

Hardwarenahe Programmierung / Angewandte Informatik

Musterlösung zu den Übungsaufgaben – 17. Oktober 2016

Aufgabe 1: Fehlerhaftes Programm

Wir betrachten das folgende C-Programm:

```
#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-1.c**:

```
#include <stdio.h>

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

Aufgabe 2: Hello, world!

Unter <https://gitlab.cvh-server.de/pgerwinski/hp/tree/master/20161017> 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.

Aufgabe 3: Primfaktorzerlegung

Schreiben Sie eine Funktion, die eine Zahl als Parameter entgegennimmt und ihre Primfaktorzerlegung ausgibt. Sie dürfen voraussetzen, daß die Funktion nur für Zahlen von 2 bis 100 aufgerufen wird. Mehrfach vorkommende Primfaktoren sollen mehrfach ausgegeben werden.

Lösung

Datei: `loesung-3.c`

```
void factor (int n)
{
    int prime[] = { 2, 3, 5, 7, 0 };
    int i = 0;
    printf ("Primfaktoren_von_%d:\n", n);
    while (n > 1 && prime[i])
    {
        if (n % prime[i] == 0)
        {
            printf ("%d\n", prime[i]);
            n /= prime[i];
        }
        else
            i++;
    }
    if (n > 1)
        printf ("%d\n", n);
}
```

Bemerkungen:

- Dieses Beispiel verwendet ein Array von Primzahlen mit dem Wert `0` als Ende-Markierung. Alternativ kann man auch die Länge des Arrays abzählen.
- Es genügt für Zahlen bis 100 tatsächlich, durch die Primzahlen bis einschließlich 7 zu dividieren. Was am Ende übrigbleibt, muß dann eine Primzahl und damit der letzte Primfaktor sein. (Allgemein gilt: Für Zahlen bis n muß man durch Primzahlen bis \sqrt{n} dividieren.)
- Anstatt nur durch Primzahlen kann man auch durch *alle* kleineren Zahlen (bis \sqrt{n}) dividieren. Das Ergebnis ist dasselbe, weil zusammengesetzte Zahlen durch ihre Primfaktoren bereits vorher „ausgesiebt“ wurden. Dies ist unnötiger Rechenaufwand, aber trotzdem eine richtige Lösung der Aufgabe.
- Wenn Sie die Faktorisierung direkt in der Funktion `main()` vornehmen, haben Sie den Aufgabentext „Schreiben Sie eine Funktion, die ...“ nicht erfüllt, es sei denn, Sie werten die Parameter des Hauptprogramms aus, was aber unverhältnismäßig aufwendig ist.
- Die Zahl 1 hat tatsächlich *keine* Primfaktoren. (Ein Produkt von null Faktoren ist per definitionem 1.)