2. Übungsblatt - C++

Henrik Gerdes, Manuel Eversmeyer

4. November 2018

```
#define N 9

/*
    Functions to provide a solution for a Soduku
    Implementation in <sodukulib.so>
*/
void print_sudoku( int sudoku[N][N]);
int solve_sodoku(int i, int j, int sudoku[N][N]);
int is_valid(int z, int i, int j, int sudoku[N][N]);
int dumb_sodoku(int sudoku[N][N]);
int is_end(int sudoku[N][N]);
```

Aufgaben/Blatt02/Final/SodukuFunctions.h

```
#include <stdio.h>
#include "SodukuFunctions.h"

/*
    @author: Henrik Gerdes, Manuel Eversmeyer

    Main for a small programm that solves a Soduku using backtracking
    The Sokoku bust be hardcoded.

    Implementation of the actual code is in a shared libary called
    <sodukulib.so>

    Command to seth path to libsodukulib.so:
    export LD_LIBRARY_PATH=/path/to/library:${LD_LIBRARY_PATH}
*/

int main(int argv, char** argc)
{

    int sudoku[N][N] = {{0,0,0,0,0,8,0,3,0},
                        {0,3,0,5,0,0,4,7,1},
                        {2,0,0,1,0,0,6,9,0},
                        {5,0,0,0,0,2,1,0,0},
                        {1,2,4,0,0,0,9,6,3},
                        {0,0,6,4,0,0,0,0,2},
                        {0,8,9,0,0,5,0,0,7},
                        {3,5,2,0,0,9,0,4,0},
                        {0,1,0,3,0,0,0,0,0}};

    printf("Eingegebenes Sodoku:\n");
    print_sudoku(sudoku);
```

```
32
33      if(solve_sodoku(0,0,sudoku))
34      {
35          printf("\nGeloestes Soduku:\n");
36          print_sudoku(sudoku);
37      }else
38      {
39          printf("\nSodoku konnte nicht geloest werden\n");
40      }
41
42      return 0;
43  }
```

Aufgaben/Blatt02/Final/SodukuSolver.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define N 9
5
6  /*
7      @author: Henrik Gerdes, Manuel Eversmeyer
8
9      Actual functions to print and solve a Sodoku
10 */
11
12
13 /*
14     Prints the Soduku-grid
15 */
16 void print_sudoku( int sudoku[N][N])
17 {
18     int i,j;
19
20     printf("\n+-------+-------+-------+\n");
21     for(i = 0; i<N;i++)
22     {
23         for(j=0;j<N;j++)
24         {
25             if(j%3 ==0)
26             {
27                 printf("|");
28             };
29             printf("%2d ", sudoku[i][j]);
30         }
31         if(i%3==2)
32         {
33             printf("|\n+-------+-------+-------+\n");
34         }else{
35             printf("|\n");
36         }
37     }
38 }
39
40 /*
41     Checks if a given number z is valid to set on position i,j in
42     a Soduku-grid
43 */
```

```c
int is_valid(int z, int i, int j, int sudoku[N][N])
{
    if(z>9 || z<1)
    {
        return 0;
    }

    int k;
    for(k=0; k<N;k++)
    {
        if(sudoku[i][k] == z)
        {
            //printf("Fehler in Zeile gefunden");
            return 0;
        }
        if(sudoku[k][j] == z)
        {
            //printf("Fehler in Spalte gefunden");
            return 0;
        }
    }


    int square_i = i/3;
    int square_j = j/3;
    square_i *=3;
    square_j *=3;

    int limit_i = square_i +3;
    int limit_j = square_j +3;

    int index_i = 0;
    int index_j = 0;

    for(index_i = square_i;index_i < limit_i;index_i++)
    {
        for(index_j = square_j;index_j < limit_j;index_j++)
        {
            if(sudoku[index_i][index_j] == z)
            {
                //printf("Fehler in Feld gefunden");
                return 0;
            }
        }
    }


    return 1;
}

/*
    Checks if there are still free positions in the grid
*/
int is_end(int sudoku[N][N])
{
    int i = 0;
    int j = 0;
    for(i = 0; i<N;i++)
```

```
102     {
103         for(j=0;j<N;j++)
104         {
105             if(sudoku[i][j]==0)
106             {
107                 return 1;
108             }
109         }
110     }
111     return 0;
112 }
113 
114 /*
115     Solves a Soduku using backtracking
116 */
117 int solve_sodoku(int i, int j, int sudoku[N][N])
118 {
119     //printf("Solve mit: i=%d   und j=%d\n", i ,j);
120     if(j==N)
121     {
122         i++;
123         j=0;
124     }
125 
126     if(!is_end(sudoku))
127     {
128         return 1;
129     }
130 
131     if(sudoku[i][j])
132     {
133         if(solve_sodoku(i,j+1,sudoku))
134         {
135             return 1;
136         }
137         return 0;
138     }
139 
140     int z = 0;
141     for(z=1;z<=N;z++)
142     {
143         if(is_valid(z,i,j,sudoku))
144         {
145             sudoku[i][j] = z;
146             if(solve_sodoku(i,j+1,sudoku))
147             {
148                 return 1;
149             }
150             sudoku[i][j] = 0;
151         }
152     }
153 
154     return 0;
155 }
156 
157 /*
158     Saves a Soduku-grid to a file
159 */
```

```c
int dumb_sodoku(int sudoku[N][N])
{
    FILE *fp;
    fp = fopen("Soduku_dumb.txt","w");

    if(fp==NULL)
    {
        return 1;
    }

    int i = 0;
    int j = 0;
    for(i = 0; i<N;i++)
    {
        for(j=0;j<N;j++)
        {
            fprintf(fp,"%d ", sudoku[i][j]);
        }
        fprintf(fp,"\n");
    }
    fclose(fp);

    return 0;
}
```

Aufgaben/Blatt02/Final/SodukuFunctions.c

```
Soduku_solver:  SodukuSolver.o sodukulib.so
    gcc -o Soduku_solver SodukuSolver.o -L. -lsodukulib
    export LD_LIBRARY_PATH=/path/to/library:${LD_LIBRARY_PATH}

sodukulib.so:   SodukuFunctions.o
     gcc -shared -o libsodukulib.so SodukuFunctions.o

SodukuFunctions.o: SodukuFunctions.c
    gcc -Wall -fPIC -c SodukuFunctions.c

SodukuSolver.o: SodukuSolver.c SodukuFunctions.h
    gcc -Wall -c SodukuSolver.c

clean:
    rm *.so
    rm *.o
    rm Soduku_solver
```

Aufgaben/Blatt02/Final/Makefile.txt

```
a) p = feld;
    Zulaessig, Felder sind auch nur Zeiger auf die eigentliche Adresse (
       Beides hat gleichen Typ)

b) feld = p;
    Compilerfehler:warning: makes integer from pointer
    Hier wird versucht ein Feld an Zeigern (bzw. erste stelle) den Wert
       eines integer zu geben.


c) p = &feld[3];
```

```
10      Zulaessig, da hier den Inhalt des Pointers, was ein int ist, einer int-
            variable zugewiesen wird
11
12  d) feld[2] = p[5];
13      Zulaessig beim compilieren da beides gleicher Typ, aber eventuell
            Laufzeitfehler wenn pointer nicht initialisiert wurde.
14
15  e) p1 = p2 + i;
16      Zulaessig, aendert aber nicht den Inhalt sondern nur die Adresse auf
            die p1 zeigt
17
18  f) p1 = i + p2;
19      Zulaessig, + wird ueberladen gibt int*. aendert aber nicht den Inhalt
            sondern nur die Adresse auf die p1 zeigt
20
21  g) i = p1 * p2;
22      Unzulaessig, keine Multiplikation auf Pointern:
23      Compilerfehler: error: invalid operands to binary * (have 'int *' and '
            int *')
24
25  h) i = p1 - p2;
26      Generell Zulaessig, liefert die Entfernung zwischen zwei Zeigern und
            liefert ein int zurueck
27
28  i) i = p1 + p2;
29  Nicht Zulaessig, Addition nicht definiert. Ergibt keinen Sinn diese
        Operation;
30      Compilerfehler: error: invalid operands to binary + (have 'int *' and '
            int *')
31
32  Erklaeren sie darueber hinaus, wie der [][]-Zugriff bei zweidimensionalen
        Arrays aussieht:
33      int feld[][] ist quasi ein int Array von einem int array. So liefert
            feld[0] ein
34      feld (bzw. Pointer) mit weiteren elementen feld[0][0] ist dann der
            eigentliche wert
35
36  Eine nette Moeglichkeit verwirrenden Programmcode zu generieren ist es,
        Array-Namen und Index zu vertauschen, d.h. feld[i] stellt denselben
        Zugriff
37  wie i[feld] dar. Warum ist das so?
38      felder sind eigentlich pointer auf die erste stelle eines Objekts, die
            intern so aussehen:
39      *(feld + i). So wird einfach nur die Adresse weiter gezaehlt. i[feld]
            ist geleichbedeutend zu
40      *(i + feld), was aequivalent ist zu *(feld + i)
```

Aufgaben/Blatt02/Final/Aufgabe_2_2.txt