Die wichtigsten C-Funktionen

Klaus Kusche

1.) Strings und Chars

`char *strcpy(char *dest, const char *src);`
Kopiert **`src`** nach **`dest`**.
Returnwert: **`dest`**

`char *strcat(char *dest, const char *src);`
Hängt **`src`** an **`dest`** an.
Returnwert: **`dest`**

`int strcmp(const char *s1, const char *s2);`
Vergleicht **`s1`** und **`s2`** nach ASCII-Zeichensatz.
Returnwert: **`< 0`** wenn **`s1 < s2`** , **`== 0`** wenn **`s1 == s2`** , **`> 0`** wenn **`s1 > s2`**

`int strcasecmp(const char *s1, const char *s2);`
Wie **`strcmp`**, aber Groß/Kleinschreibung wird ignoriert.

`int strcoll(const char *s1, const char *s2);`
Wie **`strcmp`**, aber Vergleich gemäß Ländereinstellungs-Zeichensatz.

`size_t strlen(const char *s);`
Returnwert: Länge von **`s`** (*ohne* abschließendem **`\0`**)

`char *strdup(const char *s);`
Macht eine mit **`malloc`** dynamisch angelegte Kopie von **`s`**.
Returnwert: Pointer auf den neu angelegten String, **`NULL`** wenn kein Platz.

`char *strchr(const char *s, int c);`
Sucht **`c`** in **`s`** (von vorne nach hinten).
Returnwert: Pointer auf **`c`** in **`s`**, **`NULL`** wenn nicht gefunden.

`char *strrchr(const char *s, int c);`
Dasselbe, von hinten nach vorne.

`char *strstr(const char *haystack, const char *needle);`
Sucht **`needle`** in **`haystack`** (von vorne nach hinten).
Returnwert: Pointer auf **`needle`** in **`haystack`**, **`NULL`** wenn nicht gefunden.

`char *strerror(int errno);`
Returnwert: Der zu **`errno`** gehörende Fehlertext.

`int isalpha(int c);`
Returnwert: Ungleich 0 wenn **`c`** Groß- oder Klein-Buchstabe ist, 0 sonst.

`int islower(int c);`
Dasselbe für Kleinbuchstabe.

`int isupper(int c);`
Dasselbe für Großbuchstabe.

`int isdigit(int c);`
Dasselbe für Ziffer.

`int isspace(int c);`
Dasselbe für "White Space" (Zwischenraum, **`\t`**, **`\r`**, **`\n`**, **`\v`**, **`\f`**).

(es gibt noch ein Dutzend weitere **`is...`**)

`int toupper(int c);`
Returnwert: Wenn **`c`** Kleinbuchstabe ist: Der entsprechende Großbuchstabe.
Sonst: **`c`** unverändert.

`int tolower(int c);`
Dasselbe, aber verwandelt in Kleinbuchstaben.

2.) File I/O

`FILE *fopen(const char *path, const char *mode);`
Öffnet den File **`path`** in Richtung **`mode`**.
Returnwert: Filepointer oder **`NULL`** bei Fehler.

`int fclose(FILE *fp);`
Schließt den File **`fp`**.
Returnwert: **`0`** wenn ok, **`EOF`** bei Fehler.

`int fflush(FILE *fp);`
Schreibt den internen Puffer von **`fp`** (wenn **`fp`** zum Schreiben offen ist).
Returnwert: **`0`** wenn ok, **`EOF`** bei Fehler.

`int fgetc(FILE *fp);`
Returnwert: Das nächste von **`fp`** gelesene Zeichen, **`EOF`** bei End of File oder Fehler.

`int getc(FILE *fp);`
Dasselbe, als Makro statt Funktion.

`int getchar(void);`
Dasselbe auf **`stdin`**.

`char *fgets(char *s, int size, FILE *fp);`
Liest eine Zeile incl. **`\n`**, aber max. (**`size - 1`**) Buchstaben, von **`fp`**, und hängt **`\0`** an.
Returnwert: **`s`** oder **`NULL`** bei End of File oder Fehler.

`char *gets(char *s);`
Dasselbe von **`stdin`**, aber ohne Längenprüfung (!) und ohne **`\n`** (!).

`int fputc(int c, FILE *fp);`
Schreibt **`c`** auf **`fp`**.
Returnwert: **`c`** oder **`EOF`** bei Fehler.

`int putc(int c, FILE *fp);`
Dasselbe als Makro.

`int putchar(int c);`
Dasselbe auf **`stdout`**.

`int fputs(const char *s, FILE *fp);`
Schreibt **`s`** auf **`fp`**.
Returnwert: **`>= 0`** wenn ok, **`EOF`** bei Fehler.

`int puts(const char *s);`
Dasselbe auf **`stdout`**, hängt zusätzlich **`\n`** an.

`int feof(FILE *fp);`
Returnwert: Ungleich **`0`** wenn das Lesen von **`fp`** End of File erreicht hat, **`0`** sonst.
Achtung: Erst gesetzt, *nachdem* eine Lesefunktion **`EOF`** geliefert hat.

`int ferror(FILE *fp);`
Returnwert: Ungleich **`0`** wenn auf **`fp`** ein Fehler aufgetreten ist, **`0`** sonst.

`int fprintf(FILE *fp, const char *format, ...);`
Gibt **`...`** gemäß **`format`** formatiert auf **`fp`** aus.
Returnwert: Anzahl der ausgegebenen Zeichen, **`< 0`** bei Fehler.

`int printf(const char *format, ...);`
Dasselbe für **`stdout`**.

`int sprintf(char *str, const char *format, ...);`
Dasselbe in einen String **`str`**.

`int snprintf(char *str, size_t size, const char *format, ...);`
Dasselbe in einen String **`str`** mit max. Länge **`size`** (**`size - 1`** Zeichen plus **`\0`**).

3.) Anderes

`int atoi(const char *str);`
Returnwert: Numerischer Wert der Zahl im String **`str`**, **`0`** wenn keine Zahl in **`str`**.

`double atof(const char *str);`
Dasselbe für Kommazahlen.

`void exit(int status);`
Beendet das Programm, meldet Exitstatus **`status`** an DOS bzw. die Linux-Shell.

`void *malloc(size_t size);`
Returnwert: Pointer auf **`size`** frisch allokierte Bytes, **`NULL`** wenn kein Platz.

`void free(void *ptr);`
Gibt den mit **`malloc`** angeforderten Block beginnend bei **`ptr`** wieder frei.

`int rand(void);`
Returnwert: Eine Zufallszahl zwischen **`0`** (incl.) und **`RAND_MAX`** (sehr große Zahl).

`void srand(unsigned int seed);`
Setzt den internen Startwert von **`rand()`**.
Typischer Aufruf: **`srand((unsigned int)(time(NULL)));`**

`int abs(int i);`
Returnwert: Absolutbetrag von **`i`**.

`double fabs(double x);`
Dasselbe für **`double`**.