ANSI C-Headerdateien

Lehrbeauftragter an der TFH-Berlin

Internet: www.tfh-berlin.de/~kempfer
www.kempfer.de
E-Mail: gerald@kempfer.de

Stand: 27. September 2004

Inhaltsverzeichnis

1. Header-Datei <assert.h> und ihre Funktionen

1.1. void assert(int expression);

Diese Funktion ist als Makro implementiert und wird üblicherweise verwendet, um Fehler in der Entwicklungsphase des Programms zu finden. Ist der Ausdruck `expression`, der als Parameter übergeben wird, gleich `0`, wird eine Meldung mit dem fehlerhaften Ausdruck, dem Namen des Programmmoduls und der Zeilennummer ausgegeben. Anschließend wird das Programm mit der Funktion `abort` (siehe auch Headerdateien `process.h` und `stdlib.h`) beendet. Ist dagegen der Ausdruck `expression` ungleich `0`, passiert gar nichts. In einem Konsolenprogramm wird die Meldung dem Datenstrom `stderr` übergeben, d.h. es wird auf dem Bildschirm ausgegeben, sofern der Datenstrom nicht umgeleitet wird. In einer grafischen Oberfläche wird die Meldung in ein Dialogfenster geschrieben.

Ist die Entwicklungsphase des Programms abgeschlossen, kann dieses Makro deaktiviert werden, indem vor der Zeile mit dem `#include <assert.h>` das Label `NDEBUG` (`#define NDEBUG`) definiert wird. Manche Compiler bieten dafür auch spezielle Kommandozeilen-Parameter für den Aufruf des Compilers an.

2. Header-Datei <ctype.h> und ihre Funktionen

2.1. int isalnum(int c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein Buchstabe (`'A' - 'Z'` oder `'a' - 'z'`, ohne deutsche Umlaute!) oder eine Ziffer (`'0' - '9'`) ist, ansonsten eine `0`.

2.2. int isalpha(int c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein Buchstabe (`'A' - 'Z'` oder `'a' - 'z'`, ohne deutsche Umlaute!) ist, ansonsten eine `0`.

2.3. int iscntrl(int c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein Steuerungszeichen (ASCII-Werte 0 bis 31 und 127) ist, ansonsten eine `0`.

2.4. int isdigit(int c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` eine Ziffer (`'0' - '9'`) ist, ansonsten eine `0`.

2.5. int isgraph(int c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein druckbares Zeichen (i.A. ASCII-Werte von 33 bis 126; ohne Leerzeichen!) ist, ansonsten eine `0`.

2.6. int isleadbyte(int c);

Liefert einen Wert ungleich `0`, wenn der Parameter `c` das erste Byte eines Multibyte-Zeichens ist, ansonsten eine `0`.

2.7. int islower(int c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein Kleinbuchstabe (`'a' - 'z'`, ohne deutsche Umlaute!) ist, ansonsten eine `0`.

2.8. int isprint(int c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein druckbares Zeichen (i.A. ASCII-Werte von 32 bis 126) ist, ansonsten eine `0`.

2.9. *int ispunct(int c);*

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein druckbares Zeichen, aber kein Leerzeichen, Buchstabe oder Ziffer ist, ansonsten eine `0`.

2.10. *int isspace(int c);*

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein "weißes Leerzeichen" (Whitespace) ist, ansonsten eine `0`. Zu den "weißen Leerzeichen" gehören der horizontale Tabulator (ASCII-Wert 9), der Zeilenvorschub (ASCII-Wert 10), der vertikale Tabulator (ASCII-Wert 11), der Seitenvorschub (ASCII-Wert 12), der Wagenrücklauf (ASCII-Wert 13) und das Leerzeichen (ASCII-Wert 32).

2.11. *int isupper(int c);*

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein Großbuchstabe (`'A' - 'Z'`, ohne deutsche Umlaute!) ist, ansonsten eine `0`.

2.12. *int isxdigit(int c);*

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` eine hexadezimale Ziffer (`'0' - '9'`, `'a' - 'f'` oder `'A' - 'F'`) ist, ansonsten eine `0`.

2.13. *int iswalnum(wint_t c);*

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein Buchstabe (`L'A' - L'Z'` oder `L'a' - L'z'`, ohne deutsche Umlaute!) oder eine Ziffer (`L'0' - L'9'`) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.14. *int iswalpha(wint_t c);*

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein Buchstabe (`L'A' - L'Z'` oder `L'a' - L'z'`, ohne deutsche Umlaute!) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.15. *int iswascii(wint_t c);*

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein "wide character" eines ASCII-Zeichens ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.16. *int iswcntrl(wint_t c);*

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein Steuerungszeichen (ASCII-Werte 0 bis 31 und 127) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.17. int iswctype(wint_t c, wctype_t desc);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") die gleichen Eigenschaften wie der Parameter `desc` hat, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.18. int iswdigit(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") eine Ziffer (`L'0' - L'9'`) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.19. int iswgraph(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein druckbares Zeichen (i.A. ASCII-Werte von 33 bis 126; ohne Leerzeichen!) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.20. int iswlower(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein Kleinbuchstabe (`L'a' - L'z'`, ohne deutsche Umlaute!) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.21. int iswprint(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein druckbares Zeichen (i.A. ASCII-Werte von 32 bis 126) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.22. int iswpunct(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein druckbares Zeichen, aber kein Leerzeichen, Buchstabe oder Ziffer ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.23. int iswspace(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein "weißes Leerzeichen" (Whitespace) ist, ansonsten eine `0`. Zu den "weißen Leerzeichen" gehören der horizontale Tabulator (ASCII-Wert 9), der Zeilenvorschub (ASCII-Wert 10), der vertikale Tabulator (ASCII-Wert 11), der Seitenvorschub (ASCII-Wert 12), der Wagenrücklauf (ASCII-Wert 13) und das Leerzeichen (ASCII-Wert 32).

Diese Funktion ist auch in der `wchar.h` enthalten.

2.24. *int iswupper(wint_t c);*

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein Großbuchstabe (`L'A' - L'Z'`, ohne deutsche Umlaute!) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.25. *int iswxdigit(wint_t c);*

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") eine hexadezimale Ziffer (`L'0' - L'9'`, `L'a' - L'f'` oder `L'A' - L'F'`) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.26. *int tolower(int c);*

Wandelt den angegebenen Buchstaben `c` in einen Kleinbuchstaben um. Dabei werden nur die Großbuchstaben von `'A'` bis `'Z'` umgewandelt, alle anderen Zeichen (auch die deutschen Umlaute) werden nicht geändert.

Diese Funktion ist auch in der `stdlib.h` enthalten.

2.27. *int toupper(int c);*

Wandelt den angegebenen Buchstaben `c` in einen Großbuchstaben um. Dabei werden nur die Kleinbuchstaben von `'a'` bis `'z'` umgewandelt, alle anderen Zeichen (auch die deutschen Umlaute) werden nicht geändert.

Diese Funktion ist auch in der `stdlib.h` enthalten.

2.28. *int towlower(wint_t c);*

Wandelt den angegebenen Buchstaben `c` ("wide character") in einen Kleinbuchstaben um. Dabei werden nur die Großbuchstaben von `L'A'` bis `L'Z'` umgewandelt, alle anderen Zeichen (auch die deutschen Umlaute) werden nicht geändert.

Diese Funktion ist auch in der `wchar.h` enthalten.

2.29. *int towupper(wint_t c);*

Wandelt den angegebenen Buchstaben `c` ("wide character") in einen Großbuchstaben um. Dabei werden nur die Kleinbuchstaben von `L'a'` bis `L'z'` umgewandelt, alle anderen Zeichen (auch die deutschen Umlaute) werden nicht geändert.

Diese Funktion ist auch in der `wchar.h` enthalten.

3. Header-Datei <io.h> und ihre Funktionen

*3.1. int remove(const char *path);*

Löscht die Datei, die im Parameter `path` angegeben ist. Die Dateiangabe kann einschließlich Laufwerk und Verzeichnis erfolgen. Die Datei darf nicht offen sein! Die Funktion liefert eine `0` zurück, wenn die Datei gelöscht wurde. Konnte die Datei dagegen nicht gelöscht werden, wird eine `-1` zurückgegeben und die Variable `errno` wird auf einen der folgenden konstanten Werte gesetzt:

`EACCES` - Datei hat Read-Only-Attribut.
`ENOENT` - Datei oder Verzeichnis wurde nicht gefunden oder der Parameter `path` gibt ein Verzeichnis und nicht eine Datei an.

Diese Funktion ist auch in der `stdio.h` enthalten.

*3.2. int rename(const char *oldname, const char *newname);*

Benennt eine Datei oder ein Verzeichnis von `oldname` nach `newname` um. Dabei kann im Parameter `oldname` auch Laufwerk und Verzeichnis angegeben werden. Wird auch beim Parameter `newname` Laufwerk und/oder Verzeichnis angegeben, wird die Datei an die angegebene Stelle verschoben und evtl. gleichzeitig umbenannt (funktioniert nur für Dateien; Verzeichnisse können nicht verschoben werden!). Die Funktion liefert eine `0` zurück, wenn die Datei bzw. das Verzeichnis umbenannt (und evtl. verschoben) wurde. Konnte die Operation dagegen nicht erfolgreich durchgeführt werden, wird ein Wert ungleich `0` zurückgegeben und die Variable `errno` wird auf einen der folgenden konstanten Werte gesetzt:

`EACCES` - Datei oder Verzeichnis im Parameter `newname` existiert bereits, oder konnte nicht erzeugt werden (z.B. ungültige Verzeichnisangabe, schreibgeschützte Diskette, usw.) oder es wurde versucht, ein Verzeichnis zu verschieben.
`ENOENT` - Datei oder Verzeichnis (Parameter `oldname`) wurde nicht gefunden.
`EINVAL` - Datei oder Verzeichnis (Parameter `newname`) beinhaltet ungültige Zeichen.

Diese Funktion ist auch in der `stdio.h` enthalten.

4. Header-Datei `<locale.h>` und ihre Funktionen

5. Header-Datei <malloc.h> und ihre Funktionen

5.1. void *calloc(size_t num, size_t size);

Reserviert einen Speicherbereich im Arbeitsspeicher. Dieser Speicherbereich ist ein Array mit `num` Elementen der Größe `size` Bytes. Jedes Byte dieses Speicherbereichs wird mit `0` initialisiert. Zurückgegeben wird ein Zeiger auf den reservierten Speicherbereich oder ein `NULL`-Zeiger, wenn nicht genügend zusammenhängender Speicherbereich vorhanden ist.

Diese Funktion ist auch in der `stdlib.h` enthalten.

5.2. void free(void *memblock);

Gibt den Speicherbereich `memblock`, der zuvor mit einer der Funktionen `calloc`, `malloc` oder `realloc` reserviert wurde, wieder frei. Wird als Parameter der `NULL`-Zeiger übergeben, wird der Parameter ignoriert und die Funktion sofort beendet. Wird dagegen ein Zeiger auf einen Speicherbereich übergeben, der nicht zuvor mit einer der oben genannten Funktionen reserviert wurde, führt zu einem Fehler.

Diese Funktion ist auch in der `stdlib.h` enthalten.

5.3. void *malloc(size_t size);

Reserviert einen Speicherbereich im Arbeitsspeicher. Dieser Speicherbereich hat die Größe `size` Bytes. Die Werte der einzelnen Bytes des reservierten Speicherbereichs sind undefiniert. Zurückgegeben wird ein Zeiger auf den reservierten Speicherbereich oder ein `NULL`-Zeiger, wenn nicht genügend zusammenhängender Speicherbereich vorhanden ist.

Diese Funktion ist auch in der `stdlib.h` enthalten.

5.4. void *realloc(void *memblock, size_t size);

Verändert die Größe des reservierten Speicherbereichs `memblock` auf `size` Bytes und gibt den Zeiger auf den reservierten Speicherbereich zurück (, der unter Umständen an anderer Stelle im Arbeitsspeicher steht). Ist `size` gleich `0`, wird ein `NULL`-Zeiger zurückgegeben und der reservierte Speicherbereich `memblock` wird wieder freigegeben. Ist der Zeiger `memblock` gleich dem `NULL`-Zeiger, verhält sich die Funktion `realloc` wie die Funktion `malloc`, d.h. es wird ein neuer Speicherbereich reserviert. Ist nicht genügend Arbeitsspeicher vorhanden, wird ein `NULL`-Zeiger zurückgegeben; der bisher reservierte Speicherbereich `memblock` wird dabei aber nicht verändert.

Diese Funktion ist auch in der `stdlib.h` enthalten.

6. Header-Datei <math.h> und ihre Funktionen

6.1. int abs(int n);

Liefert den Absolutwert des Wertes n vom Typ int.

Diese Funktion ist auch in der stdlib.h enthalten.

6.2. double acos(double x);

Liefert den Arcuscosinus von x. Der Parameter x muß zwischen -1 und +1 liegen. Das Ergebnis liegt zwischen 0 und Pi. Fehler können über die Funktion _matherr-Funktion abgefragt werden.

6.3. double asin(double x);

Liefert den Arcussinus von x. Der Parameter x muß zwischen -1 und +1 liefen. Das Ergebnis liegt zwischen -Pi/2 und +Pi/2. Fehler können über die Funktion _matherr-Funktion abgefragt werden.

6.4. double atan(double x);

Liefert den Arcustangens von x. Das Ergebnis liegt zwischen -Pi/2 und +Pi/2. Fehler können über die Funktion _matherr-Funktion abgefragt werden.

6.5. double atan2(double y, double x);

Liefert den Arcustangens von y/x. Das Ergebnis liegt zwischen -Pi und +Pi. Fehler können über die Funktion _matherr-Funktion abgefragt werden.

6.6. double atof(const char *string);

Konvertiert einen String in eine Fließkommazahl des Typs double. Das Ergebnis ist gleich 0.0, wenn aus dem String keine Fließkommazahl konvertiert werden kann. Liegt die Zahl im String außerhalb des Wertebereichs von double, so ist das Ergebnis nicht definiert.

Diese Funktion ist auch in der stdlib.h enthalten.

6.7. double _cabs(struct _complex z);

Liefert den absoluten Wert der komplexen Zahl z (bestehend aus Real- und Imaginärteil). Fehler können über die Funktion _matherr-Funktion abgefragt werden.

6.8. double ceil(double x);

Rundet den Wert x auf die nächste ganze Zahl auf.

6.9. double cos(double x);

Liefert den Cosinus von `x`. Das Ergebnis liegt zwischen -1 und +1. Fehler können über die Funktion `_matherr`-Funktion abgefragt werden.

6.10. double cosh(double x);

Liefert den Cosinus Hyperbolicus von `x`. Fehler können über die Funktion `_matherr`-Funktion abgefragt werden.

6.11. double exp(double x);

Liefert den Expontialwert von `x`, also e^x.

6.12. double fabs(double x);

Liefert den Absolutwert der Fließkommazahl `x`.

6.13. double floor(double x);

Rundet den Wert `x` auf die nächste ganze Zahl ab.

6.14. double fmod(double x, double y);

Berechnet den Divisionsrest (Modulo) von den zwei Fließkommazahlen `x` und `y`. Das Ergebnis hat das gleiche Vorzeichen wie der Parameter `x`.

6.15. double frexp(double x, int *expptr);

Liefert die Mantisse und den Exponenten des Wertes `x`, d.h. `x = Mantisse * 2`$^{\texttt{Exponent}}$. Der Exponent wird in der ganzen Zahl gespeichert, auf die der Zeiger `expptr` zeigt. Ist `x` gleich 0, so sind Mantisse und Exponent gleich 0.

6.16. long labs(long n);

Liefert den Absolutwert des `long`-Wertes `n`.

Diese Funktion ist auch in der `stdlib.h` enthalten.

6.17. double ldexp(double x, int exp);

Berechnet eine Fließkommazahl aus Mantisse `x` und Exponenten `exp`, also `x * 2`$^{\texttt{exp}}$.

6.18. double log(double x);

Liefert den natürlichen Logarithmus von `x`. Fehler können über die Funktion `_matherr`-Funktion abgefragt werden.

6.19. double log10(double x);

Liefert den dekadischen (10er) Logarithmus von `x`. Fehler können über die Funktion `_matherr`-Funktion abgefragt werden.

6.20. int _matherr(struct _exception *except);

Zeigt einen Fehler in mathematischen Berechnungen an: Ist das Ergebnis gleich 0, so ist ein Fehler aufgetreten, der in der Struktur `*except` näher beschrieben ist. Ferner wird der Wert `errno` auf den entsprechenden Fehlerwert gesetzt. Ist das Ergebnis ungleich 0, so ist kein Fehler aufgetreten.

Die Struktur `struct _exception` ist in `math.h` wie folgt definiert:

```
struct _exception
{  int type;          // Fehlertyp (siehe unten)
   char *name;        // Funktionsname
   double arg1, arg2; // Werte der Parameter
   double retval;     // Funktionsergebnis
};
```

Folgende Werte sind für das Element `type` definiert:

`_DOMAIN` - Fehler im Argument
`_SING` - Singularität
`_OVERFLOW` - Ergebnis zu groß
`_PLOSS` - teilweiser Informationsverlust
`_TLOSS` - kompletter Informationsverlust
`_UNDERFLOW` - Ergebnis zu klein

6.21. double modf(double x, double *intptr);

Splittet die Fließkommazahl `x` auf in den Nachkommaanteil und den Ganzzahlanteil. Der Ganzzahlanteil wird in der Fließkommazahl, auf die der Parameter `*intptr` zeigt, gespeichert.

6.22. double pow(double x, double y);

Liefert die Potenz x^y. Bei `y == 0.0` ist das Ergebnis gleich `1`, bei `y < 0` ist das Ergebnis gleich `INF`.

6.23. double sin(double x);

Liefert den Sinus von `x`. Fehler können über die Funktion `_matherr`-Funktion abgefragt werden.

6.24. double sinh(double x);

Liefert den Sinus Hyperbolicus von `x`. Fehler können über die Funktion `_matherr`-Funktion abgefragt werden.

6.25. double sqrt(double x);

Liefert die Wurzel von `x`. Der Parameter `x` darf nicht negativ sein! Fehler können über die Funktion `_matherr`-Funktion abgefragt werden.

6.26. double tan(double x);

Liefert den Tangens von `x`. Fehler können über die Funktion `_matherr`-Funktion abgefragt werden.

6.27. double tanh(double x);

Liefert den Tangens Hyperbolicus von `x`.

7. Header-Datei <memory.h> und ihre Funktionen

7.1. void *memchr(const void *buf, int c, size_t count);

Sucht in den ersten `count` Zeichen des Puffers `buf` nach dem Zeichen `c`. Wurde das Zeichen gefunden, gibt die Funktion einen Zeiger auf das gefundene Zeichen zurück, ansonsten einen `NULL`-Zeiger.

Diese Funktion ist auch in der `string.h` enthalten.

7.2. int memcmp(const void *buf1, const void *buf2, size_t count);

Vergleicht die ersten `count` Zeichen (auch über das Stringende-Zeichen `'\0'` hinaus) der Puffer `buf1` und `buf2`. Das Funktionsergebnis kann folgende Werte annehmen:

`< 0` - `buf1` ist kleiner als `buf2`
`0` - `buf1` ist gleich `buf2`
`> 0` - `buf1` ist größer als `buf2`

Diese Funktion ist auch in der `string.h` enthalten.

7.3. void *memcpy(void *dest, const void *src, size_t count);

Kopiert die ersten `count` Zeichen (auch über das Stringende-Zeichen `'\0'` hinaus) von Puffer `src` nach `dest`. Zurückgegeben wird der Zeiger `dest`. Überlappen sich die Puffer `src` und `dest`, ist nicht sichergestellt, dass der überlappende Bereich erst kopiert wird, bevor er überschrieben wird. In diesem Fall sollte stattdessen die Funktion `memmove` verwendet werden.

Diese Funktion ist auch in der `string.h` enthalten.

7.4. void *memset(void *dest, int c, size_t count);

Setzt die ersten `count` Zeichen (auch über das Stringende-Zeichen `'\0'` hinaus) von Puffer `dest` auf das angegebene Zeichen `c`. Zurückgegeben wird der Zeiger `dest`.

Diese Funktion ist auch in der `string.h` enthalten.

8. Header-Datei <process.h> und ihre Funktionen

8.1. void abort(void);

Beendet das Programm (bzw. Prozess) mit dem Status 3 (abnormal program termination). Dabei werden offene Datenströme nicht geschlossen!

Diese Funktion ist auch in der `stdlib.h` enthalten.

8.2. void exit(int status);

Beendet das Programm (bzw. Prozess) mit dem angegebenen Status. Alle offenen Datenströme werden zuvor geschlossen.

Diese Funktion ist auch in der `stdlib.h` enthalten.

8.3. int system(const char *command);

Führt den angegebenen Befehl `command` auf der Kommandozeilenebene (DOS-Fenster bzw. Konsole) aus. Ist der Befehl gleich dem `NULL`-Zeiger, so wird nur geprüft, ob ein Kommandozeilen-Interpreter vorhanden (Ergebnis ungleich `0`) oder nicht vorhanden (Ergebnis gleich `0`) ist. Ansonsten liefert diese Funktion das Ergebnis des Kommandozeilen-Interpreters. Ist ein Fehler aufgetreten, wird als Ergebnis eine `-1` zurückgegeben und `errno` wird auf einen der folgenden Konstanten gesetzt:

`E2BIG` - Argumentenliste ist zu lang
`ENOENT` - keinen Kommandozeilen-Interpreter gefunden
`ENOEXEC` - falscher Kommandozeilen-Interpreter
`ENOMEM` - nicht genügend Arbeitsspeicher

Diese Funktion ist auch in der `stdlib.h` enthalten.

9. Header-Datei <search.h> und ihre Funktionen

*9.1. void *bsearch(const void *key, const void *base, size_t num, size_t width, int (__cdecl *compare) (const void *elem1, const void *elem2));*

Führt eine binäre Suche in einem sortierten Array durch. Dabei ist der Parameter `key` ein Zeiger auf das zu suchende Element, `base` der Zeiger auf das erste Element des Arrays, `num` die Anzahl der Elemte im Array, `width` die Größe eines Array-Elementes und `compare` ein Zeiger auf eine Funktion, die zwei Elemente (`elem1` und `elem2`) miteinander vergleicht. Die Funktion `bsearch` liefert einen Zeiger auf das gesuchte Element im Array oder einen `NULL`-Zeiger, wenn das gesuchte Element nicht gefunden wurde. Ist das Array nicht sortiert oder enthält das Array doppelte Elemente mit gleichen Inhalten, ist das Funktionsergebnis nicht vorherzusagen.

Diese Funktion ist auch in der `stdlib.h` enthalten.

*9.2. void qsort(void *base, size_t num, size_t width, int (__cdecl *compare)(const void *elem1, const void *elem2));*

Sortiert ein Array beliebiger Elemente nach dem Quicksort-Verfahren. Dabei zeigt der Parameter `base` auf das erste Element des Arrays, `num` ist die Anzahl der Elemente im Array, `width` ist die Größe eines Elementes und `compare` ein Zeiger auf eine benutzerdefinierte Vergleichsfunktion, die zwei Elemente des Arrays vergleicht. Diese Funktion muss einen Wert kleiner 0 zurückgeben, wenn `elem1` kleiner als `elem2` ist, eine 0, wenn `elem1` und `elem2` identisch sind und einen Wert größer 0, wenn `elem1` größer als `elem2` ist. Damit wird in dem Array aufsteigend sortiert. Um eine absteigende Sortierung zu erhalten, muss in der Vergleichsfunktion das "kleiner als" und das "größer als" vertauscht werden.

Diese Funktion ist auch in der `stdlib.h` enthalten.

10. Header-Datei `<setjmp.h>` und ihre Funktionen

11. Header-Datei `<signal.h>` und ihre Funktionen

12. Header-Datei <stdio.h> und ihre Funktionen

*12.1. void clearerr(FILE *stream);*

Setzt die Fehlermarke für den angegebenen Datenstrom zurück.

*12.2. int fclose(FILE *stream);*

Schließt den angegebenen Datenstrom. Als Ergebnis wird entweder eine `0` (kein Fehler) oder die Konstante `EOF` (Fehler beim Schließen) zurückgegeben.

12.3. int _fcloseall();

Schließt alle offenen Datenströme. Als Ergebnis wird entweder die Anzahl der Datenströme, die geschlossen wurden, (kein Fehler) oder die Konstante `EOF` (Fehler beim Schließen) zurückgegeben. Die Datenströme `stdin`, `stdout`, `stderr` usw. werden durch diese Funktion nicht geschlossen!

*12.4. int feof(FILE *stream);*

Testet den angegebenen Datenstrom auf Dateiende. Die Funktion liefert einen Wert ungleich `0`, wenn das Dateiende erreicht ist, sonst eine `0`.

*12.5. int ferror(FILE *stream);*

Testet den angegebenen Datenstrom auf Fehler. Die Funktion liefert einen Wert ungleich `0`, wenn ein Fehler aufgetreten ist, sonst eine `0`.

*12.6. int fflush(FILE *stream);*

Leert den Puffer eines Datenstromes. Als Ergebnis wird eine `0` zurückgegeben, wenn der Puffer erfolgreich geleert wurde, ansonsten wird die Konstante `EOF` zurückgegeben. Das Leeren des `stdio`-Puffers funktioniert nicht mit jedem Compiler, da diese Funktion eigentlich nur für Datenströme gedacht ist, in die etwas hineingeschrieben wird!

*12.7. int fgetc(FILE *stream);*

Liest ein Zeichen aus dem angegebenen Datenstrom als ganze Zahl. Dabei sind alle Zahlen von 0 bis 255 als Zeichen bzw. ASCII-Wert des gelesenen Zeichens zu sehen. Ist ein Fehler oder Dateiende aufgetreten, wird die Konstante `EOF` zurückgegeben. In diesem Fall geben die Funktionen `feof` und `ferror` über die Fehlerart Aufschluss.

*12.8. int fgetpos(FILE *stream, fpos_t *pos);*

Liefert die Position innerhalb eines Datenstromes. Diese wird in `*pos` gespeichert. Das Funktionsergebnis liefert eine `0`, wenn die Position erfolgreich ermittelt wurde, ansonsten einen Wert ungleich `0`. Im Fehlerfall

wird die Fehlerart in errno abgelegt. Die Fehlerart kann folgende konstante Werte (definiert in errno.h) annehmen:

EBADF - kein Zugriff auf Datenstrom
EINVAL - Datenstrom nicht gültig

12.9. char *fgets(char *string, int n, FILE *stream);

Liest eine - mit \0 abgeschlossene - Zeichenkette vom angegebenen Datenstrom und gibt diese zurück. Gleichzeitig wird die eingelesene Zeichenkette in string gespeichert. Die Zeichenkette wird eingelesen bis einschließlich zum nächsten Zeilenumbruch (\n), bis zum Dateiende oder bis zur maximal zu lesenden Anzahl von Zeichen (Parameter n); je nachdem, was zuerst eintrifft. Im Fehlerfall wird ein NULL-Zeiger zurückgegeben. In diesem Fall geben die Funktionen feof und ferror über die Fehlerart Aufschluss.

12.10. wint_t fgetwc(FILE *stream);

Liest ein Zeichen ("wide character") aus dem angegebenen Datenstrom. Dabei sind alle positiven Zahlen einschließlich der 0 als Zeichen bzw. ASCII-Wert des gelesenen Zeichens zu sehen. Ist ein Fehler oder Dateiende aufgetreten, wird die Konstante WEOF zurückgegeben. In diesem Fall geben die Funktionen feof und ferror über die Fehlerart Aufschluss.

Diese Funktion ist auch in der wchar.h enthalten.

12.11. wchar_t *fgetws(wchar_t *string, int n, FILE *stream);

Liest eine - mit \0 abgeschlossene - Zeichenkette (mit "wide characters") vom angegebenen Datenstrom und gibt diese zurück. Gleichzeitig wird die eingelesene Zeichenkette in string gespeichert. Die Zeichenkette wird eingelesen bis einschließlich zum nächsten Zeilenumbruch (\n), bis zum Dateiende oder bis zur maximal zu lesenden Anzahl von Zeichen (Parameter n); je nachdem, was zuerst eintrifft. Im Fehlerfall wird ein NULL-Zeiger zurückgegeben. In diesem Fall geben die Funktionen feof und ferror über die Fehlerart Aufschluss.

Diese Funktion ist auch in der wchar.h enthalten.

12.12. FILE *fopen(const char *filename, const char *mode);

Öffnet eine Datei mit dem angegebenen Dateinamen und Modus. Die Funktion liefert einen Zeiger auf den Datenstrom der geöffneten Datei bzw. einen NULL-Zeiger, wenn die Datei nicht geöffnet werden konnte. Der Modus kann folgende Werte beinhalten:

"r" - Lesen (Datei muss existieren!)
"w" - Schreiben (Eine evtl. vorhandene Datei wird überschrieben!)
"a" - Anhängen (Datei wird angelegt, wenn sie noch nicht existiert!)
"r+" - Lesen und Schreiben (Datei muss existieren!)
"w+" - Lesen und Schreiben (Eine evtl. vorhandene Datei wird überschrieben!)
"a+" - Lesen und Anhängen (Datei wird angelegt, wenn sie noch nicht existiert!)

Diese Werte können mit folgenden Werten kombiniert werden:

"t" - Textmodus
"b" - Binärmodus

Im Textmodus wird das Zeichen STRG+Z (ASCII-Wert 26) als Dateiende-Zeichen interpretiert, im Binärmodus ist dies ein ganz normales Zeichen. Ferner wird im Textmodus beim Lesen die Kombination

von Wagenrücklauf (Carriage Return, ASCII-Wert 13) und Zeilenvorschub (Line Feed, ASCII-Wert 10) als ein Zeichen - nämlich `\n` - umgesetzt (beim Schreiben entsprechend umgedreht), während im Binärmodus es zwei Zeichen bleiben.

*12.13. int fprintf(FILE *stream, const char *format [, argument] ...);*

Schreibt formatierte Daten in den angegebenen Datenstrom. Zurückgegeben wird die Anzahl der geschriebenen Bytes, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

*12.14. int fputc(int c, FILE *stream);*

Schreibt das Zeichen `c` (angegeben als positive ganze Zahl) in den angegebenen Datenstrom. Zurückgegeben wird das geschriebene Zeichen oder die Konstante `EOF`, wenn ein Fehler aufgetreten ist.

*12.15. int fputs(const char *string, FILE *stream);*

Schreibt die Zeichenkette `string` in den angegebenen Datenstrom. Zurückgegeben wird bei Erfolg eine nicht-negative Zahl oder bei einem Fehler die Konstante `EOF`.

*12.16. wint_t fputwc(wint_t c, FILE *stream);*

Schreibt das Zeichen `c` ("wide character") in den angegebenen Datenstrom. Zurückgegeben wird das geschriebene Zeichen oder die Konstante `WEOF`, wenn ein Fehler aufgetreten ist.

Diese Funktion ist auch in der `wchar.h` enthalten.

*12.17. int fputws(const wchar_t *string, FILE *stream);*

Schreibt die Zeichenkette `string` (als "wide characters") in den angegebenen Datenstrom. Zurückgegeben wird bei Erfolg eine nicht-negative Zahl oder bei einem Fehler die Konstante `WEOF`.

Diese Funktion ist auch in der `wchar.h` enthalten.

*12.18. size_t fread(void *buffer, size_t size, size_t count, FILE *stream);*

Liest Daten vom angegebenen Datenstrom `stream` in den angegebenen Puffer `buffer`. Es werden maximal `count` Daten gelesen, die jeweils die Größe von `size` Bytes haben. Zurückgegeben wird die Anzahl der vollständig gelesenen Daten, im Idealfall also `count`. Tritt ein Fehler auf oder wird das Dateiende erreicht, kann die Anzahl der gelesenen Daten auch unterhalb von `count` liegen. In diesem Fall geben die Funktionen `feof` und `ferror` über die Fehlerart Aufschluss. Ist `size` oder `count` gleich `0`, gibt die Funktion auch eine `0` zurück und der Puffer wird nicht verändert.

*12.19. FILE *freopen(const char *path, const char *mode, FILE *stream);*

Weist einem Datenstrom eine neue Datei (Parameter `path`; bestehend aus Pfad und Dateinamen) zu. Der alte Datenstrom wird zuvor geschlossen. Diese Funktion wird üblicherweise verwendet, um die vordefinierten Datenströme `stdin`, `stdout` `stderr` in Dateien umzuleiten. Der Zugriffsmodus `mode` für

den neuen Datenstrom kann die gleichen Werte annehmen wie bei der Funktion `fopen`. Zurückgegeben wird ein Zeiger auf den neuen Datenstrom oder der `NULL`-Zeiger, wenn ein Fehler aufgetreten ist.

*12.20. int fscanf(FILE *stream, const char *format [, argument] ...);*

Liest formatierte Daten aus dem angegebenen Datenstrom. Die Funktion gibt die Anzahl der erfolgreich eingelesenen Daten (nicht die Anzahl der Bytes!) zurück. Das Funktionsergebnis `0` zeigt an, dass die Daten nicht zum angegebenen Format passen, also keine Daten gelesen wurden, obwohl Daten vorhanden sind. Tritt ein Lesefehler auf oder wurde das Dateiende erreicht, wird als Funktionsergebnis die Konstante `EOF` zurückgegeben.

*12.21. int fseek(FILE *stream, long offset, int origin);*

Verschiebt den Zeiger innerhalb der Datei (Dateistrom `stream`) zu der Position, die `offset` Bytes hinter der Position `origin` liegt. Dabei kann `origin` nur einen der folgenden konstanten Werte annehmen:

`SEEK_CUR` - aktuelle Position des Datei-Zeigers
`SEEK_END` - Dateiende
`SEEK_SET` - Dateianfang

Die Funktion liefert eine `0`, wenn der Datei-Zeiger erfolgreich gesetzt werden konnte, ansonsten einen Wert ungleich `0`.

Wenn eine Datei zum Anhängen (Modus `"a"` oder `"a+"`) geöffnet wurde, hängt die aktuelle Position des Datei-Zeigers von der letzten Ein-/Ausgabe-Operation ab, nicht davon, wo die nächsten Daten hingeschrieben werden würden. Wurde noch keine Ein-/Ausgabe-Operation mit der Datei durchgeführt, so ist die aktuelle Position des Datei-Zeigers am Anfang der Datei!

Bei Dateien, die im Textmodus geöffnet wurden, arbeitet die Funktion `fseek` aufgrund der Umsetzung des Wagenrücklaufs und des Zeilenumbruchs nur dann richtig, wenn einer der folgenden Bedingungen erfüllt ist:

- `offset` muss gleich `0` sein (`origin` beliebig) oder
- es wird am Dateianfang gestartet (`origin == SEEK_SET`) und es wird ein `offset` verwendet, der durch die Funktion `ftell` ermittelt wurde.

Ferner wird das Dateiende-Zeichen (`STRG+Z` bzw. ASCII-Wert 26) in Dateien, die im Textmodus zum Lesen und Schreiben (Modus `"r+"` oder `"w+"`) geöffnet wurden, entfernt. Dies ist nötig, da sonst die Funktionen `fseek` und `ftell` falsche Ergebnisse liefern.

*12.22. int fsetpos(FILE *stream, const fpos_t *pos);*

Setzt die Position `*pos` innerhalb des Datenstromes `stream`. Das Funktionsergebnis liefert eine `0`, wenn die Position erfolgreich gesetzt wurde ansonsten einen Wert ungleich `0`. Im Fehlerfall wird die Fehlerart in `errno` abgelegt. Die Fehlerart kann folgende konstante Werte (definiert in `errno.h`) annehmen:

`EBADF` - kein Zugriff auf Datenstrom
`EINVAL` - Datenstrom nicht gültig

*12.23. long ftell(FILE *stream);*

Liefert die aktuelle Position des Datei-Zeigers im angegebenen Datenstrom als Offset vom Dateianfang. Im Fehlerfall wird eine `-1L` zurückgegeben und die Variable `errno` ist auf einen der folgenden konstanten Werte (definiert in `errno.h`) gesetzt:

EBADF - kein Zugriff auf Datenstrom
EINVAL - Datenstrom nicht gültig

Bei Dateien, die im Textmodus geöffnet sind, entspricht das Funktionsergebnis aufgrund der Umsetzung von Wagenrücklauf und Zeilenumbruch nicht der Position in Bytes.

*12.24. int fwprintf(FILE *stream, const wchar_t *format [, argument] ...);*

Schreibt formatierte Daten (mit "wide characters") in den angegebenen Datenstrom. Zurückgegeben wird die Anzahl der geschriebenen wide characters, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

Diese Funktion ist auch in der `wchar.h` enthalten.

*12.25. size_t fwrite(const void *buffer, size_t size, size_t count, FILE *stream);*

Schreibt die Daten vom Puffer `buffer` in den angegebenen Datenstrom. Es werden `count` Daten mit jeweils `size` Bytes Größe geschrieben. Zurückgegeben wird die Anzahl der komplett geschriebenen Daten, dies entspricht - wenn kein Fehler aufgetreten ist - dem Parameter `count`. Ist die Datei im Textmodus geöffnet worden, wird jeder Zeilenumbruch `\n` in die zwei Zeichen Wagenrücklauf und Zeilenumbruch umgewandelt. Das Funktionsergebnis wird dadurch aber nicht verändert.

*12.26. int fwscanf(FILE *stream, const wchar_t *format [, argument] ...);*

Liest formatierte Daten ("wide characters") aus dem angegebenen Datenstrom. Die Funktion gibt die Anzahl der erfolgreich eingelesenen Daten (nicht die Anzahl der Bytes!) zurück. Das Funktionsergebnis `0` zeigt an, dass die Daten nicht zum angegebenen Format passen, also keine Daten gelesen wurden, obwohl Daten vorhanden sind. Tritt ein Lesefehler auf oder wurde das Dateiende erreicht, wird als Funktionsergebnis die Konstante `WEOF` zurückgegeben.

Diese Funktion ist auch in der `wchar.h` enthalten.

*12.27. int getc(FILE *stream);*

Liest ein Zeichen von dem angegebenen Datenstrom, wobei der ASCII-Wert des gelesenen Zeichens (ohne Vorzeichen, also von 0 bis 255) zurückgegeben wird. Tritt ein Fehler auf oder ist das Dateiende erreicht, wird die Konstante `EOF` zurückgegeben.

12.28. int getchar(void);

Liest ein Zeichen vom Datenstrom `stdin` (d.h. im allgemeinen von der Tastatur), wobei der ASCII-Wert des gelesenen Zeichens (ohne Vorzeichen, also von 0 bis 255) zurückgegeben wird. Tritt ein Fehler auf oder ist das Dateiende erreicht, wird die Konstante `EOF` zurückgegeben.

*12.29. char *gets(char *buffer);*

Liest eine Zeichenkette vom Datenstrom `stdin` in den Puffer `buffer`. Eingelesen wird solange, bis ein Zeilenumbruch `\n` einschließlich eingelesen wurde. Die Funktion ersetzt dann den Zeilenumbruch durch das Stringende-Zeichen `\0` (im Gegensatz zur Funktion `fgets`, bei der am Ende das Stringende-Zeichen angehangen und der Zeilenumbruch nicht ersetzt wird!). Zurückgegeben wird der Parameter selber, wenn das

Einlesen erfolgreich war. Wird ein NULL-Zeiger zurückgegeben, ist ein Fehler aufgetreten oder das Dateiende erreicht. Mit Hilfe der Funktionen ferror und feof kann die Art des Fehlers ermittelt werden.

12.30. wint_t getwc(FILE *stream);

Liest ein Zeichen ("wide character") von dem angegebenen Datenstrom, wobei der ASCII-Wert des gelesenen Zeichens (ohne Vorzeichen) zurückgegeben wird. Tritt ein Fehler auf oder ist das Dateiende erreicht, wird die Konstante WEOF zurückgegeben.

Diese Funktion ist auch in der wchar.h enthalten.

12.31. wint_t getwchar(void);

Liest ein Zeichen ("wide character") vom Datenstrom stdin (d.h. im allgemeinen von der Tastatur), wobei der ASCII-Wert des gelesenen Zeichens (ohne Vorzeichen) zurückgegeben wird. Tritt ein Fehler auf oder ist das Dateiende erreicht, wird die Konstante WEOF zurückgegeben.

Diese Funktion ist auch in der wchar.h enthalten.

12.32. wchar_t *_getws(wchar_t *buffer);

Liest eine Zeichenkette ("wide character") vom Datenstrom stdin in den Puffer buffer. Eingelesen wird solange, bis ein Zeilenumbruch \n einschließlich eingelesen wurde. Die Funktion ersetzt dann den Zeilenumbruch durch das Stringende-Zeichen \0 (im Gegensatz zur Funktion fgetws, bei der am Ende das Stringende-Zeichen angehangen und der Zeilenumbruch nicht ersetzt wird!). Zurückgegeben wird der Parameter selber, wenn das Einlesen erfolgreich war. Wird ein NULL-Zeiger zurückgegeben, ist ein Fehler aufgetreten oder das Dateiende erreicht. Mit Hilfe der Funktionen ferror und feof kann die Art des Fehlers ermittelt werden.

12.33. void perror(const char *string);

Gibt den angegebenen String plus einem Doppelpunkt und dem aktuellen Fehler auf dem Datenstrom stderr aus. Der aktuelle Fehler ist in der globalen Variable errno vom Typ int (Headerdatei errno.h) gespeichert.

Diese Funktion ist auch in der stdlib.h enthalten.

12.34. int printf(const char *format [, argument] ...);

Schreibt formatierte Daten in den Datenstrom stdout (d.h. im allgemeinen auf den Bildschirm). Zurückgegeben wird die Anzahl der geschriebenen Bytes, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

12.35. int putc(int c, FILE *stream);

Schreibt das Zeichen c (ASCII-Wert ohne Vorzeichen, also von 0 bis 255; bei größeren Zahlen werden nur die unteren 8 Bit geschrieben) in den angegebenen Datenstrom stream. Zurückgegeben wird das geschriebene Zeichen oder die Konstante EOF, wenn ein Fehler aufgetreten oder das Dateiende erreicht ist. Mit Hilfe der Funktionen ferror und feof erhält man die Art des Fehlers.

12.36. int putchar(int c);

Schreibt das Zeichen `c` (ASCII-Wert ohne Vorzeichen, also von 0 bis 255; bei größeren Zahlen werden nur die unteren 8 Bit geschrieben) in den Datenstrom `stdout` (d.h. im allgemeinen auf den Bildschirm). Zurückgegeben wird das geschriebene Zeichen oder die Konstante `EOF`, wenn ein Fehler aufgetreten oder das Dateiende erreicht ist. Mit Hilfe der Funktionen `ferror` und `feof` erhält man die Art des Fehlers.

*12.37. int puts(const char *string);*

Schreibt die Zeichenkette `string` in den Datenstrom `stdout` (d.h. im allgemeinen auf den Bildschirm). Zurückgegeben wird ein nicht-negativer Wert oder die Konstante `EOF`, wenn ein Fehler aufgetreten. Das Stringende-Zeichen `\0` wird durch einen Zeilenumbruch `\n` ersetzt.

*12.38. wint_t putwc(wint_t c, FILE *stream);*

Schreibt das Zeichen `c` ("wide character", ASCII-Wert ohne Vorzeichen) in den angegebenen Datenstrom `stream`. Zurückgegeben wird das geschriebene Zeichen oder die Konstante `WEOF`, wenn ein Fehler aufgetreten oder das Dateiende erreicht ist. Mit Hilfe der Funktionen `ferror` und `feof` erhält man die Art des Fehlers.

12.39. wint_t putwchar(wint_t c);

Schreibt das Zeichen `c` ("wide character", ASCII-Wert ohne Vorzeichen) in den Datenstrom `stdout` (d.h. im allgemeinen auf den Bildschirm). Zurückgegeben wird das geschriebene Zeichen oder die Konstante `WEOF`, wenn ein Fehler aufgetreten oder das Dateiende erreicht ist. Mit Hilfe der Funktionen `ferror` und `feof` erhält man die Art des Fehlers.

*12.40. int _putws(const wchar_t *string);*

Schreibt die Zeichenkette `string` ("wide characters") in den Datenstrom `stdout` (d.h. im allgemeinen auf den Bildschirm). Zurückgegeben wird ein nicht-negativer Wert oder die Konstante `WEOF`, wenn ein Fehler aufgetreten. Das Stringende-Zeichen `\0` wird durch einen Zeilenumbruch `\n` ersetzt.

*12.41. int remove(const char *path);*

Löscht die Datei, die im Parameter `path` angegeben ist. Die Dateiangabe kann einschließlich Laufwerk und Verzeichnis erfolgen. Die Datei darf nicht offen sein! Die Funktion liefert eine `0` zurück, wenn die Datei gelöscht wurde. Konnte die Datei dagegen nicht gelöscht werden, wird eine `-1` zurückgegeben und die Variable `errno` wird auf einen der folgenden konstanten Werte gesetzt:

`EACCES` - Datei hat Read-Only-Attribut.
`ENOENT` - Datei oder Verzeichnis wurde nicht gefunden oder der Parameter `path` gibt ein Verzeichnis und nicht eine Datei an.

Diese Funktion ist auch in der `io.h` enthalten.

12.42. int rename(const char *oldname, const char *newname);

Benennt eine Datei oder ein Verzeichnis von `oldname` nach `newname` um. Dabei kann im Parameter `oldname` auch Laufwerk und Verzeichnis angegeben werden. Wird auch beim Parameter `newname` Laufwerk und/oder Verzeichnis angegeben, wird die Datei an die angegebene Stelle verschoben und evtl. gleichzeitig umbenannt (funktioniert nur für Dateien; Verzeichnisse können nicht verschoben werden!). Die Funktion liefert eine `0` zurück, wenn die Datei bzw. das Verzeichnis umbenannt (und evtl. verschoben) wurde. Konnte die Operation dagegen nicht erfolgreich durchgeführt werden, wird ein Wert ungleich `0` zurückgegeben und die Variable `errno` wird auf einen der folgenden konstanten Werte gesetzt:

`EACCES` - Datei oder Verzeichnis im Parameter `newname` existiert bereits, oder konnte nicht erzeugt werden (z.B. ungültige Verzeichnisangabe, schreibgeschützte Diskette, usw.) oder es wurde versucht, ein Verzeichnis zu verschieben.
`ENOENT` - Datei oder Verzeichnis (Parameter `oldname`) wurde nicht gefunden.
`EINVAL` - Datei oder Verzeichnis (Parameter `newname`) beinhaltet ungültige Zeichen.

Diese Funktion ist auch in der `io.h` enthalten.

12.43. void rewind(FILE *stream);

Setzt den Datei-Zeiger an den Anfang der Datei. Der Aufruf dieser Funktion ist ähnlich dem Aufruf der Funktion `fseek(stream, 0L, SEEK_SET)`. Dabei gibt es aber folgende Unterschiede: Die Funktion `rewind` setzt Fehler- und Dateiende-Status zurück. Ferner gibt die Funktion `rewind` kein Ergebnis über den Erfolg oder Nichterfolg zurück.

Um z.B. den Tastaturpuffer zu löschen, kann die Funktion `rewind` mit dem Datenstrom `stdin` aufgerufen werden.

12.44. int scanf(const char *format [, argument] ...);

Liest formatierte Daten aus dem Datenstrom `stdin` (d.h. im allgemeinen von der Tastatur). Die Funktion gibt die Anzahl der erfolgreich eingelesenen Daten (nicht die Anzahl der Bytes!) zurück. Das Funktionsergebnis `0` zeigt an, dass die Daten nicht zum angegebenen Format passen, also keine Daten gelesen wurden, obwohl Daten vorhanden sind. Tritt ein Lesefehler auf oder wurde das Dateiende erreicht, wird als Funktionsergebnis die Konstante `EOF` zurückgegeben.

12.45. void setbuf(FILE *stream, char *buffer);

Setzt einen benutzerdefinierten Puffer für den angegebenen Datenstrom `stream`. Der Puffer muss ein Speicherbereich der Größe von `BUFSIZ` (Konstante, die in `stdio.h` definiert ist) haben. Wird als Puffer der `NULL`-Zeiger übergeben, ist der Datenstrom ungepuffert. Die Funktion muss nach dem Öffnen des Datenstromes und vor der ersten Ein-/Ausgabe-Operation aufgerufen werden. Z.B. der Datenstrom `stderr` ist standardmäßig ein ungepufferter Datenstrom, aber mit dieser Funktion kann daraus ein gepufferter Datenstrom gemacht werden.

Die Funktion `setbuf` ist durch die Funktion `setvbuf` ersetzt worden und ist nur noch aus Kompatibilitätsgründen verfügbar.

12.46. int setvbuf(FILE *stream, char *buffer, int mode, size_t size);

Setzt einen benutzerdefinierten Puffer der Größe `size` Bytes (erlaubte Werte: 2 ... 32768; intern wird der Wert zur nächsten Zweierpotenz abgerundet!) für den angegebenen Datenstrom `stream`. Zurückgegeben wird eine `0`, wenn die Funktion erfolgreich war, ansonsten ein Wert ungleich `0`. Die Funktion muss nach

dem Öffnen des Datenstromes und vor der ersten Ein-/Ausgabe-Operation aufgerufen werden. Wird als Puffer der NULL-Zeiger übergeben, reserviert sich die Funktion automatisch einen Speicherbereich der Größe size / 2 * 2 Bytes (wichtiger Unterschied zur Funktion setbuf!). Für den Puffer-Modus mode können folgende konstante Werte verwendet werden:

_IOFBF - Pufferung entsprechend der Parameter buffer und size
_IOLBF - für MS-DOS, sonst wie _IOFBF
_IONBF - keine Pufferung des Datenstroms

*12.47. int sprintf(char *buffer, const char *format [, argument] ...);*

Schreibt formatierte Daten in den angegebenen Puffer buffer, an die das Stringende-Zeichen \0 angehangen wird. Zurückgegeben wird die Anzahl der im Puffer gespeicherten Bytes.

*12.48. int sscanf(const char *buffer, const char *format [, argument] ...);*

Liest formatierte Daten aus der Zeichenkette buffer. Die Funktion gibt die Anzahl der erfolgreich eingelesenen Daten (nicht die Anzahl der Bytes!) zurück. Das Funktionsergebnis 0 zeigt an, dass die Daten nicht zum angegebenen Format passen, also keine Daten gelesen wurden, obwohl Daten vorhanden sind. Tritt ein Fehler auf oder wurde das Stringende erreicht, bevor der erste Wert eingelesen wurde, wird als Funktionsergebnis die Konstante EOF zurückgegeben.

*12.49. int swprintf(wchar_t *buffer, const wchar_t *format [, argument] ...);*

Schreibt formatierte Daten ("wide characters") in den angegebenen Puffer buffer, an die das Stringende-Zeichen \0 angehangen wird. Zurückgegeben wird die Anzahl der im Puffer gespeicherten Bytes.

Diese Funktion ist auch in der wchar.h enthalten.

*12.50. int swscanf(const wchar_t *buffer, const wchar_t *format [, argument] ...);*

Liest formatierte Daten ("wide characters") aus der Zeichenkette (ebenfalls "wide characters") buffer. Die Funktion gibt die Anzahl der erfolgreich eingelesenen Daten (nicht die Anzahl der Bytes!) zurück. Das Funktionsergebnis 0 zeigt an, dass die Daten nicht zum angegebenen Format passen, also keine Daten gelesen wurden, obwohl Daten vorhanden sind. Tritt ein Fehler auf oder wurde das Stringende erreicht, bevor der erste Wert eingelesen wurde, wird als Funktionsergebnis die Konstante WEOF zurückgegeben.

Diese Funktion ist auch in der wchar.h enthalten.

*12.51. FILE *tmpfile(void);*

Erzeugt eine temporäre Datei (Dateiname wird automatisch ermittelt) im aktuellen Verzeichnis. Wird diese Datei geschlossen oder das Programm beendet, wird sie automatisch auch wieder gelöscht. Die temporäre Datei wird im Modus "w+b" geöffnet. Ist die Funktion erfolgreich, gibt sie einen Zeiger auf den Datenstrom zurück. Im Fehlerfall wird der NULL-Zeiger zurückgegeben.

*12.52. char *tmpnam(char *string);*

Liefert einen automatisch generierten Namen für eine temporäre Datei, der im Parameter string gespeichert wird und gleichzeitig zurückgegeben wird (d.h. es wird der Parameter string zurückgegeben).

Wird als Paramter der NULL-Zeiger übergeben, wird der automatisch generierte Name in einem statischen Puffer gespeichert, dessen Adresse als Funktionsergebnis zurückgegeben wird. Kann kein Name für eine temporäre Datei ermittelt werden, wird der NULL-Zeiger zurückgegeben.

*12.53. int ungetc(int c, FILE *stream);*

Packt das zuletzt gelesene Zeichen c zurück in den angegebenen Datenstrom und löscht einen evtl. Dateiende-Status. Wie der Name es schon andeutet, macht diese Funktion die letzte Leseoperation mit getc rückgängig. Wird anschließend von dem Datenstrom gelesen, wird als erstes wieder das Zeichen c gelesen. Der Datenstrom muss zum Lesen geöffnet sein. Der Versuch, EOF als Zeichen in den Datenstrom zu packen, wird ignoriert. Konnte das Zeichen erfolgreich in den Datenstrom gepackt werden, wird das Zeichen als Funktionsergebnis zurückgegeben, ansonsten wird EOF zurückgegeben. Zeichen, die mit ungetc in den Datenstrom gepackt wurden, werden beim Aufruf von Funktionen wie fflush, fseek, fsetpos oder rewind wieder gelöscht. Die Funktion ungetc kann nicht zweimal hintereinander aufgerufen werden. Auch nach dem Aufruf von fscanf kann ungetc nicht aufgerufen werden, da fscanf selber ungetc aufruft.

*12.54. wint_t ungetwc(wint_t c, FILE *stream);*

Packt das zuletzt gelesene Zeichen c ("wide character") zurück in den angegebenen Datenstrom und löscht einen evtl. Dateiende-Status. Wie der Name es schon andeutet, macht diese Funktion die letzte Leseoperation mit getwc rückgängig. Wird anschließend von dem Datenstrom gelesen, wird als erstes wieder das Zeichen c gelesen. Der Datenstrom muss zum Lesen geöffnet sein. Der Versuch, WEOF als Zeichen in den Datenstrom zu packen, wird ignoriert. Konnte das Zeichen erfolgreich in den Datenstrom gepackt werden, wird das Zeichen als Funktionsergebnis zurückgegeben, ansonsten wird WEOF zurückgegeben. Zeichen, die mit ungetwc in den Datenstrom gepackt wurden, werden beim Aufruf von Funktionen wie fflush, fseek, fsetpos oder rewind wieder gelöscht. Die Funktion ungetwc kann nicht zweimal hintereinander aufgerufen werden. Auch nach dem Aufruf von fwscanf kann ungetwc nicht aufgerufen werden, da fwscanf selber ungetwc aufruft.

Diese Funktion ist auch in der wchar.h enthalten.

*12.55. int vfprintf(FILE *stream, const char *format, va_list argptr);*

Schreibt formatierte Daten in den angegebenen Datenstrom. Die zu schreibenden Daten werden in einer Arguemtenliste (Parameter argptr) übergeben. Zurückgegeben wird die Anzahl der geschriebenen Bytes, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

*12.56. int vfwprintf(FILE *stream, const wchar_t *format, va_list argptr);*

Schreibt formatierte Daten (mit "wide characters") in den angegebenen Datenstrom. Die zu schreibenden Daten werden in einer Arguemtenliste (Parameter argptr) übergeben. Zurückgegeben wird die Anzahl der geschriebenen wide characters, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

Diese Funktion ist auch in der wchar.h enthalten.

*12.57. int vprintf(const char *format, va_list argptr);*

Schreibt formatierte Daten in den Datenstrom stdout (d.h. im allgemeinen auf den Bildschirm). Die zu schreibenden Daten werden in einer Arguemtenliste (Parameter argptr) übergeben. Zurückgegeben wird

die Anzahl der geschriebenen Bytes, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

*12.58. int vsprintf(char *buffer, const char *format, va_list argptr);*

Schreibt formatierte Daten in den angegebenen Puffer `buffer`, an die das Stringende-Zeichen `\0` angehangen wird. Die zu schreibenden Daten werden in einer Arguemtenliste (Parameter `argptr`) übergeben. Zurückgegeben wird die Anzahl der im Puffer gespeicherten Bytes.

*12.59. int vswprintf(wchar_t *buffer, const wchar_t *format, va_list argptr);*

Schreibt formatierte Daten ("wide characters") in den angegebenen Puffer `buffer`, an die das Stringende-Zeichen `\0` angehangen wird. Die zu schreibenden Daten werden in einer Arguemtenliste (Parameter `argptr`) übergeben. Zurückgegeben wird die Anzahl der im Puffer gespeicherten Bytes.

Diese Funktion ist auch in der `wchar.h` enthalten.

*12.60. int vwprintf(const wchar_t *format, va_list argptr);*

Schreibt formatierte Daten ("wide characters") in den Datenstrom `stdout` (d.h. im allgemeinen auf den Bildschirm). Die zu schreibenden Daten werden in einer Arguemtenliste (Parameter `argptr`) übergeben. Zurückgegeben wird die Anzahl der geschriebenen Bytes, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

Diese Funktion ist auch in der `wchar.h` enthalten.

*12.61. int wprintf(const wchar_t *format [, argument] ...);*

Schreibt formatierte Daten ("wide characters") in den Datenstrom `stdout` (d.h. im allgemeinen auf den Bildschirm). Zurückgegeben wird die Anzahl der geschriebenen Bytes, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

Diese Funktion ist auch in der `wchar.h` enthalten.

*12.62. int wscanf(const wchar_t *format [, argument] ...);*

Liest formatierte Daten ("wide characters") aus dem Datenstrom `stdin` (d.h. im allgemeinen von der Tastatur). Die Funktion gibt die Anzahl der erfolgreich eingelesenen Daten (nicht die Anzahl der Bytes!) zurück. Das Funktionsergebnis `0` zeigt an, dass die Daten nicht zum angegebenen Format passen, also keine Daten gelesen wurden, obwohl Daten vorhanden sind. Tritt ein Lesefehler auf oder wurde das Dateiende erreicht, wird als Funktionsergebnis die Konstante `WEOF` zurückgegeben.

Diese Funktion ist auch in der `wchar.h` enthalten.

13. Header-Datei <stdlib.h> und ihre Funktionen

13.1. void abort(void);

Beendet das Programm (bzw. Prozess) mit dem Status 3 (abnormal program termination). Dabei werden offene Datenströme nicht geschlossen!

Diese Funktion ist auch in der `process.h` enthalten.

13.2. int abs(int n);

Liefert den Absolutwert des Wertes `n` vom Typ `int`.

Diese Funktion ist auch in der `math.h` enthalten.

*13.3. int atexit(void (__cdecl *func) (void));*

Gibt eine Funktion an, die bei Programmende ausgeführt werden soll. Dazu wird ein Zeiger auf die Funktion als Parameter übergeben. Diese Funktion darf keine Parameter und keinen Rückgabewert haben. Wird die Funktion mehrmals aufgerufen, werden alle übergebenen Funktionen nach dem LIFO-(Last-In_First-Out-)-Prinzip gespeichert, d.h. bei Programmende wird die zuletzt angegebene Funktion als erstes und die zuerst angegebene als letzte aufgerufen. Die Funktion `atexit` liefert eine `0` zurück, wenn sie erfolgreich war, ansonsten einen Wert ungleich `0`.

*13.4. double atof(const char *string);*

Konvertiert einen String in eine Fließkommazahl des Typs `double`. Das Ergebnis ist gleich `0.0`, wenn aus dem String keine Fließkommazahl konvertiert werden kann. Liegt die Zahl im String außerhalb des Wertebereichs von `double`, so ist das Ergebnis nicht definiert.

Diese Funktion ist auch in der `math.h` enthalten.

*13.5. int atoi(const char *string);*

Konvertiert einen String in eine ganze Zahl des Typs `int`. Das Ergebnis ist gleich `0`, wenn aus dem String keine ganze Zahl konvertiert werden kann. Liegt die Zahl im String außerhalb des Wertebereichs von `int`, so ist das Ergebnis nicht definiert.

*13.6. long atol(const char *string);*

Konvertiert einen String in eine ganze Zahl des Typs `long`. Das Ergebnis ist gleich `0L`, wenn aus dem String keine ganze Zahl konvertiert werden kann. Liegt die Zahl im String außerhalb des Wertebereichs von `long`, so ist das Ergebnis nicht definiert.

*13.7. void *bsearch(const void *key, const void *base, size_t num, size_t width, int (__cdecl *compare) (const void *elem1, const void *elem2));*

Führt eine binäre Suche in einem sortierten Array durch. Dabei ist der Parameter `key` ein Zeiger auf das zu suchende Element, `base` der Zeiger auf das erste Element des Arrays, `num` die Anzahl der Elemte im Array,

`width` die Größe eines Array-Elementes und `compare` ein Zeiger auf eine Funktion, die zwei Elemente (`elem1` und `elem2`) miteinander vergleicht. Die Funktion `bsearch` liefert einen Zeiger auf das gesuchte Element im Array oder einen `NULL`-Zeiger, wenn das gesuchte Element nicht gefunden wurde. Ist das Array nicht sortiert oder enthält das Array doppelte Elemente mit gleichen Inhalten, ist das Funktionsergebnis nicht vorherzusagen.

Diese Funktion ist auch in der `search.h` enthalten.

*13.8. void *calloc(size_t num, size_t size);*

Reserviert einen Speicherbereich im Arbeitsspeicher. Dieser Speicherbereich ist ein Array mit `num` Elementen der Größe `size` Bytes. Jedes Byte dieses Speicherbereichs wird mit `0` initialisiert. Zurückgegeben wird ein Zeiger auf den reservierten Speicherbereich oder ein `NULL`-Zeiger, wenn nicht genügend zusammenhängender Speicherbereich vorhanden ist.

Diese Funktion ist auch in der `malloc.h` enthalten.

13.9. div_t div(int numer, int denom);

Berechnet den Quotienten und den Divisionsrest von zwei ganzen Zahlen. Der Datentyp des Rückgabewertes ist eine Struktur, die wie folgt definiert ist:

```
typedef struct _div_t {
   int quot; // Quotient
   int rem;  // Divisionsrest
} div_t;
```

Ist der Parameter `denom` gleich `0`, wird das Programm mit einer entsprechenden Fehlermeldung abgebrochen.

13.10. void exit(int status);

Beendet das Programm (bzw. Prozess) mit dem angegebenen Status. Alle offenen Datenströme werden zuvor geschlossen.

Diese Funktion ist auch in der `process.h` enthalten.

*13.11. void free(void *memblock);*

Gibt den Speicherbereich `memblock`, der zuvor mit einer der Funktionen `calloc`, `malloc` oder `realloc` reserviert wurde, wieder frei. Wird als Parameter der `NULL`-Zeiger übergeben, wird der Parameter ignoriert und die Funktion sofort beendet. Wird dagegen ein Zeiger auf einen Speicherbereich übergeben, der nicht zuvor mit einer der oben genannten Funktionen reserviert wurde, führt zu einem Fehler.

Diese Funktion ist auch in der `malloc.h` enthalten.

*13.12. char *getenv(const char *varname);*

Liefert den Wert der angegebenen Umgebungsvariable `varname`. Wurde die Umgebungsvariable nicht gefunden, wird ein `NULL`-Zeiger zurückgegeben. Das Funktionsergebnis kann nicht verwendet werden, um die Umgebungsvariable zu verändern!

13.13. long labs(long n);

Liefert den Absolutwert des `long`-Wertes `n`.

Diese Funktion ist auch in der `math.h` enthalten.

13.14. ldiv_t ldiv(long numer, long denom);

Berechnet den Quotienten und den Divisionsrest von zwei ganzen Long-Zahlen. Der Datentyp des Rückgabewertes ist eine Struktur, die wie folgt definiert ist:

```
typedef struct _ldiv_t {
   long quot; // Quotient
   long rem;  // Divisionsrest
} ldiv_t;
```

Ist der Parameter `denom` gleich `0`, wird das Programm mit einer entsprechenden Fehlermeldung abgebrochen.

13.15. void *malloc(size_t size);

Reserviert einen Speicherbereich im Arbeitsspeicher. Dieser Speicherbereich hat die Größe `size` Bytes. Die Werte der einzelnen Bytes des reservierten Speicherbereichs sind undefiniert. Zurückgegeben wird ein Zeiger auf den reservierten Speicherbereich oder ein `NULL`-Zeiger, wenn nicht genügend zusammenhängender Speicherbereich vorhanden ist.

Diese Funktion ist auch in der `malloc.h` enthalten.

13.16. int mblen(const char *mbstr, size_t count);

Ermittelt die Größe des Multibyte-Zeichens `mbstr` in Bytes. Ist `mbstr` gleich dem `NULL`-Zeiger, ist das Funktionsergebnis gleich `0`. Zeigt `mbstr` innerhalb der ersten `count` Bytes nicht auf ein gültiges Multibyte-Zeichen, wird eine `-1` zurückgegeben. Ist der Parameter `count` größer als die Konstante `MB_CUR_MAX`, so werden nur die ersten `MB_CUR_MAX` Bytes untersucht.

13.17. size_t mbstowcs(wchar_t *wcstr, const char *mbstr, size_t count);

Konvertiert eine Zeichenkette (maximal die ersten `count` Zeichen) von Multibyte-Zeichen `mbstr` in eine Zeichenkette von "wide characters" `wcstr`. Zurückgegeben wird die Anzahl der konvertierten Multibyte-Zeichen. Ist der Parameter `wcstr` gleich dem `NULL`-Zeiger, wird die benötigte Größe des Zielstrings in Bytes zurückgegeben. Enthält die Multibyte-Zeichenkette `mbstr` ein ungültiges Zeichen, wird eine `-1` zurückgegeben. Die Zeichenkette `wcstr` enthält nach dem Funktionsaufruf nur dann ein Stringende-Zeichen (`L'\0'`), wenn in der Multibyte-Zeichenkette `mbstr` das Stringende-Zeichen (`'\0'`) innerhalb der ersten `count` Zeichen enthalten ist (und damit auch mit konvertiert wurde).

13.18. int mbtowc(wchar_t *wchar, const char *mbchar, size_t count);

Konvertiert das Multibyte-Zeichen `mbchar` in das "wide character" `wchar`. Zurückgegeben wird die Länge des konvertierten Multibyte-Zeichens in Bytes. Ist `mbchar` gleich dem `NULL`-Zeiger, liefert die Funktion eine `0`. Zeigt der Parameter `mbchar` innerhalb der ersten `count` Zeichen auf ein ungültiges Multibyte-Zeichen, ist das Funktionsergebnis gleich `-1`.

13.19. void perror(const char *string);

Gibt den angegebenen String plus einem Doppelpunkt und dem aktuellen Fehler auf dem Datenstrom `stderr` aus. Der aktuelle Fehler ist in der globalen Variable `errno` vom Typ `int` (Headerdatei `errno.h`) gespeichert.

Diese Funktion ist auch in der `stdio.h` enthalten.

13.20. void qsort(void *base, size_t num, size_t width, int (__cdecl *compare)(const void *elem1, const void *elem2));

Sortiert ein Array beliebiger Elemente nach dem Quicksort-Verfahren. Dabei zeigt der Parameter `base` auf das erste Element des Arrays, `num` ist die Anzahl der Elemente im Array, `width` ist die Größe eines Elementes und `compare` ein Zeiger auf eine benutzerdefinierte Vergleichsfunktion, die zwei Elemente des Arrays vergleicht. Diese Funktion muss einen Wert kleiner 0 zurückgeben, wenn `elem1` kleiner als `elem2` ist, eine 0, wenn `elem1` und `elem2` identisch sind und einen Wert größer 0, wenn `elem1` größer als `elem2` ist. Damit wird in dem Array aufsteigend sortiert. Um eine absteigende Sortierung zu erhalten, muss in der Vergleichsfunktion das "kleiner als" und das "größer als" vertauscht werden.

Diese Funktion ist auch in der `search.h` enthalten.

13.21. int rand(void);

Liefert eine Pseudo-Zahlfallszahl im Bereich von `0` und der Konstanten `RAND_MAX`. Mit Hilfe der Funktion `srand` wird zuvor der Zufallszahlen-Generator auf einen beliebigen Startwert gesetzt.

13.22. void *realloc(void *memblock, size_t size);

Verändert die Größe des reservierten Speicherbereichs `memblock` auf `size` Bytes und gibt den Zeiger auf den reservierten Speicherbereich zurück (, der unter Umständen an anderer Stelle im Arbeitsspeicher steht). Ist `size` gleich `0`, wird ein `NULL`-Zeiger zurückgegeben und der reservierte Speicherbereich `memblock` wird wieder freigegeben. Ist der Zeiger `memblock` gleich dem `NULL`-Zeiger, verhält sich die Funktion `realloc` wie die Funktion `malloc`, d.h. es wird ein neuer Speicherbereich reserviert. Ist nicht genügend Arbeitsspeicher vorhanden, wird ein `NULL`-Zeiger zurückgegeben; der bisher reservierte Speicherbereich `memblock` wird dabei aber nicht verändert.

Diese Funktion ist auch in der `malloc.h` enthalten.

13.23. int _set_error_mode(int modeval);

Verändert die Variable `__error_mode`. Diese bestimmt, wohin die Fehlermeldungen von Laufzeitfehlern, die unter Umständen das Programm beenden, hingeschrieben werden. Der Parameter `modeval` kann dabei folgende konstante Werte annehmen:

`_OUT_TO_DEFAULT` - Ziel der Fehlermeldung hängt von der Variable `__app_type` ab
`_OUT_TO_STDERR` - Fehlermeldung wird nach `stderr` geschrieben
`_OUT_TO_MSGBOX` - Fehlermeldung wird in eine Messagebox geschrieben
`_REPORT_ERRMODE` - gibt den aktuellen Wert von `__error_mode` zurück

Die Funktion liefert den alten Wert von `__error_mode` oder eine `-1`, wenn ein Fehler aufgetreten ist.

13.24. void srand(unsigned int seed);

Setzt den Startwert für die Erzeugung einer Reihe von Pseudo-Zufallszahlen (siehe Funktion `rand`). Wird für den Parameter `seed` eine 1 übergeben, wird der Zufallszahlen-Generator zurückgesetzt. Das hat den gleichen Effekt, als wenn die Funktion `srand` vor dem ersten Aufruf der Funktion `rand` gar nicht aufgerufen worden wäre.

*13.25. double strtod(const char *nptr, char **endptr);*

Wandelt einen String in eine Fließkommazahl des Typs `double` um. Dabei ist `**endptr` ein Zeiger auf den restlichen String, der nicht umgewandelt wurde. Im Fall eines Überlaufs wird die Konstante +/- `HUGE_VAL` zurückgegeben, wobei das Vorzeichen dem der zu großen Zahl entspricht. Im Fall eines Unterlaufs ist das Funktionsergebnis gleich `0`. Tritt ein Überlauf oder Unterlauf auf, wird die Variable `errno` auf die Konstante `ERANGE` gesetzt.

*13.26. long strtol(const char *nptr, char **endptr, int base);*

Wandelt einen String in eine ganze Zahl des Typs `long` um. Dabei ist `**endptr` ein Zeiger auf den restlichen String, der nicht umgewandelt wurde. Im Fall eines Überlaufs wird eine der beiden Konstanten `LONG_MAX` oder `LONG_MIN` zurückgegeben. Im Fall eines Unterlaufs ist das Funktionsergebnis gleich `0`. Tritt ein Überlauf oder Unterlauf auf, wird die Variable `errno` auf die Konstante `ERANGE` gesetzt. Der Parameter `base` gibt die Zahlenbasis für die Umwandlung an.

*13.27. unsigned long strtoul(const char *nptr, char **endptr, int base);*

Wandelt einen String in eine ganze Zahl des Typs `unsigned long` um. Dabei ist `**endptr` ein Zeiger auf den restlichen String, der nicht umgewandelt wurde. Im Fall eines Überlaufs wird die Konstante `ULONG_MAX` zurückgegeben. Im Fall eines Unterlaufs ist das Funktionsergebnis gleich `0`. Tritt ein Überlauf oder Unterlauf auf, wird die Variable `errno` auf die Konstante `ERANGE` gesetzt. Der Parameter `base` gibt die Zahlenbasis für die Umwandlung an.

*13.28. int system(const char *command);*

Führt den angegebenen Befehl `command` auf der Kommandozeilenebene (DOS-Fenster bzw. Konsole) aus. Ist der Befehl gleich dem `NULL`-Zeiger, so wird nur geprüft, ob ein Kommandozeilen-Interpreter vorhanden (Ergebnis ungleich `0`) oder nicht vorhanden (Ergebnis gleich `0`) ist. Ansonsten liefert diese Funktion das Ergebnis des Kommandozeilen-Interpreters. Ist ein Fehler aufgetreten, wird als Ergebnis eine `-1` zurückgegeben und `errno` wird auf einen der folgenden Konstanten gesetzt:

`E2BIG` - Argumentenliste ist zu lang
`ENOENT` - keinen Kommandozeilen-Interpreter gefunden
`ENOEXEC` - falscher Kommandozeilen-Interpreter
`ENOMEM` - nicht genügend Arbeitsspeicher

Diese Funktion ist auch in der `process.h` enthalten.

13.29. int tolower(int c);

Wandelt den angegebenen Buchstaben `c` in einen Kleinbuchstaben um. Dabei werden nur die Großbuchstaben von `'A'` bis `'Z'` umgewandelt, alle anderen Zeichen (auch die deutschen Umlaute) werden nicht geändert.

Diese Funktion ist auch in der `ctype.h` enthalten.

13.30. int toupper(int c);

Wandelt den angegebenen Buchstaben `c` in einen Großbuchstaben um. Dabei werden nur die Kleinbuchstaben von `'a'` bis `'z'` umgewandelt, alle anderen Zeichen (auch die deutschen Umlaute) werden nicht geändert.

Diese Funktion ist auch in der `ctype.h` enthalten.

*13.31. double wcstod(const wchar_t *nptr, wchar_t **endptr);*

Wandelt einen String mit "wide characters" in eine Fließkommazahl des Typs `double` um. Dabei ist `**endptr` ein Zeiger auf den restlichen String, der nicht umgewandelt wurde. Im Fall eines Überlaufs wird die Konstante +/-`HUGE_VAL` zurückgegeben, wobei das Vorzeichen dem der zu großen Zahl entspricht. Im Fall eines Unterlaufs ist das Funktionsergebnis gleich `0`. Tritt ein Überlauf oder Unterlauf auf, wird die Variable `errno` auf die Konstante `ERANGE` gesetzt.

Diese Funktion ist auch in der `wchar.h` enthalten.

*13.32. long wcstol(const wchar_t *nptr, wchar_t **endptr, int base);*

Wandelt einen String mit "wide characters" in eine ganze Zahl des Typs `long` um. Dabei ist `**endptr` ein Zeiger auf den restlichen String, der nicht umgewandelt wurde. Im Fall eines Überlaufs wird eine der beiden Konstanten `LONG_MAX` oder `LONG_MIN` zurückgegeben. Im Fall eines Unterlaufs ist das Funktionsergebnis gleich `0`. Tritt ein Überlauf oder Unterlauf auf, wird die Variable `errno` auf die Konstante `ERANGE` gesetzt. Der Parameter `base` gibt die Zahlenbasis für die Umwandlung an.

Diese Funktion ist auch in der `wchar.h` enthalten.

*13.33. size_t wcstombs(char *mbstr, const wchar_t *wcstr, size_t count);*

Konvertiert eine Zeichenkette (maximal die ersten `count` Zeichen) von "wide characters" `wcstr` in eine Zeichenkette von Multibyte-Zeichen `mbstr`. Zurückgegeben wird die Anzahl der konvertierten "wide characters". Ist der Parameter `mbstr` gleich dem `NULL`-Zeiger, wird die benötigte Größe des Zielstrings in Bytes zurückgegeben. Enthält die Zeichenkette der "wide characters" `wcstr` ein ungültiges Zeichen, wird eine `-1` zurückgegeben. Die Zeichenkette `mbstr` enthält nach dem Funktionsaufruf nur dann ein Stringende-Zeichen (`'\0'`), wenn in der "wide character"-Zeichenkette `wcstr` das Stringende-Zeichen (`L'\0'`) innerhalb der ersten `count` Zeichen enthalten ist (und damit auch mit konvertiert wurde).

*13.34. unsigned long wcstoul(const wchar_t *nptr, wchar_t **endptr, int base);*

Wandelt einen String mit "wide characters" in eine ganze Zahl des Typs `unsigned long` um. Dabei ist `**endptr` ein Zeiger auf den restlichen String, der nicht umgewandelt wurde. Im Fall eines Überlaufs wird die Konstante `ULONG_MAX` zurückgegeben. Im Fall eines Unterlaufs ist das Funktionsergebnis gleich `0`.

Tritt ein Überlauf oder Unterlauf auf, wird die Variable `errno` auf die Konstante `ERANGE` gesetzt. Der Parameter `base` gibt die Zahlenbasis für die Umwandlung an.

Diese Funktion ist auch in der `wchar.h` enthalten.

*13.35. int wctomb(char *mbchar, wchar_t wchar);*

Konvertiert das "wide character"-Zeichen `wchar` in das Multibyte-Zeichen `*mbchar`. Zurückgegeben wird die Länge des konvertierten "wide character"-Zeichens in Bytes. Ist `wchar` gleich dem Stringende-Zeichen (`L'\0'`), liefert die Funktion eine `0`. Ist der Parameter `wchar` ein ungültiges "wide character", ist das Funktionsergebnis gleich `-1`.

14. Header-Datei <string.h> und ihre Funktionen

*14.1. void *memchr(const void *buf, int c, size_t count);*

Sucht in den ersten `count` Zeichen (auch über das Stringende-Zeichen `'\0'` hinaus) des Puffers `buf` nach dem Zeichen `c`. Wurde das Zeichen gefunden, gibt die Funktion einen Zeiger auf das gefundene Zeichen zurück, ansonsten einen `NULL`-Zeiger.

Diese Funktion ist auch in der `memory.h` enthalten.

*14.2. int memcmp(const void *buf1, const void *buf2, size_t count);*

Vergleicht die ersten `count` Zeichen (auch über das Stringende-Zeichen `'\0'` hinaus) der Puffer `buf1` und `buf2`. Das Funktionsergebnis kann folgende Werte annehmen:

`< 0` - `buf1` ist kleiner als `buf2`
`0` - `buf1` ist gleich `buf2`
`> 0` - `buf1` ist größer als `buf2`

Diese Funktion ist auch in der `memory.h` enthalten.

*14.3. void *memcpy(void *dest, const void *src, size_t count);*

Kopiert die ersten `count` Zeichen (auch über das Stringende-Zeichen `'\0'` hinaus) von Puffer `src` nach `dest`. Zurückgegeben wird der Zeiger `dest`. Überlappen sich die Puffer `src` und `dest`, ist nicht sichergestellt, dass der überlappende Bereich erst kopiert wird, bevor er überschrieben wird. In diesem Fall sollte stattdessen die Funktion `memmove` verwendet werden.

Diese Funktion ist auch in der `memory.h` enthalten.

*14.4. void *memmove(void *dest, const void *src, size_t count);*

Verschiebt die ersten `count` Zeichen (auch über das Stringende-Zeichen `'\0'` hinaus) von Puffer `src` nach `dest`. Zurückgegeben wird der Zeiger `dest`. Überlappen sich die Puffer `src` und `dest`, wird sichergestellt, dass der überlappende Bereich erst kopiert wird, bevor er überschrieben wird.

*14.5. void *memset(void *dest, int c, size_t count);*

Setzt die ersten `count` Zeichen (auch über das Stringende-Zeichen `'\0'` hinaus) von Puffer `dest` auf das angegebene Zeichen `c`. Zurückgegeben wird der Zeiger `dest`.

Diese Funktion ist auch in der `memory.h` enthalten.

*14.6. char *strcat(char *strDestination, const char *strSource);*

Hängt die Zeichenkette `strSource` an die Zeichenkette `strDestination` an. Dabei wird das Stringende-Zeichen von `strDestination` vom ersten Zeichen von `strSource` überschrieben. Zurückgegeben wird der Zeiger `strDestination`, der dann die komplette Zeichenkette enthält. Überlappen sich die beiden Zeichenketten `strSource` und `strDestination`, ist das Ergebnis nicht vorhersehbar.

14.7. char *strchr(const char *string, int c);

Sucht das Zeichen `c` in der Zeichenkette `string`. Wird das Zeichen gefunden, wird ein Zeiger auf die erste Stelle, an der das gesuchte Zeichen steht, zurückgegeben. Wird das gesuchte Zeichen nicht gefunden, wird der `NULL`-Zeiger zurückgegeben. Das Stringende-Zeichen ist in der Suche mit eingeschlossen.

14.8. int strcmp(const char *string1, const char *string2);

Vergleicht die beiden Zeichenketten `string1` und `string2`. Das Funktionsergebnis kann folgende Werte annehmen:

`< 0` - `string1` ist kleiner als `string2`
`0` - `string1` ist gleich `string2`
`> 0` - `string1` ist größer als `string2`

14.9. int strcoll(const char *string1, const char *string2);

Vergleicht die beiden Zeichenketten `string1` und `string2` unter Berücksichtung der aktuellen Code Page. Das Funktionsergebnis kann folgende Werte annehmen:

`< 0` - `string1` ist kleiner als `string2`
`0` - `string1` ist gleich `string2`
`> 0` - `string1` ist größer als `string2`

14.10. char *strcpy(char *strDestination, const char *strSource);

Kopiert die Zeichenkette `strSource` nach `strDestination`. Zurückgegeben wird der Zeiger `strDestination`. Überlappen sich die Zeichenketten `strSource` und `strDestination`, ist nicht sichergestellt, dass der überlappende Bereich erst kopiert wird, bevor er überschrieben wird.

14.11. size_t strcspn(const char *string, const char *strCharSet);

Gibt die Länge der Zeichenkette `string` zurück, in der keins der Zeichen von `strCharSet` vorkommt. Beginnt die Zeichenkette mit einem Zeichen, das in `strCharSet` vorkommt, wird eine `0` zurückgegeben. Das abschließende Stringende-Zeichen ist in der Suche mit eingeschlossen.

14.12. char *strerror(int errnum);

Liefert einen Zeiger auf eine Zeichenkette mit einer Fehlermeldung, die zu der angegebenen Fehlernummer `errnum` gehört. Durch erneutes Aufrufen der Funktion wird diese Zeichenkette überschrieben.

14.13. size_t strlen(const char *string);

Liefert die Anzahl der Zeichen in der angegebenen Zeichenkette `string`. Dabei wird das abschließende Stringende-Zeichen nicht mitgezählt.

*14.14. char *strncat(char *strDest, const char *strSource, size_t count);*

Hängt die ersten `count` Zeichen der Zeichenkette `strSource` an die Zeichenkette `strDestination` an. Dabei wird das Stringende-Zeichen von `strDestination` vom ersten Zeichen von `strSource` überschrieben. Hat die Zeichenkette `strSource` weniger Zeichen als `count`, wird die Anzahl der Zeichen anstelle von `count` verwendet. Zurückgegeben wird der Zeiger `strDestination`, der dann die komplette Zeichenkette enthält. Überlappen sich die beiden Zeichenketten `strSource` und `strDestination`, ist das Ergebnis nicht vorhersehbar.

*14.15. int strncmp(const char *string1, const char *string2, size_t count);*

Vergleicht die ersten `count` Zeichen der beiden Zeichenketten `string1` und `string2`. Das Funktionsergebnis kann folgende Werte annehmen:

`< 0` - `string1` ist kleiner als `string2`
`0` - `string1` ist gleich `string2`
`> 0` - `string1` ist größer als `string2`

*14.16. char *strncpy(char *strDest, const char *strSource, size_t count);*

Kopiert die ersten `count` Zeichen der Zeichenkette `strSource` nach `strDestination`. Ist `count` kleiner oder gleich der Länge von `strSource`, wird nicht automatisch ein Stringende-Zeichen die kopierte Zeichenkette angehangen. Ist `count` größer als die Länge von `strSource`, werden an die kopierte Zeichenkette noch so viele Stringende-Zeichen angehangen, bis insgesamt `count` Zeichen kopiert wurden. Zurückgegeben wird der Zeiger `strDestination`. Überlappen sich die Zeichenketten `strSource` und `strDestination`, ist nicht sichergestellt, dass der überlappende Bereich erst kopiert wird, bevor er überschrieben wird.

*14.17. char *strpbrk(const char *string, const char *strCharSet);*

Sucht das erste Vorkommen eines der Zeichen von `strCharSet` in der Zeichenkette `string`. Zurückgegeben wird ein Zeiger auf das erste Vorkommen eines der gesuchten Zeichen in der Zeichenkette `string`. Wurde kein Zeichen gefunden, wird der `NULL`-Zeiger zurückgegeben. Das abschließende Stringende-Zeichen ist in der Suche nicht mit eingeschlossen.

*14.18. char *strrchr(const char *string, int c);*

Sucht das letzte Vorkommen des Zeichens `c` in der Zeichenkette `string`. Wird das Zeichen gefunden, wird ein Zeiger auf die letzte Stelle, an der das gesuchte Zeichen steht, zurückgegeben. Wird das gesuchte Zeichen nicht gefunden, wird der `NULL`-Zeiger zurückgegeben. Das Stringende-Zeichen ist in der Suche mit eingeschlossen.

*14.19. size_t strspn(const char *string, const char *strCharSet);*

Gibt die Länge der Zeichenkette `string` zurück, in der die Zeichen von `strCharSet` vorkommen. Beginnt die Zeichenkette mit einem Zeichen, das nicht in `strCharSet` vorkommt, wird eine `0` zurückgegeben. Das abschließende Stringende-Zeichen ist in der Suche nicht mit eingeschlossen.

*14.20. char *strstr(const char *string, const char *strCharSet);*

Sucht nach der Zeichenkette `strCharSet` in der Zeichenkette `string`. Zurückgegeben wird ein Zeiger auf das erste Vorkommen der Zeichenkette `strCharSet` in `string`. Wird die Zeichenkette `strCharSet` nicht gefunden, wird der `NULL`-Zeiger zurückgegeben. Ist `strCharSet` eine Zeichenkette mit der Länge 0, wird der Zeiger `string` zurückgegeben.

*14.21. char *strtok(char *strToken, const char *strDelimit);*

Liefert den ersten Teil der Zeichenkette `strToken`, der mit einem der Zeichen aus `strDelimit` begrenzt ist. In der Zeichenkette `strToken` werden durch den Funktionsaufruf die gefundenen Zeichen aus `strDelimit` durch Stringende-Zeichen ersetzt. Um den nächsten Teil zu erhalten, muss die Funktion erneut aufgerufen werden; dabei muss aber für den Parameter `strToken` der `NULL`-Zeiger eingesetzt werden. Gibt die Funktionen einen `NULL`-Zeiger zurück, wurde kein weiterer Teil gefunden.

*14.22. size_t strxfrm(char *strDest, const char *strSource, size_t count);*

Wandelt die ersten `count` Zeichen der Zeichenkette `strSource` entsprechend der lokalen Informationen (d.h. welcher Zeichensatz aktuell gültig ist) nach `strDest` um. Die Funktion liefert die Anzahl der umgewandelten Zeichen (ohne Stringende-Zeichen) zurück. Ist der Rückgabewert größer oder gleich dem Parameter `count`, ist die Zeichenkette `strDest` unvorhersehbar. Im Fehlerfall wird eine `(size_t) -1` zurückgegeben und die globale Variable `errno` wird entsprechend gesetzt.

*14.23. wchar_t *wcscat(wchar_t *strDestination, const wchar_t *strSource);*

Hängt die Zeichenkette `strSource` an die Zeichenkette `strDestination` (beides "wide characters") an. Dabei wird das Stringende-Zeichen von `strDestination` vom ersten Zeichen von `strSource` überschrieben. Zurückgegeben wird der Zeiger `strDestination`, der dann die komplette Zeichenkette enthält. Überlappen sich die beiden Zeichenketten `strSource` und `strDestination`, ist das Ergebnis nicht vorhersehbar.

Diese Funktion ist auch in der `wchar.h` enthalten.

*14.24. wchar_t *wcschr(const wchar_t *string, wint_t c);*

Sucht das Zeichen `c` in der Zeichenkette `string` (beides "wide characters"). Wird das Zeichen gefunden, wird ein Zeiger auf die erste Stelle, an der das gesuchte Zeichen steht, zurückgegeben. Wird das gesuchte Zeichen nicht gefunden, wird der `NULL`-Zeiger zurückgegeben. Das Stringende-Zeichen ist in der Suche mit eingeschlossen.

Diese Funktion ist auch in der `wchar.h` enthalten.

*14.25. int wcscmp(const wchar_t *string1, const wchar_t *string2);*

Vergleicht die beiden Zeichenketten `string1` und `string2` (beides "wide characters"). Das Funktionsergebnis kann folgende Werte annehmen:

`< 0` - `string1` ist kleiner als `string2`
`0` - `string1` ist gleich `string2`
`> 0` - `string1` ist größer als `string2`

Diese Funktion ist auch in der `wchar.h` enthalten.

*14.26. int wcscoll(const wchar_t *string1, const wchar_t *string2);*

Vergleicht die beiden Zeichenketten `string1` und `string2` (beides "wide characters") unter Berücksichtigung der aktuellen Code Page. Das Funktionsergebnis kann folgende Werte annehmen:

`< 0` - `string1` ist kleiner als `string2`
`0` - `string1` ist gleich `string2`
`> 0` - `string1` ist größer als `string2`

Diese Funktion ist auch in der `wchar.h` enthalten.

*14.27. wchar_t *wcscpy(wchar_t *strDestination, const wchar_t *strSource);*

Kopiert die Zeichenkette `strSource` nach `strDestination` (beides "wide characters"). Zurückgegeben wird der Zeiger `strDestination`. Überlappen sich die Zeichenketten `strSource` und `strDestination`, ist nicht sichergestellt, dass der überlappende Bereich erst kopiert wird, bevor er überschrieben wird.

Diese Funktion ist auch in der `wchar.h` enthalten.

*14.28. size_t wcscspn(const wchar_t *string, const wchar_t *strCharSet);*

Gibt die Länge der Zeichenkette `string` zurück, in der keins der Zeichen von `strCharSet` vorkommt (beides "wide characters"). Beginnt die Zeichenkette mit einem Zeichen, das in `strCharSet` vorkommt, wird eine `0` zurückgegeben. Das abschließende Stringende-Zeichen ist in der Suche mit eingeschlossen.

Diese Funktion ist auch in der `wchar.h` enthalten.

*14.29. size_t wcslen(const wchar_t *string);*

Liefert die Anzahl der Zeichen in der angegebenen Zeichenkette `string` ("wide characters"). Dabei wird das abschließende Stringende-Zeichen nicht mitgezählt.

Diese Funktion ist auch in der `wchar.h` enthalten.

*14.30. wchar_t *wcsncat(wchar_t *strDest, const wchar_t *strSource, size_t count);*

Hängt die ersten `count` Zeichen der Zeichenkette `strSource` an die Zeichenkette `strDestination` (beides "wide characters") an. Dabei wird das Stringende-Zeichen von `strDestination` vom ersten Zeichen von `strSource` überschrieben. Hat die Zeichenkette `strSource` weniger Zeichen als `count`, wird die Anzahl der Zeichen anstelle von `count` verwendet. Zurückgegeben wird der Zeiger `strDestination`, der dann die komplette Zeichenkette enthält. Überlappen sich die beiden Zeichenketten `strSource` und `strDestination`, ist das Ergebnis nicht vorhersehbar.

Diese Funktion ist auch in der `wchar.h` enthalten.

14.31. int wcsncmp(const wchar_t *string1, const wchar_t *string2, size_t count);

Vergleicht die ersten `count` Zeichen der beiden Zeichenketten `string1` und `string2` (beides "wide characters"). Das Funktionsergebnis kann folgende Werte annehmen:

`< 0` - `string1` ist kleiner als `string2`
`0` - `string1` ist gleich `string2`
`> 0` - `string1` ist größer als `string2`

Diese Funktion ist auch in der `wchar.h` enthalten.

14.32. wchar_t *wcsncpy(wchar_t *strDest, const wchar_t *strSource, size_t count);

Kopiert die ersten `count` Zeichen der Zeichenkette `strSource` nach `strDestination` (beides "wide characters"). Ist `count` kleiner oder gleich der Länge von `strSource`, wird nicht automatisch ein Stringende-Zeichen die kopierte Zeichenkette angehangen. Ist `count` größer als die Länge von `strSource`, werden an die kopierte Zeichenkette noch so viele Stringende-Zeichen angehangen, bis insgesamt `count` Zeichen kopiert wurden. Zurückgegeben wird der Zeiger `strDestination`. Überlappen sich die Zeichenketten `strSource` und `strDestination`, ist nicht sichergestellt, dass der überlappende Bereich erst kopiert wird, bevor er überschrieben wird.

Diese Funktion ist auch in der `wchar.h` enthalten.

14.33. wchar_t *wcspbrk(const wchar_t *string, const wchar_t *strCharSet);

Sucht das erste Vorkommen eines der Zeichen von `strCharSet` in der Zeichenkette `string` (beides "wide characters"). Zurückgegeben wird ein Zeiger auf das erste Vorkommen eines der gesuchten Zeichen in der Zeichenkette `string`. Wurde kein Zeichen gefunden, wird der `NULL`-Zeiger zurückgegeben. Das abschließende Stringende-Zeichen ist in der Suche nicht mit eingeschlossen.

Diese Funktion ist auch in der `wchar.h` enthalten.

14.34. char *wcsrchr(const wchar_t *string, int c);

Sucht das letzte Vorkommen des Zeichens `c` in der Zeichenkette `string` (beides "wide characters"). Wird das Zeichen gefunden, wird ein Zeiger auf die letzte Stelle, an der das gesuchte Zeichen steht, zurückgegeben. Wird das gesuchte Zeichen nicht gefunden, wird der `NULL`-Zeiger zurückgegeben. Das Stringende-Zeichen ist in der Suche mit eingeschlossen.

Diese Funktion ist auch in der `wchar.h` enthalten.

14.35. size_t wcsspn(const wchar_t *string, const wchar_t *strCharSet);

Gibt die Länge der Zeichenkette `string` zurück, in der die Zeichen von `strCharSet` vorkommen (beides "wide characters"). Beginnt die Zeichenkette mit einem Zeichen, das nicht in `strCharSet` vorkommt, wird eine `0` zurückgegeben. Das abschließende Stringende-Zeichen ist in der Suche nicht mit eingeschlossen.

Diese Funktion ist auch in der `wchar.h` enthalten.

*14.36. wchar_t *wcsstr(const wchar_t *string, const wchar_t *strCharSet);*

Sucht nach der Zeichenkette `strCharSet` in der Zeichenkette `string` (beides "wide characters"). Zurückgegeben wird ein Zeiger auf das erste Vorkommen der Zeichenkette `strCharSet` in `string`. Wird die Zeichenkette `strCharSet` nicht gefunden, wird der `NULL`-Zeiger zurückgegeben. Ist `strCharSet` eine Zeichenkette mit der Länge 0, wird der Zeiger `string` zurückgegeben.

Diese Funktion ist auch in der `wchar.h` enthalten.

*14.37. wchar_t *wcstok(wchar_t *strToken, const wchar_t *strDelimit);*

Liefert den ersten Teil der Zeichenkette `strToken`, der mit einem der Zeichen aus `strDelimit` begrenzt ist (beides "wide characters"). In der Zeichenkette `strToken` werden durch den Funktionsaufruf die gefundenen Zeichen aus `strDelimit` durch Stringende-Zeichen ersetzt. Um den nächsten Teil zu erhalten, muss die Funktion erneut aufgerufen werden; dabei muss aber für den Parameter `strToken` der `NULL`-Zeiger eingesetzt werden. Gibt die Funktionen einen `NULL`-Zeiger zurück, wurde kein weiterer Teil gefunden.

Diese Funktion ist auch in der `wchar.h` enthalten.

*14.38. size_t wcsxfrm(wchar_t *strDest, const wchar_t *strSource, size_t count);*

Wandelt die ersten `count` Zeichen der Zeichenkette `strSource` entsprechend der lokalen Informationen (d.h. welcher Zeichensatz aktuell gültig ist) nach `strDest` (beides "wide characters") um. Die Funktion liefert die Anzahl der umgewandelten Zeichen (ohne Stringende-Zeichen) zurück. Ist der Rückgabewert größer oder gleich dem Parameter `count`, ist die Zeichenkette `strDest` unvorhersehbar. Im Fehlerfall wird eine `(size_t) -1` zurückgegeben und die globale Variable `errno` wird entsprechend gesetzt. Diese Funktion ist auch in der `wchar.h` enthalten.

15. Header-Datei <time.h> und ihre Funktionen

*15.1. char *asctime(const struct tm *timeptr);*

Konvertiert die Zeit, die in dem Parameter `*timeptr` übergeben wird, in eine Zeichenkette um. Die Struktur `struct tm` hat folgende Felder:

```
int tm_sec   - Sekunden (0 - 59)
int tm_min   - Minuten (0 - 59)
int tm_hour  - Stunden (0 - 23)
int tm_mday  - Tag (1 - 31)
int tm_mon   - Monate seit Januar (0 - 11)
int tm_year  - Jahre seit 1900
int tm_wday  - Anzahl Tage seit Sonntag (0 - 6)
int tm_yday  - Anzahl Tage seit dem 1. Januar (0 - 365)
int tm_isdst - Sommerzeit-Flag (DST: Daylight Saving Time)
               >0: Sommerzeit, == 0 keine Sommerzeit, <0: unbekannt)
```

Die Zeichenkette, die durch diese Funktion erzeugt wird, ist 26 Zeichen lang. Davon sind die letzten beiden Zeichen ein Zeilenumbruch und das Stringende-Zeichen (`"\n\0"`). Ein erneuter Aufruf der Funktion oder ein Aufruf der Funktion `ctime` überschreibt die Zeichenkette.

15.2. clock_t clock(void);

Berechnet die Prozessorzeit (Anzahl der "clock ticks"), die seit Start des Programms vergangen ist (ist meist als `long` definiert). Ist die Prozessorzeit nicht ermittelbar, wird eine `(clock_t) -1` zurückgegeben. Wird die Prozessorzeit durch die Konstante `CLOCKS_PER_SEC` dividiert, erhält man (ungefähr) die Zeit, die seit dem Programmstart vergangen ist.

*15.3. char *ctime(const time_t *timer);*

Konvertiert die Zeit, die in dem Parameter `timer` übergeben wird, in eine Zeichenkette um (gleiches Format wie bei der Funktion `asctime`). Der Parameter `timer` gibt die Anzahl der Sekunden an, die seit 0:00 Uhr vom 1. Januar 1970 (Universal Time UTC) vergangen sind (ist meist als `long` definiert). Die lokale Zeitzone wird bei der Konvertierung berücksichtigt. Ein erneuter Aufruf der Funktion oder ein Aufruf der Funktion `asctime` überschreibt die Zeichenkette.

15.4. double difftime(time_t timer1, time_t timer0);

Berechnet die Differenz zwischen der Endzeit `timer1` und der Anfangszeit `timer0` in Sekunden. Beide Zeiten sind jeweils die Anzahl von Sekunden, die seit 0:00 Uhr vom 1. Januar 1970 (Universal Time UTC) vergangen sind.

*15.5. struct tm *gmtime(const time_t *timer);*

Konvertiert die Zeit `timer` (Anzahl von Sekunden, die seit 0:00 Uhr vom 1. Januar 1970 (Universal Time UTC) vergangen sind) in die Struktur `struct tm` (Aufbau der Struktur: siehe Funktion `asctime`). Die in der Struktur gespeicherte Zeit ist die Universal Time und nicht die lokale Zeit. Ist der Parameter negativ, wird der `NULL`-Zeiger zurückgegeben. Ein erneuter Aufruf der Funktion oder ein Aufruf der Funktion `localtime` oder `mktime` überschreibt das bisherige Ergebnis.

15.6. struct tm *localtime(const time_t *timer);

Konvertiert die Zeit `timer` (Anzahl von Sekunden, die seit 0:00 Uhr vom 1. Januar 1970 (Universal Time UTC) vergangen sind) in die Struktur `struct tm` (Aufbau der Struktur: siehe Funktion `asctime`). Die in der Struktur gespeicherte Zeit ist die Zeit in der lokalen Zeitzone. Ist der Parameter negativ, wird der `NULL`-Zeiger zurückgegeben. Ein erneuter Aufruf der Funktion oder ein Aufruf der Funktion `gmtime` oder `mktime` überschreibt das bisherige Ergebnis.

15.7. time_t mktime(struct tm *timeptr);

Konvertiert die Zeit, die in dem Parameter `*timeptr` (die lokale Zeitzone wird berücksichtigt) übergeben wird, in eine Anzahl von Sekunden, die seit 0:00 Uhr vom 1. Januar 1970 (Universal Time UTC) vergangen sind. Ist in `*timeptr` ein Datum vor dem 1. Januar 1970 bzw. ein Datum nach dem 18. Januar 2038, 19:14:07 Uhr oder ein ungültiges Datum enthalten, liefert die Funktion den Wert `(time_t) -1` zurück. Sind die Werte in `*timeptr` unvollständig, werden sie vervollständigt und normalisiert (d.h. `*timeptr` wird unter Umständern verändert!).

15.8. size_t strftime(char *strDest, size_t maxsize, const char *format, const struct tm *timeptr);

Erzeugt eine Zeichenkette mit maximale `maxsize` Zeichen, die entsprechend des benutzerdefinierten Formats `format` Zeit und/oder Datum aus `*timeptr` (Aufbau der Struktur: siehe Funktion `asctime`) enthält, und speichert diese in `strDest`. Zurückgegeben wird die tatsächliche Anzahl von Zeichen einschließlich des Stringende-Zeichens in `strDest`, wenn die erzeugte Zeichenkette nicht länger als `maxsize` ist. Ansonsten wird eine `0` zurückgegeben und der Inhalt von `strDest` ist undefiniert. Formatierungssequenzen für das benutzerdefiniert Format `format`: siehe Funktion `wcsftime`.

15.9. time_t time(time_t *timer);

Holt die Uhrzeit und Datum (Anzahl von Sekunden seit 0:00 Uhr vom 1. Januar 1970 (Universal Time UTC)) von der Systemuhr und speichert diese im Parameter `*timer`. Gleichzeitig wird dieser Wert als Funktionsergebnis zurückgegeben. Ist der Parameter gleich dem `NULL`-Zeiger, wird das Ergebnis nicht gespeichert, sondern nur zurückgegeben.

15.10. size_t wcsftime(wchar_t *strDest, size_t maxsize, const wchar_t *format, const struct tm *timeptr);

Erzeugt eine Zeichenkette mit maximale `maxsize` Zeichen ("wide characters"), die entsprechend des benutzerdefinierten Formats `format` (auch "wide characters") Zeit und/oder Datum aus `*timeptr` (Aufbau der Struktur: siehe Funktion `asctime`) enthält, und speichert diese in `strDest`. Zurückgegeben wird die tatsächliche Anzahl von Zeichen einschließlich des Stringende-Zeichens in `strDest`, wenn die erzeugte Zeichenkette nicht länger als `maxsize` ist. Ansonsten wird eine `0` zurückgegeben und der Inhalt von `strDest` ist undefiniert. Das benutzerdefiniert Format `format` können neben normalem Text noch folgende Formatierungssequenzen verwendet werden:

`%a` - abgekürtzter Name des Wochentags
`%A` - kompletter Name des Wochentags
`%b` - abgekürtzter Monatsname
`%B` - kompletter Monatsname

`%c` - komplettes Datum mit Uhrzeit (lokale Zeitzone)
`%d` - Tag im Monat (01 - 31)
`%H` - Stunden im 24-Stundenformat (00 - 23)
`%I` - Stunden im 12-Stundenformat (00 - 12)
`%j` - Tag im Jahr (001 - 366)
`%m` - Monat (01 - 12)
`%M` - Minuten (00 - 59)
`%p` - A.M. / P.M. für 12-Stundenanzeige
`%S` - Sekunden (00 - 59)
`%U` - Woche im Jahr (1. Woche beginnt mit erstem Sonntag) (00 - 53)
`%w` - Wochentag (0 - 6; Sonntag ist 0)
`%W` - Woche im Jahr (1. Woche beginnt mit erstem Montag) (00 - 53)
`%x` - komplettes Datum (lokale Zeitzone)
`%X` - komplette Uhrzeit (lokale Zeitzone)
`%y` - Jahreszahl ohne Jahrhundert (00 - 99)
`%Y` - Jahreszahl mit Jahrhundert
`%z` - Name der Zeitzone
`%Z` - abgekürzter Name der Zeitzone
`%%` - Prozentzeichen

Mit Hilfe des Flags `#` zwischen dem Prozentzeichen `%` und dem Formatierungsbuchstaben werden bei den Zahlen die führenden Nullen (wenn vorhanden) weggelassen; bei den Formatierungen `%c` und `%x` wird das lange Datumsformat verwendet.

Diese Funktion ist auch in der `wchar.h` enthalten.

16. Header-Datei <wchar.h> und ihre Funktionen

*16.1. wint_t fgetwc(FILE *stream);*

Liest ein Zeichen ("wide character") aus dem angegebenen Datenstrom. Dabei sind alle positiven Zahlen einschließlich der 0 als Zeichen bzw. ASCII-Wert des gelesenen Zeichens zu sehen. Ist ein Fehler oder Dateiende aufgetreten, wird die Konstante `WEOF` zurückgegeben. In diesem Fall geben die Funktionen `feof` und `ferror` über die Fehlerart Aufschluss.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.2. wchar_t *fgetws(wchar_t *string, int n, FILE *stream);*

Liest eine - mit `\0` abgeschlossene - Zeichenkette (mit "wide characters") vom angegebenen Datenstrom und gibt diese zurück. Gleichzeitig wird die eingelesene Zeichenkette in `string` gespeichert. Die Zeichenkette wird eingelesen bis einschließlich zum nächsten Zeilenumbruch (`\n`), bis zum Dateiende oder bis zur maximal zu lesenden Anzahl von Zeichen (Parameter `n`); je nachdem, was zuerst eintrifft. Im Fehlerfall wird ein `NULL`-Zeiger zurückgegeben. In diesem Fall geben die Funktionen `feof` und `ferror` über die Fehlerart Aufschluss.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.3. wint_t fputwc(wint_t c, FILE *stream);*

Schreibt das Zeichen `c` ("wide character") in den angegebenen Datenstrom. Zurückgegeben wird das geschriebene Zeichen oder die Konstante `WEOF`, wenn ein Fehler aufgetreten ist.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.4. int fputws(const wchar_t *string, FILE *stream);*

Schreibt die Zeichenkette `string` (als "wide characters") in den angegebenen Datenstrom. Zurückgegeben wird bei Erfolg eine nicht-negative Zahl oder bei einem Fehler die Konstante `WEOF`.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.5. int fwprintf(FILE *stream, const wchar_t *format [, argument] ...);*

Schreibt formatierte Daten (mit "wide characters") in den angegebenen Datenstrom. Zurückgegeben wird die Anzahl der geschriebenen wide characters, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.6. int fwscanf(FILE *stream, const wchar_t *format [, argument] ...);*

Liest formatierte Daten ("wide characters") aus dem angegebenen Datenstrom. Die Funktion gibt die Anzahl der erfolgreich eingelesenen Daten (nicht die Anzahl der Bytes!) zurück. Das Funktionsergebnis `0` zeigt an, dass die Daten nicht zum angegebenen Format passen, also keine Daten gelesen wurden, obwohl Daten vorhanden sind. Tritt ein Lesefehler auf oder wurde das Dateiende erreicht, wird als Funktionsergebnis die Konstante `WEOF` zurückgegeben.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.7. wint_t getwc(FILE *stream);*

Liest ein Zeichen ("wide character") von dem angegebenen Datenstrom, wobei der ASCII-Wert des gelesenen Zeichens (ohne Vorzeichen) zurückgegeben wird. Tritt ein Fehler auf oder ist das Dateiende erreicht, wird die Konstante `WEOF` zurückgegeben.

Diese Funktion ist auch in der `stdio.h` enthalten.

16.8. wint_t getwchar(void);

Liest ein Zeichen ("wide character") vom Datenstrom `stdin` (d.h. im allgemeinen von der Tastatur), wobei der ASCII-Wert des gelesenen Zeichens (ohne Vorzeichen) zurückgegeben wird. Tritt ein Fehler auf oder ist das Dateiende erreicht, wird die Konstante `WEOF` zurückgegeben.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.9. wchar_t *_getws(wchar_t *buffer);*

Liest eine Zeichenkette ("wide character") vom Datenstrom `stdin` in den Puffer `buffer`. Eingelesen wird solange, bis ein Zeilenumbruch `\n` einschließlich eingelesen wurde. Die Funktion ersetzt dann den Zeilenumbruch durch das Stringende-Zeichen `\0` (im Gegensatz zur Funktion `fgetws`, bei der am Ende das Stringende-Zeichen angehangen und der Zeilenumbruch nicht ersetzt wird!). Zurückgegeben wird der Parameter selber, wenn das Einlesen erfolgreich war. Wird ein `NULL`-Zeiger zurückgegeben, ist ein Fehler aufgetreten oder das Dateiende erreicht. Mit Hilfe der Funktionen `ferror` und `feof` kann die Art des Fehlers ermittelt werden.

Diese Funktion ist auch in der `stdio.h` enthalten.

16.10. int iswalnum(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein Buchstabe (`L'A' - L'Z'` oder `L'a' - L'z'`, ohne deutsche Umlaute!) oder eine Ziffer (`L'0' - L'9'`) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.11. int iswalpha(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein Buchstabe (`L'A' - L'Z'` oder `L'a' - L'z'`, ohne deutsche Umlaute!) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.12. int iswascii(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ein "wide character" eines ASCII-Zeichens ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.13. int iswcntrl(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein Steuerungszeichen (ASCII-Werte 0 bis 31 und 127) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.14. int iswctype(wint_t c, wctype_t desc);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") die gleichen Eigenschaften wie der Parameter `desc` hat, ansonsten eine `0`.

Diese Funktion ist auch in der `wchar.h` enthalten.

16.15. int iswdigit(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") eine Ziffer (`L'0' - L'9'`) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.16. int iswgraph(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein druckbares Zeichen (i.A. ASCII-Werte von 33 bis 126; ohne Leerzeichen!) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.17. int iswlower(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein Kleinbuchstabe (`L'a' - L'z'`, ohne deutsche Umlaute!) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.18. int iswprint(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein druckbares Zeichen (i.A. ASCII-Werte von 32 bis 126) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.19. int iswpunct(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein druckbares Zeichen, aber kein Leerzeichen, Buchstabe oder Ziffer ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.20. int iswspace(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein "weißes Leerzeichen" (Whitespace) ist, ansonsten eine `0`. Zu den "weißen Leerzeichen" gehören der horizontale Tabulator (ASCII-Wert 9), der Zeilenvorschub (ASCII-Wert 10), der vertikale Tabulator (ASCII-Wert 11), der Seitenvorschub (ASCII-Wert 12), der Wagenrücklauf (ASCII-Wert 13) und das Leerzeichen (ASCII-Wert 32).

Diese Funktion ist auch in der `ctype.h` enthalten.

16.21. int iswupper(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") ein Großbuchstabe (`L'A' - L'Z'`, ohne deutsche Umlaute!) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.22. int iswxdigit(wint_t c);

Liefert einen Wert ungleich `0`, wenn das angegebene Zeichen `c` ("wide character") eine hexadezimale Ziffer (`L'0' - L'9'`, `L'a' - L'f'` oder `L'A' - L'F'`) ist, ansonsten eine `0`.

Diese Funktion ist auch in der `ctype.h` enthalten.

*16.23. wint_t putwc(wint_t c, FILE *stream);*

Schreibt das Zeichen `c` ("wide character", ASCII-Wert ohne Vorzeichen) in den angegebenen Datenstrom `stream`. Zurückgegeben wird das geschriebene Zeichen oder die Konstante `WEOF`, wenn ein Fehler aufgetreten oder das Dateiende erreicht ist. Mit Hilfe der Funktionen `ferror` und `feof` erhält man die Art des Fehlers.

Diese Funktion ist auch in der `stdio.h` enthalten.

16.24. wint_t putwchar(wint_t c);

Schreibt das Zeichen `c` ("wide character", ASCII-Wert ohne Vorzeichen) in den Datenstrom `stdout` (d.h. im allgemeinen auf den Bildschirm). Zurückgegeben wird das geschriebene Zeichen oder die Konstante `WEOF`, wenn ein Fehler aufgetreten oder das Dateiende erreicht ist. Mit Hilfe der Funktionen `ferror` und `feof` erhält man die Art des Fehlers.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.25. int _putws(const wchar_t *string);*

Schreibt die Zeichenkette `string` ("wide characters") in den Datenstrom `stdout` (d.h. im allgemeinen auf den Bildschirm). Zurückgegeben wird ein nicht-negativer Wert oder die Konstante `WEOF`, wenn ein Fehler aufgetreten. Das Stringende-Zeichen `\0` wird durch einen Zeilenumbruch `\n` ersetzt.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.26. int swprintf(wchar_t *buffer, const wchar_t *format [,argument]...);*

Schreibt formatierte Daten ("wide characters") in den angegebenen Puffer `buffer`, an die das Stringende-Zeichen `\0` angehangen wird. Zurückgegeben wird die Anzahl der im Puffer gespeicherten Bytes.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.27. int swscanf(const wchar_t *buffer, const wchar_t *format [, argument] ...);*

Liest formatierte Daten ("wide characters") aus der Zeichenkette (ebenfalls "wide characters") `buffer`. Die Funktion gibt die Anzahl der erfolgreich eingelesenen Daten (nicht die Anzahl der Bytes!) zurück. Das Funktionsergebnis `0` zeigt an, dass die Daten nicht zum angegebenen Format passen, also keine Daten gelesen wurden, obwohl Daten vorhanden sind. Tritt ein Fehler auf oder wurde das Stringende erreicht, bevor der erste Wert eingelesen wurde, wird als Funktionsergebnis die Konstante `WEOF` zurückgegeben.

Diese Funktion ist auch in der `stdio.h` enthalten.

16.28. int towlower(wint_t c);

Wandelt den angegebenen Buchstaben `c` ("wide character") in einen Kleinbuchstaben um. Dabei werden nur die Großbuchstaben von `L'A'` bis `L'Z'` umgewandelt, alle anderen Zeichen (auch die deutschen Umlaute) werden nicht geändert.

Diese Funktion ist auch in der `ctype.h` enthalten.

16.29. int towupper(wint_t c);

Wandelt den angegebenen Buchstaben `c` ("wide character") in einen Großbuchstaben um. Dabei werden nur die Kleinbuchstaben von `L'a'` bis `L'z'` umgewandelt, alle anderen Zeichen (auch die deutschen Umlaute) werden nicht geändert.

Diese Funktion ist auch in der `ctype.h` enthalten.

*16.30. wint_t ungetwc(wint_t c, FILE *stream);*

Packt das zuletzt gelesene Zeichen `c` ("wide character") zurück in den angegebenen Datenstrom und löscht einen evtl. Dateiende-Status. Wie der Name es schon andeutet, macht diese Funktion die letzte Leseoperation mit `getwc` rückgängig. Wird anschließend von dem Datenstrom gelesen, wird als erstes wieder das Zeichen `c` gelesen. Der Datenstrom muss zum Lesen geöffnet sein. Der Versuch, `WEOF` als Zeichen in den Datenstrom zu packen, wird ignoriert. Konnte das Zeichen erfolgreich in den Datenstrom gepackt werden, wird das Zeichen als Funktionsergebnis zurückgegeben, ansonsten wird `WEOF` zurückgegeben. Zeichen, die mit `ungetwc` in den Datenstrom gepackt wurden, werden beim Aufruf von Funktionen wie `fflush`, `fseek`, `fsetpos` oder `rewind` wieder gelöscht. Die Funktion `ungetwc` kann nicht zweimal hintereinander aufgerufen werden. Auch nach dem Aufruf von `fwscanf` kann `ungetwc` nicht aufgerufen werden, da `fwscanf` selber `ungetwc` aufruft.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.31. int vfwprintf(FILE *stream, const wchar_t *format, va_list argptr);*

Schreibt formatierte Daten (mit "wide characters") in den angegebenen Datenstrom. Die zu schreibenden Daten werden in einer Arguemtenliste (Parameter `argptr`) übergeben. Zurückgegeben wird die Anzahl der

geschriebenen wide characters, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.32. int vswprintf(wchar_t *buffer, const wchar_t *format, va_list argptr);*

Schreibt formatierte Daten ("wide characters") in den angegebenen Puffer `buffer`, an die das Stringende-Zeichen `\0` angehangen wird. Die zu schreibenden Daten werden in einer Arguemtenliste (Parameter `argptr`) übergeben. Zurückgegeben wird die Anzahl der im Puffer gespeicherten Bytes.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.33. int vwprintf(const wchar_t *format, va_list argptr);*

Schreibt formatierte Daten ("wide characters") in den Datenstrom `stdout` (d.h. im allgemeinen auf den Bildschirm). Die zu schreibenden Daten werden in einer Arguemtenliste (Parameter `argptr`) übergeben. Zurückgegeben wird die Anzahl der geschriebenen Bytes, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.34. wchar_t *wcscat(wchar_t *strDestination, const wchar_t *strSource);*

Hängt die Zeichenkette `strSource` an die Zeichenkette `strDestination` (beides "wide characters") an. Dabei wird das Stringende-Zeichen von `strDestination` vom ersten Zeichen von `strSource` überschrieben. Zurückgegeben wird der Zeiger `strDestination`, der dann die komplette Zeichenkette enthält. Überlappen sich die beiden Zeichenketten `strSource` und `strDestination`, ist das Ergebnis nicht vorhersehbar.

Diese Funktion ist auch in der `string.h` enthalten.

*16.35. wchar_t *wcschr(const wchar_t *string, wint_t c);*

Sucht das Zeichen `c` in der Zeichenkette `string` (beides "wide characters"). Wird das Zeichen gefunden, wird ein Zeiger auf die erste Stelle, an der das gesuchte Zeichen steht, zurückgegeben. Wird das gesuchte Zeichen nicht gefunden, wird der `NULL`-Zeiger zurückgegeben. Das Stringende-Zeichen ist in der Suche mit eingeschlossen.

Diese Funktion ist auch in der `string.h` enthalten.

*16.36. int wcscmp(const wchar_t *string1, const wchar_t *string2);*

Vergleicht die beiden Zeichenketten `string1` und `string2` (beides "wide characters"). Das Funktionsergebnis kann folgende Werte annehmen:

`< 0` - `string1` ist kleiner als `string2`
`0` - `string1` ist gleich `string2`
`> 0` - `string1` ist größer als `string2`

Diese Funktion ist auch in der `string.h` enthalten.

*16.37. int wcscoll(const wchar_t *string1, const wchar_t *string2);*

Vergleicht die beiden Zeichenketten `string1` und `string2` (beides "wide characters") unter Berücksichtigung der aktuellen Code Page. Das Funktionsergebnis kann folgende Werte annehmen:

`< 0` - `string1` ist kleiner als `string2`
`0` - `string1` ist gleich `string2`
`> 0` - `string1` ist größer als `string2`

Diese Funktion ist auch in der `string.h` enthalten.

*16.38. wchar_t *wcscpy(wchar_t *strDestination, const wchar_t *strSource);*

Kopiert die Zeichenkette `strSource` nach `strDestination` (beide "wide characters"). Zurückgegeben wird der Zeiger `strDestination`. Überlappen sich die Zeichenketten `strSource` und `strDestination`, ist nicht sichergestellt, dass der überlappende Bereich erst kopiert wird, bevor er überschrieben wird.

Diese Funktion ist auch in der `string.h` enthalten.

*16.39. size_t wcscspn(const wchar_t *string, const wchar_t *strCharSet);*

Gibt die Länge der Zeichenkette `string` zurück, in der keins der Zeichen von `strCharSet` vorkommt (beides "wide characters"). Beginnt die Zeichenkette mit einem Zeichen, das in `strCharSet` vorkommt, wird eine `0` zurückgegeben. Das abschließende Stringende-Zeichen ist in der Suche mit eingeschlossen.

Diese Funktion ist auch in der `string.h` enthalten.

*16.40. size_t wcsftime(wchar_t *strDest, size_t maxsize, const wchar_t *format, const struct tm *timeptr);*

Erzeugt eine Zeichenkette mit maximale `maxsize` Zeichen ("wide characters"), die entsprechend des benutzerdefinierten Formats `format` (auch "wide characters") Zeit und/oder Datum aus `*timeptr` (Aufbau der Struktur: siehe Funktion `asctime`) enthält, und speichert diese in `strDest`. Zurückgegeben wird die tatsächliche Anzahl von Zeichen einschließlich des Stringende-Zeichens in `strDest`, wenn die erzeugte Zeichenkette nicht länger als `maxsize` ist. Ansonsten wird eine `0` zurückgegeben und der Inhalt von `strDest` ist undefiniert. Das benutzerdefiniert Format `format` können neben normalem Text noch folgende Formatierungssequenzen verwendet werden:

`%a` - abgekürtzter Name des Wochentags
`%A` - kompletter Name des Wochentags
`%b` - abgekürtzter Monatsname
`%B` - kompletter Monatsname
`%c` - komplettes Datum mit Uhrzeit (lokale Zeitzone)
`%d` - Tag im Monat (01 - 31)
`%H` - Stunden im 24-Stundenformat (00 - 23)
`%I` - Stunden im 12-Stundenformat (00 - 12)
`%j` - Tag im Jahr (001 - 366)
`%m` - Monat (01 - 12)
`%M` - Minuten (00 - 59)
`%p` - A.M. / P.M. für 12-Stundenanzeige
`%S` - Sekunden (00 - 59)
`%U` - Woche im Jahr (1. Woche beginnt mit erstem Sonntag) (00 - 53)

%w - Wochentag (0 - 6; Sonntag ist 0)
%W - Woche im Jahr (1. Woche beginnt mit erstem Montag) (00 - 53)
%x - komplettes Datum (lokale Zeitzone)
%X - komplette Uhrzeit (lokale Zeitzone)
%y - Jahreszahl ohne Jahrhundert (00 - 99)
%Y - Jahreszahl mit Jahrhundert
%z - Name der Zeitzone
%Z - abgekürzter Name der Zeitzone
%% - Prozentzeichen

Mit Hilfe des Flags # zwischen dem Prozentzeichen % und dem Formatierungsbuchstaben werden bei den Zahlen die führenden Nullen (wenn vorhanden) weggelassen; bei den Formatierungen %c und %x wird das lange Datumsformat verwendet.

Diese Funktion ist auch in der time.h enthalten.

16.41. size_t wcslen(const wchar_t *string);

Liefert die Anzahl der Zeichen in der angegebenen Zeichenkette string ("wide characters"). Dabei wird das abschließende Stringende-Zeichen nicht mitgezählt.

Diese Funktion ist auch in der string.h enthalten.

16.42. wchar_t *wcsncat(wchar_t *strDest, const wchar_t *strSource, size_t count);

Hängt die ersten count Zeichen der Zeichenkette strSource an die Zeichenkette strDestination (beides "wide characters") an. Dabei wird das Stringende-Zeichen von strDestination vom ersten Zeichen von strSource überschrieben. Hat die Zeichenkette strSource weniger Zeichen als count, wird die Anzahl der Zeichen anstelle von count verwendet. Zurückgegeben wird der Zeiger strDestination, der dann die komplette Zeichenkette enthält. Überlappen sich die beiden Zeichenketten strSource und strDestination, ist das Ergebnis nicht vorhersehbar.

Diese Funktion ist auch in der string.h enthalten.

16.43. int wcsncmp(const wchar_t *string1, const wchar_t *string2, size_t count);

Vergleicht die ersten count Zeichen der beiden Zeichenketten string1 und string2 (beides "wide characters"). Das Funktionsergebnis kann folgende Werte annehmen:

< 0 - string1 ist kleiner als string2
0 - string1 ist gleich string2
> 0 - string1 ist größer als string2

Diese Funktion ist auch in der string.h enthalten.

16.44. wchar_t *wcsncpy(wchar_t *strDest, const wchar_t *strSource, size_t count);

Kopiert die ersten count Zeichen der Zeichenkette strSource nach strDestination (beides "wide characters"). Ist count kleiner oder gleich der Länge von strSource, wird nicht automatisch ein Stringende-Zeichen die kopierte Zeichenkette angehangen. Ist count größer als die Länge von strSource, werden an die kopierte Zeichenkette noch so viele Stringende-Zeichen angehangen, bis

insgesamt `count` Zeichen kopiert wurden. Zurückgegeben wird der Zeiger `strDestination`. Überlappen sich die Zeichenketten `strSource` und `strDestination`, ist nicht sichergestellt, dass der überlappende Bereich erst kopiert wird, bevor er überschrieben wird.

Diese Funktion ist auch in der `string.h` enthalten.

*16.45. wchar_t *wcspbrk(const wchar_t *string, const wchar_t *strCharSet);*

Sucht das erste Vorkommen eines der Zeichen von `strCharSet` in der Zeichenkette `string` (beides "wide characters"). Zurückgegeben wird ein Zeiger auf das erste Vorkommen eines der gesuchten Zeichen in der Zeichenkette `string`. Wurde kein Zeichen gefunden, wird der `NULL`-Zeiger zurückgegeben. Das abschließende Stringende-Zeichen ist in der Suche nicht mit eingeschlossen.

Diese Funktion ist auch in der `string.h` enthalten.

*16.46. char *wcsrchr(const wchar_t *string, int c);*

Sucht das letzte Vorkommen des Zeichens `c` in der Zeichenkette `string` (beides "wide characters"). Wird das Zeichen gefunden, wird ein Zeiger auf die letzte Stelle, an der das gesuchte Zeichen steht, zurückgegeben. Wird das gesuchte Zeichen nicht gefunden, wird der `NULL`-Zeiger zurückgegeben. Das Stringende-Zeichen ist in der Suche mit eingeschlossen.

Diese Funktion ist auch in der `string.h` enthalten.

*16.47. size_t wcsspn(const wchar_t *string, const wchar_t *strCharSet);*

Gibt die Länge der Zeichenkette `string` zurück, in der die Zeichen von `strCharSet` vorkommen (beides "wide characters"). Beginnt die Zeichenkette mit einem Zeichen, das nicht in `strCharSet` vorkommt, wird eine `0` zurückgegeben. Das abschließende Stringende-Zeichen ist in der Suche nicht mit eingeschlossen.

Diese Funktion ist auch in der `string.h` enthalten.

*16.48. wchar_t *wcsstr(const wchar_t *string, const wchar_t *strCharSet);*

Sucht nach der Zeichenkette `strCharSet` in der Zeichenkette `string` (beides "wide characters"). Zurückgegeben wird ein Zeiger auf das erste Vorkommen der Zeichenkette `strCharSet` in `string`. Wird die Zeichenkette `strCharSet` nicht gefunden, wird der `NULL`-Zeiger zurückgegeben. Ist `strCharSet` eine Zeichenkette mit der Länge 0, wird der Zeiger `string` zurückgegeben.

Diese Funktion ist auch in der `string.h` enthalten.

*16.49. double wcstod(const wchar_t *nptr, wchar_t **endptr);*

Wandelt einen String mit "wide character" in eine Fließkommazahl des Typs `double` um. Dabei ist `**endptr` ein Zeiger auf den restlichen String, der nicht umgewandelt wurde.

Diese Funktion ist auch in der `stdlib.h` enthalten.

*16.50. wchar_t *wcstok(wchar_t *strToken, const wchar_t *strDelimit);*

Liefert den ersten Teil der Zeichenkette `strToken`, der mit einem der Zeichen aus `strDelimit` begrenzt ist (beides "wide characters"). In der Zeichenkette `strToken` werden durch den Funktionsaufruf die

gefundenen Zeichen aus `strDelimit` durch Stringende-Zeichen ersetzt. Um den nächsten Teil zu erhalten, muss die Funktion erneut aufgerufen werden; dabei muss aber für den Parameter `strToken` der `NULL`-Zeiger eingesetzt werden. Gibt die Funktionen einen `NULL`-Zeiger zurück, wurde kein weiterer Teil gefunden.

Diese Funktion ist auch in der `string.h` enthalten.

*16.51. long wcstol(const wchar_t *nptr, wchar_t **endptr, int base);*

Wandelt einen String mit "wide character" in eine ganze Zahl des Typs `long` um. Dabei ist `**endptr` ein Zeiger auf den restlichen String, der nicht umgewandelt wurde. Der Parameter `base` gibt die Zahlenbasis für die Umwandlung an.

Diese Funktion ist auch in der `stdlib.h` enthalten.

*16.52. unsigned long wcstoul(const wchar_t *nptr, wchar_t **endptr, int base);*

Wandelt einen String mit "wide characters" in eine ganze Zahl des Typs `unsigned long` um. Dabei ist `**endptr` ein Zeiger auf den restlichen String, der nicht umgewandelt wurde. Im Fall eines Überlaufs wird die Konstante `ULONG_MAX` zurückgegeben. Im Fall eines Unterlaufs ist das Funktionsergebnis gleich `0`. Tritt ein Überlauf oder Unterlauf auf, wird die Variable `errno` auf die Konstante `ERANGE` gesetzt. Der Parameter `base` gibt die Zahlenbasis für die Umwandlung an.

Diese Funktion ist auch in der `stdlib.h` enthalten.

*16.53. size_t wcsxfrm(wchar_t *strDest, const wchar_t *strSource, size_t count);*

Wandelt die ersten `count` Zeichen der Zeichenkette `strSource` entsprechend der lokalen Informationen (d.h. welcher Zeichensatz aktuell gültig ist) nach `strDest` (beides "wide characters") um. Die Funktion liefert die Anzahl der umgewandelten Zeichen (ohne Stringende-Zeichen) zurück. Ist der Rückgabewert größer oder gleich dem Parameter `count`, ist die Zeichenkette `strDest` unvorhersehbar. Im Fehlerfall wird eine `(size_t) -1` zurückgegeben und die globale Variable `errno` wird entsprechend gesetzt.

Diese Funktion ist auch in der `string.h` enthalten.

*16.54. int wprintf(const wchar_t *format [,argument]...);*

Schreibt formatierte Daten ("wide characters") in den Datenstrom `stdout` (d.h. im allgemeinen auf den Bildschirm). Zurückgegeben wird die Anzahl der geschriebenen Bytes, bei einem negativen Wert als Funktionsergebnis ist ein Ausgabefehler aufgetreten.

Diese Funktion ist auch in der `stdio.h` enthalten.

*16.55. int wscanf(const wchar_t *format [,argument]...);*

Liest formatierte Daten ("wide characters") aus dem Datenstrom `stdin` (d.h. im allgemeinen von der Tastatur). Die Funktion gibt die Anzahl der erfolgreich eingelesenen Daten (nicht die Anzahl der Bytes!) zurück. Das Funktionsergebnis `0` zeigt an, dass die Daten nicht zum angegebenen Format passen, also keine Daten gelesen wurden, obwohl Daten vorhanden sind. Tritt ein Lesefehler auf oder wurde das Dateiende erreicht, wird als Funktionsergebnis die Konstante `WEOF` zurückgegeben.

Diese Funktion ist auch in der `stdio.h` enthalten.