6.2 Extras/Lesen aus Dateien

6.2.1 Problem

Aus einer Datei sollen Spannungswerte eingelesen werden. Anschließend sollen Anzahl, Minimum, Maximum, AMW und Standardabweichung ermittelt und ausgegeben werden.

Ohne Dateibefehle kann man Probleme dieser Art auf Konsolenebene mit der Technik der Eingabeumleitung lösen:

```
a.out < spannungswerte.txt
```

Hiermit werden alle Eingabe statt von der Tastatur aus einer Datei gelesen. Manchmal muss man aber in einem Programm aus mehreren Dateien lesen (oder aus einer Datei und von der Tastatur), und dann ist diese Lösung nicht mehr zu gebrauchen.

6.2.2 Dateibefehle

In der Sprache C funktioniert das Lesen aus einer Datei ganz ähnlich wie das Schreiben in eine Datei. Man muss also nicht viel Neues lernen:

a) Öffnen der Datei mit `fopen(dateiname, "r")`

b) Lesen

- Zeichen für Zeichen mit `fgetc()`
- Zahl für Zahl mit `fscanf("%lf")`
- Zeile für Zeile mit `fgets()`

c) Schließen der Datei mit `fclose()`

Der Unterschied zwischen Lesen und Schreiben einer Datei liegt darin, dass man im Normalfall Daten so lange schreiben kann, wie man will[1].

Beim Lesen dagegen hat es nur Sinn, so lange zu lesen, bis die Datei fertig gelesen wurde: Hinter dem Dateiende befinden sich keine sinnvollen Daten mehr, und irgendwann soll das Lesen ja auch beendet sein.

In der Praxis sieht das so aus, dass man so oft lesen muss, bis der Lesebefehl fehlschlägt. Wenn das passiert ist, dann wurde das Dateiende erreicht, oder es ist ein Fehler aufgetreten. *Vor* diesem Fehlschlag kann man das Dateiende nicht erkennen. Das Ergebnis des fehlgeschlagenen Lesebefehls darf man natürlich nicht verwenden.

Was wird gelesen	Befehl	Dateiende oder Fehler bei
Zeichen	`i=fgetc(f);`	`i==-1`
Zahl	`rc=fscanf(f,"%lf",&x);`	`rc==-1`
Zeile	`rs=fgets(str,80,f);`	`rs==NULL`

Tabelle 1: Lesebefehle

Leider hat jeder Lesebefehl in C eine etwas andere Art der Rückmeldung. Abbildung 1 gibt eine Übersicht.

[1] Sollte im Ausnahmefall der Datenträger voll sein, hat das Schreiben eben nicht funktioniert und man gibt eine Fehlermeldung aus.

6.2.3 Beispiel: Zeichen einlesen, Programmstruktur A

Nun stellt sich die Frage, wie eine passende Programmstruktur für das Lesen aus Dateien findet. Am einfachsten ist das Lesen einzelner Bytes bzw. Zeichen aus einer Datei.

Dazu soll hier die Funktion `fgetc` verwendet werden. Als Parameter bekommt sie den Dateimerker. Beim Rückgabewert gibt es zwei Fälle:

a) `EOF` – das Dateiende wurde erreicht, oder ein Fehler ist aufgetreten.

b) Alle anderen Werte – der Rückgabewert ist gleich dem gelesenen Byte bzw. Zeichen.

Und nun das Beispiel: Das Programm `src/zeichenzaehler1a.c` zählt die Zeichen in einer Datei.

```
1  #include <stdio.h>
2  int main(void)
3  {
4      FILE *datei;
5      char dateiname[256]="";
6      int zaehler=0, zeichen; /* zeichen ist int ! */
7
8      printf("Dateiname:");
9      scanf("%255[^\n]", dateiname);
10     datei = fopen(dateiname,"r");
11     if(datei==NULL)
12     {
13         perror(dateiname);
14         return 1;
15     }
16
17     zeichen=fgetc(datei);
18     while(zeichen!=EOF)
19     {
20         /* printf("%c", (char)zeichen); */
21         ++zaehler;
22         zeichen=fgetc(datei);
23     }
24     printf("Die Datei hat %i Zeichen.\n", zaehler);
25     return 0;
26 }
```

Zeile 10 Die Datei soll zum Lesen geöffnet werden, zu diesem Zweck wird der Modus `"r"` (=*read*) benutzt.

Zeile 17 Das erste Zeichen soll eingelesen werden.

Zeile 18 Nur dann, wenn das Dateiende nicht erreicht wurde (und auch kein Fehler aufgetreten ist), wird der Schleifenrumpf ausgeführt, ansonsten ginge es sofort mit Zeile 24 weiter. `EOF` ist hier wieder die symbolische Konstante mit dem Wert `-1`. `EOF` steht zwar für *end of file*; trotzdem kann die Rückgabe von `EOF` auch bedeuten, dass ein Fehler aufgetreten ist.

Zeile 21 Hier sollen die eingelesenen Zeichen gezählt werden. Das ist die Nutzanwendung unseres Programms.

Zeile 22 Das nächste Zeichen soll eingelesen werden. Weiter geht es mit der Abfrage in Zeile 18.

Zeile 24 Ausgabe des Ergebnisses

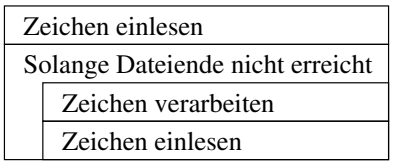

Abbildung 1: Programmstruktur A

Für die Lesevorgänge wird die Struktur aus Abbildung 1 benutzt. Man erkennt, dass hier ein kopfgesteuerte Schleife verwendet wird – und das, obwohl mindestens einmal gelesen werden muss. Das mindestens-einmal-Lesen erreicht man dadurch, dass der erste Lesevorgang bereits vor der Schleife ausgeführt wird. Am Beginn des Schleifenrumpfes wird (beim ersten Durchgang) dieses Zeichen benutzt. Und am Ende (!) des Schleifenrumpfes wird das Zeichen *für den nächsten Durchgang* gelesen.

Bei dieser Programmstruktur ist es etwas lästig, dass man den Lesebefehl zweimal eintippen muss – einmal vor der Schleife und einmal am Ende des Schleifenrumpfes. Zum Ausgleich enthält diese Programmstruktur nur eine einzige Schleife.

6.2.4 Beispiel: Zeichen einlesen, Programmstruktur B

Und hier das Programm `src/zeichenzaehler1b.c`. Es liefert das gleiche Ergebnis, hat aber eine andere Programmstruktur.

```
1  #include <stdio.h>
2  int main(void)
3  {
4      FILE *datei;
5      char dateiname[256]="";
6      int zaehler=0,  zeichen; /* zeichen ist int ! */
7
8      printf("Dateiname:");
9      scanf("%255[^\n]", dateiname);
10     datei = fopen(dateiname,"r");
11     if(datei==NULL)
12     {
13         perror(dateiname);
14         return 1;
15     }
16
17     do{
18         zeichen=fgetc(datei);
19         if(zeichen!=EOF)
20         {
21             /* printf("%c", (char)zeichen); */
22             ++zaehler;
23         }
24     }while(zeichen!=EOF)
25     printf("Die Datei hat %i Zeichen.\n", zaehler);
26     return 0;
27 }
```

Zeile 17 Hier beginnt eine fußgesteuerte Schleife.

Zeile 18 Ein Zeichen soll eingelesen werden.

Zeile 19 Nur dann, wenn das Dateiende nicht erreicht wurde (und auch kein Fehler aufgetreten ist), wird zur Nutzanwendung verzweigt, ansonsten ginge es sofort mit Zeile 24 weiter.

Zeile 22 Hier sollen die eingelesenen Zeichen gezählt werden. Das ist die Nutzanwendung unseres Programms.

Zeile 24 Nur dann, wenn das Dateiende nicht erreicht wurde (und auch kein Fehler aufgetreten ist), wird zum Beginn der Schleife (Zeile 18) zurückgesprungen.

Zeile 25 Ausgabe des Ergebnisses

Hier wird für die Lesevorgänge die Struktur aus Abbildung 2 benutzt. Hier braucht man also

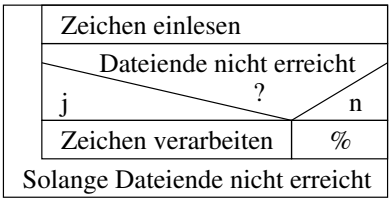

Abbildung 2: Programmstruktur B

nur einen einzigen Einlesebefehl. Allerdings muss man stattdessen die Abfrage für das Dateiende (bzw. den Lesefehler) zweimal eintippen – einmal in der Schleifenbedingung und einmal in der Verzweigung. Viel schlimmer wiegt die Tatsache, dass es passieren kann, dass man die Verzweigung vergisst. Und damit zählt man jedes Mal ein Zeichen zuviel.

6.2.5 Beispiel: Zahlen einlesen mit `fscanf`

Element zum Einlesen	Platzhalter bei fscanf	Beispiel
Zeichen, Byte	`%c`	`fscanf(dm, "%c", &ch);`
Ganze Zahl	`%i`	`fscanf(dm, "%i", &n);`
Gleitkommazahl	`%lf`	`fscanf(dm, "%lf", &x);`
Wort	`%s`	`fscanf(dm, "%79s", s);`
Zeile	`%[^\n]`	`fscanf(dm, "%79[^\n]", z);`
Anzahl von Bytes	`%c`	`fscanf(dm, "%79c", s);`

Tabelle 2: Möglichkeiten für `fscanf`

Die Funktion `fscanf` liest Elemente aus einer Datei. Sie entspricht damit genau der Funktion `scanf`, die von der Tastatur liest. `fscanf` hat aber einen Parameter mehr, nämlich den Dateimerker der Datei, aus der gelesen werden soll. Mit der Funktion `fscanf` kann man daher – wie mit `fscanf` – Zeichen, Zahlen, Worte und ganze Zeilen einlesen, je nach Art des Platzhalters (Abbildung 2).

Beim Rückgabewert gibt es prinzipiell zwei Fälle:

a) `EOF` – Dateiende oder Fehler

b) Sonst – der Rückgabewert ist gleich der Zahl der erfolgreich eingelesenen Platzhalter

Wenn man – wie es sinnvoll ist – mit jedem `scanf`-Befehl nur eine Variable einliest, dann hat man nur einen Platzhalter. Damit kommt man praktisch auf genau drei Fälle:

a) `EOF` (=-1) – Dateiende oder Fehler

b) `0` – Das, was aus der Datei gelesen wurde, konnte nicht so umgewandelt werden, wie der Platzhalter es vorsieht. Dieser Fall tritt auf, wenn eine Zahl eingelesen werden sollte, in der Datei an der Stelle aber nur Buchstaben oder andere unpassende Zeichen stehen. Die Datei hat dann ein falsches Format oder ist „kaputt“. Das Einlesen könnte jetzt mit einer Fehlermeldung abgebrochen werden.

c) `1` – Die Variable konnte erfolgreich eingelesen werden. An der Stelle, auf die der dritte Parameter zeigt, liegt jetzt der eingelesene Wert.

Das folgende Programm `lieszahlen.c` liest Gleitkommazahlen aus einer Datei. Es berechnet die Summe der Zahlen und gibt das Ergebnis aus.

```
1  #include <stdio.h>
2  int main(void)
3  {
4      FILE *datei;
5      char dateiname[]="lieszahlen.txt";
6      int rc;
7      double summe=0.0, x;
8
9      datei = fopen(dateiname, "r"); /* Datei oeffnen */
10     if(datei==NULL)
11     {
12         perror(dateiname);
13         return 1;
14     }
15
16     rc=fscanf(datei, "%lf", &x);
17     while(rc>0)
18     {
19         printf("Gelesener Wert: %lf\n", x);
20         summe = summe + x;
21         rc=fscanf(datei, "%lf", &x);
22     }
23     printf("Summe=%lf\n", summe);
24
25     fclose(datei); /* Datei schliessen */
26     return 0;
27 }
```

Zeile 16 Einlesen der ersten Zahl

Zeile 17 Das Ergebnis wird nur dann verwendet, wenn `fscanf` eine Zahl erfolgreich nach `x` einlesen konnte.

Zeile 19 Nur zur Kontrolle, kann wegfallen

Zeile 20 Der Wert `x` wird verwendet.

Zeile 21 Einlesen der Zahl für den nächsten Durchlauf

Zeile 23 Ausgabe des Ergebnisses

6.2.6 Beispiel: Zeilen einlesen mit `fscanf`

Mit `fscanf` kann man auch eine ganze Zeile in einen String einlesen. Aber: Man muss auf die Stringlänge achten. Das Zeilenende (das Zeichen `'\n'`) wird nicht eingelesen und deshalb auch nicht im String abgelegt. Hier ist ein Beispiel in `lieszeilen1.c`:

```
1  #include <stdio.h>
2  int main(void)
3  {
4      FILE *datei;
5      int rc;
6      char str[21]="";
7      datei=fopen("lieszeilen.txt", "r");
8      if(datei==NULL)
9      {
10         perror("lieszeilen.txt");
11         return 1;
12     }
13
14     rc=fscanf(datei,"%20[^\n]",str);
15     do{rc=fgetc(datei);}while(rc!='\n'&&rc>=0);
16     while(rc>=0)
17     {
18         printf("gelesen:>>%s<<\n",str);
19
20         rc=fscanf(datei,"%20[^\n]",str);
21         do{rc=fgetc(datei);}while(rc!='\n'&&rc>=0);
22     }
23     fclose(datei);
24     return 0;
25 }
```

Zeile 14 Einlesen der ersten Zeile, maximal 20 Zeichen plus Terminator

Zeile 15 Leeren des Eingabepuffers: So lange lesen, bis das Zeilenende oder das Dateiende erreicht wurden. Damit werden zu lange Zeilen einfach kommentarlos abgekürzt ohne Meldung. In jedem Fall muss man mindestens einmal `rc=fgetc(datei)` ausführen, damit das Zeilenende (das Zeichen `'\n'`) aus dem Lesepuffer entnommen wird.

Zeile 16 Nur dann, wenn kein Dateiende und kein Fehler vorliegen, in den Schleifenrumpf springen

Zeile 18 Nutzanwendung: Zeile ausgeben

Zeile 20 Einlesen der Zeile für den nächsten Durchgang

Zeile 21 Leeren des Eingabepuffers

6.2.7 Beispiel: Strings einlesen mit `fgets`

Die Funktion `fgets` dient zum Einlesen einer Zeile aus einer Datei. Mit dieser Funktion *kann* man zu lange Zeilen erkennen. Wenn man will, kann man die fehlenden Teile im nächsten Schritt bearbeiten. Man kann aber auch (wie im obigen Beispiel mit `fscanf`) zu lange Zeilen einfach kommentarlos kürzen.

`fgets` hat die Besonderheit, dass es das Zeilenende (das Zeichen `'\n'`) mit einliest. Das ist einerseits gut, weil man es dann nicht selbst einlesen muss, wie das bei `fscanf` der Fall ist.

Andererseits legt `fgets` das Zeilenende im String ab. Will man es nicht dabeihaben, muss man es dort von Hand löschen, indem man es durch den Terminator ersetzt.

Ein Beispiel zeigt `lieszeilen2.c`.

```
1  #include <stdio.h>
2  #include <string.h>
3  int main(void)
4  {
5      FILE *datei;
6      char *rs;
7      char str[21]="";
8      datei=fopen("lieszeilen.txt", "r");
9      if(datei==NULL)
10     {
11         perror("lieszeilen.txt");
12         return 1;
13     }
14
15     rs=fgets(str,21,datei);
16     if(rs!=NULL)
17     {
18         if(str[0]!='\0' && str[strlen(str)-1]=='\n')
19         {
20             str[strlen(str)-1]='\0';
21             printf("Ganze Zeile ");
22         }
23         printf("gelesen:>>%s<<\n",str);
24
25         rs=fgets(str,21,datei);
26     }
27     fclose(datei);
28     return 0;
29 }
```

Zeile 6 Der Rückgabetyp von `fgets` ist `char *`.

Zeile 15 Die Variable `str` hat die Größe 21. Daher muss hier (anders als bei `fscanf` der Wert 21 eingegeben werden.

Zeile 16 Nur dann, wenn kein Dateiende und kein Fehler vorliegen, in den Schleifenrumpf springen

Zeile 18 Falls das Zeilenende im String liegt, wurde eine ganze Zeile eingelesen.

Zeile 20 Zeilenende durch Terminator `'\0'` überschreiben

Zeile 21 Hier nur als Hinweis

Zeile 23 Nutzanwendung: Zeile ausgeben

Hier werden überlange Zeilen einfach in mehreren Durchläufen ausgegeben. Will man sie stattdessen kürzen, muss man das Programm modifizieren:

```
1      rs=fgets(str,21,datei);
2      if(rs!=NULL)
3      {
4          int rc;
```

```
5         if(str[0]!='\0' && str[strlen(str)-1]=='\n')
6             str[strlen(str)-1]='\0';
7         else
8             do{rc=fgetc(datei);}while(rc!='\n'&&rc>=0);
9         printf("gelesen:>>%s<<\n",str);
10
11        rs=fgets(str,21,datei);
12    }
```

Zeile 8 Lesen (und Wegwerfen der Zeichen) bis zum Zeilenende

Erfahrene Programmierer benutzen zum Einlesen von Zeilen meistens `fgets` und nicht `fscanf`.

6.2.8 Beispiel: Records einlesen

Das Einlesen mehrerer Records in einer großen Programmschleife wird schnell sehr unübersichtlich. Es bietet sich an, top-down zu arbeiten und für das Einlesen eines einzigen Records eine Funktion zu schreiben.

In dieser Funktion werden die einzelnen Felder des Records gelesen. Beim Auftreten des Dateiendes wird die Funktion mit Fehlerwert-Rückgabe verlassen. Konnte dagegen ein ganzes Record eingelesen werden, wird es in den Parameter kopiert und die Funktion mit Erfolgswert-Rückgabe beendet.

Im Programm `liesrecord.c` findet sich ein kurzes Beispiel dazu:

```
1  #include <stdio.h>
2
3  struct messpunkttyp{
4      int nummer;
5      double temperatur;
6  };
7  /*********************************************************/
8  /* Nur ein Record einlesen: */
9  int liesmesspunkt(FILE *datei, struct messpunkttyp *pmesspunkt)
10 {
11     int rc;
12     struct messpunkttyp messpunkt_tmp;
13                                 /* 1. Komponente bearbeiten: */
14     rc = fscanf(datei, "%i", &messpunkt_tmp.nummer);
15     if(rc<0) return 0;
16     do{rc=fgetc(datei);}while(rc!='\n' && rc!=EOF);
17                                 /* 2. Komponente bearbeiten: */
18     rc = fscanf(datei, "%lf", &messpunkt_tmp.temperatur);
19     if(rc<0) return 0;
20     do{rc=fgetc(datei);}while(rc!='\n' && rc!=EOF);
21                                 /* Ergebnis uebertragen: */
22     *pmesspunkt = messpunkt_tmp;
23     return 1;                   /* Erfolg */
24 }
25 /*********************************************************/
26 int main(void)
27 {
28     FILE *datei;
29     int lauf=0, rc;
30     struct messpunkttyp mp;
```

```
31
32      datei = fopen("liesrecord.txt", "r");
33      if(datei==NULL)
34      {
35          perror("liesrecord.txt");
36          return 1;
37      }
38
39      rc=liesmesspunkt(datei, &mp); /* erstes Record einlesen */
40      while(rc!=0)
41      {
42          ++lauf;
43          rc=liesmesspunkt(datei, &mp); /* weiteres Record einlesen */
44      }
45      printf("Es waren %i Messpunkte.\n", lauf);
46      return 0;
47  }
```

6.2.9 Unterscheiden zwischen Dateiende und Fehler

Wenn eine der Lesefunktionen kein Element mehr einliest (erkennbar durch den entsprechenden Rückgabwert, so hat das genau eine von zwei Ursachen:

a) Der Lesezeiger hat das Ende der Datei erreicht.

b) Es ist ein Fehler aufgetreten.

Zustand des Datenstroms	Rückgabe `feof(dm)`	Rückgabe `ferror(dm)`
Element konnte eingelesen werden	0	0
Das Dateiende wurde erreicht	1	0
Ein Fehler ist aufgetreten ile	0	1

Tabelle 3: Verhalten von `feof` und `ferror`

Zur Unterscheidung gibt es zwei Funktionen, nämlich `feof` und `ferror`. `feof` gibt den Wert 1 zurück, wenn das Dateiende erreicht wurde; `ferror` gibt den Wert 1 zurück, wenn ein Fehler aufgetreten ist. Abbildung 3 gibt eine Übersicht.

Man sieht: Falls man den Rückgabewert der letzten Leseoperation noch hat, reicht es aus, eine der beiden Funktionen aufzurufen[2]:

```
1       int zeichen=fgetc(dm);
2       while(zeichen!=EOF)
3       {
4           zeichen=fgetc(dm);
5       }
6       /* ab hier ist zeichen==EOF */
7       if(ferror(dm))
8           perror("Fehler!"); /* es ist ein Fehler aufgetreten */
9       else
10          printf("Das Dateiende wurde erreicht");
```

[2]denn wenn das Dateiende erreicht wurde, ist kein Fehler aufgetreten, und wenn ein Fehler aufgetreten ist, konnte das Dateiende deshalb nicht erreicht werden.