

Hardwarenahe Programmierung / Angewandte Informatik

Übungsaufgaben – 15. Januar 2018

Diese Übung enthält Punkteangaben wie in einer Klausur. Um zu „bestehen“, müssen Sie innerhalb von 80 Minuten unter Verwendung ausschließlich zugelassener Hilfsmittel 13 Punkte (von insgesamt 27) erreichen.

Aufgabe 1: Objektorientierte Programmierung mit dem C-Datentyp `union`

Aus der Vorlesung ist der Verbund-Datentyp `union` bekannt:

```
typedef union
{
    int number;
    char *name;
    uint8_t bytes[4];
} data;
```

Von der Syntax her entspricht die `union` genau dem `struct`. Insbesondere lassen sich die Datenfelder über einen Punkt ansprechen, bei Zeigern auch mit `→`. Der Unterschied zum `struct` besteht darin, daß sich in einer `union` die Datenfelder *dieselben Speicherzellen teilen*. Alle Datenfelder einer `union` haben *dieselbe Speicheradresse*. (In einem `struct` beginnt das nächste Datenfeld immer dort, wo das vorherige aufhört.)

Bei Zuweisung eines Wertes an *ein* Datenfeld einer `union` ändern sich daher die Werte *aller* Datenfelder. Dies kann man nutzen, um Speicher zu sparen (wenn man immer nur eines der Datenfelder auf einmal nutzen möchte) oder um „verwandte“ Datentypen zu konstruieren (im Sinne der objektorientierten Programmierung) oder um die Byte-Muster von Variablen zu untersuchen.

- (a) Was bewirkt das folgende Programm ([aufgabe-1a.c](#)), wenn es auf einem LittleEndian-Rechner ausgeführt wird, und warum? (4 Punkte)

```
#include <stdio.h>
#include <stdint.h>

typedef union
{
    uint32_t number;
    char *name;
    uint8_t bytes[4];
} data;

int main (void)
{
    data x;
    x.number = 303108111;
    for (int i = 0; i < 4; i++)
        printf ("%d_", x.bytes[i]);
    printf ("\n");
    printf ("%s\n", x.name);
    return 0;
}
```

- (b) Was würde dasselbe Programm auf einem BigEndian-Rechner ausgeben? (1 Punkt)

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

```
#include <stdio.h>

#define POINT 0
#define CIRCLE 1
#define TEXT 2

typedef union
{
    int radius;
    char *text;
} extra_data;

typedef struct graphics_object
{
    int type;
    void (*draw) (struct graphics_object *this);
    int x, y;
    extra_data data;
} graphics_object;

void draw_point (struct graphics_object *this)
{
    printf ("point_at_(%d,%d)\n", this->x, this->y);
}

void draw_circle (struct graphics_object *this)
{
    printf ("circle_at_(%d,%d)_with_radius_%d\n",
        this->x, this->y, this->data.radius);
}

void draw_text (struct graphics_object *this)
{
    printf ("text_at_(%d,%d):_\"%s\"\n",
        this->x, this->y, this->data.text);
}

int main (void)
{
    graphics_object a_point = { POINT, draw_point, 35, 17 };
    graphics_object a_circle = { CIRCLE, draw_circle, 20, 30 };
    a_circle.data.radius = 12;
    graphics_object some_text = { TEXT, draw_text, 42, 23 };
    some_text.data.text = "Hello,_world!";
    graphics_object *g[3] = { &a_point, &a_circle, &some_text };
    for (int i = 0; i < 3; i++)
        g[i]->draw (g[i]);
    return 0;
}
```

Dieses Programm verwaltet verschiedenartige Grafikobjekte in einem Array `graphics_object *g[3]`. Anstatt tatsächlich zu zeichnen, gibt es der Einfachheit halber als Text aus, was ggf. zu sehen wäre.

Der Datentyp `graphics_object` ist ein **struct**, in dem eins der Datenfelder eine **union** (mit weiteren Datenfeldern) ist.

- (c) Beschreiben Sie (in Worten, evtl. mit Skizze) die in diesem Programm realisierte Objekt-Hierarchie. Worum handelt es sich insbesondere bei `void (*draw) (struct graphics_object *this)`? (3 Punkte)
 - (d) Was passiert, wenn man im Hauptprogramm `main()` die an `a_circle` übergebene Funktion `draw_circle` durch `draw_text` ersetzt ([aufgabe-1d.c](#)) und warum? (2 Punkte)
 - (e) Erweitern Sie das Programm so, daß es einen weiteren Grafikobjekttyp `SQUARE` kennt, bei dem man eine Kantenlänge `a` angibt und das beim „Zeichnen“ den Text „square at (x,y) with edge length a“ (mit den korrekten Zahlen anstelle von `x`, `y` und `a`) ausgibt. (5 Punkte)
- (Hinweis für die Klausur: Abgabe auf Datenträger ist erlaubt und erwünscht, aber nicht zwingend.)

Aufgabe 2: Objektorientierte Tier-Datenbank

Diese Übung ist eine Ergänzung zu Aufgabe 3 der Übungen vom 8. 1. 2018 ([hp-uebung-20180108.pdf](#)).

Wir betrachten das korrigierte Programm (wahlweise Ihre eigene Lösung von Teilaufgabe (d) oder eine der Musterlösungen [loesung-3-1.c](#) oder [loesung-3-2.c](#)).

- (e) Schreiben Sie das Programm so um, daß es keine expliziten Typumwandlungen mehr benötigt.

Hinweis: Verwenden Sie `union`. (4 Punkte)

- (f) Schreiben Sie das Programm weiter um, so daß es die Objektinstanzen `duck` und `cow` dynamisch erzeugt.

Hinweis: Verwenden Sie `malloc()` und schreiben Sie Konstruktoren. (4 Punkte)

- (g) Schreiben Sie das Programm weiter um, so daß die Ausgabe nicht mehr direkt im Hauptprogramm erfolgt, sondern stattdessen eine virtuelle Methode `print()` aufgerufen wird.

Hinweis: Verwenden Sie in den Objekten Zeiger auf Funktionen, und initialisieren Sie diese in den Konstruktoren. (4 Punkte)

Viel Erfolg!