2. Übungsblatt - C++

Henrik Gerdes, Manuel Eversmeyer

4. November 2018

```c
#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

```c
#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);

```

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

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      int i,j;
18
19      printf("\n+-------+-------+-------+\n");
20      for(i = 0; i<N;i++){
21          for(j=0;j<N;j++){
22              if(j%3 ==0){
23                  printf("|");
24              };
25              printf("%2d ", sudoku[i][j]);
26          }
27          if(i%3==2){
28              printf("|\n+-------+-------+-------+\n");
29          }else{
30              printf("|\n");
31          }
32      }
33  }
34
35  /*
36      Checks if a given number z is valid to set on position i,j in
37      a Soduku-grid
38  */
39  int is_valid(int z, int i, int j, int sudoku[N][N]){
40      if(z>9 || z<1) {
41          return 0;
42      }
43
44      int k;
45      for(k=0; k<N;k++){
46          if(sudoku[i][k] == z){
47              //printf("Fehler in Zeile gefunden");
```

```
48              return 0;
49          }
50          if(sudoku[k][j] == z){
51              //printf("Fehler in Spalte gefunden");
52              return 0;
53          }
54      }
55
56
57      int square_i = i/3;
58      int square_j = j/3;
59      square_i *=3;
60      square_j *=3;
61
62      int limit_i = square_i +3;
63      int limit_j = square_j +3;
64
65      int index_i, index_j;
66
67
68      for(index_i = square_i;index_i < limit_i;index_i++){
69          for(index_j = square_j;index_j < limit_j;index_j++){
70
71              if(sudoku[index_i][index_j] == z){
72                  //printf("sq_i: %d    sq_j: %d", index_i,index_j);
73                  //printf("Fehler in Feld gefunden");
74                  return 0;
75              }
76          }
77      }
78
79
80      return 1;
81  }
82
83  /*
84      Checks if there are still free positions in the grid
85  */
86  int is_end(int sudoku[N][N]){
87      int i, j;
88      for(i = 0; i<N;i++){
89          for(j=0;j<N;j++){
90              if(sudoku[i][j]==0){
91                  return 1;
92              }
93          }
94      }
95      return 0;
96  }
97
98  /*
99      Solves a Soduku using backtracking
100 */
101 int solve_sodoku(int i, int j, int sudoku[N][N]){
102     //printf("Solve mit: i=%d   und j=%d\n", i ,j);
103     if(j==N){
104         i++;
105         j=0;
```

```
106     }
107
108     if(!is_end(sudoku)){
109         return 1;
110     }
111
112     int z;
113     if(sudoku[i][j]){
114         if(solve_sodoku(i,j+1,sudoku)){
115             return 1;
116         }
117         return 0;
118     }
119
120     for(z=1;z<=N;z++){
121         if(is_valid(z,i,j,sudoku)){
122             sudoku[i][j] = z;
123             if(solve_sodoku(i,j+1,sudoku)){
124                 return 1;
125             }
126             sudoku[i][j] = 0;
127         }
128     }
129
130     return 0;
131 }
132
133 /*
134     Saves a Soduku-grid to a file
135 */
136 int dumb_sodoku(int sudoku[N][N]){
137     FILE *fp;
138     fp = fopen("Soduku_dumb.txt","w");
139
140     if(fp==NULL){
141         return 1;
142     }
143     int i,j;
144
145     for(i = 0; i<N;i++){
146         for(j=0;j<N;j++){
147             fprintf(fp,"%d ", sudoku[i][j]);
148         }
149         fprintf(fp,"\n");
150     }
151     fclose(fp);
152
153     return 0;
154 }
```

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];
    Zulaessig, da hier den Inhalt des Pointers, was ein int ist, einer int-
       variable zugewiesen wird

d) feld[2] = p[5];
    Zulaessig beim compilieren da beides gleicher Typ, aber eventuell
       Laufzeitfehler wenn pointer nicht initialisiert wurde.

e) p1 = p2 + i;
    Zulaessig, aendert aber nicht den Inhalt sondern nur die Adresse auf
       die p1 zeigt

f) p1 = i + p2;
    Zulaessig, + wird ueberladen gibt int*. aendert aber nicht den Inhalt
       sondern nur die Adresse auf die p1 zeigt

g) i = p1 * p2;
    Unzulaessig, keine Multiplikation auf Pointern:
    Compilerfehler: error: invalid operands to binary * (have 'int *' and '
       int *')

h) i = p1 - p2;
    Generell Zulaessig, liefert die Entfernung zwischen zwei Zeigern und
       liefert ein int zurueck

i) i = p1 + p2;
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