

Hardwarenahe Programmierung

Musterlösung zu den Übungsaufgaben – 19. November 2018

Aufgabe 1: Arrays mit Zahlen

Wir betrachten das folgende Programm
(Datei: [aufgabe-1.c](#)):

```
#include <stdio.h>

void f (int *s0, int *s1)
{
    while (*s0 >= 0)
    {
        int *s = s1;
        while (*s >= 0)
            if (*s0 == *s++)
                printf ("%d_", *s0);
        s0++;
    }
    printf ("\n");
}

int main (void)
{
    int a[] = { 10, 4, 3, 7, 12, 0, 1, -1 };
    int b[] = { 7, 14, 0, 8, 9, 22, 10, -1 };
    f (a, b);
    return 0;
}
```

- (a) Was bewirkt die Funktion `f`, und wie funktioniert sie? (4 Punkte)
- (b) Was passiert, wenn Sie beim Aufruf der Funktion für einen der Parameter den Wert `NULL` übergeben und warum? (2 Punkte)
- (c) Was kann passieren, wenn Sie das Hauptprogramm wie folgt abändern ([aufgabe-1d.c](#)) und warum?

```
int main (void)
{
    int a[] = { 10, 4, 3, 7, 12, 0, 1 };
    int b[] = { 7, 14, 0, 8, 9, 22, 10 };
    f (a, b);
    return 0;
}
```

(2 Punkte)

Lösung

- (a) Was bewirkt die Funktion `f` und wie funktioniert sie?

Die Funktion gibt alle Zahlen aus, die sowohl im Array `s0` als auch im Array `s1` vorkommen (Schnittmenge).

Dies geschieht, indem der Zeiger `s0` das gesamte Array durchläuft (äußere Schleife). Für jedes Element des ersten Arrays durchläuft der Zeiger `s` das gesamte zweite Array (innere Schleife). Auf diese Weise wird jedes Element von `s0` mit jedem von `s1` verglichen und bei Gleichheit ausgegeben.

Um die Schleifen abbrechen zu können, enthalten beide Arrays als Ende-Markierung eine negative Zahl (`-1`).

- (b) Was passiert, wenn Sie beim Aufruf der Funktion für einen der Parameter den Wert `NULL` übergeben und warum?

In dem Moment, wo auf den jeweiligen Parameter-Zeiger zugegriffen wird (`while (*s0 >= 0)` für `s0` bzw. `int *s = s1; while (*s >= 0)` für `s1`), kommt es zu einem Absturz (Speicherzugriffsfehler). Die Dereferenzierung eines Zeigers mit dem Wert `NULL` ist nicht zulässig.

- (c) Was kann passieren, wenn Sie das Hauptprogramm wie folgt abändern ([aufgabe-1d.c](#)) und warum?

```
int main (void)
{
    int a[] = { 10, 4, 3, 7, 12, 0, 1 };
    int b[] = { 7, 14, 0, 8, 9, 22, 10 };
    f (a, b);
    return 0;
}
```

Durch die fehlenden Ende-Markierungen der Arrays laufen die Schleifen immer weiter, bis sie irgendwann zufällig auf Speicherzellen stoßen, die sich als Ende-Markierungen interpretieren lassen (negative Zahlen). Dadurch kann es zu einem Lesezugriff auf Speicher kommen, für den das Programm kein Lesezugriffsrecht hat, also zu einem Absturz (Speicherzugriffsfehler).

Aufgabe 2: Fehlerhaftes Programm: Hüpfender Ball

Das auf der nächsten Seite abgedruckte GTK+-Programm (Datei: [aufgabe-2.c](#)) soll einen hüpfenden Ball darstellen, ist jedoch fehlerhaft.

- (a) Warum sieht man lediglich ein leeres Fenster? Welchen Befehl muß man ergänzen, um diesen Fehler zu beheben? (3 Punkte)
- (b) Nach der Fehlerbehebung in Aufgabenteil (a) zeigt das Programm einen unbeweglichen Ball. Welchen Befehl muß man ergänzen, um diesen Fehler zu beheben, und warum? (2 Punkte)
- (c) Erklären Sie das merkwürdige Hüpfverhalten des Balls. Wie kommt es zustande? Was an diesem Verhalten ist korrekt, und was ist fehlerhaft? (5 Punkte)
- (d) Welche Befehle muß man in welcher Weise ändern, um ein realistischeres Hüpf-Verhalten zu bekommen? (2 Punkte)

Hinweis: Das Hinzuziehen von Beispiel-Programmen aus der Vorlesung ist ausdrücklich erlaubt – auch in der Klausur.

Allgemeiner Hinweis: Wenn Sie die Übungsaufgaben zu dieser Lehrveranstaltung als PDF-Datei betrachten und darin auf die Dateinamen klicken, können Sie die Beispiel-Programme direkt herunterladen. Dadurch vermeiden Sie Übertragungsfehler.

Lösung

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

Die für das Zeichnen zuständige Callback-Funktion wurde zwar geschrieben, aber nicht installiert. Um dies zu beheben, ergänze man den folgenden Befehl im Hauptprogramm:

`g_signal_connect (drawing_area, "draw", G_CALLBACK (draw), NULL);`

Dies erkennt man sehr schnell durch Vergleich mit dem Beispiel-Programm [gtk-13.c](#).

- (b) **Nach der Fehlerbehebung in Aufgabenteil (a) zeigt das Programm einen unbeweglichen Ball. Welchen Befehl muß man ergänzen, um diesen Fehler zu beheben, und warum?**

Die Timer-Callback-Funktion wurde zwar geschrieben, aber nicht installiert. Um dies zu beheben, ergänze man den folgenden Befehl im Hauptprogramm:

`g_timeout_add (50, (GSourceFunc) timer, drawing_area);`

Auch dies erkennt man sehr schnell durch Vergleich mit dem Beispiel-Programm [gtk-13.c](#).

- (c) **Erklären Sie das merkwürdige Hüpfverhalten des Balls. Wie kommt es zustande? Was an diesem Verhalten ist korrekt, und was ist fehlerhaft?**

Die Geschwindigkeit in y -Richtung wächst immer weiter. Der Grund dafür ist, daß die y -Komponente der Geschwindigkeit nicht auf physikalisch sinnvolle Weise berechnet wird. In der dafür zuständigen Zeile `vy = 0.5 * g * (t * t);` wird stattdessen der Weg in y -Richtung bei einer gleichmäßig beschleunigten Bewegung berechnet und als Geschwindigkeit verwendet.

- (d) **Welche Befehle muß man in welcher Weise ändern, um ein realistischeres Hüpf-Verhalten zu bekommen?**

Da der Ball am Boden abprallen soll, ist es *nicht* sinnvoll, die y -Komponente der Geschwindigkeit über die bekannte physikalische Formel $v_y = -g \cdot t$ für die Geschwindigkeit in einer gleichmäßig beschleunigten Bewegung zu berechnen.

Stattdessen ist es sinnvoll, die *Geschwindigkeitsänderung* innerhalb des Zeitintervalls dt zur Geschwindigkeitskomponente zu addieren: `vy += g * dt;`

Auch dies erkennt man sehr schnell durch Vergleich mit dem Beispiel-Programm [gtk-13.c](#).

```

#include <gtk/gtk.h>

#define WIDTH 320
#define HEIGHT 240

double t = 0.0;
double dt = 0.2;

int r = 5;

double x = 10;
double y = 200;
double vx = 20;
double vy = -60;
double g = 9.81;

gboolean draw (GtkWidget *widget, cairo_t *c, gpointer data)
{
    GdkRGBA blue = { 0.0, 0.5, 1.0, 1.0 };

    gdk_cairo_set_source_rgba (c, &blue);
    cairo_arc (c, x, y, r, 0, 2 * G_PI);
    cairo_fill (c);

    return FALSE;
}

gboolean timer (GtkWidget *widget)
{
    t += dt;
    x += vx * dt;
    y += vy * dt;
    vx = vx;
    vy = 0.5 * g * (t * t);
    if (y + r >= HEIGHT)
        vy = -vy * 0.9;
    if (x + r >= WIDTH)
        vx = -vx * 0.9;
    if (x - r <= 0)
        vx = -vx * 0.9;
    gtk_widget_queue_draw_area (widget, 0, 0, WIDTH, HEIGHT);
    g_timeout_add (50, (GSourceFunc) timer, widget);
    return FALSE;
}

int main (int argc, char **argv)
{
    gtk_init (&argc, &argv);

    GtkWidget *window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_widget_show (window);
    gtk_window_set_title (GTK_WINDOW (window), "Hello");
    g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);

    GtkWidget *drawing_area = gtk_drawing_area_new ();
    gtk_widget_show (drawing_area);
    gtk_container_add (GTK_CONTAINER (window), drawing_area);
    gtk_widget_set_size_request (drawing_area, WIDTH, HEIGHT);

    gtk_main ();
    return 0;
}

```