

Hardwarenahe Programmierung

Musterlösung zu den Übungsaufgaben – 30. November 2023

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 ("      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)

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 $\mathcal{O}(n)$.