

Hardwarenahe Programmierung / Angewandte Informatik

Übungsaufgaben – 12. Dezember 2016

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 28) erreichen.

Aufgabe 1: Daten im Speicher

Das folgende C-Programm `aufgabe-1.c` gibt den Speicherbereich, in dem sich seine Variablen befinden, als eine Folge von 8-Bit-Zahlen aus:

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

int16_t a = -1;
int32_t b = 8320;

int main (void)
{
    uint8_t *p = &a;
    for (int i = 0; i < 8; i++)
        printf ("%d", p[i]);
    printf ("\n");
}
```

Das Programm wird ohne Optimierung auf einem 32-Bit-Rechner kompiliert (mit Warnung) und gestartet:

```
$ gcc -std=c99 -Wall aufgabe-1.c -o aufgabe-1
aufgabe-1.c: In function "main":
aufgabe-1.c:9:16: warning: initialization from incompatible pointer type [...]
$ ./aufgabe-1
255 255 0 0 128 32 0 0
```

- (a) Welche Endianness hat der verwendete Rechner und warum? (2 Punkte)
- (b) Erklären Sie die ausgegebenen Zahlen.
Zu welcher Variablen gehört jeweils die Zahl? (4 Punkte)
- (c) Wie würde die Ausgabe auf einem 16-Bit-Rechner mit entgegengesetzter Endianness lauten und warum? (3 Punkte)
- (x) **Freiwillige Zusatzaufgabe:** Erklären Sie, was sich ändert und warum, wenn das Programm auf einem 32-Bit-Rechner *mit* Optimierung (`-O`) kompiliert wird. (5 Extrapunkte)

Hinweis: Aus anderen Lehrveranstaltungen sollte Ihnen der Begriff des Zweierkomplements bekannt sein. Falls nicht, gilt eine Recherche nach diesem Begriff als zugelassenes Hilfsmittel.

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 -std=c99 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: XBM-Grafik

Bei einer XBM-Grafikdatei handelt es sich um ein als C-Quelltext abgespeichertes Array, das die Bildinformationen enthält:

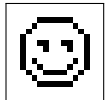
- Jedes Bit entspricht einem Pixel.
- Nullen stehen für Weiß, Einsen für Schwarz.
- LSB first.
- Jede Zeile des Bildes wird auf ganze Bytes aufgefüllt.
- Breite und Höhe des Bildes sind als Konstantendefinitionen (**#define**) in der Datei enthalten.

Sie können eine XBM-Datei sowohl mit einem Texteditor als auch mit vielen Grafikprogrammen öffnen und bearbeiten.

Beispiel ([aufgabe-3.xbm](#)):

```
#define aufgabe_3_width 14
#define aufgabe_3_height 14
static unsigned char aufgabe_3_bits[] = {
    0x00, 0x00, 0xf0, 0x03, 0x08, 0x04, 0x04, 0x08, 0x02, 0x10, 0x32, 0x13,
    0x22, 0x12, 0x02, 0x10, 0x0a, 0x14, 0x12, 0x12, 0xe4, 0x09, 0x08, 0x04,
    0xf0, 0x03, 0x00, 0x00 };

```



Ein C-Programm, das eine XBM-Grafik nutzen will, kann die [.xbm](#)-Datei mit **#include** `"..."` direkt einbinden.

Schreiben Sie ein Programm, das die XBM-Datei als ASCII-Grafik ausgibt, z. B.:

```

  * * * * *
 *           *
*           *
*   *   *   *
*   *   *   *
*   *   *   *
*   *   *   *
*   *   *   *
*   *   *   *
*   *   *   *
*   *   *   *
*   *   *   *
  * * * * *
```

(8 Punkte)

(Hinweis für die Klausur: Abgabe auf Datenträger ist erlaubt und erwünscht, aber nicht zwingend.)

Viel Erfolg!