FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

Algorithmen & Datenstrukturen ILV
Backtracking

Abgabe: Details zu den Abgabemodalitäten finden Sie auf Moodle. Voraussetzung für die Bewertung ist, dass das Programm kompiliert und ausführbar ist. Die Lösungen werden jeweils am Anfang der nächsten (online) Übungseinheit von zufällig gewählten Studierenden vorgestellt.

Ziele:

- Backtracking
- Rekursion

Hinweise zum Autograding

Die Übungsaufgaben werden in ***sudoku.cpp*** gelöst und in *main.cpp* getestet. Alle anderen Dateien bleiben unberührt! Zeile 2 in *main.cpp* - #define TEST 1 - bestimmt ob Tests ausgeführt werden, oder nicht. 0 bedeutet, dass nicht getestet wird. Für die Abgabe muss die Konstante immer den Wert 1 aufweisen.

In der *main* Funktion dürfen **keine eigenen return statements** gesetzt werden (K.O. Kriterium). Sonst werden keine Testfälle ausgeführt.

Entfernen Sie vor der automatischen Bewertung **sämtliche Eingabeaufforderungen (scanf, gets,...)** von der Konsole. Diese führen zu einem Timeout und verhindern das Autograding auf Github Classroom (K.O. Kriterium).

Das Autograding dient lediglich als **Unterstützung zur Bewertung**. Es ist notwendig Algorithmen in geeigneter (oft sehr spezieller) Weise und wie angegeben umzusetzen. Daher wird in einigen Fällen eine manuelle Überprüfung durch die Lektoren vorgenommen, wodurch die Punkte auf Moodle von denen auf Github Classroom abweichen können!

Vorbereitung

Sudoku spielen...

Noch nie Sudoku gerätselt? Lösen Sie zunächst folgendes Sudoku und machen Sie sich mit dem Regelwerk im Internet vertraut.

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

FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

Übungsaufgabe

Backtracking
(15 Classroom Punkte)

Schreiben Sie einen rekursiven Backtracking Algorithmus, der für ein beliebiges Sudoku Rätsel eine gültige Lösung zurückliefert.

Gegeben ist ein Sudoku-Grundgerüst (*sudoku.cpp*) inklusive Header-Datei (*sudoku.hpp*). Das Sudoku Grundgerüst besitzt folgende Variablen und Funktionen:

```
int field[SIZE][SIZE];
```
Sudoku Spielfeld.
```
int initial[SIZE][SIZE];
```
Kopie des Spielfeldes mit den Initialwerten.

```
void print();
```
Gibt das Sudoku Spielfeld auf der Konsole formatiert aus.

```
int checkValueInField(int value, int row, int col);
```
Überprüft ob eine übergebene Zahl an die entsprechende Position gesetzt werden darf. Liefert "true" zurück, wenn die Zahl gültig ist.

```
int setValueInField(int value, int row, int col);
```
Setzt einen Wert an die entsprechende Position im Sudoku oder liefert "false" zurück, wenn der Wert nicht gesetzt werden kann.

```
int removeValueFromField(int row, int col);
```
Löscht einen Wert aus dem Sudokuspielfeld.

```
int getValueFromField(int row, int col);
```
Holt einen Wert aus dem Spielfeld.

```
int solve(int row, int col);
```
Implementiert die Lösung für das Sudoku mit gültigen Werten. Der Algorithmus soll an der Stelle [0][0] im Array nach einer Lösung zu suchen beginnen.

Testen Sie ihre Implementierung in *main()* indem Sie folgende Schritte durchführen:

1. Deklarieren und initialisieren Sie ein zweidimensionales Array mit den Startwerten des Sudoku. Alle übrigen Werte im Array werden mit 0 initialisiert.
2. Rufen Sie die Funktion *init* mit dem gerade erzeugten Array als Parameter auf.
3. Rufen Sie *solve* auf.
4. Geben Sie das Sudoku Spielfeld auf der Konsole aus.

Hinweis: Als Hilfestellung sei auf das 8 Damen Problem aus dem Vorlesungsteil verwiesen.