PR: Programmiersprache C (Linux)
Einheit 7: I/O, Stringverarbeitung

Roland A. Eggetsberger
Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria

I/O

Standarddatei-I/O
Typisierte Dateien
System-I/O

Standarddatei-I/O (I)

- Stream
  Variable vom Typ **`FILE *`**
  Daher auch streamorientierte I/O

Standarddatei-I/O (II)

- Datei öffnen

```
#include <stdio.h>
FILE *fopen(const char *name,
            const char *modus);
```

Im Fehlerfall wird **`NULL`** zurückgegeben

Standarddatei-I/O (III)

- Bearbeitungsmodi (I)

```
r ... Lesen (NULL falls keine Datei oder
      keine Berechtigung)
w ... Anlegen zum Schreiben
a ... Öffnen zum Schreiben
```

Standarddatei-I/O (IV)

- Bearbeitungsmodi (II)

```
r+ ... Lesen (NULL falls keine Datei oder
       keine Berechtigung)
w+ ... Anlegen zum Schreiben
a+ ... Öffnen zum Schreiben
```

Standarddatei-I/O (V)

- Bearbeitungsmodi (III)

```
b ... Binärmodus (Keine Konvertierung)
t ... Textmodus (Dabei wird \r\n
      konvertiert)
```

Standarddatei-I/O (VI)

- Zeichenweise

```
#include <stdio.h>
int putc(int c, FILE *f);
int getc(FILE *f);
```

Standarddatei-I/O (VII)

- Schließen

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

Im Fehlerfall wird EOF zurückgegeben

Standarddatei-I/O (VIII)

- Formatierte I/O

```
#include <stdio.h>
int fprintf(FILE *f, const char *format,
            ...);
int fscanf(FILE *f, const char *format,
            ...);
```

Standarddatei-I/O (IX)

- Standarddateien (I)

```
stdin .... Standardeingabe
stdout ... Standardausgabe
stderr ... Standardfehlerkanal
```

Der Fehlerkanal dient dazu bei Ausgabeumleitung Ausgabe und Fehler zu trennen

Standarddatei-I/O (X)

- Standarddateien (II)

Umleitung des Fehlerkanals

```
prog <input.txt >output.txt 2>errors.txt
```

Standarddatei-I/O (XI)

- Puffer leeren

```
#include <stdio.h>
int fflush(FILE *f);
```

Im Fehlerfall wird **EOF** zurückgegeben

Standarddatei-I/O (XII)

- Dateizeiger zurücksetzen

```
#include <stdio.h>
void rewind(FILE *f);
```

Dateizeiger wird auf Dateianfang zurückgesetzt

Bei der nächsten Schreib-/Leseoperation wird mit dem ersten Zeichen begonnen

Standarddatei-I/O (XIII)

- Dateizeiger Positionieren (I)

```
#include <stdio.h>
void fseek(FILE *f, long offset, int
           origin);
```

Beliebiges Positionieren des Dateizeigers

```
origin ... Bezugspunkt
offset ... Entfernung
```

Standarddatei-I/O (XIV)

- Dateizeiger Positionieren (II)

Für den Bezugspunkt sind symbolische Konstanten definiert

```
SEEK_SET (=0) ... Anfang der Datei
SEEK_CUR (=1) ... Aktuelle Position
SEEK_END (=2) ... Ende der Datei
```

Standarddatei-I/O (XV)

- Dateizeiger positionieren (III)

Die Entfernung wird in Zeilen angegeben (auch negativ möglich)

Im Erfolgsfall gibt **fseek** 0 zurück (sonst einen Wert ungleich 0)

Standarddatei-I/O (XVI)

- Position des Dateizeigers

```
#include <stdio.h>
long ftell(FILE *f);
```

Liefert aktuelle Position des Satzzeigers vom Beginn der Datei weg

Im Fehlerfall Rückgabe von **-1L**

Typisierte Dateien (I)

- Datei als Folge identisch strukturierter Datensätze
- Datei nur durch Lese-/Schreiboperationen gekennzeichnet

Typisierte Dateien (II)

- Schreiben (I)

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t
              size, size_t nmemb,
              FILE *stream);

ptr ..... Datensätze selbst
size .... Größe eines Datensatzes
nmemb ... Anzahl der Datensätze
```

Typisierte Dateien (III)

- Schreiben (II)
  Rückgabewert ist die Anzahl der geschriebenen Datensätze

Typisierte Dateien (IV)

- Lesen

```
#include <stdio.h>
size_t fread(void *ptr, size_t size,
             size_t nmemb, FILE *stream);
```

Die Parameter sind analog wie beim Schreiben
Für den Pointer **ptr** muss entsprechend Speicher allokiert werden

System-I/O (I)

- Auf manchen Systemen gibt es eine eigene Header-Datei für Low-Level-I/O
- Im Allgemeinen werden jedoch einige Header benötigt

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

System-I/O (II)

- Datei öffnen (I)

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags,
         mode_t mode);

pathname ... Pfad
flags ...... Dateimodus
mode ....... Berechtigung
```

System-I/O (III)

- Datei öffnen (II)
  Rückgabewert ist ein Handle für die Datei (i.e. ein positiver Integer-Wert)
  Im Fehlerfall wird -1 zurückgegeben

System-I/O (IV)

- Dateimodi (I)
  Elementare Modi

```
O_RDONLY ... Lesen
O_WRONLY ... Schreiben
O_RDWR ..... Lesen und Schreiben
```

System-I/O (V)

- Dateimodi (II)
  Zusätzliche Modi

```
O_CREAT .... File erzeugen
O_TRUNC .... File löschen
O_APPEND ... Dateizeiger auf Ende setzen
```

  Diese Modi werden mit den elementaren durch oder verknüpft

System-I/O (VI)

- Berechtigungen (I)
  Vergeben von Zugriffsrechten auf eine Datei
  Userspezifisch

```
S_IREAD .... Lesen
S_IWRITE ... Schreiben
S_IEXEC .... Ausführen
```

System-I/O (VII)

- Berechtigungen (II)
  Allgemein zum Beispiel

```
S_IRGRP ... Lesen (für die Gruppe)
S_IXOTH ... Ausführen (für alle)
```

System-I/O (VIII)

- Schreiben (I)

```
#include <unistd.h>
ssize_t write(int fd, const void *buf,
              size_t count);

fd ..... Datei-Handle
buf ..... Puffer
count ... Anzahl der zu schreibenden Bytes
```

System-I/O (IX)

- Schreiben (II)
  Rückgabewert ist die Anzahl der tatsächlich geschriebenen Bytes
  Im Fehlerfall wird -1 zurückgegeben

System-I/O (X)

- Lesen

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t
             count);
```

Parameter und Fehlerfälle wie beim Schreiben

System-I/O (XI)

- Dateizeiger Positionieren (I)

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fildes, off_t offset, int
            whence);

fildes ... Datei-Handle
offset ... Entfernung
whence ... Bezugspunkt
```

System-I/O (XII)

- Dateizeiger Positionieren (II)
  Die Bezugspunkte können wie bei der Standard-I/O verwendet werden (siehe S. 17)

  Rückgabe ist ein Datei-Handle oder -1 (im Fehlerfall)

System-I/O (XIII)

- Datei schließen

```
#include <unistd.h>
int close(int fd);
```

Rückgabewert ist 0 oder -1 (im Fehlerfall)

Stringverarbeitung

Elementare Stringfunktionen

Suchfunktionen

Elementare Stringfunktionen (I)

- Kopieren

```
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src,
              size_t n);
```

Von **src** nach **dest** kopieren
Kann auf die ersten **n** Zeichen beschränkt werden

Elementare Stringfunktionen (II)

- Länge

```
size_t strlen(const char *s);
```

Länge des Strings (ohne **'\0'**)

Elementare Stringfunktionen (III)

- Vergleichen (I)

```
int strcmp(const char *s1, const char
           *s2);
int strncmp(const char *s1, const char
            *s2, size_t n);
```

Strings (oder erste **n** Zeichen) vergleichen

Elementare Stringfunktionen (IV)

- Vergleichen (II)

```
int strcasecmp(const char *s1, const char
               *s2);
int strcasencmp(const char *s1, const char
                *s2, size_t n);
```

Vergleichen ohne die Groß-/Kleinschreibung zu berücksichtigen

Elementare Stringfunktionen (V)

- Verketten

```
char *strcat(char *dest, const char *src);
char *strncat(char *dest, const char *src,
              size_t n);
```

String **src** an String **dest** anhängen (oder erste **n** Zeichen davon)

Suchfunktionen (I)

- Zeichen suchen (I)

```
char *strchr(const char *s, int c);
```

Erstes Auftreten des Zeichens im String suchen

Suchfunktionen (II)

- Zeichen suchen (II)

```
char *strrchr(const char *s, int c);
```

Letztes Auftreten des Zeichens im String suchen

Suchfunktionen (III)

- Zeichen suchen (III)

```
char *strpbrk(const char *s, const char
              *accept);
```

Erstes Auftreten eines Zeichens aus **accept** im String **s**

Suchfunktionen (IV)

- Substrings suchen

```
char *strstr(const char *haystack, const
             char *needle);
```

Nadel im Heuhaufen

Suchfunktionen (V)

- Gültige Länge (I)

```
size_t strspn(const char *s, const char
              *accept);
```

Länge des Substrings von **s** der Zeichen aus **accept** enthält

Suchfunktionen (VI)

- Gültige Länge (II)

```
size_t strcspn(const char *s, const char
               *reject);
```

Länge des Substrings von **s** der keine Zeichen aus **reject** enthält

Suchfunktionen (VII)

- Tokenizer (I)

```
char *strtok(char *s, const char *delim);
```

Ein Token ist ein Substring der keine Zeichen aus **delim** enthalten darf, aber durch **'\0'** oder ein Zeichen aus **delim** beendet wird

Suchfunktionen (VIII)

- Tokenizer (II)
  Beim ersten Aufruf wird mit einem String initialisiert
  Alle weiteren Aufrufe erfolgen mit **`NULL`**
  Der Tokenizer sucht dann jeweils den nächsten Token