Übung 5.1: Matrix erstellen

Schreiben Sie die Funktion createMatrix(…), welche eine Matrix erzeugt und die Matrix mit einem vorgegebenen Werte initialisiert (der Wert wird als dritter Übergabeparameter übergeben).

main.c

```
/**============================================================================================
Hochschule Luzern - Technik & Architektur                              Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                               http://hslu.ximit.ch
--------------------------------------------------------------------------------------------
Project     : Uebung_5-1
Name        : main.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-27v1.0
Description : Matrix erstellen
              Schreiben Sie die Funktion createMatrix(…), welche eine Matrix erzeugt und die
              Matrix mit einem vorgegebenen Werte initialisiert (der Wert wird als dritter
              Übergabeparameter übergeben).
============================================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include "matrix.h"

/**
 * The main procedure to run the application.
 * @param argc Number of command line arguments
 * @param argv The forwarded command line arguments
 * @return Application return (error) code
 */
int main(int argc, char** argv) {

  // set number of rows, cols and the initialization value
  const int rows = 4;
  const int cols = 3;
  const int value = 3;

  // create 3x4 matrix with initialization value 3
  int **m = createMatrix(rows,cols,value);

  if(m != NULL) {
    // print matrix to standard output
    printMatrix(m, rows, cols);

    // free memory
    freeMatrix(m, rows);
  }

    return (EXIT_SUCCESS);
}
```

matrix.h

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                        Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                          http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Project     : Uebung_5-1
Name        : matrix.h
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-27v1.0
Description : Matrix erstellen
============================================================================================================*/
#ifndef MATRIX_H
#define MATRIX_H

/**
 * Creates a matrix by a given number of rows and a given number of columns.
 * The matrix is filled with the values given by the third parameter.
 * Note: Don't forget to dealloc space after using!
 *
 * @param rows Number of rows for the matrix
 * @param cols Number of columns for the matrix
 * @param value Initialization value of the cells
 * @return <rows> x <cols> matrix filled with <value>
 * @see freeMatrix(int **matrix, int rows, int cols)
 */
int **createMatrix(int rows, int cols, int value);

/**
 * Filles a previously created matrix with a given value.
 *
 * @param matrix The matrix to be filled
 * @param rows Number of rows of the matrix
 * @param cols Number of columns of the matrix
 * @param value Initialization value of the cells
 * @return nothing
 */
void fillMatrix(int **matrix, int rows, int cols, int value);

/**
 * Prints a previously created matrix to the standard output
 *
 * @param matrix The matrix to be printed
 * @param rows Number of rows for the matrix
 * @param cols Number of columns for the matrix
 * @return nothing
 */
void printMatrix(int *const *matrix, int rows, int cols);

/**
 * Deallocates previously reserved memory
 *
 * @param matrix The matrix to deallocate
 * @param rows Number of rows for the matrix
 * @return <rows> x <cols> matrix filled with <value>
 */
void freeMatrix(int **matrix, int rows);

#endif  /* MATRIX_H */
```

matrix.c

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                        Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                         http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Project     : Uebung_5-1
Name        : matrix.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-27v1.0
Description : Matrix erstellen
============================================================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include "matrix.h"

int** createMatrix(int rows, int cols, int value) {
  int row;                // used for iteration
  int** matrix = NULL;    // placeholder for matrix

  // allocate memory
  matrix = (int**) malloc(rows*sizeof(int*));
  if(matrix != NULL) {    // if succesfull allocated

    // allocate rows
    for(row=0; row<rows; row++) {
      matrix[row] = (int*) malloc(cols*sizeof(int));
      if(matrix[row] == NULL) {
        printf("Unable to allocate memory!\n");
        // free if unsuccesfull allocation
        freeMatrix(matrix, row);
        return NULL;
      }
    }

    // fill matrix with values
    fillMatrix(matrix, rows, cols, value);
  }
  else printf("Unable to allocate memory!\n");

  return matrix;
}

void fillMatrix(int** matrix, int rows, int cols, int value) {
  int row, col;   // used for iteration

  // fill matrix
  for(row = 0; row < rows; row++)
    for(col = 0; col < cols; col++)
      matrix[row][col] = value;
}

void printMatrix(int* const* matrix, int rows, int cols) {
  int row, col;   // used for iteration

  for(row = 0; row < rows; row++) {
    printf("{");   // print row boundary
    for(col = 0; col < cols; col++) {
      // print cell. separator only if first cell of row
      printf(col == 0 ? "%6d" : ",%6d", matrix[row][col]);
    }
    printf("}\n");   // print row boundary
  }
}

void freeMatrix(int** matrix, int rows) {
  int row;        // used for iteration

  // free columns first
  for (row = 0; row < rows; row++){
    free(matrix[row]);
  }

  // free rows
  free(matrix);
}
```

Übung 5.2: Entwurf einer Übung zum Thema Zeigerarithmetik

Entwerfen Sie eine Übung analog der Aufgabe in Kapitel 3 wo Sie Zeigerarithmetik einsetzen.

main.c

```
/**=============================================================================================
Hochschule Luzern - Technik & Architektur                                  Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                   http://hslu.ximit.ch
-----------------------------------------------------------------------------------------------
Project     : Uebung_5-2
Name        : main.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-27v1.0
Description : Entwurf einer Übung zum Thema Zeigerarithmetik
              Entwerfen Sie eine Übung analog der Aufgabe in Kapitel 3 wo Sie Zeigerarithmetik
              einsetzen.
=============================================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include "zeigerarithmetic.h"

/*
 * The main procedure to run the application.
 * @param argc Number of command line arguments
 * @param argv The forwarded command line arguments
 * @return Application return (error) code
 */
int main(int argc, char** argv)
{
  int field[SIZE][SIZE];
  int posX = 0;
  int posY = 0;
  char input;

  /* fill the entire field with zero */
  memset(field, 0, sizeof(field));

  /* set a one in the first field */
  field[posY][posX] = 1;

  /* loop until a 'x' has been pressed (see below) */
  do
  {
    /* clear the screen - this makes the "matrix" appear at the
       same position all the time (very nice ;-) ) */
    //system("cls");
    system("cmd.exe /c cls"); // Cygwin hack

    /* prompt */
    printf("Zug [w hoch, a links, s runter, " \
           "d rechts] eingeben; beenden mit x\n");

    /* print the entire field on the console */
    printField(&field[0][0]);

    /* wait until a key has been pressed */
    while (!isalnum(input = (char)getchar()));

    /* move the "one" in the field, according to input */
    move(&field[0][0], &posX, &posY, input);
  } while(input != 'x');

  /* exit */
        return(EXIT_SUCCESS);
}
```

zeigerarithmetic.h

```
/**=============================================================================================
Hochschule Luzern - Technik & Architektur                                  Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                   http://hslu.ximit.ch
-----------------------------------------------------------------------------------------------
Project     : Uebung_5-2
Name        : zeigerarithmetic.h
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-27v1.0
Description : Matrix erstellen
              Schreiben Sie die Funktion createMatrix(…), welche eine Matrix erzeugt und die
              Matrix mit einem vorgegebenen Werte initialisiert (der Wert wird als dritter
              Übergabeparameter übergeben).
=============================================================================================*/

#ifndef ZEIGERARITHMETIC_H
#define ZEIGERARITHMETIC_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* function prototypes */
void printField(const int *field);
void move(int *field, int *posX, int *posY, char in);

/* define the size of the field */
#define SIZE 7

/* print an error message for mysterious SIZE values */
#if SIZE < 2
#error "SIZE IS SENSELESS"
#elif SIZE > 39
#error "SIZE IS LARGER THAN CONSOLE :-)"
#endif



#endif  /* ZEIGERARITHMETIC_H */
```

zeigerarithmetic.c

```
/**=============================================================================================
Hochschule Luzern - Technik & Architektur                                  Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                   http://hslu.ximit.ch
-----------------------------------------------------------------------------------------------
Project     : Uebung_5-2
Name        : zeigerarithmetic.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-27v1.0
Description : Matrix erstellen
              Schreiben Sie die Funktion createMatrix(…), welche eine Matrix erzeugt und die
              Matrix mit einem vorgegebenen Werte initialisiert (der Wert wird als dritter
              Übergabeparameter übergeben).
=============================================================================================*/
#include <stdio.h>
#include "zeigerarithmetic.h"

/* print the field */
void printField(const int *field)
{
  int row, col;

  /* newline */
  printf("\n");

  /* loop through all the rows */
  for(row = 0; row < SIZE; row++)
  {
    /* loop through all the columns */
    for(col = 0; col < SIZE; col++)
    {
      printf("%d ", *(field + row * SIZE + col));
    }
    printf("\n");
  }
}


/* moves the "one" around */
void move(int *field, int *posX, int *posY, char in)
{
  /* overwrite the old position with zero */
  *(field + *posY * SIZE + *posX) = 0;

  /* calculate the new position */
  switch(in)
  {
    case 'w':
    {
      (*posY)--;
    }
    break;

    case 'a':
    {
      (*posX)--;
    }
    break;

    case 's':
    {
      (*posY)++;
    }
    break;

    case 'd':
    {
      (*posX)++;
    }
    break;

    /* should not happen, normally :-) */
    default:
      return;
  }
```

```
  /* check each of the coordinates against upper / lower bounds */
  if(*posX < 0)
  {
    *posX = SIZE - 1;
  }

  if(*posX > SIZE - 1)
  {
    *posX = 0;
  }

  if(*posY < 0)
  {
    *posY = SIZE - 1;
  }

  if(*posY > SIZE - 1)
  {
    *posY = 0;
  }

  /* set the new position */
  *(field + *posY * SIZE + *posX) = 1;
}
```

Übung 5.3: Zeiger auf Funktion

Implementieren Sie in der vorgegebenen Ampelsteuerung eine Callback-Funktion, damit die Zeiten der einzelnen Phasen ‚von aussen' eingestellt werden können.

Studieren Sie den vorgegebenen Code und schauen Sie sich an, wie das Programm bis jetzt läuft. Kopieren Sie dazu den Code von unten in die drei Dateien semaphore.h, semaphore.c und main.c. Anschliessend machen Sie die Änderungen.

main.c

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                        Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                         http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Project     : Uebung_5-3
Name        : main.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-27v1.0
Description : Zeiger auf Funktion / Threads
              Implementieren Sie in der vorgegebenen Ampelsteuerung eine Callback-Funktion, damit
              die Zeiten der einzelnen Phasen ‚von aussen' eingestellt werden können.
============================================================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include "semaphore.h"

/**
* Determins the length of each phase of the semaphor.
* @param phase actuel phase
* @return Time in ms for the corresponding phase
*/
int getPhaseTime(SemaphorStates_t phase)
{
        int rc = 1;
        switch (phase) {
        case GREEN:          rc = 6;  break;
        case GREEN_BLINKING: rc = 2;  break;
        case YELLOW:         rc = 2;  break;
        case RED:            rc = 10; break;
        default:             rc = 2;  break;
        }
        return rc;
}

/**
* Starts the semaphor and defines with the the implementation of a callback
* function the length of the various periods.
*/
int main(int argc, char** argv)
{
        //Set callback function
        setSemaphorCallbackFunction(getPhaseTime);

        printf("Starting the semaphor...\n");
        startSemaphore();

        // Let the semaphor run for 1 minute
        sleep(60);
        printf("Stopping the semaphor...\n");
        stopSemaphor();
        printf("Semaphor stopped...\n");

        return (EXIT_SUCCESS);
}
```

semaphore.h

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                       Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                        http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Project     : Uebung_5-3
Name        : semaphore.h
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-27v1.0
Description : Zeiger auf Funktion / Threads
============================================================================================================*/
#ifndef SEMAPHORE_H
#define SEMAPHORE_H

/**
 * Implementation of a semaphor with a thread.
 * Uses a callback function to define the various times.
 */
typedef enum SemaphorStates_ {
    GREEN = 1,
    GREEN_BLINKING,
    YELLOW,
    RED
} SemaphorStates_t;

/**
 * Set the callback function that defines the time of each phase.
 * @param b pointer to function that returns the phase length in seconds.
 *        The function gets as Parameter the current semaphore state.
 */
void setSemaphorCallbackFunction(int (*b) (SemaphorStates_t phase));

/**
 * Starts the semaphor.
 * Within a thread, the semaphor changes its state from GREEN to GREEN_BLINKING
 * to YELLOW to RED to GREEN to ... .
 * After each state change, the callbackfunction defined by setCallbackFunction(...)
 * will be called to get the length of the corresponding phase.
 */
void startSemaphore();

/**
 * Stop the semaphor.
 */
void stopSemaphor();

/**
 * Definiert die Semaphorezeiten
 */
int getPhaseTime(SemaphorStates_t phase);

#endif  /* SEMAPHORE_H */
```

semaphore.c

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                        Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                         http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Project     : Uebung_5-3
Name        : semaphore.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-27v1.0
Description : Zeiger auf Funktion / Threads
============================================================================================================*/
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include "semaphore.h"

static int threadRunning = 0;
static SemaphorStates_t state = RED;
static pthread_t threadId; // Thread stuff
static pthread_attr_t threadAttr;
int (*f)(SemaphorStates_t phase);
int sleepTime;

/**
 * Printout current thread state
 */
static void printThreadState(int sleepTime) {
    char stateStr[64];

    switch (state) {
        case GREEN: strcpy(stateStr, "GREEN");
            break;
        case GREEN_BLINKING: strcpy(stateStr, "GREEN BLINKING");
            break;
        case YELLOW: strcpy(stateStr, "YELLOW");
            break;
        case RED: strcpy(stateStr, "RED");
            break;
        default: strcpy(stateStr, "UNDEFINED");
            break;
    }
    printf("The semaphor is %-14s for %2d seconds\n", stateStr, sleepTime);
    fflush(stdout);
}

static void* threadFunction(void* args) {
    while (threadRunning) {
        switch (state) {
            case GREEN:
                state = GREEN_BLINKING;
                break;
            case GREEN_BLINKING:
                state = YELLOW;
                break;
            case YELLOW:
                state = RED;
                break;
            case RED:
                state = GREEN;
                break;
            default: // Should not happen, but added error recovery
                state = RED;
                break;
        }

        if (f != NULL) {
            sleepTime = f(state);
        }

        printThreadState(sleepTime);
        sleep(sleepTime);
    }
    printf("Semaphor thread stopped\n");

    return NULL;
}
```

```
void setSemaphorCallbackFunction(int (*b) (SemaphorStates_t phase)) {
    f = b;
}

void startSemaphore() {
    threadRunning = 1;

    // Thread starten
    pthread_attr_init(&threadAttr);
    // Hier wird als drittes Argument ein Zeiger auf die Thread-Funktion übergeben.
    pthread_create(&threadId, &threadAttr, threadFunction, NULL);
}

void stopSemaphor() {
    threadRunning = 0;

    // Wait until thread is finished
    pthread_join(threadId, NULL);
    pthread_attr_destroy(&threadAttr);
}
```

Übung 5.4: Mehrdimensionales Array mit Druckmesswerten

Ein Messfeld von Drucksensoren in einer 2D-Ebene liefert Ihnen Druckmesswerte, welche Sie in einem 2-dimensionalen Array speichern sollen. Die Messwerte bestehen jeweils aus einem Druckwert (float) und einem Einheitsvorsatz (Mega, Kilo, Zero, Milli, Mikro).

Definieren Sie eine passende Datenstruktur für den Druckmesswert.

Kreieren Sie dynamisch ein variables 2-dimensionales Array mit diesen Druckmesswerten und initialisieren diese allgemein mit den Werten wie im Beispiel unten angegeben. Verwenden Sie bitte Pointer-auf-Pointer, Enum und Strukturen wie in der Vorlesung gezeigt.

Schreiben Sie zusätzlich eine Funktion, welche das Array entsprechend formatiert auf der Konsole ausgibt.

main.c

```
/**============================================================================================
Hochschule Luzern - Technik & Architektur                              Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                               http://hslu.ximit.ch
--------------------------------------------------------------------------------------------
Project     : Uebung_5-4
Name        : main.c
Author      : StalderJ
Version     : 2012-10-27v1.0
Description : Mehrdimensionales Array mit Druckmesswerten
              Ein Messfeld von Drucksensoren in einer 2D-Ebene liefert Ihnen Druckmesswerte, welche
              Sie in einem 2-dimensionalen Array speichern sollen. Die Messwerte bestehen jeweils
              aus einem Druckwert (float) und einem Einheitsvorsatz (Mega, Kilo, Zero, Milli,Mikro).
              Definieren Sie eine passende Datenstruktur für den Druckmesswert.
              Kreieren Sie dynamisch ein variables 2-dimensionales Array mit diesen Druckmesswerten
              und initialisieren diese allgemein mit den Werten wie im Beispiel unten angegeben.
              Verwenden Sie bitte Pointer-auf-Pointer, Enum und Strukturen wie in der Vorlesung
              gezeigt. Schreiben Sie zusätzlich eine Funktion, welche das Array entsprechend
              formatiert auf der Konsole ausgibt.
============================================================================================*/
#include <stdlib.h>
#include "sensorgrid.h"

/*
 * Program control
 */
int main(int argc, char** argv) {

    // Sensordata Struct
    sdata_t** grid;

    const int rows = 3;
    const int cols = 4;

    // Create the array
    createGrid(&grid, rows, cols);

    // fill in some sensless data
    writeToGrid(grid, 0, 0, 0.00001);
    writeToGrid(grid, 0, 1, 0.1);
    writeToGrid(grid, 0, 2, 0.2);
    writeToGrid(grid, 0, 3, 0.3);

    writeToGrid(grid, 1, 0, 1.0);
    writeToGrid(grid, 1, 1, 1.1);
    writeToGrid(grid, 1, 2, 1.2);
    writeToGrid(grid, 1, 3, 1.3);

    writeToGrid(grid, 2, 0, 2.0);
    writeToGrid(grid, 2, 1, 2000.1);
    writeToGrid(grid, 2, 2, 2000000.2);
    writeToGrid(grid, 2, 3, 2.3);

    // put it out to console
    displayGrid(grid, rows, cols);

    // Free what would be free-ed anyway
    free(grid);
    return (EXIT_SUCCESS);
}
```

sensorgrid.h

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                        Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                          http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Project     : Uebung_5-4
Name        : sensorgrid.h
Author      : StalderJ
Version     : 2012-10-27v1.0
Description : Mehrdimensionales Array mit Druckmesswerten
============================================================================================================*/
#ifndef SENSORGRID_H
#define SENSORGRID_H

/**
 * Defines the fraction.
 */
typedef enum fraction {
    Mega = 'M',
    Kilo = 'K',
    Zero = ' ',
    Milli = 'm',
    Mikro = '\xb5'
} fraction_t;

/**
 * Defines the data structure.
 */
typedef struct sensordata {
    float pressure;
    fraction_t frac;
} sdata_t;

/**
 * Creates a grid of sensor data containers.
 * @param grid Reference to the grid that will be filled.
 * @param x First Dimension of the grid.
 * @param y Second Dimensions of the grid.
 */
void createGrid(sdata_t ***grid, unsigned int x, unsigned int y);

/**
 * Displays the grid on console.
 * @param grid Reference to the grid.
 * @param x Size of the grid in x direction.
 * @param y Size of the grid in y direction.
 */
void displayGrid(sdata_t **grid, unsigned int x, unsigned int y);

/**
 * Writes a value to the grid at position [x,y].
 * @param grid Reference to the grid.
 * @param x X-Dimension parameter.
 * @param y Y-Dimension parameter.
 * @param val Measure value.
 */
void writeToGrid(sdata_t **grid, unsigned int x, unsigned int y, float val);

#endif  /* SENSORGRID_H */
```

sensorgrid.c

```
/**=============================================================================================================
Hochschule Luzern - Technik & Architektur                                        Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                          http://hslu.ximit.ch
-------------------------------------------------------------------------------------------------------------
Project     : Uebung_5-4
Name        : sensorgrid.c
Author      : StalderJ
Version     : 2012-10-27v1.0
Description : Mehrdimensionales Array mit Druckmesswerten
=============================================================================================================*/
#include "sensorgrid.h"

void createGrid(sdata_t ***g1, unsigned int x, unsigned int y){
    int i;
    sdata_t **grid;

    // Create the x dimension of the grid
    grid = (sdata_t**) malloc(x* sizeof(sdata_t*));
    // Create an y-dimension in every x direction
    for(i=0;i < x;i++){
        *(grid+i) = (sdata_t*) malloc(y * sizeof(sdata_t));
    }

    // Write the address where the first grid node is stored
    // directly to the pointer-var in the caller function
    (*g1) = grid;
}

void displayGrid(sdata_t **grid, unsigned int x, unsigned int y){
    int i, j;

    // Walk through every row
    for(i=0;i < x;i++){
        printf("|");
        // Walk through every element in the row
        for(j=0;j < y;j++){
            printf("%10g%c |", grid[i][j].pressure, grid[i][j].frac);
        }
        printf("\n");
    }
}

void writeToGrid(sdata_t **grid, unsigned int x, unsigned int y, float val) {
    // Searches for a matching fraction and store value with fraction
    if (val > 1000000) {
        grid[x][y].pressure = val / (1000 * 1000);
        grid[x][y].frac = Mega;
    } else if(val > 1000) {
        grid[x][y].pressure = val / 1000;
        grid[x][y].frac = Kilo;
    } else if(val < 0.001) {
        grid[x][y].pressure = val * (1000 * 1000);
        grid[x][y].frac = Mikro;
    } else if(val < 1) {
        grid[x][y].pressure = val * 1000;
        grid[x][y].frac = Milli;
    } else {
        grid[x][y].pressure = val;
        grid[x][y].frac = Zero;
    }
}
```