

Hardwarenahe Programmierung

Übungsaufgaben – 9. Januar 2023

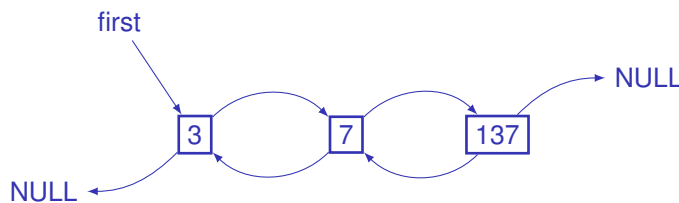
Diese Übung enthält Punkteangaben wie in einer Klausur. Um zu „bestehen“, müssen Sie innerhalb von 120 Minuten unter Verwendung ausschließlich zugelassener Hilfsmittel 20 Punkte (von insgesamt 41) erreichen.

Aufgabe 1: Einfach und doppelt verkettete Listen

Das Beispiel-Programm `aufgabe-1.c` demonstriert zwei Funktionen zur Verwaltung einfach verketteter Listen: `output_list()` zum Ausgeben der Liste auf den Bildschirm und `insert_into_list()` zum Einfügen in die Liste.

- (a) Ergänzen Sie eine Funktion `delete_from_list()` zum Löschen eines Elements aus der Liste mit Freigabe des Speicherplatzes. (5 Punkte)
- (b) Ergänzen Sie eine Funktion `reverse_list()` die die Reihenfolge der Elemente in der Liste umdreht. (3 Punkte)

Eine doppelt verkettete Liste hat in jedem Knotenpunkt (`node`) *zwei* Zeiger – einen auf das nächste Element (`next`) und einen auf das vorherige Element (z. B. `prev` für „previous“). Dadurch ist es leichter als bei einer einfach verketteten Liste, die Liste in umgekehrter Reihenfolge durchzugehen.



Der Rückwärts-Zeiger (`prev`) des ersten Elements zeigt, genau wie der Vorwärts-Zeiger (`next`) des letzten Elements, auf *nichts*, hat also den Wert `NULL`.

- (c) Schreiben Sie das Programm um für doppelt verkettete Listen. (5 Punkte)

Aufgabe 2: Ternärer Baum

Der in der Vorlesung vorgestellte *binäre Baum* ist nur ein Spezialfall; im allgemeinen können Bäume auch mehr als zwei Verzweigungen pro Knotenpunkt haben. Dies ist nützlich bei der Konstruktion *balancierter Bäume*, also solcher, die auch im *Worst Case* nicht zu einer linearen Liste entarten, sondern stets eine – möglichst flache – Baumstruktur behalten.

Wir betrachten einen Baum mit bis zu drei Verzweigungen pro Knotenpunkt, einen sog. *ternären Baum*. Jeder Knoten enthält dann nicht nur einen, sondern *zwei* Werte als Inhalt:

```
typedef struct node
{
    int content_left, content_right;
    struct node *left, *middle, *right;
} node;
```

Wir konstruieren nun einen Baum nach folgenden Regeln:

- Innerhalb eines Knotens sind die Werte sortiert: `content_left` muß stets kleiner sein als `content_right`.
- Der Zeiger `left` zeigt auf Knoten, deren enthaltene Werte durchweg kleiner sind als `content_left`.
- Der Zeiger `right` zeigt auf Knoten, deren enthaltene Werte durchweg größer sind als `content_right`.
- Der Zeiger `middle` zeigt auf Knoten, deren enthaltene Werte durchweg größer sind als `content_left`, aber kleiner als `content_right`.
- Ein Knoten muß nicht immer mit zwei Werten voll besetzt sein; er darf auch *nur einen* gültigen Wert enthalten.

Der Einfachheit halber lassen wir in diesem Beispiel nur positive Zahlen als Werte zu. Wenn ein Knoten nur einen Wert enthält, setzen wir `content_right = -1`, und der Zeiger `middle` wird nicht verwendet.

- Wenn wir neue Werte in den Baum einfügen, werden *zuerst* die nicht voll besetzten Knoten aufgefüllt und *danach erst* neue Knoten angelegt und Zeiger gesetzt.
- Beim Auffüllen eines Knotens darf nötigenfalls `content_left` nach `content_right` verschoben werden. Ansonsten werden einmal angelegte Knoten nicht mehr verändert.

(In der Praxis dürfen Knoten gemäß speziellen Regeln nachträglich verändert werden, um Entartungen gar nicht erst entstehen zu lassen – siehe z. B. https://en.wikipedia.org/wiki/2-3_tree.)

- Zeichnen Sie ein Schaubild, das veranschaulicht, wie die Zahlen 7, 137, 3, 5, 6, 42, 1, 2 und 12 nacheinander und in dieser Reihenfolge in den oben beschriebenen Baum eingefügt werden – analog zu den Vortragsfolien (<hp-20221219.pdf>), Seite 33. (3 Punkte)
- Dasselbe, aber in der Reihenfolge 2, 7, 42, 12, 1, 137, 5, 6, 3. (3 Punkte)
- Beschreiben Sie in Worten und/oder als C-Quelltext-Fragment, wie eine Funktion aussehen müßte, um den auf diese Weise entstandenen Baum sortiert auszugeben. (4 Punkte)

Aufgabe 3: Fehlerhaftes Programm: Hüpfender Ball

Das auf der nächsten Seite abgedruckte GTK+-Programm (Datei: [aufgabe-3.c](#)) soll einen hüpfenden Ball darstellen, ist jedoch fehlerhaft.

- Warum sieht man lediglich ein leeres Fenster? Welchen Befehl muß man ergänzen, um diesen Fehler zu beheben? (3 Punkte)
- 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)
- Erklären Sie das merkwürdige Hüpfverhalten des Balls. Wie kommt es zustande? Was an diesem Verhalten ist korrekt, und was ist fehlerhaft? (5 Punkte)
- 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.

```

#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;
}

```

Aufgabe 4: Hexdumps

Das folgende Programm ([aufgabe-4.c](#)) liest einen String ein und gibt die ASCII-Werte der Buchstaben hexadezimal aus. (Anders als z. B. `scanf()` akzeptiert die Funktion `fgets()` zum Lesen von Strings auch Leerzeichen, und sie vermeidet Pufferüberläufe.)

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5      char buffer[100];
6      fgets (buffer, 100, stdin);
7      for (char *p = buffer; *p; p++)
8          printf ("%02x", *p);
9      printf ("\n");
10 }
```

Beispiel: Bei der Eingabe von `Dies ist ein Test.` erscheint die Ausgabe
`44696573206973742065696e20546573742e0a.`

Schreiben Sie ein Programm, das diese Umwandlung in umgekehrter Richtung vornimmt, also z. B. bei Eingabe von `44696573206973742065696e20546573742e0a` wieder `Dies ist ein Test.` ausgibt.

(6 Punkte)

Hinweis für die Klausur: Abgabe in digitaler Form ist erwünscht, aber nicht zwingend.

Viel Erfolg – auch in den Prüfungen!