Programmieren in C
für Elektrotechniker

Kapitel 9: Ein-/Ausgabe

- **Dateisystem**

Ein-/Ausgabe, Dateisystem

- **E/A-Konzept in UNIX und C**
  - UNIX (und damit auch C) verwendet Datenströme (streams) als Konzept zur Verbindung von Programmen mit E/A-Geräten und Dateien
    → einheitliches Architekturkonzept.
  - Datenströme für Geräte und Dateien abstrahieren vom aktuellen Betriebssystem.
  - Kein Standard für grafische Benutzeroberflächen
    (→ spezielle C-Bibliotheken; nicht in dieser Vorlesung).
  - **`<stdio.h>`** enthält nützliche Funktionen für (nur) Text-E/A.

  - Dateisystem
    Teil eines Betriebssystems zur Verwaltung eines Massenspeichers
    (z.B. Festplatte)
    → nicht jedes Betriebssystem besitzt ein Dateisystem.

Ein-/Ausgabe, Dateisystem

E/A-Konzept in UNIX und C

- **Datenstrom (Datei)**
  Mit einem Namen versehener Datensatz beliebiger Länge, der aus Bytes besteht.
  - Byte-Array
  - Interpretation des Byte-Stroms als komplexere Strukturen erfolgt durch den Anwender *(→ Verwendung von Bibliotheksfunktionen)*
- file pointer
  → Zugriff auf eine Datei *(oder ein Gerät)* über eine Dateivariable.
- file pointer wird beim Öffnen einer Datei definiert, z.B. `fp = fopen("datei.txt", "w");`
  - **`fp`** realisiert einen Kanal zwischen Programm und Festplatte.

Standard-Ein- und -Ausgabe

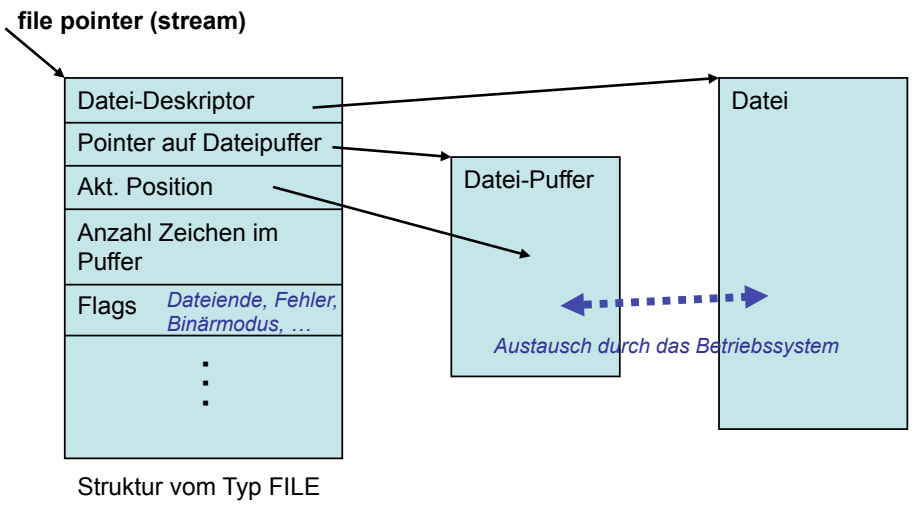
- Das Laufzeitsystem des C-Compilers kennt 3 Standardkanäle
  - Standard-Eingabe *(typ. Tastatur)*: globaler file pointer **`stdin`**
  - Standard-Ausgabe *(typ. Bildschirm)*: globaler file pointer **`stdout`**
  - Standard-Fehlerausgabe *(typ. Bildschirm)*: globaler file pointer **`stderr`**
    Beispiel: `fprintf (stderr, "Fehlerausgabe");`
- Standardkanäle können über das Betriebssystem auf Dateien umgelenkt werden:
  - `myprog > test.out`
    Ausgaben *(z.B. über `prinf()`)* werden in **`test.out`** statt auf den Bildschirm geschrieben.
  - `myprog < test.in`
    Eingaben *(z.B. über `scan()`)* werden aus **`test.in`** statt von der Tatstatur gelesen.
  - `myprog <test.in > test.out`
    Umlenkung der Ein- und Ausgabe.

Standard-Ein- und -Ausgabe

- Die Standard-Fehlerausgabe kann davon unabhängig umgelenkt werden
  - `myprog 2> eror.log`
  - Fehlerausgabe sollte immer über **`stderr`** geschrieben werden, damit die Umlenkung von **`stdout`** auf eine Datei weiterhin Fehlermeldungen auf den Bildschirm erlaubt.
- Zwei Funktionsklassen für die Standard-E/A
  - High-Level-Funktionen:
    „verstecken“ das Betriebssystem *(z.B. **`printf()`**, **`scanf()`**)*.
  - Low-Level-Funktionen:
    elementare, betriebssystemabhängige Funktionen *(nicht in der Vorlesung)*
    → Programm nicht mehr unbedingt portabel.

file pointer und Dateipuffer

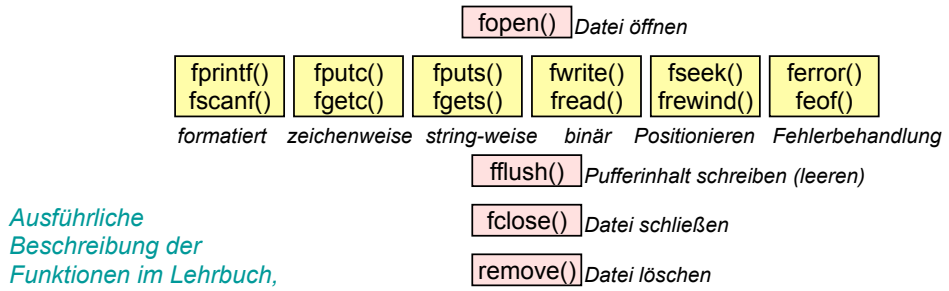
- High-level Dateizugriffe sind immer gepuffert, d.h. die Funktionen greifen auf einen Puffer zwischen Programm und Datei zu.
- Der Austausch zwischen Puffer und Datei erfolgt über das Betriebssystem.



Struktur vom Typ FILE

▪ High-Level Dateizugriffe

- **<stdio.h>** bietet mehrere Funktionsklassen für Dateizugriffe



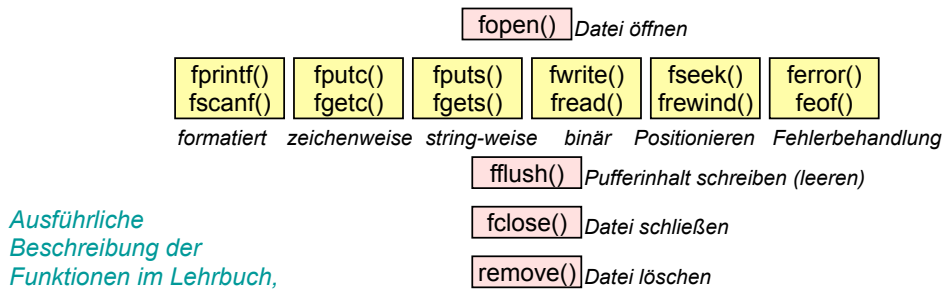
Ausführliche Beschreibung der Funktionen im Lehrbuch, Kapitel 14-8.

- Formatierte Ein-/Ausgabe, z.B.
  - `int fprintf (FILE * stream, const char * format, ...);`
  - `int fscanf (FILE * stream, const char * format, ...);`

  → *ausführliche Beschreibung der Funktionen **`fprintf`** und **`fscanf`** mit allen Formatangaben im Lehrbuch Kapitel 14.7.*

▪ High-Level Dateizugriffe

- **<stdio.h>** bietet mehrere Funktionsklassen für Dateizugriffe



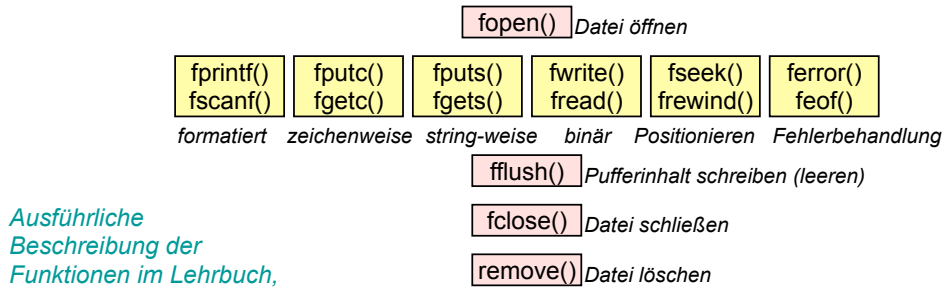
Ausführliche Beschreibung der Funktionen im Lehrbuch, Kapitel 14-8.

- Zeichenweise Ein-/Ausgabe, z.B.
  - `int fputc (int c, FILE * stream);`
  - `int fgetc (FILE * stream);`
- String-weise Ein-/Ausgabe, z.B.
  - `int fputs (const char * s, FILE * stream);`
  - `char * fgets (char * s, int max, FILE * stream);`

→ Rückgabe des ein-/ausgegebenen Zeichens bzw. ein Fehlercode **`EOF`** (**`fgets`**: **`NULL`**) *(→ Fehler-Flag im stream, s.u.).*

▪ High-Level Dateizugriffe

- **<stdio.h>** bietet mehrere Funktionsklassen für Dateizugriffe



Ausführliche Beschreibung der Funktionen im Lehrbuch, Kapitel 14-8.

- Blockweise Ein-/Ausgabe, z.B.
  - `size_t fwrite (const void * ptr, size_t size, size_t nmemb, FILE * stream);`
  - `size_t fread (void * ptr, size_t size, size_t nmemb, FILE * stream);`

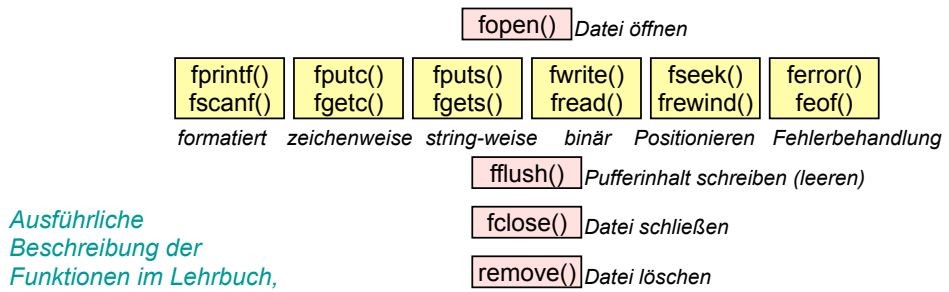
→ E/A eines Arrays *(**`nmemb`** Objekte der Größe **`size`**)*, auf das **`ptr`** zeigt.

→ *Rückgabe der korrekt verarbeiteten Objekte; im Fehlerfall weniger als **`nmemb`**.*

Programmieren in C Ein-/Ausgabe

▪ High-Level Dateizugriffe

- **<stdio.h>** bietet mehrere Funktionsklassen für Dateizugriffe

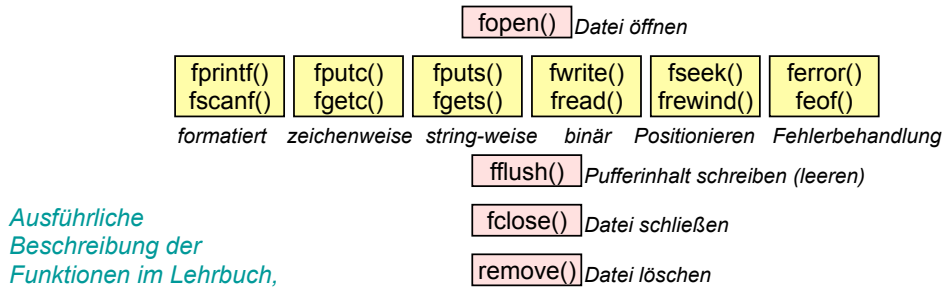


Ausführliche Beschreibung der Funktionen im Lehrbuch, Kapitel 14-8.

- file pointer positionieren, z.B.
  - `int fseek (FILE * stream, long offset, int whence);`
    setzt den file pointer der durch **`stream`** definierten Datei auf die Position, die **`offset`** Bytes von **`whence`** entfernt ist *(**`whence`**: Dateianfang/akt. Position/Dateiende)*.
  - `void rewind (FILE * stream);`
    setzt den file pointer an den Dateianfang.

High-Level Dateizugriffe

- **<stdio.h>** bietet mehrere Funktionsklassen für Dateizugriffe



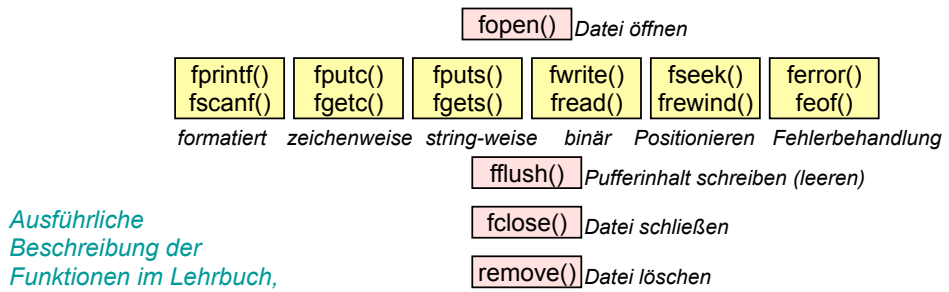
Ausführliche Beschreibung der Funktionen im Lehrbuch, Kapitel 14-8.

- Fehlerbehandlung, z.B.
  - `int ferror (FILE * stream);`
    Gibt zurück, ob das Fehler-Flag im file pointer gesetzt ist.
  - `int feof (FILE * stream);`
    Gibt zurück, ob das Flag **EOF** im file pointer gesetzt ist.

Programmieren in C Ein-/Ausgabe

High-Level Dateizugriffe

- **<stdio.h>** bietet mehrere Funktionsklassen für Dateizugriffe



Ausführliche Beschreibung der Funktionen im Lehrbuch, Kapitel 14-8.

- Dateioperationen, z.B.
  - `int fflush (FILE * stream);`
    Alle noch nicht geschriebenen Daten im Dateipuffer werden in die Datei geschrieben.
  - `int fclose (FILE * stream);`
    Daten werden in die Datei geschrieben und diese geschlossen. Noch vorhandene Eingabedaten werden gelöscht.
    Rückgabe: `0` oder **EOF**.
  - `int remove (const char *filename);`
    Löscht die angegebene Datei.

Beispiel

- Anfügen einer Zeile zu einer Datei und Ausgabe des Dateiinhalts.

```
#include <stdio.h>
#define STR_LEN 80

int main (void) {
    char str [STR_LEN];
    FILE * fp;
    const char * const filename = "bsp.txt";

Datei schreibend öffnen
    if ((fp = fopen (filename, "a")) == NULL) {
Fehlerbehandlung
        fprintf (stderr, "Datei '%s' konnte nicht zum Anhaengen"
                         " geoeffnet werden!\n", filename);
        return 1;
    }
Daten anhängen
    fprintf (fp, "Noch eine Zeile...\n");
Datei schließen
    fclose (fp);
Datei lesend öffnen
    if ((fp = fopen (filename, "r")) == NULL) {
Fehlerbehandlung
        fprintf (stderr, "Datei '%s' konnte nicht zum Lesen"
                         "geoeffnet werden!\n", filename);
        return 1;
    }
Ausgabe
    while (fgets (str, STR_LEN, fp)) printf (str);
    fclose (fp);
    return 0;
}
```


Einige weitere Funktionen aus <stdio.h>

- `int remove (const char *filename);`
  → Löscht Datei „**`filename`**“ aus dem Dateisystem.
- `int fflush (FILE * stream);`
  → Schreibt den Inhalt des Dateipuffers in die Datei.
- `int fseek (FILE * stream, long offset, int whence);`
  → Setzt Dateipositionszeiger auf die Position, die **`offset`** Bytes von **`whence`** (→ Dateianfang/akt. Position/Dateiende) entfernt ist.
- `void rewind (FILE * stream);`
  → Setzt Dateipositionszeiger an den Dateianfang.
- `int feof (FILE * stream);`
  → Überprüft das Dateiende-Flag.
- `int ferror (FILE * stream);`
  → Überprüft das Fehler-Flag.
- `void clearerr (FILE * stream);`
  → setzt das Dateiende- und das Fehlerflag zurück.

Beispiel

- Leeren des Dateipuffers *(zur Vorsicht).*

```
#include <stdio.h>
#define STR_LEN 80

int main (void) {
    FILE * fp;
    const char * const filename = "bsp.txt";

    if ((fp = fopen (filename, "a")) == NULL) {
        fprintf (stderr, "Datei '%s' konnte nicht zum Anhaengen"
                         " geoeffnet werden!\n", filename);
        return 1;
    }
    fprintf (fp, "Wichtige Information...\n");

    if (fflush (fp) != 0) {
        fprintf (stderr, "Fehler bei fflush. "
            "Achtung: Daten in %s nicht gesichert!\n", filename);
        return 1;
    }
    {
        /* Schlecht programmiert ... Provozierter Absturz. */
        int i = 0, j = 1 / i;
        /* Datei ist durch fflush() gesichert. */
    }
    return 0;
}
```


Beispiel

- Fehlerausgabe und Zurücksetzen der Fehler-Flags.

```
#include <stdio.h>

int main (void) {
    int c;
    int err;

    putc ('c', stdin);
    if ((err = ferror (stdin))) {
        fprintf (stderr, "Provozierter Fehler %d.\n", err);
        clearerr (stdin);
    }
    printf ("Bitte weitere Eingaben... Immer noch Fehler?\n");

    c = getc (stdin);
    if ((err = ferror (stdin))) {
        fprintf (stderr, "Ja, echter Fehler %d,\n", err);
        clearerr (stdin);
    } else {
        printf ("Nein, alles OK.\n");
    }
    return 0;
}
```