

Hardwarenahe Programmierung

Musterlösung zu den Übungsaufgaben 9 – 19. Dezember 2024

Aufgabe 1: PBM-Grafik

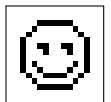
Bei einer PBM-Grafikdatei handelt es sich um ein abgespeichertes C-Array von Bytes (`uint8_t`), das die Bildinformationen enthält:

- Die Datei beginnt mit der Kennung `P4`, danach folgen Breite und Höhe in Pixel als ASCII-Zahlen, danach ein Trennzeichen und die eigentlichen Bilddaten.
- Jedes Bit entspricht einem Pixel.
- Nullen stehen für Weiß, Einsen für Schwarz.
- MSB first.
- Jede Zeile des Bildes wird auf ganze Bytes aufgefüllt.

Viele Grafikprogramme können PBM-Dateien öffnen und bearbeiten. Der Anfang der Datei (Kennung, Breite und Höhe) ist auch in einem Texteditor lesbar.

Beispiel ([aufgabe-1.pbm](#)):

```
P4
14 14
<Bilddaten>
```



In dem untenstehenden Programmfragment ([aufgabe-1.c](#)) wird eine Grafik aus Textzeilen zusammengesetzt, so daß man mit einem Texteditor „malen“ kann:

```
#include <stdio.h>

/* ... */

int main (void)
{
    pbm_open (14, 14, "test.pbm");
    pbm_line ("                ");
    pbm_line ("        XXXXXX        ");
    pbm_line ("    _X_          _X_   ");
    pbm_line ("  _X_          _X_   ");
    pbm_line (" _X_          _X_   ");
    pbm_line ("X_          _X_   ");
    pbm_line (" _X_ XX_ XX_ _X_   ");
    pbm_line (" _X_ _X_ _X_ _X_   ");
    pbm_line (" _X_          _X_   ");
    pbm_line (" _X_X_          _X_   ");
    pbm_line (" _X_ _X_          _X_   ");
    pbm_line (" _X_ XXXX_ _X_   ");
    pbm_line ("    _X_          _X_   ");
    pbm_line ("    XXXXXX        ");
    pbm_line ("                ");
    pbm_close ();
    return 0;
}
```

Ergänzen Sie das Programmfragment so, daß es eine Datei `test.pbm` erzeugt, die die Grafik enthält.

Das Programm soll typische Benutzerfehler abfangen (z. B. weniger Zeilen als in `pbm_open` angegeben), keine fehlerhaften Ausgaben produzieren oder abstürzen, sondern stattdessen sinnvolle Fehlermeldungen ausgeben.

Zum Vergleich liegt eine Datei [aufgabe-1.pbm](#) mit dem gewünschten Ergebnis bei, und die Datei [aufgabe-1.png](#) enthält dasselbe Bild.

(10 Punkte)

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

Lösung

Die Datei [loesung-1.c](#) enthält eine richtige Lösung. Man beachte die Aufrufe der Funktion [error\(\)](#) im Falle von falscher Benutzung der Bibliotheksfunktionen. Weitere Erklärungen finden Sie als Kommentare im Quelltext.

Die Datei [loesung-1f.c](#) enthält eine falsche Lösung. (Beide Dateien unterscheiden sich nur in Zeile 46.) Dieses Programm speichert die Bits in den Bytes von rechts nach links (LSB first). Richtig wäre von links nach rechts (MSB first). Das erzeugte Bild ist dementsprechend fehlerhaft.

Aufgabe 2: Fakultät

Die Fakultät $n!$ einer ganzen Zahl $n \geq 0$ ist definiert als:

$$\begin{aligned} &1 \quad \text{für } n = 0, \\ &n \cdot (n - 1)! \quad \text{für } n > 0. \end{aligned}$$

Mit anderen Worten: $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$.

Die folgende Funktion [fak\(\)](#) berechnet die Fakultät *rekursiv* (Datei: [aufgabe-2.c](#)):

```
int fak (int n)
{
    if (n <= 0)
        return 1;
    else
        return n * fak (n - 1);
}
```

- (a) Schreiben Sie eine Funktion, die die Fakultät *iterativ* berechnet, d. h. mit Hilfe einer Schleife anstelle von Rekursion. (3 Punkte)
- (b) Wie viele Multiplikationen (Landau-Symbol) erfordern beide Versionen der Fakultätsfunktion in Abhängigkeit von n ? Begründen Sie Ihre Antwort. (2 Punkte)
- (c) Wieviel Speicherplatz (Landau-Symbol) erfordern beide Versionen der Fakultätsfunktion in Abhängigkeit von n ? Begründen Sie Ihre Antwort. (3 Punkte)

Lösung

- (a) **Schreiben Sie eine Funktion, die die Fakultät *iterativ* berechnet, d. h. mit Hilfe einer Schleife anstelle von Rekursion.**

Datei: [loesung-2.c](#)

```
int fak (int n)
{
    int f = 1;
    for (int i = 2; i <= n; i++)
        f *= i;
    return f;
}
```

- (b) **Wie viele Multiplikationen (Landau-Symbol) erfordern beide Versionen der Fakultätsfunktion in Abhängigkeit von n ? Begründen Sie Ihre Antwort.**

In beiden Fällen werden n Zahlen miteinander multipliziert – oder $n - 1$, wenn man Multiplikationen mit 1 ausspart. In jedem Fall hängt die Anzahl der Multiplikationen linear von n ab; es sind $\mathcal{O}(n)$ Multiplikationen. Insbesondere arbeiten also beide Versionen gleich schnell.

- (c) **Wieviel Speicherplatz (Landau-Symbol) erfordern beide Versionen der Fakultätsfunktion in Abhängigkeit von n ? Begründen Sie Ihre Antwort.**

Die iterative Version der Funktion benötigt 2 Variable vom Typ [int](#), nämlich n und f . Dies ist eine konstante Zahl; der Speicherplatzverbrauch ist daher $\mathcal{O}(1)$.

Die rekursive Version der Funktion erzeugt jedesmal, wenn sie sich selbst aufruft, eine zusätzliche Variable n . Es sind $n + 1$ Aufrufe; die Anzahl der Variablen n hängt linear von n ab; der Speicherplatzverbrauch ist also $\mathcal{O}(n)$.

Aufgabe 3: Speicherformate von Zahlen

Wir betrachten das folgende Programm (aufgabe-3.c):

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

typedef struct
{
    uint32_t a;
    uint64_t b;
    uint8_t c;
} three_numbers;

int main (void)
{
    three_numbers xyz = { 1819042120, 2410670883059281007, 0 };
    printf ("%s\n", &xyz);
    return 0;
}
```

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)

Lösung

- (a) **Erklären Sie die beim Compilieren auftretende Warnung.**

Die Funktion `printf()` mit der Formatspezifikation `%s` erwartet als Parameter einen String, d. h. einen Zeiger auf `char`. Die Adresse (`&`) der Variablen `xyz` ist zwar ein Zeiger, aber nicht auf `char`, sondern auf einen `struct` vom Typ `three_numbers`. Eine implizite Umwandlung des Zeigertyps ist zwar möglich, aber normalerweise nicht das, was man beabsichtigt.

Der Inhalt der Füll-Bytes ist undefiniert. Im Beispiel aus der Aufgabenstellung entsteht hier die Ausgabe 5v, was den (zufälligen) hexadezimalen Werten 35 56 entspricht:

48	61	6C	6C	35	56	00	??	6F	2C	20	57	65	6C	74	21	00
a				Füll-Bytes				b				c				

Da danach die Ausgabe aufhört, muß an der nächsten Stelle ein Null-Symbol stehen, das das Ende des Strings anzeigt. Der Inhalt der darauf folgenden Speicherzelle bleibt unbekannt.