Ausgewählte Bibliotheksfunktionen

fopen

Bibliothek:	`#include <stdio.h>`
Prototyp:	`FILE *fopen(const char *name, const char *modus);`
Rückgabewert:	File-Pointer
Beschreibung:	öffnet eine Datei `name` in der Zugriffsart `modus`. Durch Anhängen des Buchstabens "t" bzw. "b" an modus kann zwischen Binär- und Textmodus gewählt werden. `FILE` ist in `stdio.h` definiert. Liste der verschiedenen Modi:

"r"	Lesen (Datei muss existieren)
"r+"	Lesen/Schreiben, open for update (Datei muss existieren)
"w"	Schreiben (wenn Datei nicht existiert, wird sie angelegt, wenn sie schon existiert, wird sie überschrieben!)
"w+"	Lesen / Schreiben (wenn Datei nicht existiert, wird sie angelegt, wenn sie schon existiert, wird sie überschrieben!)
"a"	Schreiben ab Dateiende (wenn Datei nicht existiert, wird sie angelegt)
"a+"	Schreiben ab Dateiende / Lesen (Datei muss existieren)

Beispiel:

```
fp = fopen("bsp.txt","r");
/* öffnet die bereits existierende Datei bsp.txt
zum Lesen (fp wird mit der Datei verbunden) */
```

fclose

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int fclose(FILE *fp);`
Rückgabewert:	Datei geschlossen: 0; ungültiger File-Pointer: EOF
Beschreibung:	schließt die mit `fp` verbundene Datei überträgt den Inhalt des Dateipuffers. Der Speicherbereich des E/A-Puffers wird wieder freigegeben.
Beispiel:	`if (fclose(fp) == 0)` `  printf("Datei wurde geschlossen\n");`

fflush

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int fflush(FILE *fp);`
Rückgabewert:	Fehlerfall: EOF; andernfalls: 0
Beschreibung:	überträgt den Inhalt des Dateipuffers: Schreibmodus: Dateipuffer wird geschrieben Lesemodus: Puffer wird geleert
Beispiel:	`if (fflush(fp) == EOF)` `  printf("Puffer nicht geschrieben\n");`

fread

Bibliothek:	`#include <stdio.h>`
Prototyp:	`size_t fread(void *buffer, size_t size, size_t n, FILE *fp);`
Rückgabewert:	Anzahl der tatsächlich gelesenen Datenobjekte (ist diese kleiner n -> Fehler oder EOF)
Beschreibung:	liest bis zu n Datenobjekte der Länge `size` aus der Datei ein und schreibt sie in den Speicherbereich mit Startadresse `puffer`. `size_t` ist in `stdio.h` als `unsigned int` definiert.
Beispiel:	`fread(puffer,sizeof(pufferelement),100,fp);` `/* liest 100 Elemente aus der Datei in den puffer */`

fwrite

Bibliothek:	`#include <stdio.h>`
Prototyp:	`size_t fwrite(void *buffer, size_t size, size_t n, FILE *fp);`
Rückgabewert:	Anzahl der Datenobjekte, die tatsächlich in die Datei geschrieben wurden.
Beschreibung:	schreibt n Datenobjekte der Größe `size` aus dem durch `buffer` adressierten Puffer in die Datei. `size_t` ist in `stdio.h` als `unsigned int` definiert.

Beispiel:	`fwrite(puffer,sizeof(pufferelement),4,fp);`
	`/* schreibt 4 pufferelemente aus puffer in`
	`die mit fp verbundene Datei */`

feof

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int feof(FILE *fp);`
Rückgabewert:	Dateiende-Flag gesetzt: Wert ungleich 0; andernfalls: 0
Beschreibung:	Das Makro prüft, ob das Dateiende erreicht ist

Beispiel:	`while (!feof(fp))`
	`  /* führt nachfolgenden Befehl solange aus,`
	`  bis das Dateiende erreicht ist */`

fgetc

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int fgetc(FILE *fp);`
Rückgabewert:	gelesenes Zeichen, Fehler oder Dateiende: EOF
Beschreibung:	liest aus der Datei das Zeichen von der aktuellen Position (Position wird um eins erhöht).
Beispiel:	`c = fgetc(fp);` `/* weist c das nächste Zeichen der Datei zu */`

fputc

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int fputc(int c,FILE *fp);`
Rückgabewert:	Fehlerfall: EOF; andernfalls: ausgegebenes Zeichen
Beschreibung:	schreibt das Zeichen c auf die aktuelle Dateiposition (wirkungsgleich mit Makro putc)

Beispiel:	`fputc(c,fp);`
	`/* schreibt das Zeichen c an die`
	`aktuelle Position in die Datei */`

fgets

Bibliothek:	`#include <stdio.h>`
Prototyp:	`char *fgets(char *string, int n, FILE *fp);`
Rückgabewert:	Im Fehlerfall ein NULL-Pointer, andernfalls einen Zeiger auf `string`
Beschreibung:	liest Zeichenfolge aus der Datei und kopiert sie in einen durch `string` adressierten Speicherbereich; am Ende wird "\0" angefügt. Eingelesen wird bis n-1 Zeichen gelesen wurden oder bis "\n" - einschließlich '\n' ! (anders als bei gets() !!)

Beispiel:	`if (fgets(name,21,fp) != 0)`
	`  printf("1. Name: %s\n", name);`
	`  /* Ausgabe: 1. Name: xxxx */`

fputs

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int fputs(const char *string, FILE *fp);`
Rückgabewert:	Fehlerfall: EOF; andernfalls: von EOF verschiedener Wert
Beschreibung:	schreibt die Zeichenkette `string` in Datei. Das Stringende-Zeichen '\0' wird nicht in die Datei geschrieben.

Beispiel:	`fputs(string,fp);`
	`  /* schreibt string ohne Stringende-Zeichen`
	`  in die mit fp verbundene Datei */`

fscanf

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int fscanf(FILE *fp, const char *formatstring` `[,arg1,arg2,..]);`
Rückgabewert:	Anzahl der tatsächlich gelesenen Datenfelder; Dateiende erreicht: EOF
Beschreibung:	liest Elemente aus der Datei, interpretiert sie unter Kontrolle des formatstrings und legt sie in den durch `arg1, arg2, ...` adressierten Speicherplätzen ab. `formatstring` wird in der bei `scanf` beschriebenen Weise gebildet `arg1, ...` ist ein Zeiger auf eine Variable, deren Typ mit der entsprechenden Typangabe im Format-String übereinstimmen muß.
Beispiel:	`fscanf(fp,"%10s %ld",name,&nummer);` `printf("name = %s nummer = %ld\n",name,nummer);`

fprintf

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int fprintf(FILE *fp,const char formatstring, ... );`
Rückgabewert:	Anzahl der ausgegebenen Zeichen; Fehlerfall: EOF
Beschreibung:	schreibt Formatstring in die mit `fp` verbundene Datei (Formatierung zulässig)

Beispiel:	`fprintf(fp,"2 * 2 = %d \n",2*2);`
	`/* Ausgabe von: 2 * 2 = 4 in die Datei,`
	`die mit fp verbunden ist */`

putc

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int putc(int c, FILE *fp);`
Rückgabewert:	ausgegebenes Zeichen; Im Fehlerfall EOF
Beschreibung:	Das Makro schreibt das Zeichen c auf die aktuelle Position der Datei

Beispiel:	`putc(c,fp);`
	`/* schreibt das Zeichen c an die aktuelle Position`
	`der Datei, die mit fp verbunden ist */`

putchar

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int putchar(int c);`
Rückgabewert:	ausgegebenes Zeichen; Im Fehlerfall EOF
Beschreibung:	Das Makro schreibt das Zeichen c auf die Standardausgabe. (putc(c,stdout) ist wirkungsgleich zu putchar(c)).
Beispiel:	`putchar(c); /* gibt das Zeichen c über stdout aus */`

puts

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int puts(const char *string);`
Rückgabewert:	nicht negativer Wert; im Fehlerfall EOF
Beschreibung:	Schreibt den String `string` auf die Standardausgabe stdout, abschließend wird ein Newline-Zeichen ausgegeben.
Beispiel:	`puts("Hello World!");`

getc

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int getc(FILE *fp);`
Rückgabewert:	eingelesenes Zeichen, bei Fehler oder Dateiende EOF
Beschreibung:	Das Makro liest das nächste Zeichen aus der Datei.
Beispiel:	`c = getc(fp);` `/* weist c das nächste Zeichen aus der Datei zu */`

getchar

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int getchar(void);`
Rückgabewert:	eingelesenes Zeichen, bei Fehler oder Dateiende EOF
Beschreibung:	Das Makro liest das nächste Zeichen von stdin
Beispiel:	`c = getchar();` `/* weist c das nächste Zeichen von stdin zu */`

gets

Bibliothek:	`#include <stdio.h>`
Prototyp:	`char *gets(char *buffer);`
Rückgabewert:	Funktion liefert ihr Argument als Rückgabe, bei Fehler oder Dateiende NULL-Zeiger
Beschreibung:	Liest eine Textzeile bis zum Newline-Zeichen von `stdin` und schreibt sie in den durch `buffer` adressierten Speicherbereich. Newline wird durch "\0" ersetzt. **Achtung:** Gefahr des 'Buffer overflow', falls die Eingabezeile länger als der Puffer ist - besser `fgets()` verwenden.

Beispiel:	`char buffer[100];`
	`gets(buffer);`
	`/* liest Textzeile in buffer ein */`

strcat

Bibliothek:	`#include <string.h>`
Prototyp:	`char *strcat(char *s1,const char *s2);`
Rückgabewert:	erstes Argument
Beschreibung:	Kettet den String `s2` an das Ende des Strings `s1` (Stringende-Zeichen von `s1` wird überschrieben). `s1` muß `s2` aufnehmen können.
Beispiel:	`strcat(s1,s2); /* hängt s2 an s1 an */`

strcpy

Bibliothek:	`#include <string.h>`
Prototyp:	`char *strcpy(char *s1,const char *s2);`
Rückgabewert:	s1
Beschreibung:	Kopiert `s2` in den von `s1` adressierten Speicher
Beispiel:	`/* kopiert s2 in den Speicher von s1 */` `strcpy(s1,s2);`

strlen

Bibliothek:	`#include <string.h>`
Prototyp:	`size_t strlen(const char *s);`
Rückgabewert:	Länge des Strings
Beschreibung:	Liefert die Länge des Strings `s` (ohne Stringende-Zeichen). `size_t` ist in `string.h` als `unsigned int` definiert.
Beispiel:	`x = strlen("Beispiel");` `/* Ergebnis: x = 8 */`

strncat

Bibliothek:	`#include <string.h>`
Prototyp:	`char *strncat(char *s1, const char *s2, size_t n);`
Rückgabewert:	s1
Beschreibung:	Hängt die ersten `n` Zeichen des Strings `s2` und das Stringende-Zeichen an den String `s1` an. `size_t` ist in `string.h` als `unsigned int` definiert.

Beispiel:	`char s1[20] = "Hello ";`
	`char s2[10] = "Worldwide";`
	`strncat(s1,s2,5);`
	`/* Ergebnis: 'Hello World' */`

strncpy

Bibliothek:	`#include <string.h>`
Prototyp:	`char *strncpy(char *s1,const char *s2, size_t n);`
Rückgabewert:	s1
Beschreibung:	Kopiert die ersten `n` Zeichen von `s2` in den durch `s1` bezeichneten Speicherbereich. Das Stringende-Zeichen wird nicht angehängt. Ist `s2` kleiner als `n` wird mit '\0' aufgefüllt.
Beispiel:	`strncpy(s1,"Testphase",4);`

strcmp

Bibliothek:	`#include <string.h>`
Prototyp:	`int strcmp(const char *s1,const char *s2);`
Rückgabewert:	< 0: `s1` kleiner `s2` = 0: `s1` gleich `s2` > 0: `s1` größer `s2`
Beschreibung:	Vergleicht die Strings `s1` und `s2` lexikographisch

Beispiel:	`x = strcmp("HALLO","hallo");`
	`/* Ergebnis: x = -32`
	`(da in ASCII die Kleinbuchstaben`
	`höheren Code haben) */`

printf

Bibliothek:	`#include <stdio.h>`
Prototyp:	`int printf(const char *formatstring,..., [arg1,...,argn]);`
Rückgabewert:	Anzahl der ausgegebenen Zeichen; im Fehlerfall EOF
Beschreibung:	Gibt `formatstring` auf `stdout` aus. Formatelemente werden durch Werte aus der Argumentenliste ersetzt. Ein Formatelement bestimmt die Interpretation des Arguments folgendermaßen: **`% [Flags] [Feldbreite] [.Genauigkeit] Typ`** **Flags:** '+' positive Zahlen mit Vorzeichen ' ' positive Zahlen mit führendem Leerzeichen '-' linksbündige Ausgabe **Feldbreite:** Länge des Ausgabefeldes (Ausgabe > Feld: Feld wird vergrößert). Bei Zahlen werden führende Leerzeichen mit Nullen aufgefüllt. Wird als Feldbreite '*' angegeben, bestimmt ein zusätzliches Argument vom Typ int die Feldbreite. **Genauigkeit:** Anzahl der Ziffern hinter dem Dezimalpunkt bei Ausgaben vom Typ f oder e. Anzahl der signifikanten Stellen bei Ausgaben mit g. Mindestzahl an auszugebenden Ziffern bei Ganzzahl-Ausgabe Höchstzahl der auszugebenden Zeichen bei Stringausgabe. **Typ:** gibt Interpretationsart für das Argument an; folgenden Angaben sind möglich

c	Zeichen(-folge) (auch blanks, tabs und newlines werden gelesen) Default (keine Feldgrößenangabe): 1 Zeichen
d	Integer (dezimal)
i	Integer (dezimal) oder oktal (mit führender "0") oder sedezimal (mit führendem "0x" bzw "0X")
o	Integer (oktal, mit oder ohne führende "0")
x	Integer (sedezimal, mit oder ohne führendem "0x" bzw "0X")
u	Integer (dezimal, nur positiv)
e, f, g	Gleitpunktzahl (beliebige Darstellung)
s	String (Ergänzung mit abschließendem '\0')
p	Pointer

h vor d,i,o,u,x	Kurze Integerzahl (short)
l vor d,i,o,u,x	Lange Integerzahl (long)
l vor e,f,g	double
L vor e,f,g	long double

Beispiel:	`printf()` wird bei den meisten Beispielen verwendet.

malloc

Bibliothek:	`#include <stdlib.h>`
Prototyp:	`void *malloc(size_t size);`
Rückgabewert:	typenloser Zeiger auf Startadresse des bereitgestellten Speichers; im Fehlerfall NULL-Zeiger
Beschreibung:	Reserviert einen Speicherbereich von `size` Bytes. `size_t` ist in `stdlib.h` als `unsigned int` definiert.
Beispiel:	`int *feld` `feld = (int *) malloc(10*sizeof(int));`

realloc

Bibliothek:	`#include <stdlib.h>`
Prototyp:	`void *realloc(void *ptr, size_t n);`
Rückgabewert:	typenloser Zeiger auf zugewiesenen Speicher; im Fehlerfall NULL-Zeiger
Beschreibung:	Verändert die Größe des durch `ptr` bezeichneten Speicherbereichs, der zuvor durch `malloc`, `calloc` oder `realloc` reserviert wurde. `n` ist die Länge des neuen Speicherbereiches `size_t` ist in `stdlib.h` als `unsigned int` definiert. Die Daten im bereits zugewiesenen Bereich bleiben unverändert.
Beispiel:	`realloc(zeiger,anzahl*sizeof(typ));`