

Hardwarenahe Programmierung / Angewandte Informatik

Übungsaufgaben – 18. Dezember 2017

Diese Übung enthält Punkteangaben wie in einer Klausur. Um zu „bestehen“, müssen Sie innerhalb von 100 Minuten unter Verwendung ausschließlich zugelassener Hilfsmittel 16 Punkte (von insgesamt 34) erreichen.

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. (3 Punkte)
- (b) Wie viele Multiplikationen (Landau-Symbol) erfordern beide Versionen der Fakultätsfunktion in Abhängigkeit von n ? (2 Punkte)
- (c) Wieviel Speicherplatz (Landau-Symbol) erfordern beide Versionen der Fakultätsfunktion in Abhängigkeit von n ? (3 Punkte)

Aufgabe 2: Arrays mit Zahlen

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

```
#include <stdio.h>

void f (int *s0, int *s1)
{
    while (*s0 >= 0)
    {
        int *s = s1;
        while (*s >= 0)
            if (*s0 == *s++)
                printf ("%d_", *s0);
        s0++;
    }
    printf ("\n");
}

int main (void)
{
    int a[] = { 10, 4, 3, 7, 12, 0, 1, -1 };
    int b[] = { 7, 14, 0, 8, 9, 22, 10, -1 };
    f (a, b);
    return 0;
}
```

- (a) Was bewirkt die Funktion `f` und warum? (4 Punkte)
- (b) Von welcher Ordnung (Landau-Symbol) ist die Funktion und warum?
Wir beziehen uns hierbei auf die Anzahl der Vergleiche in Abhängigkeit von der Länge der Eingabedaten `s0` und `s1`. Für die Rechnung dürfen Sie beide Längen mit n gleichsetzen, obwohl sie normalerweise nicht gleich sind. (2 Punkte)
- (c) Was passiert, wenn Sie beim Aufruf der Funktion für einen der Parameter den Wert `NULL` übergeben und warum? (2 Punkte)
- (d) Was passiert, wenn Sie das Hauptprogramm wie folgt abändern ([aufgabe-2d.c](#)) und warum?

```
int main (void)
{
    int a[] = { 10, 4, 3, 7, 12, 0, 1 };
    int b[] = { 7, 14, 0, 8, 9, 22, 10 };
    f (a, b);
    return 0;
}
```

(2 Punkte)

Aufgabe 3: Länge von Strings

Diese Aufgabe ist eine Neuauflage von Aufgabe 3 der Übung vom 20. November 2017, ergänzt in den Teilaufgaben (e), (f) und (g).

Strings werden in der Programmiersprache C durch Zeiger auf **char**-Variable realisiert.

Beispiel: **char** *hello_world = "Hello,_world!\n"

Die Systembibliothek stellt eine Funktion **strlen()** zur Ermittlung der Länge von Strings zur Verfügung (**#include <string.h>**).

- (a) Auf welche Weise ist die Länge eines Strings gekennzeichnet?
(1 Punkt)
- (b) Wie lang ist die Beispiel-String-Konstante "Hello,_world!\n", und wieviel Speicherplatz belegt sie?
(2 Punkte)
- (c) Schreiben Sie eine eigene Funktion **int strlen (char *s)**, die die Länge eines Strings zurückgibt.
(3 Punkte)

Wir betrachten nun die folgenden Funktionen (Datei: **aufgabe-3.c**):

```
int fun_1 (char *s)
{
    int x = 0;
    for (int i = 0; i < strlen (s); i++)
        x += s[i];
    return x;
}
```

```
int fun_2 (char *s)
{
    int i = 0, x = 0;
    int len = strlen (s);
    while (i < len)
        x += s[i++];
    return x;
}
```

- (d) Was bewirken die beiden Funktionen? (2 Punkte)
- (e) Von welcher Ordnung (Landau-Symbol) sind die beiden Funktionen hinsichtlich der Anzahl ihrer Zugriffe auf die Zeichen im String – und warum? Sie dürfen für **strlen()** Ihre eigene Version der Funktion voraussetzen. (3 Punkte)
- (f) Schreiben Sie eine eigene Funktion, die dieselbe Aufgabe erledigt wie **fun_2()**, nur effizienter. (4 Punkte)
- (g) Von welcher Ordnung (Landau-Symbol) ist Ihre effizientere Funktion – und warum? (1 Punkt)

Viel Erfolg!