Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Technik & Architektur

Mikrocontroller, C-Programmierung

Selbststudium und Übungen 5

Peter Sollberger, Martin Vogel V7.1

Am Ende dieser Semesterwoche wissen Sie, wie man mit mehrdimensionalen Arrays umgeht. Sie können mit Zeiger auf Zeigern umgehen und Zeiger auf Funktionen einsetzen.

1. Selbststudium

Gehen Sie nochmals die Folien durch.

2. Übungen

(zu diesen Übungen werden durch die Studierenden Musterlösungen präsentiert und abgegeben).

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).

Der untenstehende Aufruf erzeugt eine 3x4 Matrix mit dem Inhalt

3	3	3	3
3	3	3	3
3	3	3	3

```
int **m;
m = createMatrix(3, 4, 3);
```

1. Parameter: Anzahl Zeilen
2. Parameter: Anzahl Spalten
3. Parameter: Initialisierungswerte

Übung 5.2: Entwurf einer Übung zum Thema Zeigerarithmetik

Entwerfen Sie eine Übung analog der Aufgabe in Kapitel 3 wo Sie Zeigerarithmetik einsetzen. Schreiben Sie eine Aufgabenstellung und erarbeiten Sie die Musterlösung.

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.

Datei `semaphore.h`

```
#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 getPhaseTimePointer to function that returns the phase length in seconds.
 *        The function gets as Parameter the current semaphore state.
 */
/* !!! To do !!!  void setSemaphorCallbackFunction(...); */

/**
 * 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();

#endif   /* _SEMAPHORE_H */
```

Datei `semaphore.c`

```
/** Implementation of a semaphor with a thread. Uses the POSIX library. */

#include <stdio.h>
#include <pthread.h>
#include "semaphore.h"

/* !!! To do: Define a variable getPhaseTimeFunction of type 'pointer-to-function' for
   the callback function and set it to NULL. */

static int threadRunning = 0;
static SemaphorStates_t state = RED;
static pthread_t       threadId;       // Thread stuff
static pthread_attr_t  threadAttr;

/**
 * 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) {
    int sleepTime = 2;
    printThreadState(sleepTime);
    sleep(sleepTime);

    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;
        }
```

```
/* !!! To do: Call here the callback function to get the sleep time
       if (getPhaseTimeFunction != NULL) {
           sleepTime = ....;
*/
        printThreadState(sleepTime);
        sleep(sleepTime);
    }
    printf("Semaphor thread stopped\n");

    return NULL;
}

/* !!! To do !!!
void setSemaphorCallbackFunction(...) {
    getPhaseTimeFunction =
}
*/

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);
}
```

Datei `main.c`

```
#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: 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) {
    /* !!! To do: Set callback function !!!
    setSemaphorCallbackFunction(...);
     */

    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);
}
```

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.

Beispiel eines 3 x 4 Arrays mit den Inhalten Druckwert und Einheitsvorsatz initialisiert.

0.0 Kilo	0.1 Kilo	0.2 Kilo	0.3 Kilo
1.0 Kilo	1.1 Kilo	1.2 Kilo	1.3 Kilo
2.0 Kilo	2.1 Kilo	2.2 Kilo	2.3 Kilo

3. Weitere freiwillige Aufgabe

Zeigerarithmetik

Auf Folie 12 von MC_SW5_C, 'Kontrollfragen B - mehrdimensionale Vektoren' wird ein zweidimensionaler Array zur Bestimmung des Jahrestages verwendet.

Schreiben Sie ein Programm, welche die Matrix **`daytab`** dynamisch alloziert (auf dem Heap) und initialisiert.
Anschliessend berechnen Sie den Jahrestag unter Zuhilfenahme von Zeigerarithzmetik (ohne [] Operator.

Schreiben Sie ein Testprogramm (main-Methode) zum Testen.

Musterlösung (x86):

```
/*
 * Author: M.Vogel HSLU (c) 2008
 */

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

static char daytab [2][13] =
{
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};

int day_of_year(int, int, int);
void month_day(int, int, int*, int*);
int atooi(char*);

/**
 * Umwandlung string in integer - ohne Fehlerprüfung
 */
int atooi(char s[])
{
    int  i, n = 0;

    for (i=0; s[i] >= '0' && s[i] <= '9'; ++i)
        n = 10 * n + (s[i] - '0');
    return n;
}


/**
 * day of year: Tag im Jahr aus Monat und Tag bestimmen
 */
int day_of_year(int year, int month, int day)
{
    int leap;
    char* p;

    leap = ((year%4 == 0) && (year%100 != 0) || (year%400 == 0));
    p = daytab[leap];
    while (--month)
        day += *(++p);
    return day;
}

/**
 * month_day: Monat und Tag aus Tag im Jahr bestimmen
 */
void month_day(int year, int yearday, int* pmonth, int* pday)
{
    int leap;
    char* p;

    leap = ((year%4 == 0) && (year%100 != 0) || (year%400 == 0));
    p = daytab[leap];
```

```
    while (yearday > *(++p))
            yearday -= *p;
    *pmonth = p - *(daytab + leap);
    *pday = yearday;
}


/**
 *  Ohne Fehlerprüfung in der Ein-/Ausgabe
 */
int main(int argc, char** argv)
{
    int day, month, year;                          // Variablen für day_of_year
    int yearyear, yearday, pmonth, pday, i=0;    // Variablen für month_day
    char s[5], t[5], c;

    // 1. Abfrage
    printf("Tag Monat Jahr eingeben (mit Space)");
    while((c = getchar()) != ' ')
        s[i++] = c;
    s[i] = '\0';
    day = atooi(s);

    i = 0;
    while((c = getchar()) != ' ')
        s[i++] = c;
    s[i] = '\0';
    month = atooi(s);

    i = 0;
    while((c = getchar()) != '\n')
        s[i++] = c;
    year = atooi(s);

    printf("Anzahl Tag im Jahr: %d\n", (day_of_year(year, month, day)));

    // 2. Abfrage
    i=0;
    printf("Bitte Jahr und Jahrtag eingeben (mit Space)");
    while((c = getchar()) != ' ')
        t[i++] = c;
    yearday = atooi(t);                //besserer Programmierstil mit atooi Funktion

    i=0;
    while((c = getchar()) != '\n')
        t[i++] = c;
    t[i] = '\0';
    yearyear =  atooi(t);

    month_day(yearday, yearyear, &pmonth, &pday);

    printf("Monat: %d Tag: %d\n", pmonth, pday );

    return (EXIT_SUCCESS);
}
```