

Hardwarenahe Programmierung

Übungsaufgaben – 16. Januar 2020

Diese Übung enthält Punkteangaben wie in einer Klausur. Um zu „bestehen“, müssen Sie innerhalb von 100 Minuten unter Verwendung ausschließlich zugelassener Hilfsmittel 19 Punkte (von insgesamt 38) erreichen. (Davon entfallen 12 Punkte, entsprechend 30 Minuten, auf die bereits früher gestellte Aufgabe 2, Teilaufgaben (a) bis (e).)

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 ? 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)

Aufgabe 2: Länge von Strings

Diese Aufgabe ist eine Neuauflage von Aufgabe 3 der Übung vom 14. November 2019, ergänzt um die Teilaufgaben (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-2.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) Schreiben Sie eine eigene Funktion, die dieselbe Aufgabe erledigt wie [fun_2\(\)](#), nur effizienter. (4 Punkte)
- (f) Von welcher Ordnung (Landau-Symbol) sind die beiden Funktionen hinsichtlich der Anzahl ihrer Zugriffe auf die Zeichen im String? Begründen Sie Ihre Antwort. Sie dürfen für [strlen\(\)](#) Ihre eigene Version der Funktion voraussetzen. (3 Punkte)
- (g) Von welcher Ordnung (Landau-Symbol) ist Ihre effizientere Funktion? Begründen Sie Ihre Antwort. (1 Punkt)

Aufgabe 3: Objektorientierte Tier-Datenbank

Das auf der nächsten Seite in Blau dargestellte Programm (Datei: [aufgabe-3a.c](#)) soll Daten von Tieren verwalten.

Beim Compilieren erscheinen die folgende Fehlermeldungen:

```
$ gcc -std=c99 -Wall -O aufgabe-2a.c -o aufgabe-2a
aufgabe-2a.c: In function 'main':
aufgabe-2a.c:31: error: 'animal' has no member named 'wings'
aufgabe-2a.c:37: error: 'animal' has no member named 'legs'
```

Der Programmierer nimmt die auf der nächsten Seite in Rot dargestellten Ersetzungen vor (Datei: [aufgabe-3b.c](#)). Daraufhin gelingt das Compilieren, und die Ausgabe des Programms lautet:

```
$ gcc -std=c99 -Wall -O aufgabe-2b.c -o aufgabe-2b
$ ./aufgabe-2b
A duck has 2 legs.
Error in animal: cow
```

- (a) Erklären Sie die o. a. Compiler-Fehlermeldungen. (2 Punkte)
- (b) Wieso verschwinden die Fehlermeldungen nach den o. a. Ersetzungen? (3 Punkte)
- (c) Erklären Sie die Ausgabe des Programms. (5 Punkte)
- (d) Beschreiben Sie – in Worten und/oder als C-Quelltext – einen Weg, das Programm so zu berichtigen, daß es die Eingabedaten ("A duck has 2 wings. A cow has 4 legs.") korrekt speichert und ausgibt. (4 Punkte)

```

#include <stdio.h>

#define ANIMAL 0
#define WITH_WINGS 1
#define WITH_LEGS 2

typedef struct animal
{
    int type;
    char *name;
} animal;

typedef struct with_wings
{
    int wings;
} with_wings;

typedef struct with_legs
{
    int legs;
} with_legs;

int main (void)
{
    animal *a[2];

    animal duck;
    a[0] = &duck;
    a[0]—>type = WITH_WINGS;
    a[0]—>name = "duck";
    a[0]—>wings = 2;  ← ((with_wings *) a[0])—>wings = 2;

    animal cow;
    a[1] = &cow;
    a[1]—>type = WITH_LEGS;
    a[1]—>name = "cow";
    a[1]—>legs = 4;  ← ((with_legs *) a[1])—>legs = 4;

    for (int i = 0; i < 2; i++)
        if (a[i]—>type == WITH_LEGS)
            printf ("A_%s_has_%d_legs.\n", a[i]—>name,
                    ((with_legs *) a[i])—> legs);
        else if (a[i]—>type == WITH_WINGS)
            printf ("A_%s_has_%d_wings.\n", a[i]—>name,
                    ((with_wings *) a[i])—> wings);
        else
            printf ("Error_in_animal:_%s\n", a[i]—>name);

    return 0;
}

```

Viel Erfolg!