Systemprogrammierung

Teil 6: Ein-/Ausgabe
Dateizugriff, Elementare Ein-/Ausgabe

Prof. Dr. H. Drachenfels
Hochschule Konstanz

Version 9.0
28.7.2017

ANSI-C Ein-/Ausgabe: <stdio.h> (1)

Bei den Ein-/Ausgabefunktionen der ANSI-C Standard-Bibliothek werden die Eingabe-Quellen und Ausgabe-Ziele mit einem FILE-Zeiger angegeben:

- **FILE** ist ein (Alias-)Name für eine Struktur,
  die den Zustand einer Eingabe-Quelle bzw. eines Ausgabe-Ziels verwaltet
  zum Zustand gehören Puffer, Lese-/Schreibposition, aufgetretene Fehler, ...
- vordefinierte globale Variablen für die Standard-Ein-/Ausgabe:

```
extern FILE *stdin;
extern FILE *stdout;
extern FILE *stderr;
```

Hinweis: stdin, stdout und stderr können auch Präprozessor -Makros sein

- **fopen** liefert Zeiger auf weitere FILE-Objekte:

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

mode `"r"` für reinen Lesezugriff, `"w"` für reinen Schreibzugriff, ...

- **fclose** schließt nicht mehr benötigte Eingabe-Quellen und Ausgabe-Ziele:

```
int fclose(FILE *fp);
```

ANSI-C Ein-/Ausgabe: <stdio.h> (2)

- Ein-/Ausgabe von Einzelzeichen:

```
int fgetc(FILE *fp);
```

liefert das nächste Zeichen (umgewandelt in int) oder EOF bei Eingabeende / Fehler

```
int fputc(int c, FILE *fp);
```

schreibt das Zeichen c und liefert c oder bei Fehler EOF

```
...
```

- Ein-/Ausgabe von Zeichenketten:

```
char *fgets(char *s, int n, FILE *fp);
```

liefert in s die nächsten maximal n - 1 Zeichen einer Zeile
und gibt s zurück, bzw. NULL bei Eingabeende / Fehler

```
int fputs(const char *s, FILE *fp);
```

schreibt die Zeichenkette s und liefert nicht-negativen Wert bzw. bei Fehler EOF

```
...
```


ANSI-C Ein-/Ausgabe: <stdio.h> (3)

- formatierte Ein-/Ausgabe:

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

versucht die in format genannten Lücken zu füllen
und liefert die Anzahl der gefüllten Lücken oder EOF bei Eingabeende

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

schreibt die Zeichenkette format inklusive der mit Werten gefüllten Lücken
und liefert die Anzahl der insgesamt geschriebenen Bytes oder bei Fehler EOF

```
...
```

- Ein-/Ausgabe von Binärdaten:

```
size_t fread(void *p, size_t size, size_t n, FILE *fp);
```

liefert in p maximal n Portionen von size Byte
und gibt die Anzahl der tatsächliche gelesenen Portionen zurück

```
size_t fwrite(const void *p, size_t size, size_t n, FILE *fp);
```

schreibt maximal n Portionen von size Byte aus p
und gibt die Anzahl der tatsächliche geschriebenen Portionen zurück

ANSI-C Ein-/Ausgabe: <stdio.h> (4)

- Fehlerbehandlung:

```
int feof(FILE *fp);
```

liefert einen von 0 verschiedenen Wert, wenn das Eingabeende erreicht wurde

```
int ferror(FILE *fp);
```

liefert einen von 0 verschiedenen Wert, wenn ein Fehler aufgetreten ist

```
void perror(const char *prefix);
```

gibt prefix gefolgt von der Fehlermeldung des aktuellen Fehlers auf stderr aus

```
void clearerr(FILE *fp);
```

Setzt den Eingabeende- und Fehlerzustand zurück

Beispiel-Programm <stdio.h>

Zählt die Zeichen in Dateien

```
#include <stdio.h>   /* fopen, fgetc, fclose */
int main(int argc, char *argv[])
{
    int i;
    for (i = 1; i < argc; ++i)
    {
        int n = 0;
        FILE *fp = fopen(argv[i], "r");
        if (fp == NULL) ... /* Fehlerbehandlung */

        while (fgetc(fp) != EOF) {
            ++n;
        }
        ... /* Fehlerbehandlung */
        printf("%s: %d Zeichen\n", argv[i], n);
        fclose(fp);
    }
    return 0;
}
```

POSIX Ein-/Ausgabe: Übersicht

POSIX (Portable Operating System Interface)
ist ein Standard für die Programmierschnittstelle von Betriebssystemen.

- der Standard legt C-Systemaufrufe und die zugehörigen Header-Dateien fest:
  über 80 Header-Dateien mit über 1000 Funktionen und Makros
  (dabei teilweise Überlappungen mit dem ANSI-C-Standard)

  Die meisten UNIX-Varianten und viele weitere Betriebssysteme
  halten sich ganz oder zumindest weitgehend an diesen Standard.

- wichtige Header-Dateien im Zusammenhang mit Ein-/Ausgabe:

  <fcntl.h> und <unistd.h>
  kein Schreibfehler!
  Umgang mit Dateien und Datenströmen (**creat**, **open**, **read**, **write**, **close**)

  <sys/stat.h> und <dirent.h>
  Umgang mit Verzeichnissen (**stat**, **mkdir**, **opendir**, **readdir**, **closedir**)

  <errno.h>
  Fehlerzustand und symbolische Namen für Fehlernummern (**errno**)

POSIX Ein-/Ausgabe: Elementare Ein-/Ausgabe (1)

Bei den elementaren Ein-/Ausgabefunktionen nach POSIX-Standard werden Eingabe-Quellen und Ausgabe-Ziele über einen Dateideskriptor angesprochen:

- ein **Dateideskriptor** ist eine nicht-negative ganze Zahl
  bei der ANSI-C Ein-/Ausgabe in der FILE-Struktur gespeichert

- vordefinierte Dateideskriptoren für die Standard-Ein-/Ausgabe:
  0 Standardeingabe
  1 Standardausgabe
  2 Standardfehlerausgabe

- **open** liefert einen Dateideskriptor für eine Datei:

```
int open(const char *dateiname, int flags); /* <fcntl.h> */
```

  liefert den kleinsten nicht belegten Dateideskriptor oder bei Fehler -1

- **close** schließt nicht mehr benötigte Eingabe-Quellen und Ausgabe-Ziele:

```
int close(int fd); /* <unistd.h> */
```

  liefert 0 oder bei Fehler -1

POSIX Ein-/Ausgabe: Elementare Ein-/Ausgabe (2)

- Ein-/Ausgabe von Bytes:

  ssize_t read(int fd, void *p, size_t n); /* <unistd.h> */

  *liefert in **p** maximal n Byte und gibt die Anzahl der tatsächliche gelesenen Bytes zurück, **0** bei Eingabeende, **-1** bei Fehler*

  ssize_t write(int fd, const void *p, size_t n); /* <unistd.h> */

  *schreibt maximal **n** Byte aus **p** und gibt die Anzahl der tatsächliche geschriebenen Bytes oder bei Fehler **-1** zurück*

  ssize_t /* <sys/types.h> */

  *Aliasname für einen ganzzahligen Typ mit Vorzeichen (**int** oder **long**)*

- Fehlerbehandlung:

  extern int errno; /* <errno.h>, errno kann auch ein Makro sein */

  POSIX-Funktionen weisen errno im Fehlerfall eine Fehlernummer ungleich 0 zu

  für die Fehlernummern sind symbolische Konstanten definiert
  (z.B: EACCES für fehlendes Zugriffsrecht auf eine Datei)

Beispiel-Programm Dateien (1)

Kopiert eine Datei

```
#include <stdio.h>      /* fprintf */
#include <string.h>     /* strerror */

#include <fcntl.h>      /* open, O_RDONLY, O_WRONLY, O_CREAT, O_EXCL */
#include <sys/stat.h>   /* mode_t, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH */
#include <unistd.h>     /* read, write */
#include <errno.h>      /* errno */

int main(int argc, char *argv[])
{
    mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH; /* Zugriffsrechte */
    int in, out; /* Dateideskriptoren */
    int n;
    unsigned char byte;

    if (argc != 3)
    {
        fprintf(stderr, "Aufruf: %s Quelle Ziel\n", argv[0]);
        return 1;
    }

    ...
```

Beispiel-Programm Dateien (2)

```
    ...

    in = open(argv[1], O_RDONLY);
    if (in   == -1) ... /* Fehlerbehandlung */

    out = open(argv[2], O_WRONLY | O_CREAT | O_EXCL, mode);
    if (out  == -1) ... /* Fehlerbehandlung */

    while ((n = read(in, &byte, 1)) > 0)
    {
        int m = write(out, &byte, 1);
        if (m != 1) ... /* Fehlerbehandlung */
    }

    if (n < 0) ... /* Fehlerbehandlung */

    close(out);

    close(in);

    return 0;
}
```


POSIX Ein-/Ausgabe: Verzeichnisse

Nach POSIX-Standard werden Verzeichnisse über DIR-Zeiger angesprochen:

- **opendir** liefert einen DIR-Zeiger für ein Verzeichnis:

  DIR *opendir(const char *verzeichnisname); /* <dirent.h> */

  liefert NULL bei Fehler

- **closedir** beendet den Verzeichniszugriff:

  int closedir(DIR *dirp); /* < dirent.h> */

  liefert 0 oder bei Fehler -1

- **readdir** liefert einen Zeiger auf den nächsten ungelesenen Verzeichniseintrag:

  struct dirent *readdir(DIR *dirp); /* < dirent.h> */

  der Verzeichniseintrag enthält unter d_name einen Dateinamen
  liefert NULL bei Verzeichnisende oder Fehler

- **stat** liefert Statusinformation zu einer Datei *(Dateityp, Zugriffsrechte, ...)*:

  int stat(const char *dateiname, struct stat *buf); /* < sys/stat.h> */

  liefert 0 oder bei Fehler -1

  Ausgabeparameter

Beispiel-Programm Verzeichnisse (1)

Listet Verzeichnisse auf

```
#include <stdio.h>   /* fprintf, printf */
#include <string.h>  /* strerror  */

#include <sys/stat.h>  /* struct stat, S_ISDIR */
#include <dirent.h>    /* DIR, struct dirent, opendir, readdir */
#include <errno.h>     /* errno  */

int main(int argc, char *argv[])
{
    struct stat s;      /* Dateistatus  */
    DIR *d;             /* geoeffnetes Verzeichnis */
    struct dirent *e;   /* gelesener Verziechniseintrag */
    int i;

    ...
```


Beispiel-Programm Verzeichnisse (2)

```
    ...
    for (i = 1; i < argc; ++i) {
        /* Datei vorhanden? */
        if (stat(argv[i], &s) == -1) ... /* Fehlerbehandlung */

        /* Dateityp Verzeichnis? */
        if (!S_ISDIR(s.st_mode)) ... /* Fehlerbehandlung */

        d = opendir(argv[i]);
        if (d == NULL) ... /* Fehlerbehandlung */

        errno = 0;
        while ((e = readdir(d)) != NULL) {
            printf("%s/%s\n", argv[i], e->d_name);
        }
        ... /* Fehlerbehandlung */

        closedir(d);
    }
    return 0;
}
```

ANSI-C Ein-/Ausgabe: Index