

Hardwarenahe Programmierung

Musterlösung zu den Übungsaufgaben – 12. November 2018

Aufgabe 1: Text-Grafik-Bibliothek

Schreiben Sie eine Bibliothek für „Text-Grafik“ mit folgenden Funktionen:

- **void clear (char c)**
Bildschirm auf Zeichen `c` löschen,
also komplett mit diesem Zeichen (z. B.: Leerzeichen) füllen
- **void put_point (int x, int y, char c)**
Punkt setzen (z. B. einen Stern (*) an die Stelle (x, y) „malen“)
- **char get_point (int x, int y)**
Punkt lesen
- **void display (void)**
das Gezeichnete auf dem Bildschirm ausgeben

Hinweise:

- Eine C-Bibliothek besteht aus (mindestens) einer `.h`-Datei und einer `.c`-Datei.
- Verwenden Sie ein Array als „Bildschirm“.
Vor dem Aufruf der Funktion `display()` ist nichts zu sehen;
alle Grafikoperationen erfolgen auf dem Array.
- Verwenden Sie Präprozessor-Konstante, z. B. `WIDTH` und `HEIGHT`,
um Höhe und Breite des „Bildschirms“ festzulegen:


```
#define WIDTH 72
#define HEIGHT 24
```
- Schreiben Sie zusätzlich ein Test-Programm, das alle Funktionen der Bibliothek benutzt,
um ein hübsches Bild (z. B. ein stilisiertes Gesicht – „Smiley“) auszugeben.

Lösung

Siehe die Dateien `textgraph.c` und `textgraph.h` (Bibliothek) sowie (Test-Programm).

Diese Lösung erfüllt zusätzlich die Aufgabe, bei fehlerhafter Benutzung (Koordinaten außerhalb des Zeichenbereichs) eine sinnvolle Fehlermeldung auszugeben, anstatt unkontrolliert Speicher zu überschreiben und abzustürzen.

Das Schlüsselwort `static` bei der Deklaration der Funktion `check_coordinates()` bedeutet, daß diese Funktion nur lokal (d. h. innerhalb der Bibliothek) verwendet und insbesondere nicht nach außen (d. h. für die Benutzung durch das Hauptprogramm) exportiert wird. Dies dient dazu, nicht unnötig Bezeichner zu reservieren (Vermeidung von „Namensraumverschmutzung“).

Man beachte die Verwendung einfacher Anführungszeichen (Apostrophe) bei der Angabe von `char`-Konstanten (`'*'`) im Gegensatz zur Verwendung doppelter Anführungszeichen bei der Angabe von String-Konstanten (String = Array von `chars`, abgeschlossen mit Null-Symbol). Um das einfache Anführungszeichen selbst als `char`-Konstante anzugeben, ist ein vorangestellter Backslash erforderlich: `'\"'` („Escape-Sequenz“). Entsprechendes gilt für die Verwendung doppelter Anführungszeichen innerhalb von String-Konstanten:

```
printf("Your_name_is:_%s'", name);
```

Aufgabe 2: Datum-Bibliothek

Zerlegen Sie die Datum-Bibliothek aus der Übungsaufgabe 3 vom 29. 10. 2018 in eine `.c`- und eine `.h`-Datei.
Hinweis: Schreiben Sie auch hierfür zusätzlich ein Test-Programm.

Lösung

Die Dateien `dates.c` und `dates.h` enthalten die Bibliothek, die Datei `test-dates.c` ein Programm zum Testen der Bibliothek.

Aufgabe 3: Kondensator

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

- (a) Stellen Sie den zeitlichen Spannungsverlauf in einer Tabelle dar.
- (b) Wie lange dauert es, bis die Spannung unter 0.1 V gefallen ist?
- (c) Vergleichen Sie die berechneten Werte mit der exakten theoretischen Entladekurve: $U(t) = U_0 \cdot e^{-\frac{t}{RC}}$

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

In dem Programm `loesung-3.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, die durch die Simulation berechnete Spannung und die Spannung $U_0 \cdot e^{-\frac{t}{RC}}$ gemäß der theoretischen Entladekurve 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 $U = Q/C$.

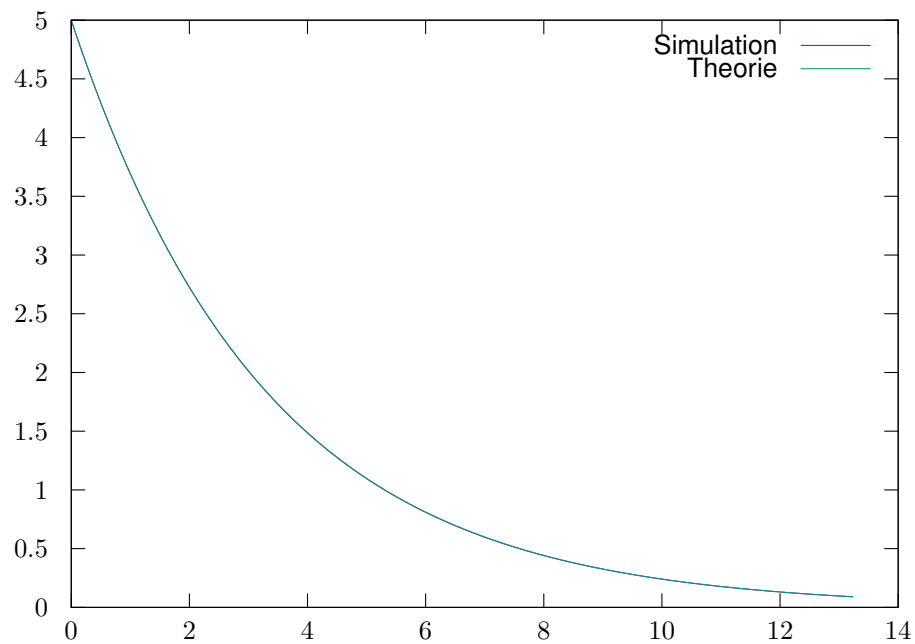
Für eine einfache Ausgabe der Tabelle verwenden wir dreimal die Formatspezifikation `%10.3lf`. Damit schreiben wir jeweils eine *lange Fließkommazahl* (`%lf`) rechtsbündig in ein Feld der Breite 10 und lassen uns 3 Nachkommastellen ausgeben.

Wir compilieren das Programm mit: `gcc -Wall -O loesung-3.c -lm -o loesung-3`
(Man beachte das `-lm` für das Einbinden der Mathematik-Bibliothek.)

Der Tabelle entnehmen wir dann, daß die Spannung bei etwa $t = 12.91 \text{ s}$ den Wert 0.1 V unterschreitet.

Ebenfalls der 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-3 > loesung-3.dat`, in einer Datei `loesung-3.dat` speichern, können Sie sich die beiden Kurven graphisch darstellen lassen, z. B. mit `gnuplot` und dem folgenden Befehl: `plot "loesung-3.dat" using 1:2 with lines title "Simulation", "loesung-3.dat" using 1:3 with lines title "Theorie"`



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