PR3 - Leseaufgabe zur Vorlesung 7

(Stand 2021-10-01 14:51)

Prof. Dr. Holger Peine
Hochschule Hannover
Fakultät IV – Abteilung Informatik
Raum 334, Tel. 0511-9296-1830
Holger.Peine@hs-hannover.de

L.7 Ein-/Ausgabe und Dateizugriffe

L.7.1 Dateien

Wir beginnen mit einem einfachen Beispiel zum Schreiben in eine Datei. Das folgende Programm öffnet eine Datei `out.txt` zum Schreiben, schreibt den Text „Hello world!“ hinein und schließt die Datei wieder:

```
#include <stdio.h>
int main(void) {
    FILE* fp;
    fp = fopen("out.txt","w");
    fprintf(fp,"Hello world!\n");
    fclose(fp);
    return 0;
}
```

Die Struktur `FILE` ist ein opaker Typ (in `stdio.h` deklariert) und enthält Informationen über den Datenstrom (Verweise auf die Datei, Verweis auf Puffer, Positionszeiger, …). Das `"w"` in der `fopen`-Anweisung bedeutet: Öffnen zum Schreiben ("write"). Analog gibt es `"r"` usw.

L.7.2 Standarddateien bzw. -datenströme

Neben eigenen, selbst zu öffnenden Dateien, gibt es drei Standarddateien, die automatisch beim Programmstart geöffnet werden (deklariert in `stdio.h`):

Name des Stroms	Bezeichnung	Default-Gerät	Strom in Java
`stdin`	Standardeingabe	Tastatur	`System.in`
`stdout`	Standardausgabe	Bildschirm	`System.out`
`stderr`	Standardfehlerausgabe	Bildschirm	`System.err`

Diese drei "Dateien" (auch "Ströme" / "streams" genannt) sind normalerweise mit der Konsole verbunden, können aber beim Programmstart auch mit anderen Dateien verbunden werden, z. B. mit der bash unter Linux mit einem Programmaufruf wie folgt:

```
./a.out <input.txt >output.txt
```

Bedeutet:

- `stdin` liefert statt Tastatureingaben die Daten aus der Datei `input.txt`
- `stdout` leitet Ausgaben nicht auf die Konsole, sondern in die Datei `output.txt`.

```
./a.out 2>errors.txt
```

Bedeutet:

- `stderr` leitet die Fehlerausgaben nicht auf die Konsole, sondern in die Datei `errors.txt` um.

Weitere Varianten entnimmt man z. B. dem bash-Howto unter http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-3.html.

L.7.3 Überblick über Ein-/Ausgabefunktionen der Standardbibliothek

Die folgende Tabelle gibt einen Überblick über die Funktionen der Standardbibliothek. Alle Funktionen, die auf Dateien operieren, beginnen mit einem `f`. Eine Besonderheit bilden die Funktionen `sscanf` unf `sprintf`, die keine Ein-/Ausgabefunktionen im eigentlichen Sinne sind, weil sie Ein- und Ausgabe aus einem bzw. in einen `char[]` simulieren.

Aktion		**Quelle / Ziel**		
		`stdin` bzw. `stdout`	**Beliebige Datei**	**Zeichen-kette**
Öffnen / Schließen / Positionieren			`fopen, fclose, feof, fflush, ftell, fseek`	
Lesen / Schreiben	**Datentyp**			
	Zeichen und Zeichenketten	`gets, puts, getchar, putchar`	`fgets, fputs, fgetc, fputc`	
	Standardtypen (`int`, `float`, ...)	`scanf, printf`	`fscanf, fprintf`	`sscanf, sprintf`
	Beliebige binäre Daten[1]		`fread, fwrite`	

Neben den genannten Funktionen bietet die Standardbibliothek weitere Funktionen an, die den Rahmen der Darstellung hier sprengen würden. Literaturempfehlungen:

- http://www2.hs-fulda.de/~klingebiel/c-stdlib/index.htm
  - nicht vollständig, aber enthält die wesentlichen Funktionen. Und ist gratis!
  - Als Download auf dem Server: fulda-stutz-klingebiel-c-stdlib.tgz
- Online-Buch: Wolf, Jürgen: C von A-Z, 3. Auflage, Rheinwerk 2006, ISBN 978-3-8362-1411-7. Gibt‘s gratis online: http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/
- Sehr gute C-Referenz: Harbison, Samuel P., Steele, Guy L.: C. A Reference Manual, 5th ed., Prentice Hall 2002..
- Klassiker: Kernighan, Brian W., Ritchie, Dennis M.: The C Programming Language, 2nd ed., Prentice Hall 1988. In deutsch zahlreich in der Bibliothek vorhanden.
- Der ANSI-Standard selbst. Eine Vorversion des Standards (Grundlage für ein öffentliches Review) finden Sie auf dem Server: ansi_c_draft_19880513.txt.
- Gute Online-Referenz: http://www.cplusplus.com/reference/clibrary/ .

Wir stellen die Funktionen der Tabelle nun jeweils kurz vor. Sie müssen die Funktionsweise der einzelnen Funktionen nicht auswendig lernen. Dieses Dokument dient eher einem ersten Einstieg und einem groben Überblick über die Möglichkeiten der Standardbibliothek.

[1] Wozu ein Binärmodus? Der Binärmous kann Zeit und Platz sparen. Ein Beispiel:
- Die Zahl 30457 belegt im Textmodus 5 Zeichen, im Binärmodus in der Regel 2 (short) oder 4 (int) Bytes.
- Liest man die Zahl im Textmodus ein, muss die Zeichenfolge '3' '0' '4' '5' '7' zeitaufwändig in die interne Byte-Darstellung eines `int` umgewandelt werden.
- Der Binärmodus ist, anders als eine Textdarstellung, unabhängig von einem bestimmten Zahlensystem (z.B. Dezimal- oder Hexadezimalsystem).

L.7.3.1 gets, puts, getchar, putchar

`int getchar(void)`
liefert ein Zeichen vom `stdin` (oder EOF)

- Liefert erst nach Eingabe von RETURN.
- Unter DOS/Windows gibt es folgende Funktionen, die unmittelbar nach einer Tastenbetätigung ein Zeichen liefern. Diese gehören nicht zum ANSI-C-Standard: `getch`, `getche`.

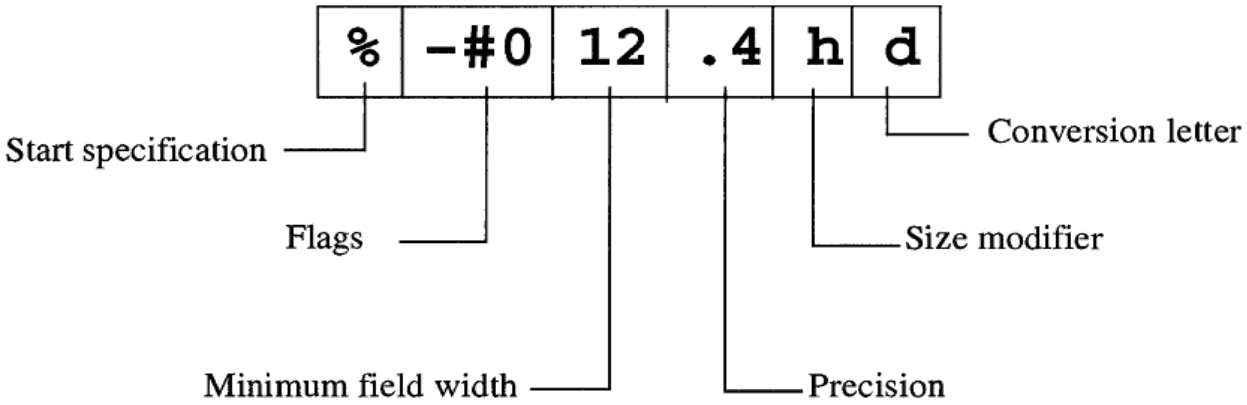
`char* gets(char* s)`
nicht mehr verwenden – führt immer zu Pufferüberlauf-Schwachstelle (stattdessen `fgets(…, stdin)` verwenden, siehte unten)

`int putchar(int c)`
schreibt ein Zeichen nach stdout und liefert dieses Zeichen zurück (oder `EOF`).

`int puts(const char* s)`
schreibt Zeichenkette ohne \0 aber mit Zeilenvorschub. Rückgabe `EOF` oder (bei Erfolg) eine Zahl >= 0.

L.7.3.2 printf

Hier der Übersicht halber nochmals der Aufbau einer printf-Formatangabe aus der Vorlesung (bis auf das % am Anfang und die Typangabe am Ende sind alle Bestandteile optional):



Hier die möglichen "size modifier":

Modifi-kator	vor Typangabe	Typ des Werts
h	d, i	short int
	u, o, x, X	unsigned short int
l	d, i	long int
	u, o, x, X	unsigned long int
L	f, e, E, g, G	long double

Hier die möglichen Datentypangaben ("conversion letters") – auswending Lernen ist nicht nötig!):

Typ-angabe	Typ des Werts	Darstellungsform
d, i	int	dezimal
u	unsigned int	dezimal
o	unsigned int	oktal (Ziffern 0...7)
x	unsigned int	hexadezimal (Ziffern 0...9 a...f)
X	unsigned int	hexadezimal (Ziffern 0...9 A...F)
f	double	dezimal mit Vor- und Nachpunktstellen
e, E	double	dezimal „Mantisse e Exponent“ bzw. „Mantisse E Exponent“
g, G	double	dezimal, je nach Größe entweder gemäß f oder e bzw. E
c	char/int	Zeichen gemäß ASCII
s	char *	Zeichenkette gemäß ASCII
p	void *	Speicheradresse

Hier die möglichen Flags:

Flag	bei Typangabe	Effekt
-	alle	Ausgabe erfolgt linksbündig (im Normalfall erfolgt die Ausgabe rechtsbündig)
+	d, i, f, e, E, g, G	vor positiven Zahlenwerten wird ein +-Zeichen ausgegeben (im Normalfall werden nur negative Vorzeichen ausgegeben)
Leer-zeichen	d, i, f, e, E, g, G	falls kein Vorzeichen auszugeben ist, wird stattdessen ein Leerzeichen ausgegeben
#	o	Ausgabe beginnt mit 0
	x, X	Ausgabe beginnt mit 0x bzw. 0X
	f, e, E, g, G	Ausgabe enthält einen Dezimalpunkt, auch wenn keine Nachpunktstellen ungleich null vorhanden sind
	g, G	Ausgabe enthält in den Nachpunktstellen abschließende Nullen (bis zur gewünschten Genauigkeit – siehe unten)
0	alle ohne c, s, p	Ausgabe wird mit führenden Nullen aufgefüllt (falls minimale Feldbreite angegeben ist – siehe unten)
	d, i, u, o, x, X	Flag 0 wird ignoriert, wenn eine Genauigkeit (siehe unten) angegeben ist

Die minimale Feldbreite ist ein dezimaler Ganzzahlwert, der die minimale Anzahl auszugebender Zeichen festlegt. Hat die Darstellung des auszugebenden Werts weniger Zeichen, so wird die Ausgabe im Normalfall links mit Leerzeichen aufgefüllt (sofern es durch Flags nicht anders festgelegt ist). Statt des Zahlenwerts kann auch ein Stern * angegeben werden; man kann dann die gewünschte Breite dynamisch in einem weiteren Parameter übergeben.

Hier die Möglichkeiten für die "precision":

bei Typangabe	Festlegung
d, i, u, o, x, X	minimale Anzahl auszugebender Ziffern (Ausgabe wird gegebenenfalls mit führenden Nullen aufgefüllt)
f, e, E	Anzahl der Ziffern nach dem Dezimalpunkt
g, G	maximale Anzahl auszugebender Ziffern (Ausgabe wird gegebenenfalls nach den führenden Ziffern abgeschnitten)
s	maximale Anzahl auszugebender Zeichen (Ausgabe wird gegebenenfalls nach einem Anfangsstück abgeschnitten)

Statt des Zahlenwerts kann auch ein Stern * angegeben werden; man kann dann die gewünschte Genauigkeit dynamisch in einem weiteren Parameter übergeben.

L.7.3.3 scanf

Hier die möglichen Konversionszeichen von `scanf`; statt der Typangabe `s` ist auch [...] möglich. In […] stehen dann die Zeichen, die bei der Eingabe akzeptiert werden. `scanf` liest dann bis zum ersten nicht akzeptierten Zeichen. Analog gibt [^...] an, welche Zeichen *nicht* akzeptiert werden.

Typ-angabe	Typ des Werts	zu interpretierende textuelle Darstellungsform
d	int	ganzzahliger Dezimalwert, möglicherweise mit Vorzeichen
i	int	ganzzahliger Wert, möglicherweise mit Vorzeichen, entweder dezimal oder oktal (mit führender 0) oder hexadezimal (mit führendem 0x oder 0X)
u	unsigned int	ganzzahliger Dezimalwert
o	unsigned int	ganzzahliger Oktalwert (mit führender 0)
x	unsigned int	ganzzahliger Hexadezimalwert (mit führendem 0x oder 0X)
e, f, g	float	Zahlenwert, möglicherweise mit Vorzeichen, Nachpunktstellen und Exponent
c	char	Zeichen
s	char *	Zeichenkette ohne Whitespace-Zeichen
p	void *	Speicheradresse

Und hier die möglichen Typmodifikatoren:

Modifi-kator	vor Typangabe	Typ des Werts
h	d, i	short int
	u, o, x	unsigned short int
l	d, i	long int
	u, o, x	unsigned long int
	e, f, g	double
L	e, f, g	long double

L.7.3.4 fopen, fclose, feof, fflush, ftell, fseek

`FILE* fopen(const char* filename, const char* mode)`
Öffnet einen Datenstrom (Datei, Netzwerkverbindung, Gerät, …):

- Bei Fehler: Rückgabe `NULL` und `errno` wird gesetzt.
- Bedeutung von `mode`:
  - r: Lesezugriff auf existierende Datei.
  - w: Schreibzugriff in neue Datei. Falls schon existent, wird die Datei gelöscht.
  - …b (z. B. "rb"): Binärmodus
  - …t (z. B. "rt"): Textmodus (t ist optional)
  - Weitere sind möglich (z. B. zum Anhängen)

`int fclose(FILE* stream)`
Schließt die Datei (Rückgabe: 0=Erfolg oder `EOF`).

`int feof(FILE* stream)`
prüft, ob das Ende einer Datei erreicht ist (Rückgabe: 1 = wahr, sonst 0).

`int fflush(FILE* stream)`
leert den Ausgabepuffer auf den Datenträger.

- Nach Standard nur für Ausgabestreams erlaubt.
- Auf vielen Systemen auch für Eingabepuffer möglich (nicht portabel!).

`long ftell(FILE* stream)`
liefert aktuelle Position in der geöffneten Datei.

`int fseek(FILE* stream, long offset, int whence)`
verschiebt die aktuelle Position einer geöffneten Datei

Details zu `ftell/fseek` finden Sie z. B. hier: http://www.cplusplus.com/reference/clibrary/cstdio/fseek/.
Insbesondere bei Text-Dateien nicht immer mit dem intuitiv erwarteten Verhalten!

L.7.3.5 fgets, fputs, fgetc, fputc

`int fgetc(FILE* stream)`
liefert ein Zeichen aus der Datei (oder `EOF`)

- Beispiel:
```
char zeichen;
FILE* fp;
fp = fopen("datei","r");
while (!feof(fp)) {
    zeichen = fgetc(fp);
    printf("%c",zeichen);
}
```

`int fputc(int c, FILE* stream)`
schreibt ein Zeichen in eine Datei.

- Rückgabe: der geschriebene ASCII-Wert oder `EOF`.

`char* fgets(char* s, int n, FILE* stream)`
liest bis zum nächsten `\n` oder `EOF`, höchstens jedoch `n-1` Zeichen. Speichert die Zeichen und ein abschließendes `\0` in `s`.

- Rückgabe: `s` (oder `NULL` im Fehlerfall)
- Beispiel:
```
char* s = malloc(13);
FILE* fp = fopen("datei","r");
fgets(s, 13, fp);
```

`int fputs(const char* s, FILE* stream)`
schreibt `s` ohne abschließendes `\0` in eine Datei.

L.7.3.6 fscanf, fprintf, sscanf, sprintf

Diese Funktionen arbeiten wie scanf und printf, nur dass sie für andere Quellen / Ziele konzipiert sind.

L.7.3.7 fread, fwrite

`size_t fread(void* ptr, size_t size, size_t nmemb, FILE* stream)`
liest `nmemb` Datenblöcke (jeder Block hat Größe `size`) aus einer Datei und legt sie im Hauptspeicher an Startadresse `ptr` ab.

- Rückgabe: Anzahl gelesener Blöcke.

`size_t fwrite(const void* ptr,size_t size, size_t nmemb, FILE* stream)`
schreibt `nmemb` aufeinanderfolgende Datenblöcke (jeder Block hat Größe `size`) aus dem Speicherbereich mit Startadresse `ptr` in die Datei

- Rückgabe: Anzahl der geschriebenen Blöcke zurück.

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

void write(void) {
    int a[] = { 4, 7, -3, 9, -8 };
    int laenge= 5;
    FILE* f = fopen("data.dat", "wb");
    fwrite(&laenge, sizeof(laenge), 1, f);
    fwrite(a, sizeof(a[0]), laenge, f);
    fclose(f);
```

```
}
void read(void) {
    int* a;
    int laenge, i;
    FILE* f = fopen("data.dat", "rb");
    fread(&laenge, sizeof(laenge), 1, f);
    a = malloc(laenge * sizeof(*a));
    fread(a, sizeof(a[0]), laenge, f);
    fclose(f);
    for (i = 0; i < laenge; i++) printf("%d\n", a[i]);
}
int main(void) {
    write();
    read();
    return 0;
}
```

L.7.4 Operationen auf dem Dateisystem

Wir stellen Ihnen noch drei häufig verwendete Funktionen vor, die Dateien im Dateisystem unbenennen, löschen oder erzeugen können:

`int remove(const char* filename)`
entfernt den Dateinamen aus dem Dateisystem.

`int rename(const char* old, const char* new)`
benennt eine Datei um

`FILE* tmpfile(void)`
erzeugt eine temporäre Datei, die bei einem nachfolgenden `fclose`-Aufruf oder bei Programmende wieder automatisch gelöscht wird.

- Die Datei wird im Binärmodus geöffnet
- Rückgabewert: Dateizeiger (oder `NULL` im Fehlerfall).