

# Hardwarenahe Programmierung / Angewandte Informatik

## Musterlösung zu den Übungsaufgaben – 14. November 2016

### Aufgabe 1: 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-1.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.
- (b) Wie viele Multiplikationen (Landau-Symbol) erfordern beide Versionen der Fakultätsfunktion?
- (c) Wieviel Speicherplatz (Landau-Symbol) erfordern beide Versionen der Fakultätsfunktion?

### Lösung

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

Datei: `loesung-1.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 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?**

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 2: Strings

Wir betrachten nochmals die Funktion aus der vorherigen Übung (Datei: [aufgabe-2.c](#)):

```
int fun_1 (char *s1, char *s2)
{
    int result = 1;
    for (int i = 0; s1[i] && s2[i]; i++)
        if (s1[i] != s2[i])
            result = 0;
    return result;
}
```

- (e) Von welcher Ordnung (Landau-Symbol) ist die Funktion `fun_1()` hinsichtlich der Anzahl ihrer Zugriffe auf die Zeichen in den Strings – und warum?
- (f) Von welcher Ordnung (Landau-Symbol) ist die von Ihnen in Aufgabenteil (d) geschriebene effizientere Version der Funktion?

### Lösung

- (e) **Von welcher Ordnung (Landau-Symbol) ist die Funktion `fun_1()` hinsichtlich der Anzahl ihrer Zugriffe auf die Zeichen in den Strings – und warum?**

Die Funktion vergleicht *in einer einzigen Schleife* jeweils zwei Zeichen der Strings bis hin zur Länge  $n$  des kleineren Strings. Die Ordnung ist daher  $\mathcal{O}(n)$ .

- (f) **Von welcher Ordnung (Landau-Symbol) ist die von Ihnen in Aufgabenteil (d) geschriebene effizientere Version der Funktion?**

Wenn die Strings überhaupt nicht übereinstimmen, bricht die effizientere Version der Funktion unmittelbar ab; im günstigsten Fall beträgt die Ordnung also  $\mathcal{O}(1)$ .

Wenn die Strings bis zur Länge  $n$  des kleineren Strings übereinstimmen, vergleicht die Funktion weiterhin alle Zeichen der Strings. Im ungünstigsten Fall beträgt die Ordnung daher weiterhin  $\mathcal{O}(n)$ .

## Aufgabe 3: Text-Grafik-Bibliothek

Ergänzen Sie die Text-Grafik-Bibliothek aus der letzten Übung um eine weitere Funktion:

- **`void fill (int x, int y, char c, char o)`**  
Fläche in der „Farbe“ `o`, die den Punkt `(x, y)` enthält, mit der „Farbe“ `c` ausmalen

Hinweise:

- Führen Sie eine Web-Recherche nach dem Begriff „Floodfill“ durch.
- Schreiben Sie ein Test-Programm, das interessante umrandete Flächen – auch mit „Loch“ – „ausmalt“.

### Lösung

Siehe die Dateien [textgraph.c](#) und [textgraph.h](#) (Bibliothek) sowie [test-textgraph.c](#) (Test-Programm).

Die Musterlösung folgt i. w. dem Algorithmus `fill4()` der Webseite <https://de.wikipedia.org/wiki/Floodfill>.

Ohne das `if` geriete die Funktion `fill()` in eine unendliche Rekursion, was zu einem Überlauf des CPU-Stacks und damit zu einem Absturz führen würde.

Wenn die umrandende Kontur nicht abgeschlossen wäre, würde `fill()` „ausbrechen“ und auf dem gesamten Bildschirm alle Punkte der Farbe `o` mit der Farbe `c` übermalen. Aus diesem Grund ist der „Smiley“ etwas eckiger als im Test-Programm zur vorherigen Übung.