

Hardwarenahe Programmierung

Übungsaufgaben – 7. Januar 2019

Diese Übung enthält Punkteangaben wie in einer Klausur. Um zu „bestehen“, müssen Sie innerhalb von 100 Minuten unter Verwendung ausschließlich zugelassener Hilfsmittel 18 Punkte (von insgesamt 36) erreichen.

Aufgabe 1: Speicherformate von Zahlen

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

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  typedef struct
5  {
6      uint32_t a;
7      uint64_t b;
8      uint8_t c;
9  } three_numbers;
10
11 int main (void)
12 {
13     three_numbers xyz = { 1819042120, 2410670883059281007, 0 };
14     printf ("%s\n", &xyz);
15     return 0;
16 }
```

Das Programm wird für einen 32-Bit-Rechner kompiliert und ausgeführt.

(Die `gcc`-Option `-m32` sorgt dafür, daß `gcc` Code für einen 32-Bit-Prozessor erzeugt.)

```
$ gcc -Wall -m32 aufgabe-2.c -o aufgabe-2
aufgabe-2.c: In function "main":
aufgabe-2.c:14:13: warning: format "%s" expects argument of type "char *", but
argument 2 has type "three_numbers * {aka struct <anonymous> *}" [-Wformat=]
    printf ("%s\n", &xyz);
                   ^
$ ./aufgabe-2
Hallo, Welt!
```

- (a) Erklären Sie die beim Compilieren auftretende Warnung. (2 Punkte)
- (b) Erklären Sie die Ausgabe des Programms. (4 Punkte)
- (c) Welche Endianness hat der verwendete Rechner? Wie sähe die Ausgabe auf einem Rechner mit entgegengesetzter Endianness aus? (2 Punkte)
- (d) Dasselbe Programm wird nun für einen 64-Bit-Rechner kompiliert und ausgeführt.
(Die `gcc`-Option `-m64` sorgt dafür, daß `gcc` Code für einen 64-Bit-Prozessor erzeugt.)

```
$ gcc -Wall -m64 aufgabe-2.c -o aufgabe-2
aufgabe-2.c: In function "main":
aufgabe-2.c:14:13: warning: format "%s" expects argument of type "char *",
but argument 2 has type "three_numbers * {aka struct <anonymous> *}"
[-Wformat=]
    printf ("%s\n", &xyz);
                   ^
$ ./aufgabe-2
Hall15V
```

(Es ist möglich, daß die konkrete Ausgabe auf Ihrem Rechner anders aussieht.)

Erklären Sie die geänderte Ausgabe des Programms. (3 Punkte)

Aufgabe 2: Zeigerarithmetik

Wir betrachten das folgende Programm ([aufgabe-2.c](#)):

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

void output (uint16_t *a)
{
    for (int i = 0; a[i]; i++)
        printf ("%d", a[i]);
    printf ("\n");
}

int main (void)
{
    uint16_t prime_numbers[] = { 2, 3, 5, 7, 11, 13, 17, 0 };

    uint16_t *p1 = prime_numbers;
    output (p1);
    p1++;
    output (p1);

    char *p2 = prime_numbers;
    output (p2);
    p2++;
    output (p2);

    return 0;
}
```

Das Programm wird kompiliert und ausgeführt:

```
$ gcc -Wall aufgabe-2.c -o aufgabe-2
aufgabe-2.c: In function 'main':
aufgabe-2.c:20:13: warning: initialization from
                  incompatible pointer type [enabled by default]
aufgabe-2.c:21:3: warning: passing argument 1 of 'output' from
                  incompatible pointer type [enabled by default]
aufgabe-2.c:4:6: note: expected 'uint16_t *' but argument is of type 'char *'
aufgabe-2.c:23:3: warning: passing argument 1 of 'output' from
                  incompatible pointer type [enabled by default]
aufgabe-2.c:4:6: note: expected 'uint16_t *' but argument is of type 'char *'
$ ./aufgabe-2
2 3 5 7 11 13 17
3 5 7 11 13 17
2 3 5 7 11 13 17
768 1280 1792 2816 3328 4352
```

- (a) Erklären Sie die Funktionsweise der Funktion `output ()`. (2 Punkte)
- (b) Begründen Sie den Unterschied zwischen der ersten (2 3 5 7 11 13 17) und der zweiten Zeile (3 5 7 11 13 17) der Ausgabe des Programms. (2 Punkte)
- (c) Erklären Sie die beim Compilieren auftretenden Warnungen und die dritte Zeile (2 3 5 7 11 13 17) der Ausgabe des Programms. (3 Punkte)
- (d) Erklären Sie die vierte Zeile (768 1280 1792 2816 3328 4352) der Ausgabe des Programms. Sie dürfen einen Little-Endian-Rechner voraussetzen. (4 Punkte)

Aufgabe 3: Objektorientierte Tier-Datenbank

```
#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;
}
```

Das oben in Blau dargestellte Programm (Datei: [aufgabe-3a.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 oben in Rot dargestellten Ersetzungen vor (Datei: [aufgabe-3b.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)

Viel Erfolg!