

Systemnahe Programmierung in C (SPiC)

22 Ergänzungen – Ein-/Ausgabe

Jürgen Kleinöder, Daniel Lohmann, Volkmar Sieh

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Sommersemester 2022

<http://sys.cs.fau.de/lehre/SS22/spic>



- E/A-Funktionalität nicht Teil der Programmiersprache
- Realisierung durch „normale“ Funktionen
 - Bestandteil der Standard-Bibliothek
 - einfache Programmierschnittstelle
 - effizient
 - portabel
 - betriebssystem-nah
- Funktionsumfang
 - Öffnen/Schließen von Dateien
 - Lesen/Schreiben von Zeichen, Zeilen oder beliebigen Datenblöcken
 - formatierte Ein-/Ausgabe



Standard-Ein-/Ausgabe

Jedes C-Programm erhält beim Start automatisch 3 E/A-Kanäle:

stdin: Standard-Eingabe

- normalerweise mit der Tastatur verbunden
- „Dateiende“ (EOF) wird durch Eingabe von CTRL-D am Zeilenanfang signalisiert
- bei Programmaufruf in der Shell auf Datei umlenkbar

```
~> prog < eingabedatei
```

stdout: Standard-Ausgabe

- normalerweise mit Bildschirm (bzw. dem Fenster in dem das Programm gestartet wurde) verbunden
- bei Programmaufruf in der Shell auf Datei umlenkbar

```
~> prog > ausgabedatei
```

stderr: Ausgabekanal für Fehlermeldungen

- normalerweise ebenfalls mit Bildschirm verbunden



■ Pipes

- Die Standardausgabe eines Programmes kann mit der Standardeingabe eines anderen Programms verbunden werden:

```
~> prog1 | prog2
```

Die Umlenkung von Standard-E/A-Kanälen ist für die aufgerufenen Programme weitgehend unsichtbar.

■ automatische Pufferung

- Eingaben von der Tastatur werden normalerweise vom Betriebssystem zeilenweise zwischengespeichert und erst bei einem **NEWLINE**-Zeichen ('`\n`') an das Programm übergeben!
- Ausgaben an den Bildschirm werden vom Programm normalerweise zeilenweise zwischengespeichert und erst beim **NEWLINE**-Zeichen wirklich auf den Bildschirm geschrieben!



Öffnen und Schließen von Dateien

- Neben den Standard-E/A-Kanälen kann ein Programm selbst weitere E/A-Kanäle öffnen
 - Zugriff auf Dateien
- Öffnen eines E/A-Kanals
 - Funktion `fopen` (File Open)
- Schließen eines E/A-Kanals
 - Funktion `fclose` (File Close)



■ Schnittstelle fopen

```
#include <stdio.h>
```

```
FILE *fopen(const char *name, const char *mode);
```

name: Pfadname der zu öffnenden Datei

mode: Art, wie Datei zu öffnen ist

"r": zum Lesen (read)

"w": zum Schreiben (write)

"a": zum Schreiben am Dateiende (append)

"rw": zum Lesen und Schreiben (read/write)

- öffnet Datei **name**
- Ergebnis von **fopen**: Zeiger auf einen Datentyp **FILE**, der einen Dateikanal beschreibt; im Fehlerfall **NULL**



■ Schnittstelle fclose

```
#include <stdio.h>

int fclose(FILE *fp);
```

- schließt E/A-Kanal fp
- Ergebnis ist entweder 0 (kein Fehler aufgetreten) oder EOF im Falle eines Fehlers



Öffnen und Schließen von Dateien – Beispiel

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fp; int ret;

    fp = fopen("test.dat", "w"); /* Open "test.dat" for writing. */
    if (fp == NULL) {
        /* Error */
        perror("test.dat"); /* Print error message. */
        exit(EXIT_FAILURE); /* Terminate program. */
    }

    ... /* Program can now write to file "test.dat". */

    ret = fclose(fp); /* Close file. */
    if (ret == EOF) {
        /* Error */
        perror("test.dat"); /* Print error message. */
        exit(EXIT_FAILURE); /* Terminate program. */
    }

    return EXIT_SUCCESS;
}
```



Zeichenweises Lesen und Schreiben

■ Lesen eines einzelnen Zeichens

- von der Standardeingabe

```
#include <stdio.h>
int getchar(void);
```

- aus einer Datei

```
#include <stdio.h>
int fgetc(FILE *fp);
```

- lesen das nächste Zeichen
- geben das Zeichen als `int`-Wert zurück
- geben bei Eingabe von CTRL-D bzw. am Ende der Datei EOF als Ergebnis zurück

■ Schreiben eines einzelnen Zeichens

- auf die Standardausgabe

```
#include <stdio.h>
int putchar(int c);
```

- in eine Datei

```
#include <stdio.h>
int fputc(int c, FILE *fp);
```

- schreiben das Zeichen `c`
- geben im Fehlerfall EOF als Ergebnis zurück



Kopierprogramm:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    FILE *src, *dst;
    int c;

    if (argc != 3) { ... }

    if ((src = fopen(argv[1], "r")) == NULL) { ... }
    if ((dst = fopen(argv[2], "w")) == NULL) { ... }

    while ((c = fgetc(src)) != EOF) {
        if (fputc(c, dst) == EOF) { ... }
    }

    if (fclose(dst) == EOF) { ... }
    if (fclose(src) == EOF) { ... }

    return EXIT_SUCCESS;
}
```



Zeilenweises Lesen und Schreiben

■ Lesen einer Zeile

```
#include <stdio.h>
char *fgets(char *buf, int bufsize, FILE *fp);
```

- liest Zeichen aus Dateikanal `fp` in das `char`-Feld `buf` bis entweder `bufsize-1` Zeichen gelesen wurden oder `'\n'` oder `EOF` gelesen wurde
- `s` wird mit `'\0'` abgeschlossen (`'\n'` wird nicht entfernt)
- gibt bei `EOF` oder Fehler `NULL` zurück
- für `fp` kann `stdin` eingesetzt werden, um von der Standardeingabe zu lesen

■ Schreiben einer Zeile

```
#include <stdio.h>
int fputs(char *buf, FILE *fp);
```

- schreibt die Zeichen im Feld `s` auf Dateikanal `fp`
- gibt im Fehlerfall `EOF` zurück
- für `fp` kann auch `stdout` oder `stderr` eingesetzt werden



■ Schnittstelle

```
#include <stdio.h>
int printf(char *format, ...);
int fprintf(FILE *fp, char *format, ...);
int sprintf(char *buf, char *format, ...);
int snprintf(char *buf, int bufsize, char *format, ...);
```

- Die statt ... angegebenen Parameter werden entsprechend der Angaben im `format`-String ausgegeben
 - bei `printf` auf der Standardausgabe
 - bei `fprintf` auf dem Dateikanal `fp`
(für `fp` kann auch `stdout` oder `stderr` eingesetzt werden)
 - `sprintf` schreibt die Ausgabe in das `char`-Feld `buf`
(achtet dabei aber nicht auf das Feldende => Pufferüberlauf möglich!)
 - `snprintf` arbeitet analog, schreibt aber nur maximal `bufsize` Zeichen
(`bufsize` sollte natürlich nicht größer als die Feldgröße sein)



- Zeichen im `format`-String können verschiedene Bedeutung haben
 - normale Zeichen:
werden einfach in die Ausgabe kopiert
 - Escape-Zeichen:
z.B. `\n` oder `\t` werden durch die entsprechenden Zeichen (hier Zeilenvorschub bzw. Tabulator) bei der Ausgabe ersetzt
 - Format-Anweisungen:
beginnen mit `%`-Zeichen und beschreiben, wie der dazugehörige Parameter in der Liste nach dem `format`-String aufbereitet werden soll
- für genauere Informationen siehe Manuals (`man 3 printf, ...`)



- Format-Anweisungen

`%d`, `%i`: `int`-Parameter als Dezimalzahl ausgeben

`%ld`, `%li`: entsprechend für `long int`

`%f`: `float`-Parameter als Fließkommazahl ausgeben
(z.B. `13.153534`)

`%lf`: entsprechend für `double`

`%e`: `float`-Parameter als Fließkommazahl in
10er-Potenz-Schreibweise ausgeben (z.B. `2.71456e+02`)

`%le`: entsprechend für `double`

`%c`: `char`-Parameter als einzelnes Zeichen ausgeben

`%s`: `char`-Feld wird ausgegeben, bis `'\0'` erreicht ist

`%%`: ein `%`-Zeichen wird ausgegeben

...: ...



Formatierte Ausgabe – Beispiel

```
int tag = 25;
int monat = 6;
int jahr = 2009;
char *name = "Michael Jackson";
printf("Am %d.%d.%d starb\n%s.\n",
      tag, monat, jahr, name);

printf("\n");

double pi = asin(1.0) * 2.0;
double e = exp(1.0);
fprintf(stdout,
      "Wichtige Werte sind:\n");
fprintf(stdout,
      "pi=%lf und e=%lf\n", pi, e);
```

```
~> ./test
Am 25.6.2009 starb
Michael Jackson.
```

```
Wichtige Werte sind:
pi=3.141593 und e=2.718282
~>
```



■ Schnittstelle

```
#include <stdio.h>
```

```
int scanf(char *format, ...);
```

```
int fscanf(FILE *fp, char *format, ...);
```

```
int sscanf(char *buf, char *format, ...);
```

Format-String analog zur formatierten Ausgabe.

Für genauere Informationen siehe Manuals (`man 3 scanf, ...`).

Aber: da Werte gelesen werden sollen, müssen Zeiger auf die zu beschreibenden Variablen übergeben werden!



Formatierte Eingabe – Beispiel

```
double pi, e;  
int ret;  
  
ret = scanf("pi=%lf, e=%lf\n", &pi, &e);  
if (ret != 2) {  
    fprintf(stderr, "Bad input!\n");  
    exit(EXIT_FAILURE);  
}  
printf("I got\n\tpi=%lf\n\te=%lf\n", pi, e);
```

```
~> ./test  
3.14 2.718  
Bad input!  
~>
```

```
~> ./test  
pi=3.14, e=2.718  
I got  
    pi=3.140000  
    e=2.718000  
~>
```

