6.1 Extras/Schreiben in Dateien

6.1.1 Problem

Ein C-Programm soll den Spannungsverlauf an einem Kondensator über mehrere Sekunden berechnen und in lesbarer Form in eine Datei schreiben. Nur: Wie schreibt man von einem C-Programm aus in eine Datei?

6.1.2 Programmbeispiel

Das folgende Programm `speicher_notiz.c` löst ein ähnliches Problem:

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      FILE   *dateimerker;
6      char   dateiname[]="notizen.txt";
7      char   notiz[801]="";
8
9      dateimerker = fopen(dateiname, "a");
10     if(dateimerker == NULL)
11     {
12         perror(dateiname);
13         return 0;
14     }
15     printf("Notiz, die in der Datei gespeichert werden soll:");
16     scanf("%800[^\n]", notiz);
17
18     fprintf(dateimerker, "Notiz: ");
19     fprintf(dateimerker, "%s\n", notiz);
20     fclose(dateimerker);
21     return 0;
22 }
```

Zeile 1: Die Dateifunktionen der C-Standardbibliothek benötigen die bekannte Include-Datei `<stdio.h>`.

Zeile 5: `FILE *dateimerker` ist eine Variable, die den Zugriff auf eine Datei ermöglicht, quasi ein Henkel (*handle*). Sie zeigt auf eine programminterne Datenstruktur in der C-Laufzeitumgebung.

Zeile 9: `dateimerker=fopen(dateiname, "a")`: Hier wird die Datei geöffnet. Der Pfadname der Datei ist der erste Parameter. Daer Pfadname darf relativ oder absolut angegeben werden. Relativ heißt: relativ zum aktuellen Verzeichnis des Prozesses.

Der zweiter Parameter ist der so genannte Modus (*mode*). Er gibt an, ob die Datei dieses Namens, falls sie schon existiert, überschrieben oder ergänzt werden soll.

Lautet der Modus `"w"` wie *write*, gilt folgendes: Falls eine Datei mit diesem Namen noch nicht existiert, wird sie beim Öffnen neu angelegt. Falls schon eine Datei mit demselben Pfadnamen existiert, wird sie überschrieben.

Lautet der Modus `"a"` wie *append*, gilt: Falls eine Datei mit diesem Namen noch nicht existiert, wird sie auch hier beim Öffnen neu angelegt. Falls schon eine Datei mit demselben Pfadnamen existiert, wird der neue Inhalt an den bisherigen Dateiinhalt angehängt.

Zeile 10: Falls die Datei aus irgendeinem Grund nicht angelegt oder geöffnet werden konnte (ungültiger Pfadname, fehlende Berechtigung, gleichnamiges Verzeichnis, ...), wird statt des Dateimerkers ein Nullzeiger zurückgegeben.

Zeile 12: Ohne einen gültigen Dateimerker kann man die Datei nicht nutzen. Man kann aber eine Fehlermeldung bekommen; dazu dient die Funktion `perror()`. Diese Funktion braucht als Parameter einen beliebigen String, der zu Beginn der Fehlermeldung mit ausgegeben wird.

Zeile 13: In diesem Fall sollte man das Programm beenden.

Zeile 18: `fprintf(dateimerker, "Notiz: ");`: Mit diesem Befehl kann ein Zeichen, eine Zahl oder ein String ausgegeben werden (je nach Formatstring), hier ist es ein String. Insofern verhält sich `fprintf()` wie `printf()`.

Zeile 19: `fprintf(dateimerker, "%s\n", notiz);`: Hier wird die Notiz ausgegeben. Wenn man `fprintf()` mehrfach aufruft, wird die Datei größer. In der Struktur, auf die der Dateimerker zeigt, wird nämlich ein Zeiger auf die aktuelle Schreibposition mitgeführt. Dieser Zeiger wird bei jedem Zugriff weitergeschoben.

Dabei ist es egal, ob die Datei einst mit dem Modus `"w"` oder mit dem Modus `"a"` aufgerufen wurde. Der Unterschied zwischen `"w"` und `"a"` bezieht sich nämlich nur auf den Moment, in dem die Datei geöffnet wurde. Danach wirken `"w"` und `"a"` gleich.

Zeile 20: Mit `fclose(dateimerker);` bringt man die Datei in einen sauberen Zustand zurück, alle Zwischenpuffer werden geschlossen und die Datei kann von anderen Programmen geöffnet werden.

6.1.3 Datenstrom

Die interne Datenstruktur, auf die die Variable `dateimerker` vom Typ `FILE *` zeigt, beinhaltet einen sogenannten Datenstrom. Ein solcher Datenstrom ist eine eigene Datenstruktur. Diese Struktur ähnelt einem Array von Bytes, aber mit einigen Unterschieden:

- Man muss den Datenstrom vor der Benutzung initialisieren (öffnen)
- Dieses „Array“ darf beliebig groß werden (solange der Platz reicht)
- Man kann aber nur an einer bestimmten Stelle in das „Array“ schreiben; es gibt praktisch einen Zugriffszeiger, der diese Stelle festlegt
- Nach jedem Zugriff wird der Zugriffszeiger um eins erhöht, so dass man auf das nächste Element zugreifen kann
- Die Stelle des letzten Zugriffs markiert die Größe des „Arrays“
- Man kann den Zugriffszeiger aber auch manuell ändern

Diese Datenstruktur ist speziell für den Zugriff auf vergleichsweise langsame Massenspeicher (Magnetbänder, Festplatten, optische Laufwerke, Netzwerk-Speicher) eingerichtet worden. Bei diesen ist es meistens so, dass ein Zugriff auf benachbarte Bytes schnell ist, ein Zugriff auf weit entfernte Bytes dagegen *sehr* langsam. Es sind eben Speichermedien mit sequentiellem Zugriff.

Andererseits spielt Speichergröße bei diesen Massenspeichern nur eine geringere Rolle als beim RAM; deshalb braucht man die Größe dieser Datenstruktur nicht zu begrenzen.

Das Prinzip des Umgangs mit Dateien in C sieht nun so aus:

a) Datei öffnen, prüfen auf Erfolg; man erhält einen Datenstrom

b) In den Datenstrom schreiben (so oft man will)

c) Datenstrom schließen, so dass die Datei anschließend den gewünschten Inhalt hat

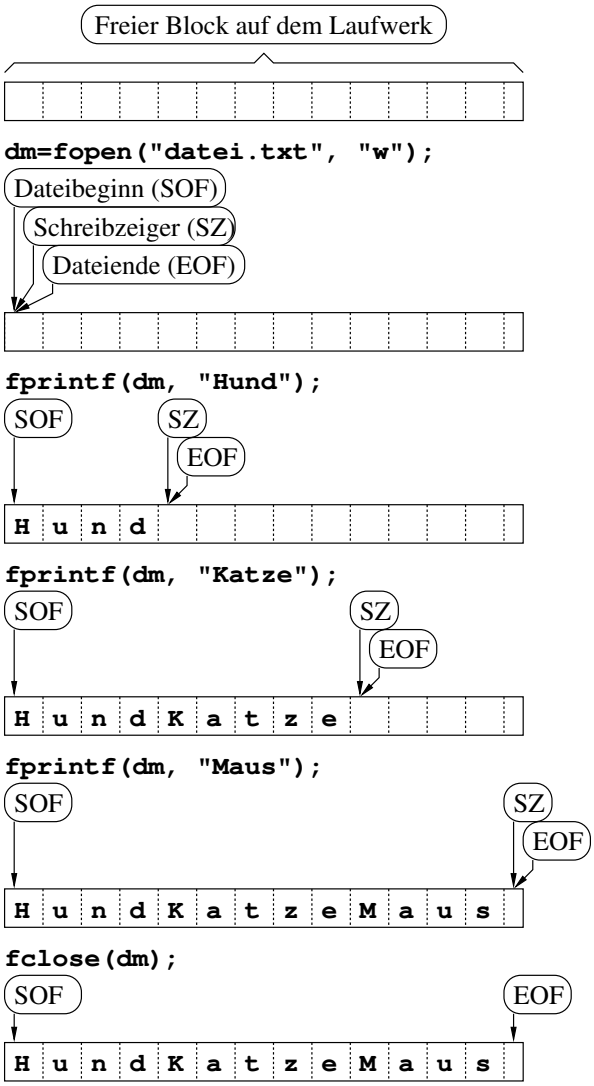

Abbildung 1: Schreiben in einen Datenstrom

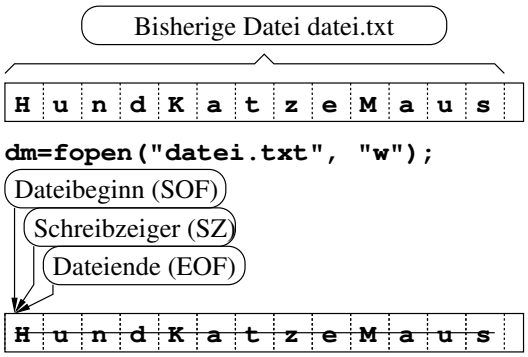

Abbildung 2: Öffnen einer bestehenden Datei im Modus `w`

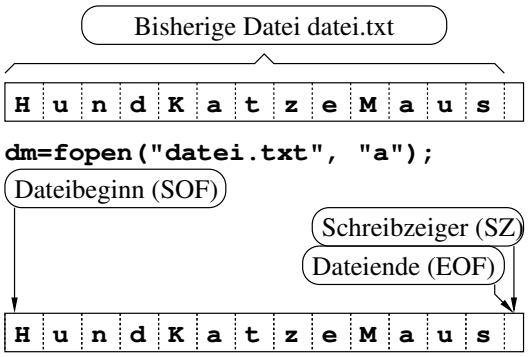

Abbildung 3: Öffnen einer bestehenden Datei im Modus `a`

Abbildung 1 zeigt ein Beispiel, in dem drei Worte in einen Datenstrom geschrieben werden, der zu einer neu angelegten Datei gehört. Die Datei hat zum Schluss genau diese drei Worte als Inhalt.

Was passiert nun, wenn diese Datei noch einmal zum Schreiben geöffnet wird? Wenn man die Datei im Modus `"w"` öffnet, wird der bisherige Inhalt ignoriert (Abbildung 2), und der nächste Schreibzugriff erfolgt wieder am Dateianfang.

Wenn man dagegen den Modus `"a"` verwendet, wird der bisherige Inhalt beibehalten (Abbildung 3), und und der nächste Schreibzugriff erfolgt am bisherigen Dateiende.

6.1.4 Ausgabefunktionen

Es gibt noch weitere Funktionen, mit denen man in C in eine Datei schreiben kann. Sie sind aufgelistet in Tabelle 1 Mit `fputc()` kann man bei einem Aufruf immer nur ein Byte schreiben.

Befehl	Was wird geschrieben?
`fputc(z, dateimerker);`	Zeichen `z`
`fprintf(dateimerker,"%lf", x);`	Zahl `x`
`fputs(str, dateimerker);`	String `str`

Tabelle 1: Funktionen zum Schreiben in Dateien

Mit `fputs()` schreibt man eine Zeichenkette (String) auf einmal in die Datei. Und `fprintf()` erlaubt alle die Ausgaben, die auch mit `printf()` möglich sind: Einzelzeichen (`%c`), Strings (`%s`) und Zahlen (`%i` usw.). Und diese drei Funktionen dürfen in einem Programm ohne Probleme gemischt verwendet werden:

```
1  #include <stdio.h>
2  int main(void)
3  {
4      FILE *dm;
5      dm=fopen("datei.txt","w");
6      if(dm == NULL)
7      {
8          perror("datei.txt");
9          return 1;
10     }
11     fputc(66, dm);
12     fputs("ielefeld", dm);
13     fprintf(dm, "␣hat␣mehr␣als␣%i␣Einwohner.\n", 330000);
14     fclose(dm);
15 }
```

Terminal
```
schueler@debian964:~$ gcc -o ausgabefunkt ausgabefunkt.c
schueler@debian964:~$ ausgabefunkt
schueler@debian964:~$ cat datei.txt # in MS-Windows: type datei.txt
Bielefeld hat mehr als 330000 Einwohner.
```

`fputc()` kann man also genauso verwenden wie `putchar()`. `fputs()` entspricht `puts()`, bis auf die Tatsache, dass `puts()` einen Zeilentrenner (`\n`) an die Ausgabe anfügt und `fputs()` nicht. Und `fprintf()` kann man genauso verwenden wie `printf()`. In allen drei Fällen braucht man aber den Dateimerker als zusätzlichen Parameter. Der ist nötig, damit die Funktion weiß, in welche Datei geschrieben werden soll. Man kann nämlich in einem Programm mehrere Dateien gleichzeitig geöffnet haben.

6.1.5 Ein Programm – mehrere Dateien

Was ist, wenn man in einem Programm mehrere Dateien schreiben will? Am einfachsten ist es, wenn man die Dateien nacheinander schreibt:

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      FILE *dm;
6      dm=fopen("datei1.txt","w");
7      if(dm==NULL)
8      {
9          perror("datei1.txt");
10         return 0;
11     }
12     fprintf(dm, "Hallo an die erste Datei\n");
13     fclose(dm);
14
15     dm=fopen("datei2.txt","w");
16     if(dm==NULL)
17     {
18         perror("datei2.txt");
19         return 0;
20     }
21     fprintf(dm, "Hallo an die zweite Datei\n");
22     fclose(dm);
23
24     return 0;
25 }
```

In den Zeilen 6 bis 13 wird der Dateimerker `dm` für die erste Datei benutzt. In den Zeilen 15 bis 22 wird derselbe Dateimerker für die zweite Datei benutzt.

Wichtig ist: Zuerst muss hier die erste Datei geschlossen werden, bevor man den Dateimerker für das Öffnen der zweiten Datei benutzt.

Terminal
```
schueler@debian964:~$ gcc -o mehrere1 mehrere1.c
schueler@debian964:~$ mehrere1
schueler@debian964:~$ cat datei1.txt
Hallo an die erste Datei
schueler@debian964:~$ cat datei2.txt
Hallo an die zweite Datei
```

6.1.6 Ein Programm – mehrere Dateien gleichzeitig

Wenn man mehrere Dateien *gleichzeitig* geöffnet haben möchte, dann braucht man einfach zwei Dateimerker:

```c
#include <stdio.h>

int main(void)
{
    FILE *dma;
    FILE *dmb;
    dma=fopen("datei1.txt","w");
    if(dma==NULL)
    {
        perror("datei1.txt");
        return 0;
    }
    dmb=fopen("datei2.txt","w");
    if(dmb==NULL)
    {
        perror("datei2.txt");
        fclose(dma);
        return 0;
    }

    fprintf(dma, "Hallo an die erste Datei\n");
    fprintf(dmb, "Hallo an die zweite Datei\n");
    fprintf(dma, "Nun wieder die erste Datei\n");
    fprintf(dmb, "Nun wieder die zweite Datei\n");
    fclose(dmb);
    fclose(dma);
    return 0;
}
```

Terminal

```
schueler@debian964:~$ gcc -o mehrere2 mehrere2.c
schueler@debian964:~$ mehrere2
schueler@debian964:~$ cat datei1.txt
Hallo an die erste Datei
Nun wieder die erste Datei
schueler@debian964:~$ cat datei2.txt
Hallo an die zweite Datei
Nun wieder die zweite Datei
```

Die Reihenfolge des Öffnens und Schließens ist hier unwichtig. Wichtig ist nur:

- Bekommt man von `fopen()` einen NULL-Zeiger, darf man damit nicht weiterarbeiten, also kein `fputc()`, `fputs()` und kein `fprintf()` mit einem NULL-Zeiger als Parameter benutzen. Selbst `fclose()` darf nicht mit einem NULL-Zeiger aufgerufen werden!
- Jede offene Datei muss auch wieder geschlossen werden. Es ist zwar so, dass beim Programmende alle offenen Dateien durch das Laufzeitsystem geschlossen werden, aber es gilt als guter Programmierstil, jede (wirklich jede) geöffnete Datei auch selbst wieder zu schließen. Der Grund: Es kann sein, dass das Programm erweitert wird; vielleicht wird die Befehlsfolge in eine Funktion übernommen. Und dann kann es sein, dass die nicht geschlossene Datei einen schwer zu findenden Fehler verursacht.

6.1.7 Ausgabekanäle

`fputc()` ist also vergleichbar mit `putchar()`, das immer nur ein ASCII-Zeichen auf den Bildschirm ausgibt. Und tatsächlich sind diese beiden Funktionen verwandt: Sie stammen beide aus `<stdio.h>`, und `putchar()` ist einfach ein Spezialfall von `fputc()`. In der Standardbibliothek von C wird nämlich der Bildschirm (wie auch die Tastatur, dazu später) als ein Sonderfall einer geöffneten Datei behandelt. Sobald man ein C-Programm in einem der üblichen Betriebssysteme compiliert und startet, bekommt der gestartete Prozess gleich drei offene Dateien geschenkt:

- Die Standard-Eingabe (Kanal 0) ist mit der Tastatur verbunden. Der dazugehörige Dateimerker in `<stdio.h>` hat den Namen `stdin`.
- Die Standard-Ausgabe (Kanal 1) ist mit dem Bildschirm verbunden. Der dazugehörige Dateimerker in `<stdio.h>` hat den Namen `stdout`.
- Die Fehler-Ausgabe (Kanal 2) ist ebenfalls mit dem Bildschirm verbunden. Der dazugehörige Dateimerker in `<stdio.h>` hat den Namen `stderr`.

Die Standard-Ausgabe (Kanal 1) wird von `putchar()`, `puts()` und `printf()` zur Ausgabe benutzt:

```
1    putchar(68);
2    fputc(68, stdout);
3    puts("Hallo");
4    fputs("Hallo\n", stdout);
5    printf("%i", 95);
6    fprintf(stdout, "%i", 95);
```

Die Zeilen 1 und 2, 3 und 4, 5 und 6 sind also jeweils gleichbedeutend.

Die Fehler-Ausgabe (Kanal 2) wird von der Funktion `perror()` zur Ausgabe benutzt:

```
1 #include <errno.h>
2 #include <strerror.h>
3    perror("meinedatei.txt");
4    fprintf(stderr, "meinedatei.txt: %s\n", strerror(errno));
```

Auch hier sind die Zeilen 3 und 4 gleichbedeutend[1].

Wozu gibt es überhaupt zwei verschiedene Kanäle, wenn beide auf den Bildschirm ausgeben?

- Durch Umleitung der einzelnen Kanäle kann man Programmausgabe und Fehlermeldungen voneinander trennen.
- Die Standard-Ausgabe ist gepuffert, die Fehler-Ausgabe nicht. Daher hat die Standard-Ausgabe eine höhere Datenrate; die Fehler-Ausgabe hat dafür eine geringer Totzeit.

  Die Standard-Ausgabe ist also schneller mit größeren Datenmengen fertig; dafür kommen die Fehlermeldungen von der Fehler-Ausgabe sofort an (wie bei einer Abkürzung, die über einen kurzen, aber schmalen Feldweg führt).

6.1.8 Fehler und Rückgabewerte beim Öffnen

Wenn man in eine Datei schreiben möchte, kann es sein, dass es nicht funktioniert. In den meisten Fällen fällt das schon beim Öffnen der Datei auf. Sobald der Rückgabewert von `fopen()` ein NULL-Zeiger ist, ist irgendein Fehler aufgetreten. Zwei Dinge sind dann nötig:

a) Der Benutzer sollte informiert werden, dass nicht alles so lief wie erhofft

b) Das Programm kann mit diesem NULL-Zeiger keine weiteren Operationen auf der Datei ausführen und muss ebenfalls reagieren, z. B. sich beenden

Im ersten Beispiel dieses Kapitels ist das so gemacht worden.

[1] errno bietet eine Fehlernummer an; strerror() zeigt die dazu passende Fehlermeldung.

6.1.9 Fehler und Rückgabewerte beim Schreiben

Wenn das Öffnen einer Datei geklappt hat, kann beim Schreiben eigentlich nur noch passieren, dass die Datei zu groß wird. Deshalb kann es sinnvoll sein, bei den Schreibfunktionen den Rückgabewert abzufragen. Tabelle 2 zeigt, was bei Erfolg und was bei einem Fehler zurückgegeben wird.

Funktion	Fehler	Erfolg
`fputc()`	−1 (=EOF)	Das Zeichen selbst
`fputs()`	−1 (=EOF)	nicht-negative Zahl
`fprintf()`	−1 (=EOF)	Anzahl ausgegebener Zeichen

Tabelle 2: Rückgabewerte der Schreibfunktionen

6.1.10 Fehler und Rückgabewerte beim Schließen

Das Schließen einer Datei gelingt immer. Deshalb wird der Rückgabewert von `fclose()` meistens auch nicht abgefragt. Man darf eine Datenstrom allerdings immer nur einmal schließen. Danach ist der Dateimerker nicht mehr gültig.

6.1.11 Lösung

Das folgende Programm ist die Lösung des oben genannten Problems:

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <locale.h>
4  int main(void)
5  {
6      double c=47e-6;  /* Kap. */    double r=220e3;  /* Wid. */
7      double t=0.0;    /* Zeit */    double u0=550.0; /* Betriebsspg. */
8      double uc=u0;    /* Kondensator-Spg. */
9      FILE *dm;
10     int rc;
11     setlocale(LC_ALL, "");
12     dm=fopen("ergebnis.txt","w");
13     if(dm==NULL)
14     {
15         perror("ergebnis.txt");
16         return 0;
17     }
18     while(uc>24.0)
19     {
20         uc=u0*(exp(-t/(r*c)));
21         rc=fprintf(dm, "t=%lg\tuc=%lg\n", t, uc);
22         if(rc==EOF)
23         {
24             perror("ergebnis.txt");
25             return 0;
26         }
27         t+=0.5;
28     }
29     fclose(dm);
30     return 0;
31 }
```