

# Übungen zu Systemnahe Programmierung in C (SPiC)

Sebastian Maier  
(Lehrstuhl Informatik 4)

## Übung 6



Wintersemester 2016/2017



## Inhalt

---

Dateien & Dateikanäle

Dateikanäle

Ein-/Ausgaben

Aufgabe: trac

Aufgabenbeschreibung

Nützliche Bibliotheksfunktionen

Mögliche Fehler

Fehlerbehandlung getchar()

Kommandozeilenparameter

Hands-on: concat



## Dateien & Dateikanäle

Dateikanäle

Ein-/Ausgaben

Aufgabe: trac

Hands-on: concat



## Dateikanäle

---

- Ein- und Ausgaben erfolgen über gepufferte Dateikanäle
- `FILE *fopen(const char *path, const char *mode);`
  - öffnet eine Datei zum Lesen oder Schreiben (je nach mode)
  - liefert einen Zeiger auf den erzeugten Dateikanal
    - `r`   Lesen
    - `r+`   Lesen & Schreiben
    - `w`   Schreiben; Datei wird ggf. erstellt oder Inhalt ersetzt
    - `w+`   Lesen & Schreiben; Datei wird ggf. erstellt oder Inhalt ersetzt
    - `a`   Schreiben am Ende der Datei; Datei wird ggf. erstellt
    - `a+`   Schreiben am Ende der Datei; Lesen am Anfang; Datei wird ggf. erstellt
- `int fclose(FILE *fp);`
  - schreibt ggf. gepufferte Ausgabedaten des Dateikanals
  - schließt anschließend die Datei



- standardmäßig geöffnete Dateikanäle
  - `stdin` Eingaben
  - `stdout` Ausgaben
  - `stderr` Fehlermeldungen
- `int fgetc(FILE *stream);`
  - liest ein einzelnes Zeichen aus der Datei
- `char *fgets(char *s, int size, FILE *stream);`
  - liest max. size Zeichen in einen Buffer ein
  - stoppt bei Zeilenumbruch und EOF
- `int fputc(int c, FILE *stream);`
  - schreibt ein einzelnes Zeichen in die Datei
- `int fputs(const char *s, FILE *stream);`
  - schreibt einen null-terminierten String (ohne das Null-Zeichen)



## Inhalt

---

### Dateien & Dateikanäle

#### Aufgabe: trac

- Aufgabenbeschreibung
- Nützliche Bibliotheksfunktionen
- Mögliche Fehler
- Fehlerbehandlung `getchar()`
- Kommandozeilenparameter

#### Hands-on: concat



## ■ TRAC: TRanslate Arbitrary Characters

```
1 $ ./trac <FIND> <REPLACE>
2
3 $ ./trac aie bei    # a -> b; i -> e; e -> i
4 dies ist ein Test  # Eingabe
5 deis est ien Tist  # Ausgabe
```

## ■ Ähnlich wie das Kommando tr(1) in Unix-artigen Betriebssystemen

## ■ Programmablauf

- Kommandozeilenparameter prüfen
  - ggf. Fehlermeldung ausgeben und Programm beenden
- Zeichen einlesen von stdin bis EOF oder Fehler
  - Zeichen ggf. entsprechend Mapping ersetzen
  - Zeichen ausgeben
- evtl. Fehler erkennen und Fehlermeldung ausgeben
- Programm beenden (mit entsprechendem exit-Status)



# Nützliche Bibliotheksfunktionen (1)

## ■ `size_t strlen(const char *s);`

- Länge einer Zeichenfolge bestimmen

→ `strlen(3)`

## ■ `int getchar(void);`

- Einzelnes Zeichen von stdin einlesen

→ `getchar(3)`

## ■ `int putchar(int c);`

- Einzelnes Zeichen auf stdout ausgeben

→ `putchar(3)`

## ■ `int fprintf(FILE *stream, const char *format, ...);`

- Formatierte Ausgaben auf einem Dateikanal (z.B. stderr)

→ `fprintf(3)`



## Nützliche Bibliotheksfunktionen (2)

- `int ferror(FILE *stream);`
  - Fehlerzustand eines Dateikanals abfragen
  - `ferror(3)`
  
- `void perror(const char *s);`
  - gibt die übergebene Zeichenfolge gefolgt von einer Beschreibung des letzten Fehlers auf `stderr` aus
  - nutzt die globale `errno` Variable (⇒ nur für Bibliotheksfehler)
  - `perror(3)`
  
- `void exit(int status);`
  - Beenden des Programms mit übergebenem `exit`-Status
  - `exit(3)`



## Mögliche Fehler

1    `$ ./trac <FIND> <REPLACE>`

- Bedienungsfehler
  - falsche Anzahl an Argumenten
  - unterschiedliche Länge von `FIND` und `REPLACE`
  - selbes Zeichen kommt mehrfach in `FIND` vor
  - ⇒ Erklärung der Verwendung auf `stderr` ausgeben und Programm beenden
  
- Ein-/Ausgabefehler
  - Zugriff auf eine Datei ist nicht möglich (umgeleiteter Dateikanal)
  - ⇒ Fehler auf `stderr` ausgeben und Programm beenden



```
1 int c;
2 while ((c=getchar()) != EOF) {
3     ...
4 }
5
6 /* EOF oder Fehler? */
7 if(ferror(stdin)) {
8     /* Fehler */
9     ...
10 }
```

- „fgetc(), getc() and getchar() return the character read as an unsigned char cast to an int **or EOF on end of file or error.**”
- Wie kann man den Fehlerfall von EOF unterscheiden?  
⇒ ferror(3)



## Wiederholung: Kommandozeilenparameter

```
1 ...
2 int main(int argc, char *argv[]){
3     strcmp(argv[argc - 1], ... )
4     ...
5     return EXIT_SUCCESS;
6 }
```

- Übergabeparameter:
  - main() bekommt vom Betriebssystem Argumente
  - argc: Anzahl der Argumente
  - argv: Vektor aus Strings der Argumente (Indices von 0 bis argc-1)
- Rückgabeparameter:
  - Rückgabe eines Wertes an das Betriebssystem
  - Zum Beispiel Fehler des Programms: return EXIT\_FAILURE;



Dateien & Dateikanäle

Aufgabe: trac

Hands-on: concat



## Hands-on: concat

---

- Konkateniert den Inhalt mehrerer Dateien  
⇒ Übergebene Dateien öffnen & nacheinander auf `stdout` ausgeben
- Hilfreiche Funktionen:
  - `fopen(3)` ⇒ Öffnen der Datei
  - `fgetc(3)` ⇒ Einlesen einzelner Zeichen
  - `fputc(3)` ⇒ Ausgeben einzelner Zeichen
  - `fclose(3)` ⇒ Schließen der Datei
- Sinnvolle Fehlerbehandlung beachten
  - Fehlende Dateien melden und überspringen
  - Fehlermeldungen auf `stderr` ausgeben
- Erweiterung
  - Behandlung von „-“ Zeichen als speziellen Parameter - vgl. `cat(1)`  
⇒ Zeichen von `stdin` einlesen (und auf `stdout` ausgeben) bis EOF
  - Verwendung: `./concat test1.txt - test2.txt`

