Referenzkarte

aus: Jürgen Wolf, *C von A bis Z* (ISBN 3-89842-643-2)

Bedeutung	Syntax
<stdio.h> – Standard-E/A-Funktionen	
Datei öffnen	`FILE *fopen(const char *pfad, const char *modus);`
Datei öffnen mit bereits vorhandenem Stream	`FILE *freopen(const char *pfad,const char *modus,FILE *fz);`
Inhalt des Datei-Puffers in einen Stream schreiben	`int fflush(FILE *fz);`
Datei schließen	`int fclose(FILE *fz);`
Datei löschen	`int remove(const char *pfadname);`
Datei umbenennen	`int rename(const char *alt, const char *neu);`
Temporäre Datei erzeugen und automatisch wieder löschen	`FILE *tmpfile(void);`
Eindeutigen Namen für eine temporäre Datei erzeugen	`char *tmpnam(char *ptr);`
Einer Datei einen Puffer zuordnen	`void setbuf(FILE *fz, char *puffer);`
Einer geöffneten Datei einen Puffer zuordnen	`int setvbuf(FILE *fz, char *puffer, int modus, size_t size);`
Formatiert aus einem Stream lesen	`int fscanf(FILE *fz, const char *format, …);`
Formatiert aus der Standardeingabe (stdin) lesen	`int scanf(const char *format, …);`
Formatiert aus einem String lesen	`int sscanf(const char *puffer, const char *format, …);`
Formatiert in einen Stream schreiben	`int fprintf(FILE *fz, const char *format, …);`
Formatiert auf die Standardausgabe (stdout) schreiben	`int printf(const char *format, …);`
Formatiert in einen String schreiben	`int sprintf(char *puffer, const char *format, …);`
Formatiert in einen Stream schreiben mit Argumentenzeiger	`int vfprintf(FILE *fz, const char *format, va_list arg);`
Formatiert auf stdout schreiben mit Argumentenzeiger	`int vprintf(const char *format, va_list arg);`
Formatiert in einen String schreiben mit Argumentenzeiger	`int vsprintf(char *puffer,const char *format,va_list arg);`
Ein Zeichen von der Standardeingabe (stdin) einlesen	`int getchar(void);`
Ein Zeichen auf die Standardausgabe (stdout) schreiben	`int putchar(int ch);`
Ein Zeichen von einem Stream einlesen	`int fgetc(FILE *fz);  /* or */ int getc(FILE *fz);`
Ein Zeichen in einen Stream schreiben	`int fputc(int ch,FILE *fz);/* or */ int putc(int ch,FILE *fz);`
Ein gelesenes Zeichen in den Stream zurückschieben	`int ungetc(int zeichen, FILE *fz);`
Eine ganze Zeile von der Standardeingabe einlesen	`int gets(char *puffer); /* sicherere Alternative fgets() */`
Eine ganze Zeile auf die Standardausgabe schreiben	`int puts(const char *puffer);`
Eine ganze Zeile aus einem Stream lesen	`int fgets(char *puffer, int n, FILE *fz);`
Eine ganze Zeile in einen Stream schreiben	`int fputs(const char *puffer, FILE *fz);`
Binäres Lesen ganzer Blöcke	`size_t fread(void *ptr,size_t size,size_t n_obj,FILE *fz);`
Binäres Schreiben ganzer Blöcke	`size_t fwrite(const void *p,size_t siz,size_t n_obj,FILE *fz);`
Einen Stream positionieren	`int fseek(FILE *fz, long offset, int origin);`
Die Position eines Streams abfragen	`int ftell(FILE *fz);`
Die Position eines Streams speichern	`int fgetpos(FILE *fz, fpos_t *pos);`

Einen Stream positionieren (2. Möglichkeit)	`int fsetpos(FILE *fz, const fpos_t *pos);`
Den Stream zum Dateianfang zurücksetzen	`void rewind(FILE *fz);`
EOF-Flag überprüfen, ob der Stream am Dateiende ist	`int feof(FILE *fz);`
Fehler-Flag überprüfen, ob beim Stream ein Fehler auftrat	`int ferror(FILE *fz);`
Das Fehler- und EOF-Flag löschen	`void clearerr(FILE *fz);`
Ausgabe einer zu errno gehörigen Fehlermeldung	`void perror(const char *str);`
<stdlib.h> – Allgemeine Hilfsfunktionen	
Einen String in einen double-Wert konvertieren	`double atof(const char *str);`
Einen String in einen int-Wert konvertieren	`int atoi(const char *str);`
Einen String in einen long int-Wert konvertieren	`long int atol(const char *str);`
Einen String in einen double-Wert konvertieren	`double strtod(const char *str, char **endptr);`
Einen String in einen long-Wert konvertieren	`long strtol(const char *str, char **endptr, int base);`
Einen String in einen unsigned long-Wert konvertieren	`long strtoul(const char *str, char **endptr, int base);`
Speicherplatz allokieren	`void *malloc(size_t size);`
Speicherplatz allokieren	`void *calloc(size_t anzahl, size_t size);`
Speicherplatz allokieren	`void *realloc(void *zeiger, size_t size);`
Allokierten Speicherplatz wieder freigeben	`void free(void *ptr);`
Erzeugt Pseudo-Zufallszahl zwischen 0 und RAND_MAX	`int rand(void);`
Legt den Startpunkt für eine Pseudo-Zufallszahl fest	`void srand(unsigned int start);`
Eine Funktion zur Ende-Behandlung eintragen	`int atexit(void (* funktion) (void));`
Normale Programmbeendigung	`void exit(int status);`
Nicht normale Programmbeendigung	`void abort(void);`
Ein Kommando zur Ausführung an die Umgebung übergeben	`int system(const char *str);`
Aktuelle Umgebungsvariablen des Systems abfragen	`char *getenv(const char *name);`
Quotient und Rest einer Division berechnen (int)	`div_t div(int zaehler, int nenner);`
Quotient und Rest einer Division berechnen (long)	`ldiv_t ldiv(long zaehler, long nenner);`
Absolutwert eines int-Arguments	`int abs(int num);`
Absolutwert eines long-Arguments	`long labs(long num);`
Binäre Suche	`void *bsearch(const void *key, const void *start, size_t n, size_t size, int (*cmp)(const void *, const void *))`
Quicksort (Sortieren)	`void qsort(void *array, size_t n, size_t size, int (*cmp)(const void *, const void *))`
<string.h> – Funktionen für Strings	
Bestimmte Anzahl Bytes in Quelle nach Ziel kopieren	`void memcpy(void *ziel,const void *quelle, size_t anzahl);`
Wie memcpy, nur wird der korrekte Kopiervorgang garantiert	`void memmove(void *ziel,const void *quelle, size_t anzahl);`
Sucht ein bestimmtes Zeichen in einem bestimmten Bereich	`void memchr(const void *ptr, int zeichen, size_t bereich);`
Füllt einen bestimmten Bereich mit einem Zeichen	`void memset(const void *ptr, int zeichen, size_t bereich);`

Vergleicht eine bestimmte Anzahl Bytes miteinander	`void memcmp(const void *s1, const void *s2, size_t anzahl);`
Einen String kopieren	`int strcpy(char *ziel, const char *quelle);`
Eine bestimmte Anzahl von Zeichen eines Strings kopieren	`char *strncpy(char *ziel, const char *quelle, size_t size);`
Einen String an einen anderen hängen	`char *strcat(char *str1, const char *str2);`
Einen String bestimmter Länge an einen anderen hängen	`char *strncat(char *str1, char *str2, size_t size);`
Zwei Strings miteinander vergleichen	`int strcmp(const char *str1, const char *str2);`
Zwei Strings bis zu einer gewissen Länge vergleichen	`int strncmp(const char *str1,const char *str2,size_t size);`
Länge eines Strings ermitteln (ohne Stringende-Zeichen \0)	`size_t strlen(const char *str);`
Ein Zeichen in einem String suchen (von vorne)	`char *strchr(const char *str, int zeichen);`
Ein Zeichen in einem String suchen (von hinten)	`char *strrchr(const char *str, int zeichen);`
Suchen einer bestimmten Stringfolge in einem String	`char *strstr(const char *str, const char *such_str);`
Suchen eines Zeichens aus einer Zeichenmenge im String	`char *strpbrk(const char *str, const char *zeichen_menge);`
Ermittelt die Länge der übereinstimmenden Zeichen	`size_t strspn(const char *str, const char *zeichen_menge);`
Ermittelt die Anzahl der nicht übereinstimmenden Zeichen	`size_t strcspn(const char *str, const char *zeichen_menge);`
Einen String nach bestimmten Zeichen zerlegen	`char *strtok(char *str, const char *zeichen);`
Gibt eine zur Fehlernummer gehörende Fehlermeldung aus	`char *strerror(int fehlernummer);`
Länderspezifische Vergleichsfunktion	`int strcoll(const char*str1, const char *str2);`
Länderspezifische Umwandlung von Zeichen	`size_t strxfrm(char *land, const char *is_land, size_t size);`
<ctype.h> – Testen und Umwandeln von Zeichen	
Testet auf alphanumerisches Zeichen	`int isalnum(int ch);`
Testet auf Zeichen des Alphabets	`int isalpha(int ch);`
Testet auf Steuerzeichen	`int iscntrl(int ch);`
Testet auf eine Ziffer	`int isdigit(int ch);`
Testet auf druckbares Zeichen (ohne Leerzeichen)	`int isgraph(int ch);`
Testet auf Kleinbuchstabe	`int islower(int ch);`
Testet auf druckbares Zeichen	`int isprint(int ch);`
Testet auf Interpunktionszeichen	`int ispunct(int ch);`
Testet auf Zwischenraum-Zeichen (\n \t \v \r \f, Leerzeichen)	`int isspace(int ch);`
Testet auf Großbuchstabe	`int isupper(int ch);`
Testet auf eine hexadezimale Ziffer	`int isxdigit(int ch);`
Testet auf ein ASCII-Zeichen	`int isascii(int ch);`
Wandelt das Zeichen in einen Kleinbuchstaben um	`int tolower(int ch);`
Wandelt das Zeichen in einen Großbuchstaben um	`int toupper(int ch);`
<time.h> – Datums- und Zeitfunktionen	
Verbrauchte CPU-Zeit seit dem Programmstart	`clock_t clock(void);`
Erfragen der aktuellen Kalenderzeit	`time_t time(time_t *time_ptr);`
Konvertieren vom time_t-Zeitformat zur struct tm-Zeit	`struct tm *gmtime(const time_t *time_ptr);` `struct tm *localtime(const time_t *time_ptr);`

Konvertieren vom struct tm-Zeitformat zur time_t-Zeit	`time_t mktime(const struct tm *tm_ptr);`
Konvertieren vom struct tm-Zeitformat in einen String	`char *asctime(const struct tm *tm_ptr);`
Konvertieren vom time_t-Zeitformat in einen String	`char *ctime(const time_t *time_ptr);`
Differenzen zweier Uhrzeiten im time_t-Format	`double difftime(time_t z2, time_t z1);`
Konvertieren vom struct tm-Zeitformat in einen benutzerdefinierten String	`size_t strftime(char *puf,size_t smax,const char *fmt, const struct tm *t_ptr);`
<assert.h> – Hilfe zur Fehlersuche	
Hinzufügen von Testpunkten im Programm	`void assert(int ausdruck);`
<setjmp.h> – Sprünge über die Funktionsgrenze hinweg	
Einen Programmzustand abspeichern	`int setjmp(jmp_buf env);`
Den Programmzustand wieder herstellen	`void longjmp(jmp_buf env, int wert)`
<signal.h> – Signale	
Signalhandler einrichten	`void (*signal (int signalnummer,void (*sighandler)(int)))(int)`
Ein Signal an das eigene Programm schicken	`int raise(int signalnummer);`
<stdarg.h> – Variable Argumentenliste	
Liste der Parameter definieren	`va_list ap;`
Argumentenliste mit dem ersten Argument initialisieren	`va_start(va_list ap, args);`
Nächsten Parameter aus der Liste lesen	`type va_arg(va_list ap, type);`
Argumentlesevorgang beenden	`void va_end(va_list ap);`
<stddef.h> – Standarddefinitionen	
Offset einzelner Strukturelemente ermitteln	`offsetof(struktur, strukturelement);`
<math.h> – Mathematische Funktionen	
Arcuscosinus	`double acos(double zahl)`
Arcussinus	`double asin(double zahl)`
Arcustangens	`double atan(double zahl)`
Arcustangens von zahl1 und zahl2	`double atan2(double zahl1, double zahl2)`
Cosinus	`double cos(double zahl)`
Sinus	`double sin(double zahl)`
Tangens	`double tan(double zahl)`
Cosinus hyperbolicus	`double cosh(double zahl)`
Sinus hyperbolicus	`double sinh(double zahl)`
Tangens hyperbolicus	`double tanh(double zahl)`
Exponentialfunktion berechnen	`double exp(double zahl)`
Logarithmus von zahl zur Basis e=2.71828 ...	`double log(double zahl)`
Logarithmus zur Basis 10	`double log10(double zahl)`
Quadratwurzel	`double sqrt(double zahl)`
Gleitpunktzahl aufrunden	`double ceil(double zahl)`
Absolutwert	`double fabs(double zahl)`
Gleitpunktzahl abrunden	`double floor(double zahl)`
Gleitpunktzahl aufteilen in Mantisse und Exponent	`double frexp(double zahl, int zahl2)`
Gleitpunktzahl in Vor- und Nachkommateil zerlegen	`double modf(double1 zahl1, double2 *zahl2)`
Potenz zahl1zahl2	`double pow(double zahl1, double zahl2)`
»float modulo« errechnet den Rest von zahl1/zahl2	`int fmod(double zahl1, double zahl2)`