5. Dateien, Ein- und Ausgabe

5.1 Syntax und Standardfunktionen

5.2 Formatierte Aus- und Eingabe

5.3 Beispiele

5.1 Syntax und Standardfunktionen

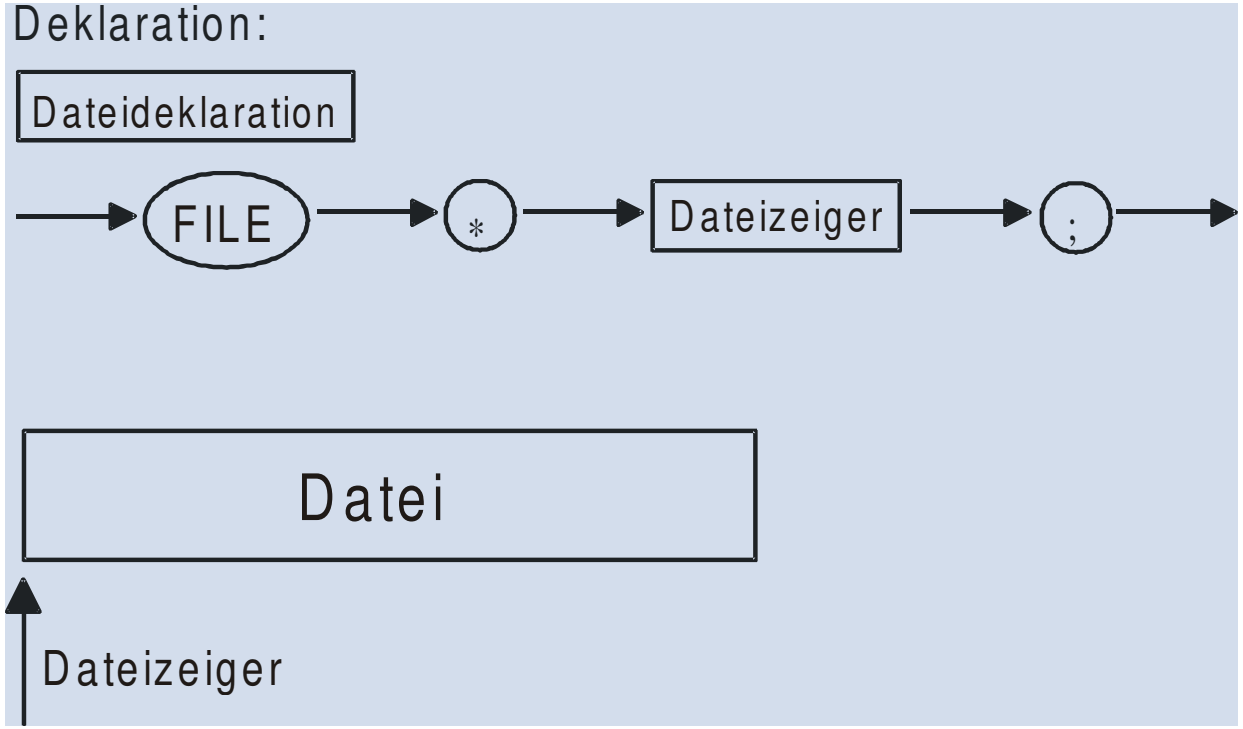
Der Datentyp FILE

Man kann Programme nicht völlig losgelöst von ihrer Umgebung, insbesondere dem Betriebssystem betrachten. Programme arbeiten in der Regel mit Dateien des Betriebssystems (stdin, stdout, stderr, Dateisystem).

Zur interne Repräsentation von Betriebssystem-Dateien dient in C der zusammengesetzte Datentyp `FILE`. `FILE` wird in <stdio.h> definiert. Die Struktur enthält auch Elemente, die den aktuellen Zustand der Datei beschreiben.

Der Zugriff auf Dateien erfolgt über **Dateizeiger**.

Syntax der FILE-Deklaration



Eine Datei im Unix-Dateisystem wird als Zeichenstrom aufgefasst, der sequenziell bearbeitet werden muss.

Standardfunktion

In der Datei <stdio.h> werden einige Funktionen für den Zugriff auf Dateien definiert.

Öffnen einer Datei

```
FILE *fopen(char *file_name, char file_mode)
```

öffnet eine Datei des angegebenen Namens im angegebenen Modus (r = read, w = write, a = append) und gibt einen Zeiger auf diese Datei zurück.

Eine Datei kann u.a. in den drei angegebenen Modi geöffnet werden. Wird eine nicht-existierende Datei zum Schreiben oder Anfügen geöffnet, so wird sie erzeugt. Bei einem Fehler wird der Wert `NULL` zurückgegeben.

Wird eine existierende Datei zum Schreiben geöffnet, geht ihr Inhalt verloren!

Schließen einer Datei

```
int fclose(FILE *file_pointer)
```

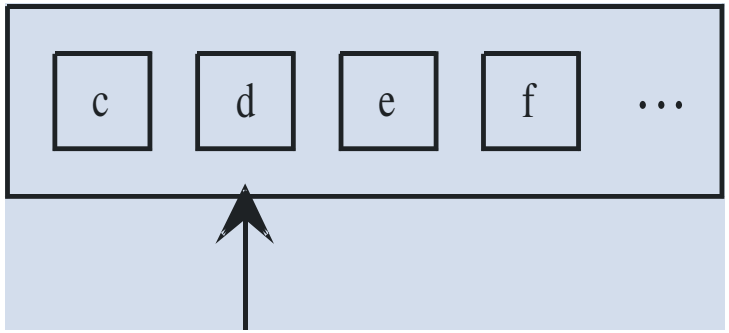
schließt die Datei, auf die der Zeiger zeigt.

Einfaches Beispiel für `open` und `close`

```
file *fp;
fp = fopen("Noten.dat","w");
if (fp == Null)
        ....
fclose(fp);
```

Lesen aus Dateien: `fgetc` (1)

```
int fgetc(FILE *file_pointer)
```



Lesen aus Dateien: `fgetc` (2)

Die Funktion `fgetc` liefert das nächste Zeichen aus der Datei, welche durch den `file_pointer` identifiziert wird. Am Dateiende oder bei Fehler ist das Resultat EOF (= -1) (EOF bedeutet “end of file“).

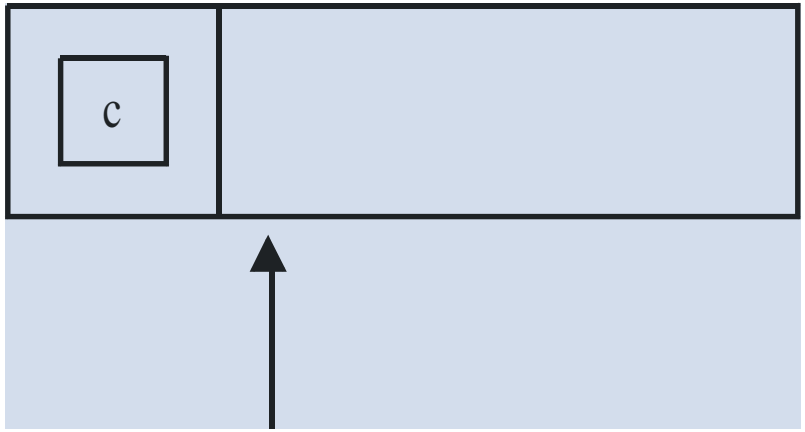
Die symbolische Konstante EOF ist in <stdio.h> definiert und hat typischerweise den Wert -1. Dies ist auch der Grund, warum `fgetc` als `int` und nicht als `char` definiert wird.

Beispiel

```
char z;
   .
   .
z = fgetc(fp);
```

Schreiben in Dateien: `fputc`

```
int fputc(int c, FILE *file_pointer)
```



`fputc` gibt das Zeichen c in die Datei aus und liefert bei Erfolg c als Resultat, bei Fehler EOF.

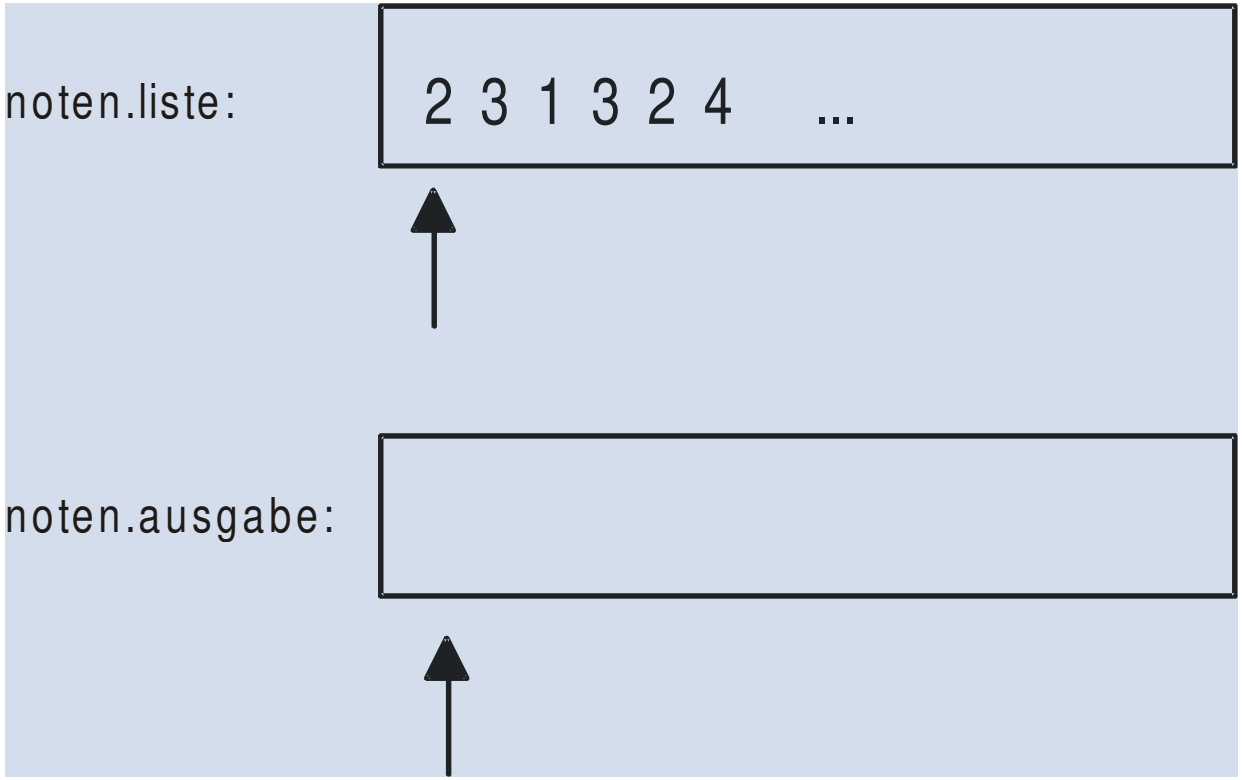
Beispiel

```
int r;
r = fputc('x',fp);
```

Beispiel für `fputc` (1)

```
FILE *datei1, *datei2;
int note;     /*  wir behandeln nur ganzzahlige Noten */
...
datei1 = fopen("noten.liste","r");
datei2 = fopen("noten.ausgabe","w");
while ((note = fgetc(datei1)) != EOF)
   fputc(note,datei2);
fclose(datei1);
fclose(datei2);
```

Beispiel für `fputc` (2)



Betriebssystemdateien (1)

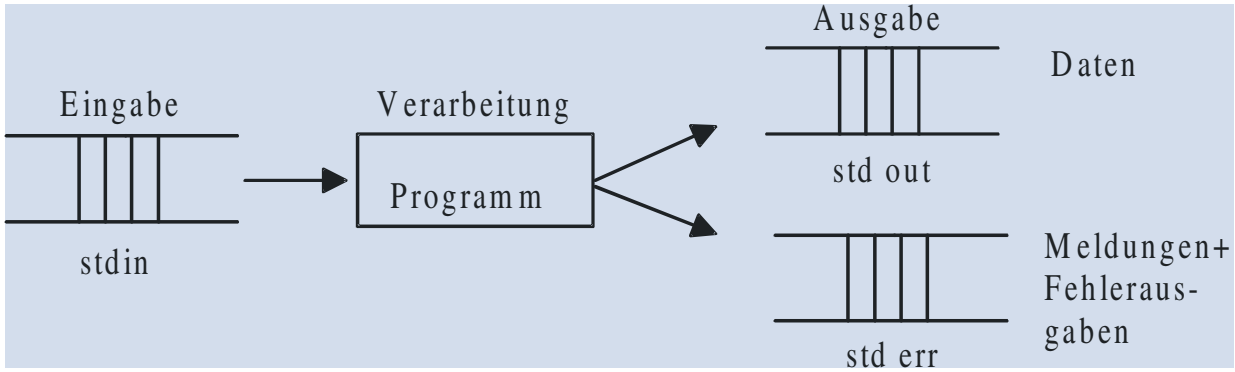
Vom Betriebssystem werden drei Standarddateien bereitgestellt:

1) stdin - standard input file (Standardeingabe)

2) stdout - standard output file (Standardausgabe)

3) stderr - standard error file (Standardfehlerausgabe)

In der Regel wird 1) von der Tastatur, 2) + 3) vom Bildschirm übernommen:



Betriebssystemdateien (2)

Die Vereinbarung der Standarddateien geschieht in <stdio.h>.

```
FILE *stdin,*stdout,*stderr
```

sind Zeiger auf die Dateien und können als solche verwendet werden.

Lesen und Schreiben einzelner Zeichen

Lesen eines einzelnen Zeichens

In <stdio.h> ist hierfür ein Makro definiert:

```
#define getchar()    fgetc(stdin)
```

Schreiben eines Zeichens auf den Bildschirm

Weiteres Makro in <stdio.h>:

```
#define putchar(c) fputc((c),stdout)
```

5.2 Formatierte Aus- und Eingabe

Formatierte Ausgabe

```
int fprintf(FILE *file_ptr, char *format, \ arg1, arg2,
...)
```

schreibt eine Zeichenkette auf einmal in die angegebene Datei. Dabei definiert die Zeichenkette "`format`" das Ausgabeformat der auszugebenden Datenwerte, die in den Variablen der Argumentenliste stehen.

Beispiel

```
fprintf(datei2,"Dies ist die erste Note: %d\n", note);
```

schreibt in datei2 eine Zeile mit dem angegebenen Text inklusive des Inhalts der Variablen "note". Das Zeichen ‘\n’ steht für Zeilenende.

Das Zeichen ‘%d’ steht als Platzhalter für die Variable "note". An dieser Stelle soll der Dezimalwert von "note" eingefügt werden, bevor die Zeichenkette "format" in die Datei datei2 ausgegeben wird.

Formatierte Eingabe

```
int fscanf(FILE *file_pointer, char *format, &arg1, \
&arg2, ...)
```

kann analog zu `fprintf` mehrere Zeichen gleichzeitig lesen. `fscanf` liest durch Zwischenraumzeichen (“blanks“) getrennte Zeichen aus der Datei `file_pointer`, interpretiert sie unter Kontrolle von "`format`" und legt die Resultate in den Variablen der Argumentenliste ab.

Merke

Bei `fscanf` müssen **Zeiger** auf die Variablen übergeben werden, damit ihr Inhalt geändert werden kann (call by reference).

`fscanf` gibt die Anzahl erfolgreich erkannter und zugewiesener Eingabefelder als Funktionswert zurück.

Beispiel für die formatierte Eingabe

In der Datei `datei1` seien immer drei Zahlen in einer Zeile gespeichert, die gemeinsam gelesen werden sollen:

```
fscanf(datei1, "%d %d %d\n", &note1, &note2, &note3);
```

Das Zeichen '%d' steht nun als Ersatz für dezimale Zeichen in der eingelesenen Zeichenkette, deren Wert an die Variablen `note1`, `note2` und `note3` übergeben werden.

Merke

`fprintf` und `fscanf` haben eine variable Anzahl von Argumenten! Diese ist abhängig von der Zahl von Variablen, deren Inhalt geschrieben/gelesen werden soll.

Umwandlungsschablonen (1)

Jede Umwandlungsschablone beginnt mit dem Zeichen ‘%’ und endet mit einem **Umwandlungszeichen**. Dazwischen sind folgende Formathilfen möglich (in der angegebenen Reihenfolge):

-	Linksausrichtung
<Zahl>	minimale Feldbreite
.<Zahl>	Genauigkeit
h	short
l	long

Umwandlungsschablonen (2)

Die wichtigsten Umwandlungszeichen sind:

d	Dezimalzahl (int)
o	oktale Zahl ohne Vorzeichen
x	hexadezimale Zahl ohne Vorzeichen
u	Dezimalzahl ohne Vorzeichen (unsigned int)
c	einzelnes Zeichen (char)
s	Zeichenkette (char*)
f,e	Gleitkommazahl (double)
p	Zeiger (void*)

Hinweis: `%%` gibt ein einfaches %-Zeichen aus.

Beispiele

`%-6.8lf`	(long double)
`%-5hd`	(short int)

Zwischenraumzeichen (1)

`fscanf` überliest bei der Zuweisung von Zeichen aus der Datei alle Zwischenraumzeichen zwischen zwei Eingaben, um zum nächsten konkreten Zeichen zu kommen.

Als Zwischenraumzeichen gelten in C:

	Leerzeichen
\t	Tabulatorzeichen
\n	Zeilentrenner
\r	Wagenrücklauf
\f	Seitenvorschub

Zwischenraumzeichen (2)

Beispiel

```
int i, return_value;
char c;
char string[15];
return_value = fscanf(stdin, \
   "%d %5s %c", &i, &string, &c)
```

Nun wird über die Betriebssystem-Datei stdin (in der Regel die Tastatur) folgende Zeile eingegeben:

45 Eingabe x

Als Ergebnis erhält man folgende Zuweisungen:

i = 45;
string = "Eing";
c = ‘x’;
return_value = 3;

Weitere Standardfunktionen (1)

In <stdio.h> sind u.a. noch folgende Vereinfachungen definiert:

```
#define   printf(...)   fprintf(stdout, ...)
#define   scanf(...)fscanf(stdin, ...)
```

Außerdem folgende Funktionen:

```
int sprintf(char* string, char* format,...)
```

gibt die Ausgabe in die Zeichenkette "string" aus.

```
int sscanf(char* string, char* format,...)
```

liest Daten von der Zeichenkette "string" nach dem angegebenen Format in die Variablen ein.

Weitere Standardfunktionen (2)

`sscanf` wird oft dazu verwendet, Eingaben aus Dateien zu verarbeiten, deren Format nicht exakt festgelegt ist. Dazu werden Dateiinhalte zeilen-weise in Zeichenketten-Variablen gelesen. Anschließend wird versucht, die Zeichenkette mittels `sscanf` zu zerlegen.

```
char* fgets(char* line, int maxchar, \
  FILE* file_pointer)
```

liest aus der Datei `file_pointer` die nächste Eingabezeile, aber maximal maxchar-1 Zeichen.

```
int fputs(char* line, FILE* file_pointer)
```

schreibt eine Zeichenkette in die Datei file_pointer

5.3 Beispiele

Beispiel 1: sequentielles Lesen

Ermitteln der Anzahl der männlichen und weiblichen Personen aus einer Personal-Datei:

```
typedef struct {
   char name[15], vorname[15];
   char geschlecht;
} PERSON;
```

(Fortsetzung auf der nächsten Seite)

Beispiel 1: sequentielles Lesen (2)

```
int main()
{
  File *datei;
  PERSON person;
  int wzahl, mzahl;
  wzahl = mzahl = 0;
  if((datei = fopen("Personal.Datei", "r")) == NULL)
   {
   fprintf (stderr,"error opening Personal.Datei \
   \n");
     return-1;
   }
```

Beispiel 1: sequentielles Lesen (3)

```
while (fscanf(datei,"%15s %15s %c", person.name, \
        person.vorname, &person.geschlecht) != EOF)
      {
        if (person.geschlecht == 'w')
              wzahl++;
        else
              mzahl++;
       }
       fclose(datei);
       fprintf(stdout,"wzahl: %d,mzahl: %d \n";  \
             wzahl, mzahl);
       return 0;
}
```

Beispiel 2: Einfügen an einer bestimmten Stelle (1)

Ein Einfügen in die Mitte einer Datei ist nicht zulässig! Deshalb: Anlegen einer neuen Datei und Umkopieren.

```
#include <stdio.h>
typedef struct {
   char name[15], vorname[15];
   char geschlecht;
} PERSON;
```

(Fortsetzung auf der nächsten Seite)

Beispiel 2: "Einfügen" an einer bestimmten Stelle (2)

```
int main()
{
  FILE *original, *kopie;
  PERSON person, einfueg = {"Kamel", "Ernst", 'm'};
  int ret_val;
  original = fopen( "Personal.Datei", "r");
  kopie    = fopen("Personal.Kopie"; "w");

  /* Bis zum Einfuegepunkt kopieren */
  while (((ret_val = fscanf (original,  \
        "%15s %15s %c", person.name, \
            person.vorname, &person.geschlecht)) \
            != EOF) && \
            (strcmp (person.name,einfueg.name) < 0))
        fprintf (kopie, "%s %s %c\n",  \
            person.name, person.vorname, \
            person.geschlecht);

    /* einzufuegende Person einfuegen */
  fprintf (kopie, "%s %s %c\n", einfueg.name,einfueg.vorname, einfueg.geschlecht);

    /* zuletzt gelesene Person noch einfuegen */
    if (ret_val != EOF)
        fprintf (kopie, "%s %s %c\n", person.name,
person.vorname, \ person.geschlecht);
```

(Fortsetzung auf der nächsten Seite)

Beispiel 2: "Einfügen" an einer bestimmten Stelle (3)

```
 /* Rest der Datei kopieren */
  while (fscanf (original, "%15s %15s %c", person.name, \
                 person.vorname, &person.geschlecht))
      fprintf (kopie, "%s %s %c\n", \
      person.name, person.vorname,  \
      person.geschlecht);
    fclose(original);
    fclose(kopie);
    return 0;
}
```

Beispiel 3: Verschmelzen von Dateien (1)

Zwei aufsteigend sortierte Dateien sollen in eine einzige aufsteigend sortierte Datei verschmolzen werden.

Zunächst wird eine Routine beschrieben, die den Rest einer Datei in eine andere kopiert:

```
void restuebertrage(char *c, FILE *von, \   FILE *nach)
{
   while (*c != EOF)
   {
      putc(*c, nach);
      *c = getc(von);
   }
   return;
}
```

Beispiel 3: Verschmelzen von Dateien (2)

Damit kann die Routine, die zwei Dateien zu einer dritten mischt, erstellt werden:

```
void mische(FILE *datei1, FILE *datei2, FILE *datei3)
{
    char z1, z2;
    datei1 = fopen("Eingabe1", "r");
    datei2 = fopen("Eingabe2", "r");
    datei3 = fopen("Ausgabe", "w");
    z1 = getc(datei1);
    z2 = getc(datei2);
    while ((z1 != EOF && z2 != EOF)) {
        if (z1 < z2) {
            putc(z1,datei3);
            z1 = getc(datei1);
        } else {
            putc(z2,datei3);
            z2 = getc(datei2);
        }
    }
```

(Fortsetzung auf der nächsten Seite)

Beispiel 3: Verschmelzen von Dateien (3)

```
    restuebertrage (&z1,datei1,datei3);
    restuebertrage (&z2,datei2,datei3);
    fclose(datei1);
    fclose(datei2);
    fclose(datei3);
}
```