Inleiding

Het oplossen van Sudoku puzzels is een leuk tijdverdrijf. Als je niet weet wat een Sudoku puzzel is kijk dan op Wikipedia[1]. In dit document gaan we stap voor stap een programma schrijven om Sudoku puzzels op te lossen.

1 Datastructuur om de puzzel in op te slaan

Een Sudoku puzzel bestaat uit een matrix van 9 bij 9 hokjes. Een hokje is leeg of bevat een cijfer (0..9). Deze data kan in C het beste worden opgeslagen in een 2-dimensionale array van integers waarbij het getal 0 wordt gebruikt om een leeg hokje weer te geven. De Sudoku puzzel die weergegeven is in figuur 1 kan in een C programma worden opgeslagen in een variabele `puzzle1` zoals gegeven in listing 1.

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

Figuur 1: Een voorbeeld van een Sudoku puzzel.

```
1   int puzzle1[9][9] =
2   {
3       {8, 6, 0, 0, 2, 0, 0, 0, 0},
4       {0, 0, 0, 7, 0, 0, 0, 5, 9},
5       {0, 0, 0, 0, 0, 0, 0, 0, 0},
6       {0, 0, 0, 0, 6, 0, 8, 0, 0},
7       {0, 4, 0, 0, 0, 0, 0, 0, 0},
8       {0, 0, 5, 3, 0, 0, 0, 0, 7},
9       {0, 0, 0, 0, 0, 0, 0, 0, 0},
10      {0, 2, 0, 0, 0, 0, 6, 0, 0},
11      {0, 0, 7, 5, 0, 9, 0, 0, 0}
12  };
```

Listing 1: De puzzel uit figuur 1 opgeslagen in de variabele `puzzle1`.

2 Printen van de puzzel

Het is handig om een Sudoku puzzel die in een C programma is opgeslagen af te kunnen drukken.

[1] Zie: `http://nl.wikipedia.org/wiki/Sudoku`

Opdracht 1: `printPuzzle`

Schrijf een functie `printPuzzle` die een Sudoku puzzel, die is opgeslagen zoals in hoofdstuk 1 gegeven is, afdrukt op het beeldscherm.

3 Uitwerking van opdracht 1

Een mogelijke implementatie van de functie `printPuzzle` die een Sudoku puzzel afdrukt, is gegeven in listing 2. Als deze functie gebruikt wordt om de Sudoku puzzel uit listing 1 af te drukken, verschijnt de uitvoer die in figuur 2 gegeven is.

```
void printPuzzle(int m[][9])
{
    for (int r = 0; r < 9; r++)
    {
        if (r % 3 == 0)
        {
            printf(" ------- ------- -------\n");
        }
        for (int c = 0; c < 9; c++)
        {
            if (c % 3 == 0)
            {
                printf("| ");
            }
            int digit = m[r][c];
            if (digit == 0)
            {
                printf("  ");
            }
            else
            {
                printf("%d ", m[r][c]);
            }
        }
        printf("|\n");
    }
    printf(" ------- ------- -------\n");
}
```

Listing 2: Een functie die een Sudoku puzzel op het beeldscherm afdrukt, zie `printSudoku.c`.

4 Is de puzzel geldig?

Om de Sudoku puzzel op te kunnen lossen moeten we kunnen controleren of de puzzel geldig is.

```
 ------- ------- -------
| 8 6   |   2   |       |
|       | 7     |   5 9 |
|       |       |       |
 ------- ------- -------
|       |   6   | 8     |
|   4   |       |       |
|     5 | 3     |     7 |
 ------- ------- -------
|       |       |       |
|   2   |       | 6     |
|     7 | 5   9 |       |
 ------- ------- -------
```

Figuur 2: Uitvoer van de functieaanroep: `printPuzzle(puzzle1);`.

Een geldige puzzel moet voldoen aan de volgende 3 voorwaarden:

- In elke rij mag elk cijfer (1..9) slechts 1x voorkomen.
- In elke kolom mag elk cijfer (1..9) slechts 1x voorkomen.
- In elk blokje van 3 bij 3, aangegeven met de dikke lijnen in figuur 1 mag elk cijfer (1..9) slechts 1x voorkomen.

Opdracht 2: `areRowsValid`

Schrijf een functie `areRowsValid` die in een Sudoku puzzel, die is opgeslagen zoals in hoofdstuk 1 gegeven is, controleert of alle rijen geldig zijn. In elke rij mag elk cijfer (1..9) slechts 1x voorkomen.

Om te kunnen controleren of de functie `areRowsValid` correct werkt is een zogenoemde unittest geschreven, zie listing 3.

Het bedenken en implementeren van de unittest voordat de te testen code zelf geschreven wordt, wordt test-driven development genoemd, zie Wikipedia[2]

```
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  bool areRowsValid(int m[][9])
5  {
6      return true;
7  }
8
9  void printTestResult(int testNumber, bool testResult)
10 {
11     if (testResult)
12     {
```

[2] Zie: `http://nl.wikipedia.org/wiki/Test-driven_development`

```
        printf("Test %d is succesvol uitgevoerd.\n", testNumber);
    }
    else
    {
        printf("Test %d is NIET succesvol uitgevoerd!\n", ↩
         ↪ testNumber);
    }
}

int main(void)
{
    int puzzle1[9][9] =
    {
        {8, 6, 0, 0, 2, 0, 0, 0, 0},
        {0, 0, 0, 7, 0, 0, 0, 5, 9},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 6, 0, 8, 0, 0},
        {0, 4, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 5, 3, 0, 0, 0, 0, 7},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 2, 0, 0, 0, 0, 6, 0, 0},
        {0, 0, 7, 5, 0, 9, 0, 0, 0}
    };
    int puzzle2[9][9] =
    {
        {8, 6, 3, 9, 2, 5, 7, 4, 1},
        {4, 1, 2, 7, 8, 6, 3, 5, 9},
        {7, 5, 9, 4, 1, 3, 2, 8, 6},
        {9, 7, 1, 2, 6, 4, 8, 3, 5},
        {3, 4, 6, 8, 5, 7, 9, 1, 2},
        {2, 8, 5, 3, 9, 1, 4, 6, 7},
        {1, 9, 8, 6, 3, 2, 5, 7, 4},
        {5, 2, 4, 1, 7, 8, 6, 9, 3},
        {6, 3, 7, 5, 4, 9, 1, 2, 8}
    };
    int puzzle3[9][9] =
    {
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0}
    };
    int puzzle4[9][9] =
    {
```

```
61          {0, 0, 0, 0, 0, 0, 0, 0, 0},
62          {0, 0, 0, 0, 0, 0, 0, 0, 0},
63          {0, 0, 0, 0, 0, 0, 0, 0, 0},
64          {0, 0, 0, 0, 0, 0, 0, 0, 0},
65          {1, 0, 0, 0, 0, 0, 0, 0, 1},
66          {0, 0, 0, 0, 0, 0, 0, 0, 0},
67          {0, 0, 0, 0, 0, 0, 0, 0, 0},
68          {0, 0, 0, 0, 0, 0, 0, 0, 0},
69          {0, 0, 0, 0, 0, 0, 0, 0, 0}
70      };
71      int puzzle5[9][9] =
72      {
73          {0, 0, 0, 0, 0, 0, 0, 0, 0},
74          {0, 0, 0, 0, 0, 0, 0, 0, 0},
75          {0, 0, 0, 0, 0, 0, 0, 0, 0},
76          {0, 0, 0, 0, 0, 0, 0, 0, 0},
77          {9, 0, 0, 0, 0, 0, 0, 0, 9},
78          {0, 0, 0, 0, 0, 0, 0, 0, 0},
79          {0, 0, 0, 0, 0, 0, 0, 0, 0},
80          {0, 0, 0, 0, 0, 0, 0, 0, 0},
81          {0, 0, 0, 0, 0, 0, 0, 0, 0}
82      };
83      int puzzle6[9][9] =
84      {
85          {0, 0, 0, 0, 0, 0, 0, 0, 0},
86          {0, 0, 0, 0, 0, 0, 0, 0, 0},
87          {0, 0, 0, 0, 0, 0, 0, 0, 0},
88          {0, 0, 0, 0, 0, 0, 0, 0, 0},
89          {1, 2, 3, 4, 5, 6, 7, 8, 8},
90          {0, 0, 0, 0, 0, 0, 0, 0, 0},
91          {0, 0, 0, 0, 0, 0, 0, 0, 0},
92          {0, 0, 0, 0, 0, 0, 0, 0, 0},
93          {0, 0, 0, 0, 0, 0, 0, 0, 0}
94      };
95
96      printTestResult(1, areRowsValid(puzzle1) == true);
97      printTestResult(2, areRowsValid(puzzle2) == true);
98      printTestResult(3, areRowsValid(puzzle3) == true);
99      printTestResult(4, areRowsValid(puzzle4) == false);
100     printTestResult(5, areRowsValid(puzzle5) == false);
101     printTestResult(6, areRowsValid(puzzle6) == false);
102
103     return 0;
104 }
```

Listing 3: unitTestAreRowsValid.c.

Opdracht 3: `areColumnsValid`

Schrijf een functie `areColumnsValid` die in een Sudoku puzzel, die is opgeslagen zoals in hoofdstuk 1 gegeven is, controleert of alle kolommen geldig zijn. In elke kolom mag elk cijfer (1..9) slechts 1x voorkomen. Om te controleren of de functie correct werkt kan gebruik gemaakt worden van het programma `unitTestAreColumnsValid.c`.

Opdracht 4: `areBlocksValid`

Schrijf een functie `areBlocksValid` die in een Sudoku puzzel, die is opgeslagen zoals in hoofdstuk 1 gegeven is, controleert of alle blokjes van 3 bij 3 geldig zijn. In elk blokje van 3 bij 3, aangegeven met de dikke lijnen in figuur 1 mag elk cijfer (1..9) slechts 1x voorkomen. Om te controleren of de functie correct werkt kan gebruik gemaakt worden van het programma `unitTestAreBlocksValid.c`.

Opdracht 5: `isValid`

Schrijf een functie `isValid` die in een Sudoku puzzel, die is opgeslagen zoals in hoofdstuk 1 gegeven is, controleert of de puzzel geldig is. Maak daarbij gebruik van de al eerder geschreven functies `areRowsValid`, `areColumnsValid` en `areBlocksValid`. Schrijf voordat je de functie `isValid` implementeert eerst een unittest waarmee gecontroleerd kan worden of de functie correct werkt.

5 Uitwerking van opdracht 2

Een mogelijke implementatie van de functie `areRowsValid` is gegeven in listing 4.

6 Uitwerking van opdracht 3

Een mogelijke implementatie van de functie `areColumnsValid` kan gevonden worden in het programma `testAreColumnsValid`.

7 Uitwerking van opdracht 4

Een mogelijke implementatie van de functie `areBlocksValid` is gegeven in listing 5.

8 Uitwerking van opdracht 5

Een mogelijke implementatie van de functie `isValid` is gegeven in listing 6.

```
bool areRowsValid(int m[][9])
{
    for (int r = 0; r < 9; r++)
    {
        bool checklist[9] = {false};
        for (int c = 0; c < 9; c++)
        {
            int digit = m[r][c];
            if (digit != 0)
            {
                if (checklist[digit - 1])
                {
                    return false;
                }
                else
                {
                    checklist[digit - 1] = true;
                }
            }
        }
    }
    return true;
}
```

Listing 4: De functie `areRowsValid`, zie programma `testAreRowsValid.c`.

9 Oplossen van de puzzel

Om de puzzel op te lossen maken we gebruik van backtracking, zie Wikipedia[3]. Bij deze methode maken we een keuze bij het invullen van het eerste lege vakje. Als deze keuze niet tot een oplossing leidt dan moeten we teruggaan (vandaar de naam backtracking) naar het keuzemoment en een andere keuze maken. Omdat een Sudoku maar één mogelijke oplossing heeft leidt deze methode uiteindelijk tot een volledig opgeloste puzzel.

De functie, genaamd `solve`, om de puzzel op te lossen werkt als volgt:

- Zoek het eerste lege vakje.
- Vul een 1 in.
  - Controleer of de puzzel nog geldig is.
  - Als de puzzel nog geldig is, los dan de resterende puzzel op.
  - Als de puzzel geldig is en de resterende puzzel opgelost kan worden, return dan true (de puzzel is opgelost).
- (Hier kom je als het invullen van de 1 een ongeldige puzzel heeft opgeleverd of als de puzzel na het invullen van de 1 niet meer opgelost kan worden. 1 was dus de verkeerde keuze en we proberen iets anders:) Vul een 2 in.
  - Controleer of de puzzel nog geldig is.
  - Als de puzzel nog geldig is, los dan de resterende puzzel op.

[3] Zie: `https://nl.wikipedia.org/wiki/Backtracking`

```
bool areBlocksValid(int m[][9])
{
    for (int rb = 0; rb < 9; rb += 3)
    {
        for (int cb = 0; cb < 9; cb += 3)
        {
            bool checklist[9] = {false};
            for (int r = rb; r < rb + 3; r++)
            {
                for (int c = cb; c < cb + 3; c++)
                {
                    int digit = m[r][c];
                    if (digit != 0)
                    {
                        if (checklist[digit - 1])
                        {
                            return false;
                        }
                        else
                        {
                            checklist[digit - 1] = true;
                        }
                    }
                }
            }
        }
    }
    return true;
}
```

Listing 5: De functie areBlocksValid, zie programma testAreBlocksValid.c.

```
bool isValid(int m[][9])
{
    return areRowsValid(m) && areColumnsValid(m) && ↩
     ↪ areBlocksValid(m);
}
```

Listing 6: De functie isValid, zie programma testIsValid.c.

  - Als de puzzel geldig is en de resterende puzzel opgelost kan worden, return dan true (de puzzel is opgelost).

- (Hier kom je als het invullen van de 2 een ongeldige puzzel heeft opgeleverd of als de puzzel na het invullen van de 2 niet meer opgelost kan worden. 2 was dus, net zoals 1, de verkeerde keuze en we proberen iets anders:) Vul een 3 in.
  - Controleer of de puzzel nog geldig is.
  - Als de puzzel nog geldig is, los dan de resterende puzzel op.
  - Als de puzzel geldig is en de resterende puzzel opgelost kan worden, return dan true (de puzzel is opgelost).

- ...

- (Hier kom je als het invullen van de 8 een ongeldige puzzel heeft opgeleverd of als de puzzel na het invullen van de 8 niet meer opgelost kan worden. 8 was dus, net zoals 1 t/m 7, de verkeerde keuze en we proberen iets anders:) Vul een 9 in.
  - Controleer of de puzzel nog geldig is.
  - Als de puzzel nog geldig is, los dan de resterende puzzel op.
  - Als de puzzel geldig is en de resterende puzzel opgelost kan worden, return dan true (de puzzel is opgelost).
- We hebben alle getallen geprobeerd, maar de puzzel kan niet worden opgelost. De puzzel is dus geen Sudoku want een Sudoku moet één enkele oplossing hebben. Return false.

In het hierboven beschreven algoritme om de Sudoku puzzel op te lossen, moet nadat het eerste nog lege hokje is ingevuld (geprobeerd worden om) de rest van de puzzel op te lossen. Dit kunnen we doen door vanuit de functie `solve` die het algoritme implementeert de functie `solve` opnieuw aan te roepen. De functie `solve` bevat dan dus een aanroep naar zichzelf. Zo'n functie die zichzelf aanroept wordt een recursieve functie genoemd, zie eventueel Wikipedia[4]. We moeten er daarbij wel voor zorgen dat de functie bij elke aanroep een stapje dichter bij de oplossing komt omdat er anders een oneindige lus ontstaat. Bedenk dat de lokale variabelen van een functie bij elke aanroep van de betreffende functie opnieuw worden aangemaakt en dat deze variabelen blijven bestaan totdat de betreffende aanroep wordt beëindigd (met een `return`).

Opdracht 6: `solve`

Schrijf een functie `solve` die een Sudoku puzzel, die is opgeslagen zoals in hoofdstuk 1 gegeven is, kan oplossen. Maak gebruik van het hierboven beschreven recursieve backtracking algoritme. Maak daarbij gebruik van de al eerder geschreven functie `isValid`. Schrijf voordat je de functie `solve` implementeert eerst een unittest waarmee gecontroleerd kan worden of de functie correct werkt.

10 Uitwerking van opdracht 6

Een mogelijke implementatie van de functie `solve` is gegeven in listing 7.

Het programma `sudokuMetMeerOplossingen.c` is een (erg beperkte) unittest voor de functie `solve`. Het eerste deel van uitvoer van dit programma is gegeven in figuur 3. We zien dat de functie `solve` de puzzel die gegeven is in figuur 1 keurig oplost.

Uit het tweede deel van de uitvoer van het programma `sudokuMetMeerOplossingen.c` blijkt echter dat de functie `solve` te goed werkt, zie figuur 4. De functie lost namelijk ook puzzels op die meerder oplossingen hebben (bijvoorbeeld een puzzel met alleen maar lege vakjes) en dat is *niet* correct omdat een geldige Sudoku puzzel slechts 1 oplossing mag hebben. Als er meerdere oplossingen zijn moet het programma aangeven dat de Sudoku onoplosbaar is.

[4] Zie: `https://nl.wikipedia.org/wiki/Recursie_%28informatica%29`

```
bool solve(int m[][9])
{
    for (int r = 0; r < 9; r++)
    {
        for (int c = 0; c < 9; c++)
        {
            if (m[r][c] == 0)
            {
                for (int digit = 1; digit <= 9; digit++)
                {
                    m[r][c] = digit;
                    if (isValid(m) && solve(m))
                    {
                        return true;
                    }
                    m[r][c] = 0;
                }
                return false;
            }
        }
    }
    return isValid(m);
}
```

Listing 7: De functie solve, zie programma sudokuMetMeerOplossingen.c.

11 Controleren of er echt maar één oplossing is

Bij het zoeken naar de oplossing wordt in het eerste vrije hokje eerst een 1 ingevuld, als dit niet tot een oplossing leidt, wordt een 2 ingevuld. Als dit niet tot een oplossing leidt, wordt een 3 ingevuld, enz. Zie hoofdstuk 9 op pagina 7. Om te controleren of er maar één enkele oplossing is lossen we de puzzel nogmaals met een andere methode op. Nu wordt in het eerste vrije hokje eerst een 9 ingevuld, als dit niet tot een oplossing leidt, wordt een 8 ingevuld. Als dit niet tot een oplossing leidt, wordt een 7 ingevuld, enz. Deze methode wordt `solveHigh` genoemd omdat deze methode de puzzel met zo hoog mogelijke getallen probeert op te lossen. De originele methode, die we `solve` hadden genoemd, zie listing 7 wordt nu hernoemd tot `solveLow` omdat deze methode de puzzel met zo laag mogelijke getallen probeert op te lossen. Als beide methoden dezelfde oplossing vinden, dan betekent dit dat er maar één mogelijke oplossing is en dat de puzzel dus een Sudoku is. Als beide methoden een andere oplossing vinden, dan betekent dit dat er minstens twee oplossingen zijn en dat de puzzel geen (geldige) Sudoku is.

Opdracht 7: `solveHigh`

Schrijf een functie `solveHigh` die een Sudoku puzzel, die is opgeslagen zoals in hoofdstuk 1 gegeven is, kan oplossen. Maak gebruik van het hierboven beschreven recursieve backtracking algoritme en probeer de puzzel met zo hoog mogelijke getallen op te lossen. Maak daarbij gebruik van de al eerder geschreven functie

```
Puzzel 1:
 ------- ------- -------
| 8 6   |   2   |       |
|       | 7     |   5 9 |
|       |       |       |
 ------- ------- -------
|       |   6   | 8     |
|   4   |       |       |
|     5 | 3     |     7 |
 ------- ------- -------
|       |       |       |
|   2   |       | 6     |
|     7 | 5   9 |       |
 ------- ------- -------
Test 1 is succesvol uitgevoerd.
Oplossing puzzel 1:
 ------- ------- -------
| 8 6 3 | 9 2 5 | 7 4 1 |
| 4 1 2 | 7 8 6 | 3 5 9 |
| 7 5 9 | 4 1 3 | 2 8 6 |
 ------- ------- -------
| 9 7 1 | 2 6 4 | 8 3 5 |
| 3 4 6 | 8 5 7 | 9 1 2 |
| 2 8 5 | 3 9 1 | 4 6 7 |
 ------- ------- -------
| 1 9 8 | 6 3 2 | 5 7 4 |
| 5 2 4 | 1 7 8 | 6 9 3 |
| 6 3 7 | 5 4 9 | 1 2 8 |
 ------- ------- -------
```

Figuur 3: Eerste deel van de uitvoer van het programma `sudokuMetMeerOplossingen.c`.

`isValid`. Schrijf voordat je de functie `solveHigh` implementeert eerst een unittest waarmee gecontroleerd kan worden of de functie correct werkt.

Opdracht 8: `solve`

Hernoem de eerder geschreven functie `solve`, zie listing 7 tot `solveLow` Schrijf een nieuwe functie `solve` die een Sudoku puzzel, die is opgeslagen zoals in hoofdstuk 1 gegeven is, kan oplossen. Als de puzzel meerdere oplossingen heeft en de puzzel dus geen (geldige) Sudoku is, dan moet de functie de waarde `false` teruggeven. Maak gebruik van het hierboven beschreven functies `solveLow` en `solveHigh`. Schrijf voordat je de functie `solve` implementeert eerst een unittest waarmee gecontroleerd kan worden of de functie correct werkt.

12 Uitwerking van opdracht 7

Een mogelijke implementatie van de functie `solveHigh` is gegeven in listing 8.

```
Puzzel 2:
 ------- ------- -------
|       |       |       |
|       |       |       |
|       |       |       |
 ------- ------- -------
|       |       |       |
|       |       |       |
|       |       |       |
 ------- ------- -------
|       |       |       |
|       |       |       |
|       |       |       |
 ------- ------- -------
Test 2 is NIET succesvol uitgevoerd!
Oplossing puzzel 2:
 ------- ------- -------
| 1 2 3 | 4 5 6 | 7 8 9 |
| 4 5 6 | 7 8 9 | 1 2 3 |
| 7 8 9 | 1 2 3 | 4 5 6 |
 ------- ------- -------
| 2 1 4 | 3 6 5 | 8 9 7 |
| 3 6 5 | 8 9 7 | 2 1 4 |
| 8 9 7 | 2 1 4 | 3 6 5 |
 ------- ------- -------
| 5 3 1 | 6 4 2 | 9 7 8 |
| 6 4 2 | 9 7 8 | 5 3 1 |
| 9 7 8 | 5 3 1 | 6 4 2 |
 ------- ------- -------
```

Figuur 4: Tweede deel van de uitvoer van het programma `sudokuMetMeerOplossingen.c`.

Het programma `sudokuMetMeerOplossingen2.c` is een (erg beperkte) unittest voor de functie `solveHigh`. Het eerste deel van uitvoer van dit programma is gegeven in figuur 5. We zien dat de functie `solveHigh` de puzzel die gegeven is in figuur 1 keurig oplost.

Uit het tweede deel van de uitvoer van het programma `sudokuMetMeerOplossingen2.c`, zie figuur 6 blijkt dat de functie `solveHigh` een andere oplossing vindt dan de functie `solveLow`, zie figuur 4.

13 Uitwerking van opdracht 8

Een mogelijke implementatie van de functie `solve` is gegeven in listing 9.

De functie `solve` maakt gebruik van de functie `copy` waarmee een puzzel gekopieerd kan worden en de functie `isEqual` waarmee bepaald kan worden of twee puzzels gelijk zijn. Zie listing 10 respectievelijk listing 11.

Het programma `sudokuUniekeOplossing.c` is een (erg beperkte) unittest voor de functie `solve`. De uitvoer van dit programma toont aan dat de functie `solve` correct werkt.

```
bool solveHigh(int m[][9])
{
    for (int r = 0; r < 9; r++)
    {
        for (int c = 0; c < 9; c++)
        {
            if (m[r][c] == 0)
            {
                for (int digit = 9; digit >= 1; digit--)
                {
                    m[r][c] = digit;
                    if (isValid(m) && solveHigh(m))
                    {
                        return true;
                    }
                    m[r][c] = 0;
                }
                return false;
            }
        }
    }
    return isValid(m);
}
```

Listing 8: De functie solveHigh, zie programma sudokuMetMeerOplossingen2.c.

```
bool solve(int puzzle[][9])
{
    int result1[9][9], result2[9][9];
    copy(result1, puzzle);
    if (solveLow(result1))
    {
        copy(result2, puzzle);
        if (solveHigh(result2) && isEqual(result1, result2))
        {
            copy(puzzle, result1);
            return true;
        }
    }
    return false;
}
```

Listing 9: De functie solve, zie programma sudokuUniekeOplossing.c.

14 Inlezen van de puzzel uit een bestand

Als we een specifieke Sudoku op willen lossen met de tot nu toe ontwikkelde programma's dan moeten we de broncode (Engels: source code) van het programma aanpassen. Dit is niet handig. Het zou handiger zijn als de puzzel in een tekstbestand kan worden geplaatst en als het programma dat de puzzel oplost dit tekstbestand inleest.

```
Puzzel 1:
 ------- ------- -------
| 8 6   |   2   |       |
|       | 7     |   5 9 |
|       |       |       |
 ------- ------- -------
|       |   6   | 8     |
|   4   |       |       |
|     5 | 3     |     7 |
 ------- ------- -------
|       |       |       |
|   2   |       | 6     |
|     7 | 5   9 |       |
 ------- ------- -------
Test 1 is succesvol uitgevoerd.
Oplossing puzzel 1:
 ------- ------- -------
| 8 6 3 | 9 2 5 | 7 4 1 |
| 4 1 2 | 7 8 6 | 3 5 9 |
| 7 5 9 | 4 1 3 | 2 8 6 |
 ------- ------- -------
| 9 7 1 | 2 6 4 | 8 3 5 |
| 3 4 6 | 8 5 7 | 9 1 2 |
| 2 8 5 | 3 9 1 | 4 6 7 |
 ------- ------- -------
| 1 9 8 | 6 3 2 | 5 7 4 |
| 5 2 4 | 1 7 8 | 6 9 3 |
| 6 3 7 | 5 4 9 | 1 2 8 |
 ------- ------- -------
```

Figuur 5: Eerste deel van de uitvoer van het programma sudokuMetMeerOplossingen2.c.

```
1  void copy(int to[][9], int from[][9])
2  {
3      for (int r = 0; r < 9; r++)
4      {
5          for (int c = 0; c < 9; c++)
6          {
7              to[r][c] = from[r][c];
8          }
9      }
10 }
```

Listing 10: De functie copy, zie programma sudokuUniekeOplossing.c.

Bestanden kunnen eenvoudig ingelezen worden in C door gebruik te maken van een `FILE *`[5] en van de functies: `fopen`[6], `fscanf`[7] en `fclose`[8].

[5] Zie: http://en.cppreference.com/w/c/io

[6] Zie: http://en.cppreference.com/w/c/io/fopen

[7] Zie: http://en.cppreference.com/w/c/io/fscanf

[8] Zie: http://en.cppreference.com/w/c/io/fclose

```
Puzzel 2:
 ------- ------- -------
|       |       |       |
|       |       |       |
|       |       |       |
 ------- ------- -------
|       |       |       |
|       |       |       |
|       |       |       |
 ------- ------- -------
|       |       |       |
|       |       |       |
|       |       |       |
 ------- ------- -------
Test 2 is NIET succesvol uitgevoerd!
Oplossing puzzel 2:
 ------- ------- -------
| 9 8 7 | 6 5 4 | 3 2 1 |
| 6 5 4 | 3 2 1 | 9 8 7 |
| 3 2 1 | 9 8 7 | 6 5 4 |
 ------- ------- -------
| 8 9 6 | 7 4 5 | 2 1 3 |
| 7 4 5 | 2 1 3 | 8 9 6 |
| 2 1 3 | 8 9 6 | 7 4 5 |
 ------- ------- -------
| 5 7 9 | 4 6 8 | 1 3 2 |
| 4 6 8 | 1 3 2 | 5 7 9 |
| 1 3 2 | 5 7 9 | 4 6 8 |
 ------- ------- -------
```

Figuur 6: Tweede deel van de uitvoer van het programma sudokuMetMeerOplossingen2.c.

```
1  bool isEqual(int m1[][9], int m2[][9])
2  {
3      for (int r = 0; r < 9; r++)
4      {
5          for (int c = 0; c < 9; c++)
6          {
7              if (m1[r][c] != m2[r][c])
8              {
9                  return false;
10             }
11         }
12     }
13     return true;
14 }
```

Listing 11: De functie isEqual, zie programma sudokuUniekeOplossing.c.

Opdracht 9: readPuzzle

Schrijf een functie `readPuzzle` die een Sudoku puzzel, die is opgeslagen in een tekstbestand zoals figuur 7 kan inlezen in een datastructuur die in hoofdstuk 1 gegeven is. Het prototype van de functie is:
`bool readPuzzle(FILE* fp, int m[][9])`
Als het inlezen gelukt is, moet de functie de waarde `true` teruggeven en als het inlezen niet gelukt is, moet de functie de waarde `false` teruggeven. Schrijf voordat je de functie `readPuzzle` implementeert eerst een unittest waarmee gecontroleerd kan worden of de functie correct werkt.

```
8 6 0 0 2 0 0 0 0
0 0 0 7 0 0 0 5 9
0 0 0 0 0 0 0 0 0
0 0 0 0 6 0 8 0 0
0 4 0 0 0 0 0 0 0
0 0 5 3 0 0 0 0 7
0 0 0 0 0 0 0 0 0
0 2 0 0 0 0 6 0 0
0 0 7 5 0 9 0 0 0
```

Figuur 7: Inhoud van de tekstfile `sudoku_2.txt`.

15 Uitwerking van opdracht 9

Een mogelijke implementatie van de functie `readPuzzle` is gegeven in listing 12.

```
1  bool readPuzzle(FILE* fp, int m[][9])
2  {
3      for (int r = 0; r < 9; r++)
4      {
5          for (int c = 0; c < 9; c++)
6          {
7              if (fscanf(fp, "%d", &m[r][c]) != 1)
8              {
9                  return false;
10             }
11         }
12     }
13     return true;
14 }
```

Listing 12: De functie `readPuzzle`, zie programma `sudokuFileRead.c`.

16 Filenaam meegeven als command line argument

Het is soms handig als we de naam van het bestand waarin zich een Sudoku puzzel bevindt als command line argument kunnen meegeven zodat dit bestand kan worden ingelezen. We kunnen het programma dan in een command window als volgt starten:

`pogrammanaam sudoku.txt`
Meer informatie over command line arguments kun je vinden op cppreference.com[9].

Opdracht 10: `readSudoku.c`

Schrijf een programma `readSudoku.c` dat een Sudoku puzzel, die is opgeslagen in een tekstbestand zoals weergegeven in figuur 7 kan inlezen in een datastructuur die in hoofdstuk 1 gegeven is. De puzzel moet vervolgens op het scherm worden afgedrukt. De bestandsnaam moet als command line argument meegegeven kunnen worden. Als er geen filenaam als command line argument meegegeven wordt, dan moet het programma aan de gebruiker vragen om alsnog een filenaam in te voeren. Als de file niet bestaat of als de file niet geopend of gelezen kan worden dan dient een passende foutmelding te worden gegeven. Maak gebruik van de eerder geschreven functies `readpuzzle` en `printpuzzle`.

17 Uitwerking van opdracht 10

Een mogelijke implementatie van de functie `main` van het programma `readSudoku.c` is gegeven in listing 13.

```
1  int main(int argc, char *argv[])
2  {
3      if (argc > 2)
4      {
5          fprintf(stderr, "Gebruik: %s [<input.txt>]\n", argv[0]);
6          return 1;
7      }
8      char* inputFilename;
9      if (argc > 1)
10     {
11         inputFilename = argv[1];
12     }
13     else
14     {
15         char filename[200];
16         printf("Geef filenaam: ");
17         scanf("%199s", filename);
18         inputFilename = filename;
19     }
20     FILE* inFile = fopen(inputFilename, "r");
21     if (inFile == NULL)
22     {
23         fprintf(stderr, "Kan bestand %s niet openen voor ↩
           ↪ lezen!\n", inputFilename);
24         perror("Error");
25         return 2;
```

[9] Zie: `http://en.cppreference.com/w/c/language/main_function`

```
26      }
27      int puzzle[9][9];
28      if (!readPuzzle(inFile, puzzle))
29      {
30          fprintf(stderr, "Fout tijdens lezen van bestand %s!", ↩
            ↪ inputFilename);
31          perror("Error");
32          return 3;
33      }
34      else
35      {
36          printPuzzle(puzzle);
37      }
38      return 0;
39  }
```

Listing 13: De functie `main`, zie programma `readSudoku.c`.

18 Programma dat Sudoku's oplost

Het is nu nog maar een kleine stap om een programma te schrijven dat een willekeurige, in een bestand opgeslagen, Sudoku puzzel kan oplossen.

Opdracht 11: `solveSudoku`

Schrijf een programma `solveSudoku.c` dat een Sudoku puzzel, die is opgeslagen in een tekstbestand zoals weergegeven in figuur 7 kan inlezen in een datastructuur die in hoofdstuk 1 gegeven is. De puzzel moet vervolgens op het scherm worden afgedrukt. Tot slot moet de oplossing van de puzzel op het scherm worden afgedrukt. Als de puzzel niet opgelost kan worden (omdat het geen geldige Sudoku is), dan moet dit worden gemeld op het scherm. De bestandsnaam moet als command line argument meegegeven kunnen worden. Als er geen filenaam als command line argument meegegeven wordt, dan moet het programma aan de gebruiker vragen om alsnog een filenaam in te voeren. Als de file niet bestaat of als de file niet geopend of gelezen kan worden dan dient een passende foutmelding te worden gegeven. Maak gebruik van het eerder geschreven programma `readSudoku.c`, zie opdracht 10 en de bij opdracht 8 ontwikkelde functie `solve` en alle daarin gebruikte functies.

Opdracht 12: Executietijd meten

Meet hoelang het programma erover doet om de volgende puzzels op te lossen of om vast te stellen dat ze onoplosbaar zijn:

- `sudoku_1.txt`
- `sudoku_2.txt`
- `sudoku_leeg.txt`

- sudoku_met_meer_oplossingen.txt

In de PowerShell van Windows kun je de executietijd, die nodig is om sudoku_1.txt op te lossen, meten met het commando:

```
(Measure-Command{.\solveSudoku .\sudoku_1.txt | ↩
 ↪ Out-Default}).TotalSeconds}
```

19 Uitwerking van opdracht 11

De functie `main` van het programma `solveSudoku.c` is grotendeels gelijk aan die van het programma `readSudoku.c`, zie listing 13. De laatste `else` in dit programma moet vervangen worden door de `else` die in listing 14 gegeven is.

```
1     else
2     {
3         printPuzzle(puzzle);
4         printf("Oplossing:\n");
5         if (solve(puzzle))
6         {
7             printPuzzle(puzzle);
8         }
9         else
10        {
11            printf("Is niet mogelijk!\n");
12        }
13    }
```

Listing 14: De laatste `else` in de functie `main` van het programma `solveSudoku.c`.

20 Uitwerking van opdracht 12

De executietijden van het programma `solveSudoku.c` die nodig zijn om de verschillende puzzels op te lossen zijn gegeven in tabel 1.

Tabel 1: Executietijden van het programma `solveSudoku.c`.

Puzzel	Executietijd (s)
sudoku_1.txt	0,07
sudoku_2.txt	20,88
sudoku_leeg.txt	0,04
sudoku_met_meer_oplossingen.txt	18,91

21 Programma dat Sudoku's sneller oplost

We zien in tabel 1 dat het programma `solveSudoku.c` sommige puzzels zeer snel oplost. Bij andere puzzels duurt het oplossen vervelend lang. Dit roept de vraag op: kunnen we het programma aanpassen zodat de puzzels sneller opgelost worden.

Een eenvoudige manier om het programma te versnellen is het gebruik van de optimizer van de compiler. In Code::Blocks kunnen we dit doen door in het menu Settings » Compiler... de optie Optimize fully (for speed) [`-O3`] aan te vinken.

Opdracht 13: Executietijd meten van het geoptimaliseerde programma

Meet hoelang het programma erover doet om de puzzels, die in opdracht 12 gegeven zijn, op te lossen of om vast te stellen dat ze onoplosbaar zijn als het programma met de optie `-O3` gecompileerd is.

22 Uitwerking van opdracht 13

De executietijden van het programma `solveSudoku.c` die nodig zijn om de verschillende puzzels op te lossen zijn gegeven in tabel 2.

Tabel 2: Executietijden van het op snelheid geoptimaliseerde programma `solveSudoku.c`.

Puzzel	Executietijd (s)
`sudoku_1.txt`	0,06
`sudoku_2.txt`	3,63
`sudoku_leeg.txt`	0,04
`sudoku_met_meer_oplossingen.txt`	3,32

23 Programma dat Sudoku's nog sneller oplost

We zien in tabel 2 dat het optimaliseren van het programma `solveSudoku.c` er voor heeft gezorgd dat alle puzzels vrij snel worden opgelost. Toch stellen we de vraag: kunnen we het programma aanpassen zodat de puzzels nog sneller opgelost worden.

Als we naar de werking kijken van `solveHigh`, zie listing 8 en `solveLow`, dan zien we dat na elk getal dat geprobeerd wordt de functie `isValid` aangeroepen wordt. Deze functie `isValid`, zie listing 6 roept op zijn beurt de functies `areRowsValid`, `areColumnsValid` en `areBlocksValid` aan, zie listings 4 en 5. Deze functies controleren respectievelijk alle rijen, alle kolommen en alle blokken. Dat is echter helemaal niet nodig. We hebben namelijk geprobeerd om een cijfer in één bepaald hokje (met een bepaald rij- en kolomnummer) in te vullen. Er hoeft dus maar één rij, één kolom en één blok gecontroleerd te worden!

Opdracht 14: `solveSudokuFast`

Schrijf een programma `solveSudokuFast.c` door het programma `solveSudoku.c` zodanig aan te passen dat na het invullen van een hokje slechts één rij, één kolom en één blok gecontroleerd wordt.

Opdracht 15: Executietijd meten van het verbeterde programma

Meet hoelang het programma `solveSudokuFast.c` erover doet om de puzzels, die in opdracht 12 gegeven zijn, op te lossen of om vast te stellen dat ze onoplosbaar zijn als het programma met de optie `-O3` gecompileerd is.

24 Uitwerking van opdracht 14

Een mogelijke implementatie van het programma `solveSudokuFast.c` is gegeven in listing 15.

```
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  bool isRowValid(int m[][9], int r)
5  {
6      bool checklist[9] = {false};
7      for (int c = 0; c < 9; c++)
8      {
9          int digit = m[r][c];
10         if (digit != 0)
11         {
12             if (checklist[digit - 1])
13             {
14                 return false;
15             }
16             else
17             {
18                 checklist[digit - 1] = true;
19             }
20         }
21     }
22     return true;
23 }
24
25 bool areRowsValid(int m[][9])
26 {
27     for (int r = 0; r < 9; r++)
28     {
29         if (!isRowValid(m, r))
```

```
30          {
31              return false;
32          }
33      }
34      return true;
35  }
36
37  bool isColumnValid(int m[][9], int c)
38  {
39      bool checklist[9] = {false};
40      for (int r = 0; r < 9; r++)
41      {
42          int digit = m[r][c];
43          if (digit != 0)
44          {
45              if (checklist[digit - 1])
46              {
47                  return false;
48              }
49              else
50              {
51                  checklist[digit - 1] = true;
52              }
53          }
54      }
55      return true;
56  }
57
58  bool areColumnsValid(int m[][9])
59  {
60      for (int c = 0; c < 9; c++)
61      {
62          if (!isColumnValid(m, c))
63          {
64              return false;
65          }
66      }
67      return true;
68  }
69
70  bool isBlockValid(int m[][9], int rb, int cb)
71  {
72      bool checklist[9] = {false};
73      for (int r = rb; r < rb + 3; r++)
74      {
75          for (int c = cb; c < cb + 3; c++)
76          {
77              int digit = m[r][c];
78              if (digit != 0)
```

```
79              {
80                  if (checklist[digit - 1])
81                  {
82                      return false;
83                  }
84                  else
85                  {
86                      checklist[digit - 1] = true;
87                  }
88              }
89          }
90      }
91      return true;
92  }
93
94  bool areBlocksValid(int m[][9])
95  {
96      for (int rb = 0; rb < 9; rb += 3)
97      {
98          for (int cb = 0; cb < 9; cb += 3)
99          {
100             if (!isBlockValid(m, rb, cb))
101             {
102                 return false;
103             }
104         }
105     }
106     return true;
107 }
108
109 bool isValidMove(int m[][9], int r, int c)
110 {
111     return isRowValid(m, r) && isColumnValid(m, c) && ↩
        ↪ isBlockValid(m, r/3 * 3, c/3 * 3);
112 }
113
114 bool isValid(int m[][9])
115 {
116     return areRowsValid(m) && areColumnsValid(m) && ↩
        ↪ areBlocksValid(m);
117 }
118
119 bool solveLow(int m[][9])
120 {
121     for (int r = 0; r < 9; r++)
122     {
123         for (int c = 0; c < 9; c++)
124         {
125             if (m[r][c] == 0)
```

```
126             {
127                 for (int digit = 1; digit <= 9; digit++)
128                 {
129                     m[r][c] = digit;
130                     if (isValidMove(m, r, c) && solveLow(m))
131                     {
132                         return true;
133                     }
134                     m[r][c] = 0;
135                 }
136                 return false;
137             }
138         }
139     }
140     return isValid(m);
141 }
142 
143 bool solveHigh(int m[][9])
144 {
145     for (int r = 0; r < 9; r++)
146     {
147         for (int c = 0; c < 9; c++)
148         {
149             if (m[r][c] == 0)
150             {
151                 for (int digit = 9; digit >= 1; digit--)
152                 {
153                     m[r][c] = digit;
154                     if (isValidMove(m, r, c) && solveHigh(m))
155                     {
156                         return true;
157                     }
158                     m[r][c] = 0;
159                 }
160                 return false;
161             }
162         }
163     }
164     return isValid(m);
165 }
166 
167 void copy(int to[][9], int from[][9])
168 {
169     for (int r = 0; r < 9; r++)
170     {
171         for (int c = 0; c < 9; c++)
172         {
173             to[r][c] = from[r][c];
174         }
```

```
175     }
176 }
177
178 bool isEqual(int m1[][9], int m2[][9])
179 {
180     for (int r = 0; r < 9; r++)
181     {
182         for (int c = 0; c < 9; c++)
183         {
184             if (m1[r][c] != m2[r][c])
185             {
186                 return false;
187             }
188         }
189     }
190     return true;
191 }
192
193 bool solve(int puzzle[][9])
194 {
195     int result1[9][9], result2[9][9];
196     copy(result1, puzzle);
197     if (solveLow(result1))
198     {
199         copy(result2, puzzle);
200         if (solveHigh(result2) && isEqual(result1, result2))
201         {
202             copy(puzzle, result1);
203             return true;
204         }
205     }
206     return false;
207 }
208
209 bool readPuzzle(FILE* fp, int m[][9])
210 {
211     for (int r = 0; r < 9; r++)
212     {
213         for (int c = 0; c < 9; c++)
214         {
215             if (fscanf(fp, "%d", &m[r][c]) != 1)
216             {
217                 return false;
218             }
219         }
220     }
221     return true;
222 }
223
```

```
224 void printPuzzle(int m[][9])
225 {
226     for (int r = 0; r < 9; r++)
227     {
228         if (r % 3 == 0)
229         {
230             printf(" ------- ------- -------\n");
231         }
232         for (int c = 0; c < 9; c++)
233         {
234             if (c % 3 == 0)
235             {
236                 printf("| ");
237             }
238             int digit = m[r][c];
239             if (digit == 0)
240             {
241                 printf("  ");
242             }
243             else
244             {
245                 printf("%d ", m[r][c]);
246             }
247         }
248         printf("|\n");
249     }
250     printf(" ------- ------- -------\n");
251 }
252
253 int main(int argc, char *argv[])
254 {
255     if (argc > 2)
256     {
257         fprintf(stderr, "Gebruik: %s [<input.txt>]\n", argv[0]);
258         return 1;
259     }
260     char* inputFilename;
261     if (argc > 1)
262     {
263         inputFilename = argv[1];
264     }
265     else
266     {
267         char filename[200];
268         printf("Geef filenaam: ");
269         scanf("%199s", filename);
270         inputFilename = filename;
271     }
272     FILE* inFile = fopen(inputFilename, "r");
```

```
273     if (inFile == NULL)
274     {
275         fprintf(stderr, "Kan bestand %s niet openen voor ↩
            ↪ lezen!\n", inputFilename);
276         perror("Error");
277         return 2;
278     }
279     int puzzle[9][9];
280     if (!readPuzzle(inFile, puzzle))
281     {
282         fprintf(stderr, "Fout tijdens lezen van bestand %s!", ↩
            ↪ inputFilename);
283         perror("Error");
284         return 3;
285     }
286     else
287     {
288         printPuzzle(puzzle);
289         printf("Oplossing:\n");
290         if (solve(puzzle))
291         {
292             printPuzzle(puzzle);
293         }
294         else
295         {
296             printf("Is niet mogelijk!\n");
297         }
298     }
299     return 0;
300 }
```

Listing 15: solveSudokuFast.c.

25 Uitwerking van opdracht 15

De executietijden van het programma solveSudokuFast.c die nodig zijn om de verschillende puzzels op te lossen zijn gegeven in tabel 3.

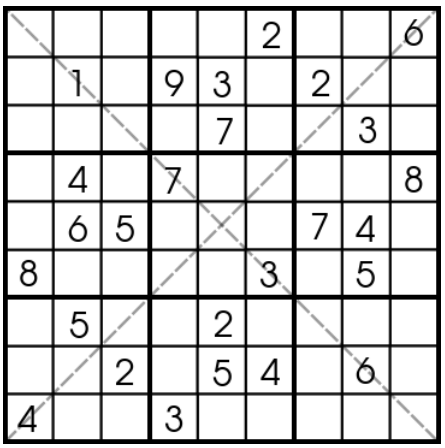
Tabel 3: Executietijden van het op snelheid geoptimaliseerde programma solveSudokuFast.c.

Puzzel	Executietijd (s)
sudoku_1.txt	0,06
sudoku_2.txt	0,73
sudoku_leeg.txt	0,04
sudoku_met_meer_oplossingen.txt	0,65

We zien dat nu alle puzzels binnen 1 seconde opgelost worden!

26 Variatie op de standaard Sudoku

We willen nu het programma `solveSudoku.c` dat we bij opdracht 11 hebben ontwikkeld, aanpassen zodat het zogenoemde "diagonaal-Sudoku" puzzels op kan lossen. Om een diagonaal-Sudoku op te lossen moet je voldoen aan dezelfde regels als bij een gewone Sudoku met als extra dat ook op de 2 diagonalen de cijfers 1 t/m 9 slechts één maal mogen voorkomen. In de figuur 8 zijn de diagonalen met een gestreepte lijn weergegeven.



Figuur 8: Een diagonaal-Sudoku, zie `sudoku_diagonal.txt`.

De in listing 6 gegeven functie `isValid` moet dus uitgebreid worden zoals in listing 16 is gegeven.

```
bool isValid(int m[][9])
{
    return areRowsValid(m) && areColumnsValid(m) && ↩
     ↪ areBlocksValid(m) && areDiagonalsValid(m);
}
```

Listing 16: De aangepaste functie `isValid`.

Opdracht 16: `areDiagonalsValid`

Schrijf een functie `areDiagonalsValid` die in een diagonaal-Sudoku puzzel, die is opgeslagen zoals in hoofdstuk 1 gegeven is, controleert of de diagonalen geldig zijn. Schrijf voordat je de functie `areDiagonalsValid` implementeert eerst een unittest waarmee gecontroleerd kan worden of de functie correct werkt. Voeg de functie daarna toe aan `solveSudoku.c` en pas de functie `isValid` aan zoals gegeven in listing 16.

27 Uitwerking van opdracht 16

Een mogelijke implementatie van de functie `areDiagonalsValid` is gegeven in listing 17. Deze functie is in het programma `solveDiagonalSudoku.c` opgenomen zodat je met dit programma diagonaal-Sudoku puzzels kunt oplossen.

```
bool areDiagonalsValid(int m[][9])
{
    bool checklist1[9] = {false}, checklist2[9] = {false};
    for(int rc = 0; rc < 9; rc++)
    {
        int digit1 = m[rc][rc];
        int digit2 = m[rc][8 - rc];
        if (digit1 != 0)
        {
            if (checklist1[digit1 - 1] == false)
            {
                checklist1[digit1 - 1] = true;
            }
            else
            {
                return false;
            }
        }
        if (digit2 != 0)
        {
            if (checklist2[digit2 - 1] == false)
            {
                checklist2[digit2 - 1] = true;
            }
            else
            {
                return false;
            }
        }
    }
    return true;
}
```

Listing 17: De functie `areDiagonalsValid`, zie programma `testAreDiagonalsValid.c`.

28 Diagonaal-Sudoku puzzels sneller oplossen

In het programma `solveDiagonalSudoku.c` wordt de functie `areDiagonalsValid` aangeroepen vanuit de functie `isValid`, zie listing 9. De functie `isValid` wordt vanuit de functies `solveHigh` (zie listing 8 en `solveLow` aangeroepen, telkens als een nieuwe waarde in een hokje ingevuld is. Het is echter alleen maar nodig om te bepalen of een diagonaal nog geldig is als een hokje op die diagonaal een nieuwe waarde heeft gekregen.

Opdracht 17: `solveDiagonalSudokuFast`

Schrijf een programma `diagonaalsgewijzet.c` door het programma `solveSudokuFast.c` zodanig aan te passen dat na het invullen van een hokje de diagonaal of diagonalen waar het hokje zich op bevindt gecontroleerd wordt.

29 Uitwerking van opdracht 16

De aanpassingen zijn gegeven in listing 18.

```
bool isDiagonal1Valid(int m[][9])
{
    bool checklist[9] = {false};
    for(int rc = 0; rc < 9; rc++)
    {
        int digit = m[rc][rc];
        if (digit != 0)
        {
            if (checklist[digit - 1] == false)
            {
                checklist[digit - 1] = true;
            }
            else
            {
                return false;
            }
        }
    }
    return true;
}

bool isDiagonal2Valid(int m[][9])
{
    bool checklist[9] = {false};
    for(int rc = 0; rc < 9; rc++)
    {
        int digit = m[rc][8 - rc];
        if (digit != 0)
        {
            if (checklist[digit - 1] == false)
            {
                checklist[digit - 1] = true;
            }
            else
            {
                return false;
            }
        }
    }
    return true;
}

bool areDiagonalsValid(int m[][9])
{
    return isDiagonal1Valid(m) && isDiagonal2Valid(m);
```

```
46 }
47
48 bool isValidMove(int m[][9], int r, int c)
49 {
50     return isRowValid(m, r) && isColumnValid(m, c) && ↩
        ↪ isBlockValid(m, r/3 * 3, c/3 * 3) && (r != c || ↩
        ↪ isDiagonal1Valid(m)) && (r != 8 - c || isDiagonal2Valid(m));
51 }
52
53 bool isValid(int m[][9])
54 {
55     return areRowsValid(m) && areColumnsValid(m) && ↩
        ↪ areBlocksValid(m) && areDiagonalsValid(m);
56 }
```

Listing 18: De functies `isDiagonal1Valid`, `isDiagonal2Valid`, `areDiagonalsValid`, `isValidMove` en `isValid` uit het programma `solveDiagonalSudokuFast.c`.

30 Een nieuwe uitdaging

Nu we een programma hebben geschreven dat Sudoku puzzels op kan lossen is het tijd voor de volgende uitdaging.

Opdracht 18: Binairo

Schrijf een programma waarmee Binairo puzzels opgelost kunnen worden. Als je niet weet wat een Binairo puzzel is, kijk dan op Wikipedia[a].

[a] Zie: `https://nl.wikipedia.org/wiki/Takuzu`