

# Hardwarenahe Programmierung

## Musterlösung zu den Übungsaufgaben – 15. Oktober 2018

### Aufgabe 1: Fibonacci-Zahlen

Die Folge der Fibonacci-Zahlen ist definiert durch:

1. Zahl: 0
2. Zahl: 1
- nächste Zahl = Summe der beiden vorherigen

Schreiben Sie ein Programm, das die ersten 50 Fibonacci-Zahlen ausgibt.

### Lösung

Zwei verschiedene richtige Lösungen finden Sie in den Dateien [loesung-1-1.c](#) und [loesung-1-2.c](#).

Die Lösung in [loesung-1-2.c](#) speichert alle berechneten Zahlen in einem Array, die in [loesung-1-1.c](#) hingegen speichert immer nur maximal drei Zahlen gleichzeitig. Sofern nicht alle berechneten Zahlen später noch benötigt werden, ist daher [loesung-1-1.c](#) zu bevorzugen.

Wichtig in [loesung-1-1.c](#) ist, daß  $f_0 + f_1$  berechnet wird, *bevor*  $f_0$  oder  $f_1$  ein neuer Wert zugewiesen wird. Dies ist nur möglich, weil das Programm eine zusätzliche Variable (hier:  $f_2$ ) verwendet.

### Aufgabe 2: Fehlerhaftes Programm

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

```
#include <stdio.h>

int main (void)
{
    for (int i = 10; i = 0; i - 1)
        printf ("%d\n", i);
    return 0;
}
```

- (a) Was bewirkt dieses Programm und warum?
- (b) Ändern Sie das Programm so, daß es einen „Countdown“ von 10 bis 0 ausgibt.

### Lösung

- (a) **Was bewirkt dieses Programm und warum?**

Dieses Programm bewirkt nichts. Die **for**-Schleife wird nicht ausgeführt.

Begründung: Die **for**-Bedingung ist eine Zuweisung des Werts **0** an die Variable **i**. Neben dem Seiteneffekt der Zuweisung liefert der Ausdruck einen Wert zurück, nämlich den zugewiesenen Wert **0**. Dieser wird von **for** als eine Bedingung mit dem konstanten Wert „falsch“ interpretiert.

(Hinweis: Ohne diese Begründung ist die Aufgabe nur zu einem kleinen Teil gelöst.)

Darüberhinaus ist die Zähl-Anwendung unwirksam: Sie berechnet den Wert  $i - 1$  und vergißt ihn wieder, ohne ihn einer Variablen (z. B. **i**) zuzuweisen.

- (b) Ändern Sie das Programm so, daß es einen „Countdown“ von 10 bis 0 ausgibt.

Datei `loesung-2.c`:

```
#include <stdio.h>

int main (void)
{
    for (int i = 10; i >= 0; i--)
        printf ("%d\n", i);
    return 0;
}
```

### Aufgabe 3: Hello, world!

Unter <https://gitlab.cvh-server.de/pgerwinski/hp/tree/2020ws/20181015> finden Sie (unter anderem) die Programme `test-1.c`, `test-2.c` und `test-3.c`.

Was bewirken diese Programme, und warum verhalten sie sich so?

### Lösung

- `test-1.c`

Hinter `return` steht ein Ausdruck mit dem Komma-Operator. Dieser bewirkt, daß der Wert vor dem Komma berechnet und ignoriert und danach der Wert nach dem Komma zurückgegeben wird.

In diesem Fall wird vor dem Komma der Wert des `printf()`-Aufrufs berechnet und ignoriert. Als Seiteneffekt gibt das Programm die Zeile `Hello, world!` aus. Anschließend wird der Wert `0` an `return` übergeben und daher `return 0` ausgeführt.

- `test-2.c`

Das Programm gibt die Zeile `Die Antwort lautet: 42` aus.

Die `if`-Bedingung ist eine Zuweisung `b = 42`, die den zugewiesenen Wert `42` zurückgibt. Weil dieser Wert ungleich Null ist, interpretiert `if` ihn als Wahrheitswert „wahr“, führt also den `if`-Zweig aus und überspringt den `else`-Zweig.

- `test-3.c`

Das Programm stürzt mit einer Fehlermeldung „Speicherzugriffsfehler“ oder „Schutzverletzung“ ab.

Der Funktionsaufruf `printf (42)` übergibt den Zahlenwert `42` als String, also als einen Zeiger auf `char`-Variable, an die Funktion `printf()`. Diese versucht, auf den Speicher ab Adresse `42` zuzugreifen, wofür aber das Programm keine Zugriffsrechte hat. Das Betriebssystem beendet daraufhin das Programm mit der o. a. Fehlermeldung.

Der String `"Die_Antwort_lautet:_"` wird nicht ausgegeben, weil Schreiboperationen aus Effizienzgründen erst nach einer abgeschlossenen Zeile (`"\n"`) durchgeführt werden.