

# Hardwarenahe Programmierung / Angewandte Informatik

## Musterlösung zu den Übungsaufgaben – 31. Oktober 2016

### Aufgabe 1: Datum-Bibliothek

Schreiben Sie eine Bibliothek `dates.c` (mit Header-Datei `dates.h`) zur Behandlung von Datumsangaben.

Diese soll enthalten:

- einen **struct**-Datentyp `date`, der eine Datumsangabe speichert,
- eine Funktion **void** `date_print (date *d)`, die ein Datum ausgibt,
- eine Funktion **int** `date_set (date *d, int day, int month, int year)`, die ein Datum auf einen gegebenen Tag setzt und zurückgibt, ob es sich um ein gültiges Datum handelt,
- eine Funktion **void** `date_next (date *d)`, die ein Datum auf den nächsten Tag vorrückt.

### Lösung

Die Dateien `dates.c` und `dates.h` enthalten die Bibliothek, die Datei `test-dates.c` ein Programm zum Testen der Bibliothek.

Eine detaillierte Anleitung, wie man auf die Funktion `date_next()` kommt, finden Sie im Skript zur Lehrveranstaltung, Datei `hp-2016ws.pdf`, ab Seite 26.

Eine **falsche** Lösung wäre eine einzige `.c`-Datei, die sowohl die `date`-Funktionen als auch die `main()` enthielte.

Man beachte, daß die Deklaration der **struct** `date` in der `.h`-Datei erfolgt und daß `dates.c` die Datei `dates.h` mit **#include** einbindet. Wenn man den Datentyp `date` direkt in `dates.c` definiert, kennt das aufrufende Programm (hier: `main()` in `test-dates.c`) ihn nicht. Wenn man ihn, anstatt `dates.h` per **#include** in `dates.c` einzubinden, in `dates.c` nochmals definiert, funktioniert es zwar, aber man hat eine Code-Verdopplung: Jede Änderung an der Datenstruktur `date` muß dann an zwei Stellen erfolgen.

### Aufgabe 2: Fehlerhaftes Programm

Das folgende OpenGL-Programm (Datei: `aufgabe-2.c`) soll einen sich gleichmäßig drehenden Würfel darstellen, ist jedoch fehlerhaft.

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include "opengl-magic.h"

float t = 0.0;

void draw (void)
{
    glClear (GL_COLOR_BUFFER_BIT
            | GL_DEPTH_BUFFER_BIT);
    set_material_color (1.0, 0.7, 0.0);
    glRotatef (t, 0.5, 1.0, 0.0);
    glutSolidCube (0.5);
    glFlush ();
    glutSwapBuffers ();
}

void timer_handler (int value)
{
    t += 0.05;
    glutPostRedisplay ();
    glutTimerFunc (50, timer_handler, 0);
}

int main (int argc, char **argv)
{
    init_opengl (&argc, argv, "Cube");
    glutMainLoop ();
    glutTimerFunc (50, timer_handler, 0);
    return 0;
}
```

(Die Datei [opengl-magic.h](#) sowie die zugehörige C-Datei [opengl-magic.c](#) sind die aus der Vorlesung bekannten Beispiel-Dateien, die Sie zusammen mit diesen Übungsaufgaben herunterladen können.)

- (a) Warum sieht man lediglich ein leeres Fenster?  
Welchen Befehl muß man ergänzen, um diesen Fehler zu beheben?

Nach der Fehlerbehebung in Aufgabenteil (a) zeigt das Programm einen sich drehenden Würfel.

- (b) Erklären Sie das Drehverhalten des Würfels.  
(c) Welche Befehle muß man ergänzen, um eine gleichförmige Drehung zu erhalten?

## Lösung

- (a) **Warum sieht man lediglich ein leeres Fenster?**  
**Welchen Befehl muß man ergänzen, um diesen Fehler zu beheben?**

Zum einen fehlt in `main()` die Übergabe der Zeichenfunktion `draw()` als Callback an die Funktion `glutDisplayFunc()`.

Zum anderen erfolgt die Übergabe der Zeitschrittfunktion `timer_handler()` als Callback an die Funktion `glutTimerFunc()` erst *nach* dem Aufruf von `glutMainLoop()`. Da das Callback jedoch innerhalb von `glutMainLoop()` bekannt sein muß, ist dies zu spät.

Das Beispiel-Programm [cube-3.c](#) aus der Vorlesung ist eine korrigierte Version des fehlerhaften Programms [aufgabe-2.c](#).

- (b) **Erklären Sie das Drehverhalten des Würfels.**

Jeder Aufruf von `glRotatef()` wirkt sich auf *alle* nachfolgenden Zeichenoperationen aus, insbesondere auch auf solche, die erst im *nächsten* Aufruf von `draw()` erfolgen.

Demzufolge wird in jedem Aufruf von `draw()` der Gesamtdrehwinkel um `t` erhöht, und wir sehen eine immer schneller werdende Drehung.

- (c) **Welche Befehle muß man ergänzen, um eine gleichförmige Drehung zu erhalten?**

Um die Wirkung von `glRotatef()` auf einen einzigen Aufruf von `draw()` zu beschränken, kann man ihn in Aufrufe der Funktionen `glPushMatrix()` und `glPopMatrix()` „einrahmen“ – siehe die Dateien [cube-4.c](#) (langsame Drehung) und [cube-5.c](#) (schnellere Drehung).

Alternativ könne man auch in jedem Aufruf von `draw()` um einen konstanten Winkel weiterdrehen und das Aufaddieren der Gesamtdrehung bewußt miteinbeziehen – siehe [cube-4a.c](#) und [cube-5a.c](#). Diese Art der Benutzung der OpenGL-Bibliothek wäre jedoch für komplexere 3d-Grafik-Szenarien ungeeignet.