9 | 1 | 7 | 3 | 4 | 2 | 6 | 8 |
#===========#===========#===========#
```

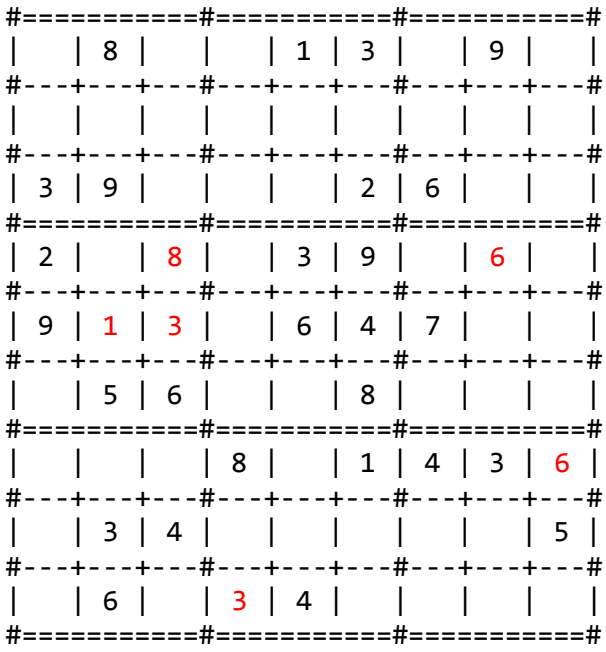
E8

```
char sudokuInput08[_CELDAS + 1] =
"0800130900000000000390002600200039000900064700056008000000801430034000005060040000";
```

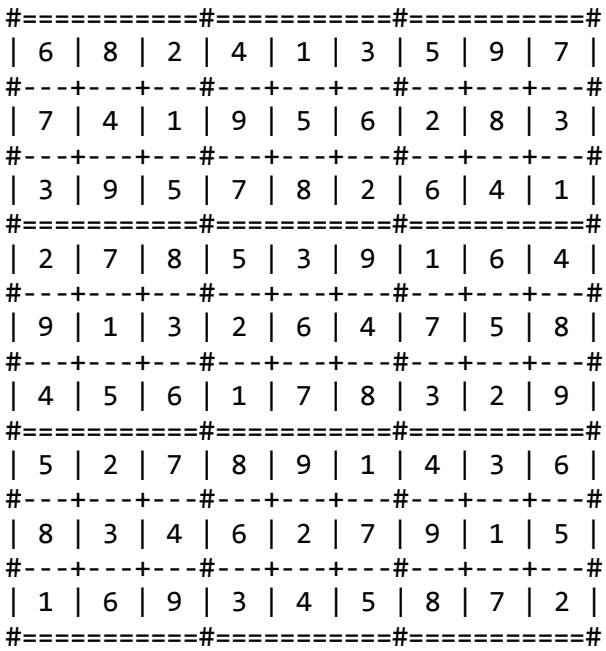
Que ofrece el siguiente Sudoku:

```
#===========#===========#===========#
|   | 8 |   |   | 1 | 3 |   | 9 |   |
#---+---+---#---+---+---#---+---+---#
|   |   |   |   |   |   |   |   |   |
#---+---+---#---+---+---#---+---+---#
| 3 | 9 |   |   |   | 2 | 6 |   |   |
#===========#===========#===========#
| 2 |   |   |   | 3 | 9 |   |   |   |
#---+---+---#---+---+---#---+---+---#
| 9 |   |   |   | 6 | 4 | 7 |   |   |
#---+---+---#---+---+---#---+---+---#
|   | 5 | 6 |   |   | 8 |   |   |   |
#===========#===========#===========#
|   |   |   |   | 8 |   | 1 | 4 | 3 |   |
#---+---+---#---+---+---#---+---+---#
|   | 3 | 4 |   |   |   |   |   | 5 |
#---+---+---#---+---+---#---+---+---#
|   | 6 |   |   | 4 |   |   |   |   |
#===========#===========#===========#
```

Y que, mediante el procedimiento de búsqueda de valores definido en la primera parte de la práctica se llega a los siguientes valores:



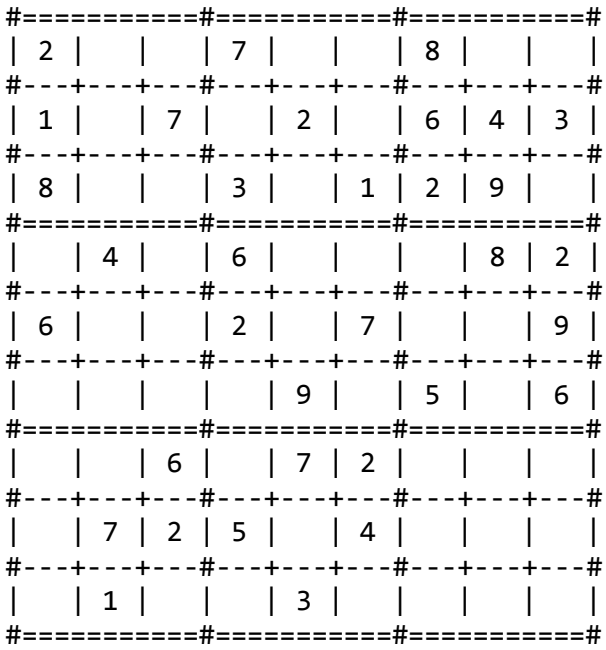
El resto del cuadro se deberá rellenar por fuerza bruta, hasta llegar a la siguiente solución:



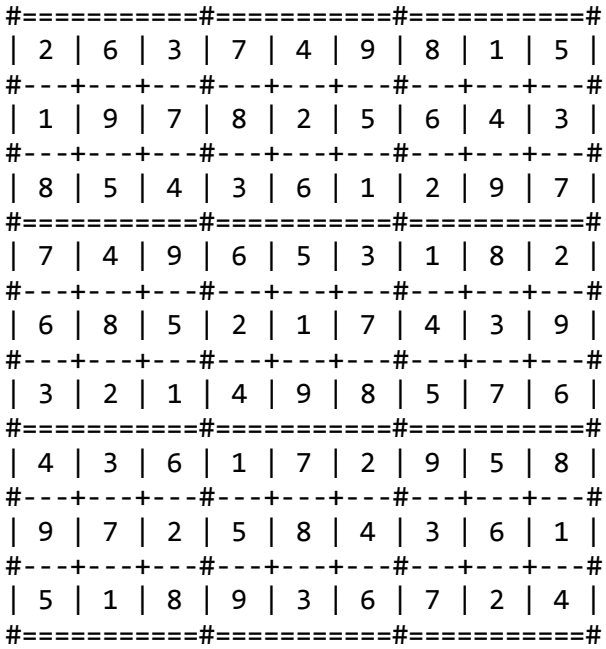
E9

```
char sudokuInput09[_CELDAS + 1] =
"200700800107020643800301290040600082600207009000090506006072000072504000010030000";
```

Que ofrece el siguiente Sudoku:



Y que, mediante el procedimiento de búsqueda de valores definido en la primera parte de la práctica se llega a la solución final del Sudoku, con los siguientes valores:

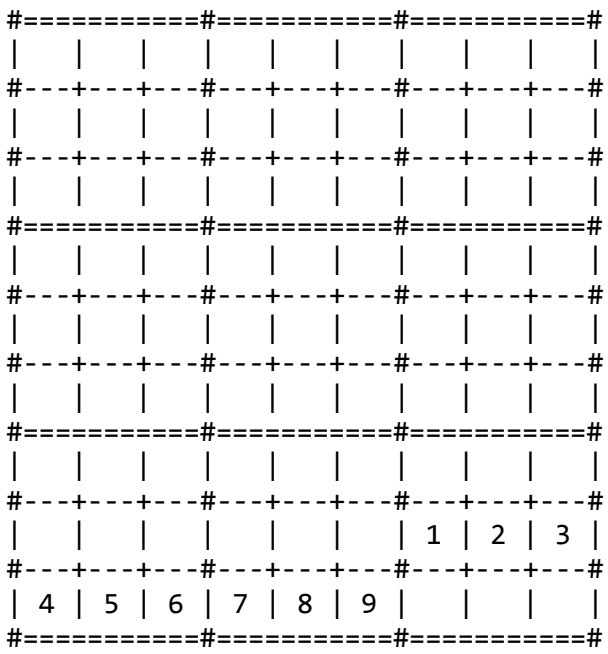


ANTI EJEMPLOS

X1

```
char sudokuInput10[_CELDAS + 1] =
"000000000000000000000000000000000000000000000000000000000000000000000000000000000123456789000";
```

Que ofrece el siguiente Sudoku:



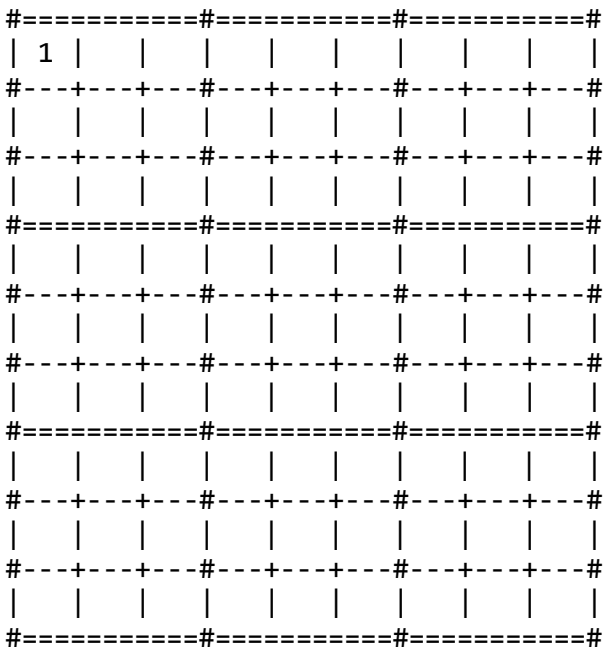
Éste es un sudoku inicialmente bien planteado porque no se halla ningún valor repetido en ninguna fila, columna o caja, pero que no tiene solución porque, por ejemplo, en la caja inferior derecha (la caja 9) en sus posiciones inferiores, no podemos insertar ningún valor sin romper las reglas de los sudokus.

Existen muchos sudokus bien planteados sin solución. La práctica optativa debería detectar esas incidencias.

X2

```
char sudokuInput10[_CELDAS + 1] =
"1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000";
```

Que ofrece el siguiente Sudoku:



Éste es un sudoku inicialmente bien planteado y que, desde luego, tiene infinidad de soluciones. Casi todos los sudokus de orden mayor de 64 podrán tener más de una solución. No existe un oren máximo a partir del cual todo sudoku tiene más de una solución. Pero no hacen muchas matemáticas para concluir que éste que ahora proponemos sí las tiene.

La práctica final optativa deberá ofrecer alguna de las múltiples posibles soluciones de este tipo de sudokus.