C Library

Ohne Anspruch auf Vollständigkeit!!!

Include-Files für `#include`:

- Wo wird gesucht: Unterschied `" "` (eigene Header) und `< >` (System-Header)
- Bitte für alle verwendeten Library-Funktionen: Header inkludieren!
- Auf Typen achten: `off_t`, `size_t`, `mode_t`, `pid_t`, ...!
- Auch Konstanten sind hilfreich, z. B. `PATH_MAX`!

Als Suchhilfe:

`stdio.h`	C-Library-File-I/O, `perror`, `popen`
`string.h`	String- und Mem-Funktionen, `strerror`
`ctype.h`	`isalpha`, ...
`sys/types.h`	Typdeklarationen
`errno.h`	`errno`, alle Werte und Makros dafür (`EPERM`, ...)
`stdlib.h`	Kreuz und quer: Makros, Typen, Funktionen: `malloc` & `free`, `exit` & `abort`, `ato`x `rand` & `srand`, `system`, `getenv`, ... Merkhilfe: `stdlib.h` enthält *betriebssystem-unabhängige* Funktionen
`unistd.h`	Viele Unix-Systemfunktionen: `exec` & `fork` & `getpid`, `getenv`, `getcwd` & `chdir`, `sleep` Unix-System-File-I/O (`read`, ...), `pipe`, ... Merkhilfe: `unistd.h` enthält *Unix-spezifische* Funktionen
`time.h`	Zeitfunktionen (auch: `sys/time.h`, `sys/times.h`)
`fcntl.h`	`open`, `creat`, `fcntl`
`sys/stat.h`	`stat`, `fstat`, Konstanten und Makros dafür
`dirent.h`	Lesen von Directories
`assert.h`	`assert`
`limits.h`	Konstanten: Limits, Min- / Max-Werte (z. B. `PATH_MAX`)
`math.h`	`sin`, ...
`setjmp.h`	`setjmp` & `longjmp` (igitt!)
`stdarg.h`	`va`-Makros (igitt!)
`signal.h`	Funktionen für Signal-Handling, Konstanten für Signale
`sys/wait.h`	`wait` und Verwandte
`sys/mman.h`	`mmap` und Verwandte
`sys/ipc.h`	Allg. Definitionen für Shm/Sem/Msg
`sys/msg.h`	Message Queues
`sys/sem.h`	Semaphore
`sys/shm.h`	Shared Memory

Unix-System-File-I/O: (aus `unistd.h` und `fcntl.h`)

- Type `int` ("Filedescriptor")
- Standardfiles: `0`, `1`, `2` (oder besser lesbar und portabel: `STDIN_FILENO`, `STDOUT_FILENO` und `STDERR_FILENO` aus `unistd.h`)
- Funktionen ohne `f`

- Returnwert `-1` für Fehler
- Direkte Systemaufrufe (Kernel, nicht Library)

`open(`*name*`, `*flags [, perm]*`)`	Öffnet File (legt ihn ev. neu an) `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_APPEND`, `O_TRUNC`, `O_CREAT`, `O_EXCL`, `O_NONBLOCK`
`creat(`*name*`, `*perm*`)`	`open(`*name*`, O_WRONLY | O_CREAT | O_TRUNC, `*perm*`)`
`dup(`*fd*`)` `dup2(`*fd1*`, `*fd2*`)`	Öffnet offenen File auf anderem Descriptor
`close(`*fd*`)`	Schließt File
`read(`*fd*`, `*buf*`, `*n*`)`	Liest
`write(`*fd*`, `*buf*`, `*n*`)`	Schreibt *n* Bytes, unformatiert ohne Rücksicht auf `\n` und `\0`
`lseek(`*fd*`, `*pos*`, `*mode*`)`	Setzt Schreib/Leseposition

C-Library-File-I/O: (aus `stdio.h`)

- Type `FILE *` (intern `typedef struct ...`) ("File-Pointer")
- Standardfiles: `stdin`, `stdout`, `stderr`
- Funktionen `f...`
- Returnwert `EOF` oder `NULL` für Fehler (`EOF` verwenden, nicht `-1`!!!)
- Achtung: ***Gepuffert***
  - `stderr` am Terminal: Unbuffered
  - `stdin` und `stdout` am Terminal: Line buffered (I/O erst bei `\n` oder bei Read!)
  - Alles andere: Block buffered (I/O erst wenn 4 KB voll)
- In Library programmiert, ruft intern Unix-System-File-I/O auf!
- Nicht beides gemischt auf einem File verwenden!!!

`fopen(`*name*`, `*type*`)`	Öffnet File (`r w a r+ w+ a+`)
`fdopen(`*fd*`, `*type*`)`	Öffnet File zu offenem Descriptor
`freopen(`*name*`, `*type*`, `*file*`)`	Öffnet File statt offenem File (für `stdin`, `stdout`, ...)
`fclose(`*file*`)`	Schließt File
`fflush(`*file*`)`	Schreibt Puffer
`getc(`*file*`)` `fgetc(`*file*`)`	1 Char lesen ***Achtung:*** Ergebnis ist `int` wegen `EOF`!!! `getc` ist Makro, `fgetc` ist Funktion
`getchar()`	`getc(stdin)`
`putc(`*char*`, `*file*`)` `fputc(`*char*`, `*file*`)`	1 Char schreiben `putc` ist Makro, `fputc` ist Funktion
`putchar(`*char*`)`	`putc(`*char*`, stdout)`
`fgets(`*str*`, `*len*`, `*file*`)`	String lesen (bis `\n`)
`gets(`*str*`)`	Dasselbe von `stdin` ***Achtung:*** Keine Längenprüfung!!! ***Achtung:*** `gets` verwirft das `\n`, `fgets` nicht!

`fputs(`*str*`,` *file*`)`	String schreiben
`puts(`*str*`)`	Dasselbe auf `stdout`
	Achtung: `puts` hängt `\n` an, `fputs` nicht!
`printf(`*format*`,` *arg* `...)`	Formatierte Ausgabe auf `stdout`
`fprintf(`*file*`,` *format*`,` *arg* `...)`	Formatierte Ausgabe auf *file*
`sprintf(`*str*`,` *format*`,` *arg* `...)`	Formatierte Ausgabe in *str*
	Achtung: Keine Längenprüfung! Keine Typprüfung!
	Keine Prüfung auf Anzahl der Argumente!
`scanf(`*format*`,` *ptr* `...)`	Formatiertes Lesen von `stdin`
`fscanf(`*file*`,` *format*`,` *ptr* `...)`	Formatiertes Lesen von *file*
`sscanf(`*str*`,` *format*`,` *ptr* `...)`	Formatiertes Lesen von *str*
	Achtung: Argumente müssen ***Pointer*** sein!
`fread(...)`	Blockweises unformatiertes Lesen
`fwrite(...)`	Blockweises unformatiertes Schreiben
`fseek(`*file*`,` *pos*`,` *mode*`)`	Aktuelle Position setzen
`rewind(`*file*`)`	Auf Fileanfang positionieren (Kein `f`!)
`ftell(`*file*`)`	Was ist die aktuelle Position?
	Alternativ: `fgetpos` und `fsetpos`
`ferror(`*file*`)`	Ist ein Fehler aufgetreten?
`feof(`*file*`)`	Ist End-of-File erreicht?
`clearerr(`*file*`)`	Setzt `ferror`- und `feof`-Flag zurück
`fileno(`*file*`)`	Zu *file* gehörender File-Descriptor

Andere File-Funktionen:

`remove(`*name*`)` `rmdir(`*name*`)`	Löschen von Files/Dir's
`mkdir(`*name*`,` *perm*`)`	Anlegen von Dirs
`rename(`*old*`,` *new*`)`	Umbenennen von Files/Dir's
`tmpfile()`	Kreiert und öffnet temporären File
`access(`*name*`,` *mode*`)`	Prüft Existenz und Zugriffsrechte
`stat(`*path*`,` *buf*`)`	Liest Status
	(Rechte, Länge, Owner, Filedatum, ...)
`fstat(`*fd*`,` *buf*`)`	Analog für *fd* (nicht *file*!)
`chmod(`*name*`,` *perm*`)`	Setzt Rechte

Andere Library-Funktionen:

String- und Mem-Funktionen:

`\0`-terminiert!

Achtung: Keine Längenprüfung!

- `strcat`, `strncat`, `strcpy`, `strncpy`: Strings anhängen / kopieren
- `strcmp`, `strncmp`, `strcasecmp`, `strncasecmp`: Strings nach ASCII-Wert vergleichen
- `strcoll`: Strings nach Locale vergleichen
- `strlen`: Länge eines Strings
- `strdup`: Kopie eines Strings via `malloc`
- `strchr`, `strrchr`, `strstr`: Suche in Strings (alt: `index`, `rindex`)
- `strspn`, `strcspn`, `strpbrk`, `strtok`, `strsep`: Parsen von Strings

- `atol`, `atof`, `strtol`, ...: Umwandlung von String in Zahl
- `memchr`, `memcmp`, `memcpy`, `memset`: Das gleiche für Speicherbereiche (Zähler statt `\0`)

Char-Funktionen:

- `isdigit`, `isalpha`, `islower`, `isupper`, `isalnum`, `isspace`, `iscntrl`, ...:
  Test von Char's (Locale!!!)
- `toupper`, `tolower`: Umwandlung groß/klein

Int-Funktionen:

- `rand`, `srand`: Zufallszahlen

Math-Funktionen:

- `sin`, `cos`, `tan`, ...
- `exp`, `log`, `log10`, `pow`, `sqrt`
- `fabs`, `floor`, `ceil`
- `drand48`, ...: Zufallszahlen

Error Handling: *Wir prüfen jeden nichttrivialen Library- und Kernel-Aufruf!*

- `extern int errno`: Fehlercode des letzten Library-Aufruf
  (oder 0, wenn alles ok) (aus `errno.h`)
  (*Achtung:* Wird automatisch gesetzt, aber nicht zurückgesetzt!)
- `EPERM`, `ENOENT`, ...: Konstanten dafür
- `perror(`*str*`)`: Gibt *str* und Fehlertext zu `errno` auf `stderr` aus.
  Empfohlen!!! (Einheitlichkeit, Lokalisierung, ...)
  Noch besser: `fprintf` auf `stderr` mit `argv[0]` und `strerror(errno)`
  (und ev. Name des Files o. ä.)!
  Übliches Format: *Programmname*: *Operation*: *strerror-text*
- `exit(`*rc*`)`: Ende des Programms mit angegebenem Returncode
- `atexit(`*func*`)`, `on_exit(`*func*`,` *arg*`)`: Installation von Exithandlern
  (Funktionen, die bei Programmende automatisch ausgeführt werden)
- `abort()`: Harter Programmabbruch mit `core`
- `assert(`*expr*`)`: Prüft *expr*, sonst Abbruch mit Zeilennummer (aus `assert.h`)
- `setjmp`, `longjmp`: (Finger weg!)

System-Funktionen:

- `malloc`, `free`: Dynamische Speicherverwaltung
- `getcwd`, `chdir`
- `getenv`: Lesen von Environment-Variablen
- `getopt`: Lesen von Commandline-Flags
- `sleep(`*n*`)`: Tut *n* Sekunden lang nichts (intern `alarm`)

Datum und Zeit:

- `time`: Liefert aktuelle Zeit (Sekunden seit 1970)
- `ctime`, `asctime`, `localtime`, `strftime`, ...: Zeit formatieren
- `mktime`: Die Gegenrichtung
- `clock_t`, `time_t`, `struct tm`: Typen dafür