

# Hardwarenahe Programmierung

## Musterlösung zu den Übungsaufgaben – 21. November 2022

### Aufgabe 1: Kondensator

Ein Kondensator der Kapazität  $C = 100 \mu\text{F}$  ist auf die Spannung  $U_0 = 5 \text{ V}$  aufgeladen und wird über einen Widerstand  $R = 33 \text{ k}\Omega$  entladen.

- (a) Schreiben Sie ein C-Programm, das den zeitlichen Spannungsverlauf in einer Tabelle darstellt. (5 Punkte)
- (b) Schreiben Sie ein C-Programm, das ermittelt, wie lange es dauert, bis die Spannung unter  $0.1 \text{ V}$  gefallen ist. (4 Punkte)
- (c) Vergleichen Sie die berechneten Werte mit der exakten theoretischen Entladekurve:  $U(t) = U_0 \cdot e^{-\frac{t}{RC}}$  (3 Punkte)

Hinweise:

- Für die Simulation zerlegen wir den Entladevorgang in kurze Zeitintervalle  $dt$ . Innerhalb jedes Zeitintervalls betrachten wir den Strom  $I$  als konstant und berechnen, wieviel Ladung  $Q$  innerhalb des Zeitintervalls aus dem Kondensator herausfließt. Aus der neuen Ladung berechnen wir die Spannung am Ende des Zeitintervalls.
- Für den Vergleich mit der exakten theoretischen Entladekurve benötigen Sie die Exponentialfunktion `exp()`. Diese finden Sie in der Mathematik-Bibliothek: `#include <math.h>` im Quelltext, beim `gcc`-Aufruf `-lm` mit angeben.
- $Q = C \cdot U$ ,  $U = R \cdot I$ ,  $I = \frac{dQ}{dt}$

### Lösung

- **Schreiben Sie ein C-Programm, das den zeitlichen Spannungsverlauf in einer Tabelle darstellt.**

In dem Programm `loesung-1a.c` arbeiten wir, dem ersten Hinweis folgend, mit einem Zeitintervall von `dt = 0.01`. Mit dieser Schrittweite lassen wir uns eine Tabelle ausgeben, die jeweils die Zeit und durch die Simulation berechnete Spannung ausgibt.

Wir simulieren, wie die Ladung  $Q = C \cdot U$  des Kondensators im Laufe der Zeit abfließt. Dazu berechnen wir in jedem Zeitschritt zunächst den Strom  $I = U/R$ , der aus dem Kondensator fließt. Dieser Strom bewirkt, daß innerhalb des Zeitintervalls  $dt$  die Ladung  $dQ = I \cdot dt$  aus dem Kondensator abfließt. Am Ende des Zeitintervalls berechnen wir die zur neuen Ladung  $Q$  gehörende neue Spannung  $U = Q/C$ .

Für eine einfache Ausgabe der Tabelle verwenden wir die Formatspezifikationen `%10.3lf` für drei bzw. `%15.8lf` für acht Nachkommastellen Genauigkeit. Damit schreiben wir jeweils eine *lange Fließkommazahl* (`%lf`) rechtsbündig in ein Feld der Breite 10 bzw. 15 und lassen uns 3 bzw. 8 Nachkommastellen ausgeben.

Wir compilieren das Programm mit: `gcc -Wall -O loesung-1a.c -o loesung-1a`

- **Schreiben Sie ein C-Programm, das ermittelt, wie lange es dauert, bis die Spannung unter  $0.1 \text{ V}$  gefallen ist.**

Wir ändern das Programm `loesung-1a.c` so ab, daß zum einen die Schleife abbricht, sobald die Spannung den Wert  $0.1 \text{ V}$  unterschreitet (`loesung-1b.c`), und daß zum anderen nicht jedesmal eine Zeile für die Tabelle ausgegeben wird, sondern erst am Ende die Zeit (und die Spannung).

Der Ausgabe entnehmen wir, daß die Spannung bei etwa  $t = 12.90 \text{ s}$  den Wert  $0.1 \text{ V}$  unterschreitet.

- **Vergleichen Sie die berechneten Werte mit der exakten theoretischen Entladekurve:**

$$U(t) = U_0 \cdot e^{-\frac{t}{RC}}$$

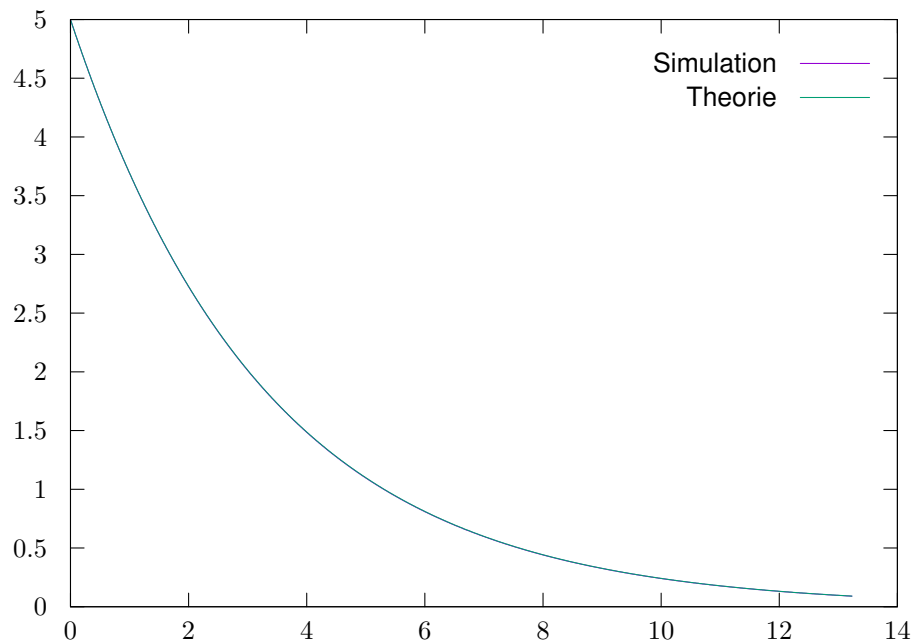
Wir ändern das Programm `loesung-1a.c` so ab, daß es zusätzlich zur Zeit und zur simulierten Spannung die exakte Spannung  $U_0 \cdot e^{-\frac{t}{RC}}$  gemäß der theoretischen Entladekurve ausgibt (`loesung-1c.c`),

Da dieses Programm die Exponentialfunktion verwendet, müssen wir nun beim Compilieren zusätzlich `-lm` für das Einbinden der Mathematik-Bibliothek angeben.

Der erweiterten Tabelle können wir entnehmen, daß die durch die Simulation berechnete Spannung mit der Spannung  $U_0 \cdot e^{-\frac{t}{RC}}$  gemäß der theoretischen Entladekurve bis auf wenige Prozent übereinstimmt. Dies ist für viele praktische Anwendungen ausreichend, wenn auch nicht für Präzisionsmessungen.

Wenn Sie die Ausgabe des Programms, z. B. mit `./loesung-1c > loesung-1c.dat`, in einer Datei `loesung-1c.dat` speichern, können Sie sich die beiden Kurven graphisch darstellen lassen, z. B. mit `gnuplot` und dem folgenden Befehl:

```
plot "loesung-1c.dat" using 1:2 with lines title "Simulation",
     "loesung-1c.dat" using 1:3 with lines title "Theorie"
```



Der Unterschied zwischen der simulierten und der theoretischen Entladungskurve ist mit bloßem Auge nicht sichtbar.

## Aufgabe 2: Personen-Datenbank

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

```
#include <stdio.h>
#include <string.h>

typedef struct
{
    char first_name[10];
    char family_name[20];
    char day, month;
    int year;
} person;

int main (void)
{
    person sls;
    sls.day = 26;
    sls.month = 7;
    sls.year = 1951;
    strcpy (sls.first_name, "Sabine");
    strcpy (sls.family_name, "Leutheusser-Schnarrenberger");
    printf ("%s_%s_wurde_am_%d.%d.%d_geboren.\n",
           sls.first_name, sls.family_name, sls.day, sls.month, sls.year);
    return 0;
}
```

Die Standard-Funktion `strcpy()` bewirkt ein Kopieren eines Strings von rechts nach links, hier also z. B. die Zuweisung der String-Konstanten "Sabine" an die String-Variable `sls.first_name[]`.

Das Programm wird für einen 32-Bit-Rechner kompiliert und ausgeführt.

(Die `gcc`-Option `-m32` sorgt dafür, daß `gcc` Code für einen 32-Bit-Prozessor erzeugt.)

```
$ gcc -Wall -O -m32 aufgabe-2.c -o aufgabe-2
$ ./aufgabe-2
Sabine Leutheusser-Schnarrenberger wurde am 110.98.1701278309 geboren.
Speicherzugriffsfehler
```

- (a) Erklären Sie die Ausgabe des Programms einschließlich der Zahlenwerte. (4 Punkte)
- (b) Welche Endianness hat der verwendete Rechner? Begründen Sie Ihre Antwort. (1 Punkt)
- (c) Wie sähe die Ausgabe auf einem Rechner mit entgegengesetzter Endianness aus? (2 Punkte)
- (d) Erklären Sie den Speicherzugriffsfehler. (Es kann sein, daß sich der Fehler auf Ihrem Rechner nicht bemerkbar macht. Er ist aber trotzdem vorhanden.) (2 Punkte)

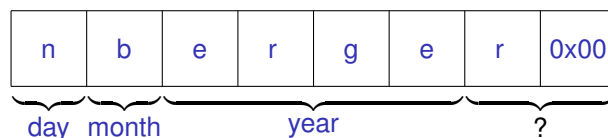
## Lösung

- (a) **Erklären Sie die Ausgabe des Programms einschließlich der Zahlenwerte.**

Der String "Leutheusser-Schnarrenberger" hat 27 Zeichen und daher mehr als die in der Variablen `sls.family_name` vorgesehenen 20 Zeichen. Das "nberger" paßt nicht mehr in die String-Variable.

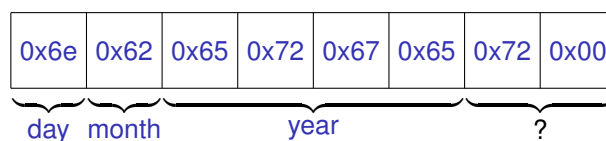
Die Zuweisung `strcpy (sls.family_name, "Leutheusser-Schnarrenberger")` überschreibt daher 8 Speicherzellen außerhalb der String-Variablen `sls.family_name` mit dem String "nberger" (7 Buchstaben zzgl. String-Ende-Symbol) – und damit insbesondere die Variablen `sls.day`, `sls.month` und `sls.year`.

Die überschriebenen Speicherzellen sehen demnach folgendermaßen aus:



(„?“ steht für zwei Speicherzellen, von denen wir nicht wissen, wofür sie genutzt werden.)

Wenn wir die ASCII-Zeichen in Hexadezimalzahlen umrechnen, entspricht dies:



Dies entspricht bereits genau den Werten 110 und 98 für die Variablen `sls.day` bzw. `sls.month`.

Für die Variable `sls.year` müssen wir ihre vier Speicherzellen unter der Berücksichtigung der Endianness des Rechners zusammenziehen. Für Big-Endian ergibt dies `0x65726765 == 1701996389`. Für Little-Endian ergibt sich der Wert `0x65677265 == 1701278309`, der auch in der Ausgabe des Programms auftaucht.

- (b) **Welche Endianness hat der verwendete Rechner? Begründen Sie Ihre Antwort.**

Wie in (a) begründet, ergibt sich die Ausgabe von 1701278309 für das Jahr aus dem Speicherformat Little-Endian.

- (c) **Wie sähe die Ausgabe auf einem Rechner mit entgegengesetzter Endianness aus?**

Wie in (a) begründet, ergäbe sich aus dem Speicherformat Big-Endian die Ausgabe von 1701996389 für das Jahr.

- (d) **Erklären Sie den Speicherzugriffsfehler. (Es kann sein, daß sich der Fehler auf Ihrem Rechner nicht bemerkbar macht. Er ist aber trotzdem vorhanden.)**

Die zwei in (a) mit „?“ bezeichneten Speicherzellen wurden ebenfalls überschrieben. Dies ist in der Ausgabe zunächst nicht sichtbar, bewirkt aber später den Speicherzugriffsfehler.

(Tatsächlich handelt es sich bei den überschriebenen Speicherzellen um einen Teil der Rücksprungadresse, die `main()` verwendet, um mit `return 0` an das Betriebssystem zurückzugeben.)

**Hinweis 1:** Um auf einen solchen Lösungsweg zu kommen, wird empfohlen, „geheimnisvolle“ Zahlen nach hexadezimal umzurechnen und in Speicherzellen (Zweiergruppen von Hex-Ziffern) zu zerlegen. Oft erkennt man dann direkt ASCII-Zeichen: Großbuchstaben beginnen mit der Hex-Ziffer 4 oder 5, Kleinbuchstaben mit 6 oder 7.

**Hinweis 2:** Um derartige Programmierfehler in der Praxis von vornherein zu vermeiden, wird empfohlen, anstelle von `strcpy()` grundsätzlich die Funktion `strncpy()` zu verwenden. Diese erwartet einen zusätzlichen Parameter, der die maximal zulässige Länge des Strings enthält. Ohne einen derartigen expliziten Parameter kann die Funktion nicht wissen, wie lang die Variable ist, in der der String gespeichert werden soll.

### Aufgabe 3: 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-3.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)

### Lösung

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

Datei: `loesung-3.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 Abhängigkeit von  $n$ ? Begründen Sie Ihre Antwort.**

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 in Abhängigkeit von  $n$ ? Begründen Sie Ihre Antwort.**

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  $\mathcal{O}(n)$ .