

# Hardwarenahe Programmierung

## Übungsaufgaben – 14. Januar 2019

Diese Übung enthält Punkteangaben wie in einer Klausur. Um zu „bestehen“, müssen Sie innerhalb von 90 Minuten unter Verwendung ausschließlich zugelassener Hilfsmittel 16 Punkte (von insgesamt 32) erreichen.

### Aufgabe 1: Iterationsfunktionen

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

```
#include <stdio.h>

void foreach (int *a, void (*fun) (int x))
{
    for (int *p = a; *p >= 0; p++)
        fun (*p);
}

void even_or_odd (int x)
{
    if (x % 2)
        printf ("%d_ist_ungerade.\n", x);
    else
        printf ("%d_ist_gerade.\n", x);
}
```

```
int main (void)
{
    int numbers[] = { 12, 17, 32, 1, 3, 16, 19, 18, -1 };
    foreach (numbers, even_or_odd);
    return 0;
}
```

- (a) Was bedeutet **void (\*fun) (int x)**, und welchen Sinn hat seine Verwendung in der Funktion **foreach()**? (2 Punkte)
- (b) Schreiben Sie das Hauptprogramm **main()** so um, daß es unter Verwendung der Funktion **foreach()** die Summe aller positiven Zahlen in dem Array berechnet. Sie dürfen dabei weitere Funktionen sowie globale Variable einführen. (4 Punkte)

### Aufgabe 2: Objektorientierte Tier-Datenbank

Das auf der nächsten Seite in Blau dargestellte Programm (Datei: [aufgabe-2a.c](#)) soll Daten von Tieren verwalten.

Beim Compilieren erscheinen die folgende Fehlermeldungen:

```
$ gcc -std=c99 -Wall -O aufgabe-2a.c -o aufgabe-2a
aufgabe-2a.c: In function 'main':
aufgabe-2a.c:31: error: 'animal' has no member named 'wings'
aufgabe-2a.c:37: error: 'animal' has no member named 'legs'
```

Der Programmierer nimmt die auf der nächsten Seite in Rot dargestellten Ersetzungen vor (Datei: [aufgabe-2b.c](#)). Daraufhin gelingt das Compilieren, und die Ausgabe des Programms lautet:

```
$ gcc -std=c99 -Wall -O aufgabe-2b.c -o aufgabe-2b
$ ./aufgabe-2b
A duck has 2 legs.
Error in animal: cow
```

- (a) Erklären Sie die o. a. Compiler-Fehlermeldungen. (2 Punkte)
- (b) Wieso verschwinden die Fehlermeldungen nach den o. a. Ersetzungen? (3 Punkte)
- (c) Erklären Sie die Ausgabe des Programms. (5 Punkte)
- (d) Beschreiben Sie – in Worten und/oder als C-Quelltext – einen Weg, das Programm so zu berichtigen, daß es die Eingabedaten ("A duck has 2 wings. A cow has 4 legs.") korrekt speichert und ausgibt. (4 Punkte)
- (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)

```
#include <stdio.h>

#define ANIMAL 0
#define WITH_WINGS 1
#define WITH_LEGS 2

typedef struct animal
{
    int type;
    char *name;
} animal;

typedef struct with_wings
{
    int wings;
} with_wings;

typedef struct with_legs
{
    int legs;
} with_legs;

int main (void)
{
    animal *a[2];

    animal duck;
    a[0] = &duck;
    a[0] -> type = WITH_WINGS;
    a[0] -> name = "duck";
    a[0] -> wings = 2;  ← ((with_wings *) a[0]) -> wings = 2;

    animal cow;
    a[1] = &cow;
    a[1] -> type = WITH_LEGS;
    a[1] -> name = "cow";
    a[1] -> legs = 4;  ← ((with_legs *) a[1]) -> legs = 4;

    for (int i = 0; i < 2; i++)
        if (a[i] -> type == WITH_LEGS)
            printf ("A_%s_has_%d_legs.\n", a[i] -> name,
                    ((with_legs *) a[i]) -> legs);
        else if (a[i] -> type == WITH_WINGS)
            printf ("A_%s_has_%d_wings.\n", a[i] -> name,
                    ((with_wings *) a[i]) -> wings);
        else
            printf ("Error_in_animal:_%s\n", a[i] -> name);

    return 0;
}
```

*Viel Erfolg!*