stormamiga_lib

Matthias Henze

COLLABORATORS			
	TITLE : stormamiga_lib		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Matthias Henze	October 23, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

stormamiga_lib

1.1 Inhalt

```
                    stormamiga.lib Version 42.01 ( 01.11.1996 )

      © Kopierrecht 1996 bei COMPIUTECK

        geschrieben von Matthias Henze


         F R E E W A R E
```

1.2 einleitung

Einleitung:
~~~~~~~~~~~

Da die Funktionen der "storm.lib" in C geschrieben sind,
werden die damit gelinkten Programme sehr groß und langsam.
Die Funktionen der "amiga.lib" sind auch nicht gerade klein
und schnell.
Aus diesem Grund habe ich mich am 18.03.1996 entschlossen
die "stormamiga.lib" zu schreiben.

Die "stormamiga.lib" ist fast komplett in Assembler geschrieben.
Dadurch werden die damit gelinkten Programme auch sehr klein
und schnell.
Mein Ziel ist es, alle Funktionen der "amiga.lib", die nicht
in den Pragmadateien enthalten sind, und alle Funktionen der
"storm.lib" durch kurze und schnelle Assemblerroutinen zu
ersetzen. Außerdem will ich einige Spezialbefehle von anderen
Compilern (zur Zeit nur vom GCC) und einige Routinen, die das
Programmieren erleichtern, in die "stormamiga.lib" integrieren.
~~~~~~~~~~~

Wichtig

In der jetzigen Version der "stormamiga.lib" sind noch nicht
alle Funktionen der "storm.lib" und der "amiga.lib" enthalten.

Die "stormamiga.lib" sollte auch mit C++ funktionieren. Allerdings
wurde das noch nicht genügend getestet.

1.3 registrierung

Registrierung:
~~~~~~~~~~~~~~

Obwohl die "stormamiga.lib" Freeware ist, was auch so bleiben
wird, können Sie sich bei
mir
registrieren lassen. Ich möchte
vor allem wissen, ob sich die Weiterentwicklung überhaupt lohnt.

Sie können mir natürlich auch Geschenke, die Sie für angemessen
halten, zuschicken. Es währe auch sehr schön, wenn Sie mir Ihr
Programm, bei dem Sie die "stormamiga.lib" verwendet haben,
zuschicken würden. Außerdem würde ich gerne Ihre Meinung zur
"stormamiga.lib" erfahren.

Für Fehlerberichte (möglichst mit dem entsprechenden Quelltext
und einer genauen Beschreibung) und Verbesserungsvorschläge bin
ich jederzeit dankbar.
Wenn Sie Fragen zur "stormamiga.lib" haben, können Sie mich gerne
anrufen oder mir schreiben. Ich werde die Antworten zu Ihren
Fragen in der Anleitung der nächsten Version veröffentlichen.
Wenn ich Ihnen die Antworten zu Ihren Fragen persönlich zuschicken
soll, dann müssen Sie mir natürlich einen frankierten Briefumschlag
mitschicken. Wenn ich Ihnen die neuste Version der "stormamiga.lib"
zuschicken soll, dann müssen Sie neben dem frankierten Briefumschlag
auch eine Diskette (HD oder DD) mitschicken.

1.4 systemanforderungen

Systemanforderungen:
~~~~~~~~~~~~~~~~~~~~

- Ein Amiga
- AmigaOS 2.0 oder höher
- MC68EC020 oder höher
- StormC Version 1.1 oder höher

1.5 installation

```
Installation:
~~~~~~~~~~~~~

Starten Sie das Installationsprogramm "stormamiga.lib HD-Install"
und führen Sie die Installation nach Ihren Wünschen und Anfor-
derungen durch.

Obwohl es von der "stormamiga.lib" mehrere Versionen, mit unter-
schiedlichen Namen ("stormamiga.lib", "stormamiga_nc.lib")
gibt, verwende ich meistens den allgemeinen Namen "stormamiga.lib".
Für die Startupcodes "stormamiga_startups.o", "stormamiga_nc_start-
ups.o", "stormamiga_C++_startups.o", "stormamiga_nc_C++_startups.o"
verwende ich meistens den allgemeinen Namen "stormamiga_startups.o"

Fügen Sie die "stormamiga.lib" an erster Stelle in das Projekt
ein. Da in dieser Version noch nicht alle Funktionen enthalten
sind, müssen Sie noch die "storm.lib" in das Projekt einfügen.
Wenn Sie die Spezialfunktionen der "stormamiga.lib" nutzen
wollen, müssen Sie noch die Includedatei "stormamiga.h", mit
"#include <stormamiga.h>", in Ihren Quelltext einbinden. Die
Includedatei "stormamiga.h" sollten Sie als  letzte einbinden.
Um auch den neuen Startupcode "stormamiga_startups.o" zu nutzen,
müssen Sie, bei den Linkeroptionen (Linker 1) des Menüs Einstellungen,
die Option "Eigener Startup-Code" einschalten und die Datei
"stormamiga_startups.o" auswählen.
```

1.6 funktionen

```
                Die Funktionen der "stormamiga.lib"
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Da die normalen Ansi-C und C++ Funktionen bereits bei StormC
beschrieben werden, erkläre ich nur die Spezialfunktionen
und -befehle.

Hinweis
Wenn Sie sich die Map-Datei, eines mit der "stormamiga.lib"
gelinkten Programms ansehen, werden Sie feststellen, daß
einige Funktionen den Vorsatz "intern__" (z.B. intern__form_in)
besitzen. Diese Funktionen werden in der "stormamiga.lib" nur
intern verwendet und können nicht als Befehle benutzt werden.



                 Der Startupcode
                AmigaDOS stdio Funktionen

                 SPRINTF
                 VSPRINTF
                stdio Funktionen

                 printf_
```

```
 fprintf_
 sprintf_
 vprintf_

 vfprintf_
 vsprintf_
 scanf_
 fscanf_

 sscanf_
 vscanf
 vscanf_
 vfscanf

 vfscanf_
 vsscanf
 vsscanf_
 setbuffer

 setlinebuf
string Funktionen

 strcoll
 strxfrm
 bcmp
 bcopy

 bzero
 ffs
 index
 rindex

 memccpy
 strsep
 swab
 strnicmp

 strcasecmp
 strncasecmp
 strlower
 strupper

 stricmp_d
 strnicmp_d
 strcasecmp_d
 strncasecmp_d

 strlower_d
 strupper_d
 strlwr_d
 strupr_d

 strdup
ctype Funktionen

 isalnum_d
 isalpha_d
```

```
 islower_d
 isupper_d

 tolower_d
 toupper_d
assert Funktionen

 assert_
time Funktionen

 strftime
 strftime_d
 asctime_d
 ctime_d
Spezial Funktionen

 muls
 muls_r
 mulu
 mulu_r

 divsl
 divsl_r
 divul
 divul_r

 muls64
 muls64_r
 mulu64
 mulu64_r

 divs64
 divs64_r
 divu64
 divu64_r

 button_al
 button_ar
 button_bl
 button_br

 button
 button_r
 waitbutton_al
 waitbutton_ar

 waitbutton_bl
 waitbutton_br
 waitbutton
 waitbutton_r
```

1.7 Der Startupcode

Der Startupcode "stormamiga_startups.o"
Die Startupcodes "stormamiga_startups.o" und "stormamiga_nc_
startups.o" sind speziell für Ansi-C geschrieben. Der Startup-
code "stormamiga_startups.o" unterstützt das große Codemodell
in Verbindung mit den Datenmodellen "FAR", "NEAR A4" und "NEAR
A6". Der Startupcode "stormamiga_nc_startups.o" unterstützt das
kleine Codemodell in Verbindung mit den Datenmodellen "FAR",
"NEAR A4" und "NEAR A6".
Der größte Unterschied, zum Startupcode "startup.o" von StormC,
ist der Aufruf der Funktionen "main" und "wbmain".

Bei einem Start aus dem CLI wird die Funktion "_main__", das
entspricht der Funktion
main__()
in Ansi-C, aufgerufen. Wenn
es diese Funktion, in Ihrem Programm, nicht gibt, wird sie aus
der "stormamiga.lib" dazugelinkt. Die Funktion "_main__" ruft
die Funktion "_main", das entspricht der Funktion "main()" in
Ansi-C, auf. Wenn es diese Funktion nicht gibt, meldet der
Linker einen Fehler.

Bei einem Start von der Workbench wird die Funktion "_wbmain",
das entspricht der Funktion "wbmain()" in Ansi-C, aufgerufen.
Wenn es diese Funktion nicht gibt, wird sie aus der "storm-
amiga.lib" dazugelinkt.

Die Startupcodes "stormamiga_C++_startups.o" und "stormamiga_nc_
C++_startups.o" sind für C++ und Ansi-C geschrieben. Der Startup-
code "stormamiga_C++_startups.o" unterstützt das große Codemodell
in Verbindung mit den Datenmodellen "FAR", "NEAR A4" und "NEAR A6".
Der Startupcode "stormamiga_nc_C++_startups.o" unterstützt das
kleine Codemodell in Verbindung mit den Datenmodellen "FAR",
"NEAR A4" und "NEAR A6".
Die Funktion entspricht dem Startupcode "startup.o" von StormC.

1.8 main__()

Die Funktion "main__()"

Wenn Sie für Ihr Programm keine Auswertung von Argumenten
benötigen oder diese Routine selber schreiben, können Sie die
Funktion "main()" in "main__()" umbenennen. Dadurch wird Ihr
Programm etwas kleiner.

Wichtig

Da der Compiler die Funktion "main__()" nicht als normale
main-Funktion erkennt, setzt er auch nicht den Returncode
auf 0. Deshalb müssen Sie den Returncode, mit "return 0",
selber auf 0 setzen.
Die Funktion "main__()" benötigt den Startupcode "stormamiga
_startups.o" oder "stormamiga_nc_startups.o" und funktioniert

```
nur in Ansi-C.
```

1.9 sprintf

```
                SPRINTF
Formatierte Ausgabe in den Stringpuffer "s".

Übersicht
#include <stormamiga.h>

r = SPRINTF (s, format, ...);

long r;
char *s;
const char *format;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Formatierte Ausgabe in den
                 Stringpuffer
                 "s". Der Formatstring
"format" beschreibt das Ausgabeformat. Danach folgen zusätzlich
angegebene Parameter.

"SPRINTF" verwendet den Befehl  RawDoFmt  der "exec.library"
und ist dadurch auch sehr klein.

Rückgabe
Die Anzahl der ausgegebenen Zeichen.
```

1.10 vsprintf

```
                VSPRINTF
Formatierte Ausgabe in den Stringpuffer "s".

Übersicht
#include <stormamiga.h>

r = VSPRINTF (s, format, vl);

long r;
char *s;
const char *format;
va_list vl;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
```

```
Formatierte Ausgabe in den
                 Stringpuffer
                 "s". Der Formatstring
"format" beschreibt das Ausgabeformat. Danach folgt eine

                 Parameterliste
                 "vl".

"VSPRINTF" verwendet den Befehl  RawDoFmt  der "exec.library"
und ist dadurch auch sehr klein.

Rückgabe
Die Anzahl der ausgegebenen Zeichen.
```

1.11 printf_

```
               printf_
Formatierte Ausgabe in die Standardausgabe "stdout".
Funktionsreduzierte Version von "printf".

Übersicht
#include <stormamiga.h>

r = printf_ (format, ...);

int r;
const char *format;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Formatierte Ausgabe in die Standardausgabe "stdout". Der

                 Ausgabeformatstring_
                 "format" beschreibt das Ausgabe-
format. Danach folgen zusätzlich angegebene Parameter.

Rückgabe
Die Anzahl der ausgegebenen Zeichen oder im Fehlerfall
eine negative Zahl.
```

1.12 fprintf_

```
               fprintf_
Formatierte Ausgabe in die Datei "f".
Funktionsreduzierte Version von "fprintf".

Übersicht
#include <stormamiga.h>
```

```
r = fprintf_ (f, format, ...);

int r;
FILE *f;
const char *format;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Formatierte Ausgabe in die Datei "f". Der
                 Ausgabeformatstring_
                "format" beschreibt das Ausgabeformat. Danach folgen zusätzlich
angegebene Parameter.

Rückgabe
Die Anzahl der ausgegebenen Zeichen oder im Fehlerfall
eine negative Zahl.
```

1.13 sprintf_

```
                sprintf_
Formatierte Ausgabe in den Stringpuffer "s".
Funktionsreduzierte Version von "sprintf".

Übersicht
#include <stormamiga.h>

r = sprintf_ (s, format, ...);

int r;
char *s;
const char *format;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Formatierte Ausgabe in den
                 Stringpuffer
                 "s". Der

                 Ausgabeformatstring_
                 "format" beschreibt das Ausgabe-
format. Danach folgen zusätzlich angegebene Parameter.

Rückgabe
Die Anzahl der ausgegebenen Zeichen oder im Fehlerfall
eine negative Zahl.
```

1.14 vprintf_

```
                vprintf_
Formatierte Ausgabe in die Standardausgabe "stdout".
Funktionsreduzierte Version von "vprintf".

Übersicht
#include <stormamiga.h>

r = vprintf_ (format, vl);

int r;
const char *format;
va_list vl;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Formatierte Ausgabe in die Standardausgabe "stdout". Der

                 Ausgabeformatstring_
                 "format" beschreibt das Ausgabe-
format. Danach folgt eine
                 Parameterliste
                 "vl".

Der Befehl "vprintf_" ist auch als
                 Inlinefunktionen
                 verfügbar.

Rückgabe
Die Anzahl der ausgegebenen Zeichen oder im Fehlerfall
eine negative Zahl.
```

1.15 vfprintf_

```
                vfprintf_
Formatierte Ausgabe in die Datei "f".
Funktionsreduzierte Version von "vfprintf".

Übersicht
#include <stormamiga.h>

r = vfprintf_ (f, format, vl);

int r;
FILE *f;
const char *format;
va_list vl;

Standard
(noch) keiner (Eigenentwicklung)
```

```
Erklärung
Formatierte Ausgabe in die Datei "f". Der

                Ausgabeformatstring_
                "format" beschreibt
das Ausgabeformat. Danach folgt eine

                Parameterliste
                "vl".

Rückgabe
Die Anzahl der ausgegebenen Zeichen oder im Fehlerfall
eine negative Zahl.
```

1.16 vsprintf_

```
               vsprintf_
Formatierte Ausgabe in den Stringpuffer "s".
Funktionsreduzierte Version von "vsprintf".

Übersicht
#include <stormamiga.h>

r = vsprintf_ (s, format, vl);

int r;
char *s;
const char *format;
va_list vl;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Formatierte Ausgabe in den
                Stringpuffer
                "s". Der

                Ausgabeformatstring_
                "format" beschreibt das
Ausgabeformat. Danach folgt eine
                Parameterliste
                "vl".

Rückgabe
Die Anzahl der ausgegebenen Zeichen oder im Fehlerfall
eine negative Zahl.
```

1.17 scanf_

```
                scanf_
Einlesen einer formatierten Eingabe aus der Standardeingabe "stdin".
Funktionsreduzierte Version von "scanf".

Übersicht
#include <stormamiga.h>

r = scanf_ (format, ...);

int r;
const char *format;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Einlesen einer formatierten Eingabe aus der Standardeingabe "stdin".
Der
                 Eingabeformatstring_
                 "format" beschreibt das Eingabeformat.
Danach folgen zusätzlich angegebene Parameter.

Rückgabe
Die Anzahl der eingelesenen Zeichen oder im Fehlerfall einen Rück-
gabewert der kleiner als die Anzahl der eingelesenen Zeichen ist.
```

1.18 fscanf_

```
                fscanf_
Einlesen einer formatierten Eingabe aus der Datei "f".
Funktionsreduzierte Version von "fscanf".

Übersicht
#include <stormamiga.h>

r = fscanf_ (f, format, ...);

int r;
FILE *f;
const char *format;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Einlesen einer formatierten Eingabe aus der Datei "f". Der

                 Eingabeformatstring_
                 "format" beschreibt das Eingabeformat.
Danach folgen zusätzlich angegebene Parameter.

Rückgabe
Die Anzahl der eingelesenen Zeichen oder im Fehlerfall einen Rück-
```

```
gabewert der kleiner als die Anzahl der eingelesenen Zeichen ist.
```

1.19 sscanf_

```
                sscanf_
Einlesen einer formatierten Eingabe aus dem Stringpuffer "s".
Funktionsreduzierte Version von "sscanf".

Übersicht
#include <stormamiga.h>

r = sscanf_ (s, format, ...);

int r;
char *s;
const char *format;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Einlesen einer formatierten Eingabe aus dem Stringpuffer "s". Der

                Eingabeformatstring_
                "format" beschreibt das Eingabeformat.
Danach folgen zusätzlich angegebene Parameter.

Rückgabe
Die Anzahl der eingelesenen Zeichen oder im Fehlerfall einen Rück-
gabewert der kleiner als die Anzahl der eingelesenen Zeichen ist.
```

1.20 vscanf

```
                vscanf
Einlesen einer formatierten Eingabe aus der Standardeingabe "stdin".

Übersicht
#include <stormamiga.h>

r = vscanf (format, vl);

int r;
const char *format;
va_list vl;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Einlesen einer formatierten Eingabe aus der Standardeingabe "stdin".
```

```
Der
                 Eingabeformatstring
                 "format" beschreibt das Eingabeformat.
Danach folgt eine
                 Parameterliste
                 "vl".

Der Befehl "vscanf" ist auch als
                 Inlinefunktionen
                 verfügbar.

Rückgabe
Die Anzahl der eingelesenen Zeichen oder im Fehlerfall einen Rück-
gabewert der kleiner als die Anzahl der eingelesenen Zeichen ist.
```

1.21 vscanf_

```
                vscanf_
Einlesen einer formatierten Eingabe aus der Standardeingabe "stdin".
Funktionsreduzierte Version von "scanf".

Übersicht
#include <stormamiga.h>

r = vscanf_ (format, vl);

int r;
const char *format;
va_list vl;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Einlesen einer formatierten Eingabe aus der Standardeingabe "stdin".
Der
                 Eingabeformatstring_
                 "format" beschreibt das Eingabeformat.
Danach folgt eine
                 Parameterliste
                 "vl".

Der Befehl "vscanf_" ist auch als
                 Inlinefunktionen
                 verfügbar.

Rückgabe
Die Anzahl der eingelesenen Zeichen oder im Fehlerfall einen Rück-
gabewert der kleiner als die Anzahl der eingelesenen Zeichen ist.
```

1.22 vfscanf

```
                vfscanf
Einlesen einer formatierten Eingabe aus der Datei "f".

Übersicht
#include <stormamiga.h>

r = vfscanf (f, format, vl);

int r;
FILE *f;
const char *format;
va_list vl;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Einlesen einer formatierten Eingabe aus der Datei "f". Der

                Eingabeformatstring
                "format" beschreibt das Eingabeformat.
Danach folgt eine
                Parameterliste
                "vl".

Rückgabe
Die Anzahl der eingelesenen Zeichen oder im Fehlerfall einen Rück-
gabewert der kleiner als die Anzahl der eingelesenen Zeichen ist.
```

1.23 vfscanf_

```
               vfscanf_
Einlesen einer formatierten Eingabe aus der Datei "f".
Funktionsreduzierte Version von "vfscanf".

Übersicht
#include <stormamiga.h>

r = vfscanf_ (f, format, vl);

int r;
FILE *f;
const char *format;
va_list vl;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Einlesen einer formatierten Eingabe aus der Datei "f". Der
```

```
                Eingabeformatstring_
                "format" beschreibt das Eingabeformat.
Danach folgt eine
                Parameterliste
                "vl".

Rückgabe
Die Anzahl der eingelesenen Zeichen oder im Fehlerfall einen Rück-
gabewert der kleiner als die Anzahl der eingelesenen Zeichen ist.
```

1.24 vsscanf

```
               vsscanf
Einlesen einer formatierten Eingabe aus dem Stringpuffer "s".

Übersicht
#include <stormamiga.h>

r = vsscanf (s, format, vl);

int r;
char *s;
const char *format;
va_list vl;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Einlesen einer formatierten Eingabe aus dem Stringpuffer "s". Der

                Eingabeformatstring
                "format" beschreibt das Eingabeformat.
Danach folgt eine
                Parameterliste
                "vl".

Rückgabe
Die Anzahl der eingelesenen Zeichen oder im Fehlerfall einen Rück-
gabewert der kleiner als die Anzahl der eingelesenen Zeichen ist.
```

1.25 vsscanf_

```
               vsscanf_
Einlesen einer formatierten Eingabe aus dem Stringpuffer "s".
Funktionsreduzierte Version von "vsscanf".

Übersicht
#include <stormamiga.h>
```

```
r = vsscanf_ (s, format, vl);

int r;
char *s;
const char *format;
va_list vl;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Einlesen einer formatierten Eingabe aus dem Stringpuffer "s". Der

                 Eingabeformatstring_
                 "format" beschreibt das Eingabeformat.
Danach folgt eine
                 Parameterliste
                 "vl".

Rückgabe
Die Anzahl der eingelesenen Zeichen oder im Fehlerfall einen Rück-
gabewert der kleiner als die Anzahl der eingelesenen Zeichen ist.
```

1.26 setbuffer

```
                setbuffer
setzen eines Dateipuffers

Übersicht
#include <stormamiga.h>

r = setbuffer (f, buf, size);

void r;
FILE *f;
char *buf;
int size;

Standard
(noch) keiner (4.3BSD)

Erklärung
Setzt für die Datei "f" den Dateipuffer "buf" der Länge "size".

Der Befehl "setbuffer" ist auch als
                 Inlinefunktionen
                 verfügbar.

Rückgabe
keine
```

1.27 setlinebuf

```
                setlinebuf
setzen eines Zeilenpuffers

Übersicht
#include <stormamiga.h>

r = setlinebuf (f);

int r;
FILE *f;

Standard
(noch) keiner (4.3BSD)

Erklärung
Setzt für die Datei "f" einen Zeilenpuffer.

Der Befehl "setlinebuf" ist auch als
                Inlinefunktionen
                verfügbar.

Rückgabe
Es wird immer 0 zurückgegeben.
```

1.28 strcoll

```
strcoll
vergleichen zweier Strings unter Beachtung der aktuellen Sprache

Übersicht
#include <stormamiga.h>

r = strcoll (s1, s2);

int r;
const char *s1;
const char *s2;

Standard
ANSI C3.159-1989 ("ANSI C")

Erklärung
Vergleicht die Strings "s1" und "s2" Zeichen für Zeichen.
Da die "stormamiga.lib" die ANSI-Lokalisierung nicht unterstützt,
wird die aktuelle Sprache nicht berücksichtigt.
Die Funktion "strcoll" ist nur aus Gründen der Kompatibilität zu
anderen Compilern (z.B.: GCC) vorhanden.

Rückgabe
< 0 wenn s1 < s2
= 0 wenn s1 = s2
```

```
> 0 wenn s1 > s2
```

1.29 strxfrm

```
strxfrm
kopieren eines Strings unter Beachtung der aktuellen Sprache

Übersicht
#include <stormamiga.h>

r = strxfrm (dest, source, n);

int r;
char *dest;
const char *source;
size_t n;

Standard
ANSI C3.159-1989 ("ANSI C")

Erklärung
Kopiert maximal "n" Bytes vom String "source" nach "dest".
Da die "stormamiga.lib" die ANSI-Lokalisierung nicht unterstützt,
wird die aktuelle Sprache nicht berücksichtigt.
Die Funktion "strxfrm" ist nur aus Gründen der Kompatibilität zu
anderen Compilern (z.B.: GCC) vorhanden.

Rückgabe
Die Länge des kopierten Strings "source" ohne Nullzeichen.
```

1.30 bcmp

```
bcmp
vergleichen zweier Speicherbereiche mit Beachtung einer Maximallänge

Übersicht
#include <stormamiga.h>

r = bcmp (b1, b2, n);

int r;
const void *b1;
const void *b2;
size_t n;

Standard
(noch) keiner (4.2BSD)

Erklärung
Vergleicht die Speicherbereiche "b1" und "b2" Byte für Byte auf
maximal "n" Bytes Länge. Die Speicherbereiche dürfen sich über-
schneiden.
```

```
Rückgabe
< 0 wenn b1 < b2
= 0 wenn b1 = b2
> 0 wenn b1 > b2
```

1.31 bcopy

```
bcopy
Speicher kopieren

Übersicht
#include <stormamiga.h>

r = bcopy (source, dest, n);

void r;
const void *source;
void *dest;
size_t n;

Standard
(noch) keiner (4.2BSD)

Erklärung
Kopiert "n" Bytes vom Speicherbereich "source" nach "dest". Die
Speicherbereiche dürfen sich überschneiden. Wenn "n" 0 ist, wird
nichts kopiert.

Rückgabe
keine
```

1.32 bzero

```
bzero
schreibt NULL-Bytes in einen Speicherbereich

Übersicht
#include <string.h>

r = bzero (b, n);

void r;
void b;
size_t n;

Standard
(noch) keiner (4.3BSD)

Erklärung
Schreibt "n" NULL-Bytes in den Speicherbereich "b".
```

```
Rückgabe
keine
```

1.33 ffs

```
ffs
findet das erste gesetzte Bit in einem Bit-String

Übersicht
#include <stormamiga.h>

r = ffs (value);

int r;
int value;

Standard
(noch) keiner (4.3BSD)

Erklärung
Findet das erste gesetzte Bit in dem Bit-String "value" und gibt
den Index davon zurück.

Rückgabe
Index des Bit-String "value"
```

1.34 index

```
index
sucht das erste Vorkommen eines Zeichens in einem String

Übersicht
#include <stormamiga.h>

r = index (s, c);

char *r;
const char *s;
int c;

Standard
(noch) keiner (Version 6 AT&T UNIX)

Erklärung
Sucht das erste Vorkommen des Zeichens "c" in dem String "s"
und gibt einen Zeiger auf das erste gefundene Zeichen "c"
zurück. Wenn das Zeichen "c" nicht gefunden wird, wird 0
zurückgegeben.

Rückgabe
Ein Zeiger auf das erste gefundene Zeichen "c" oder 0.
```

1.35 rindex

```
rindex
sucht das letzte Vorkommen eines Zeichens in einem String

Übersicht
#include <stormamiga.h>

r = rindex (s, c);

char *r;
const char *s;
int c;

Standard
(noch) keiner (Version 6 AT&T UNIX)

Erklärung
Sucht das letzte Vorkommen des Zeichens "c" in dem String "s"
und gibt einen Zeiger auf das letzte gefundene Zeichen "c"
zurück. Wenn das Zeichen "c" nicht gefunden wird, wird 0
zurückgegeben.

Rückgabe
Ein Zeiger auf das letzte gefundene Zeichen "c" oder 0.
```

1.36 memccpy

```
memccpy
Speicher kopieren

Übersicht
#include <stormamiga.h>

r = memccpy (dest, source, c, n);

void *r;
void *dest;
const void *source;
int c;
size_t n;

Standard
(noch) keiner (4.3BSD)

Erklärung
Die Funktion kopiert den Speicherbereich "source" in den Speicher-
bereich "dest". Wenn das Zeichen "c" im Speicherbereich "source"
vorkommt, wird der Kopiervorgang an dieser Stelle gestoppt und ein
Zeiger auf das Byte hinter der Kopie des Zeichens "c" im Speicher-
bereich "dest" zurückgegeben. Ansonsten werden "n" Bytes kopiert
und 0 zurückgegeben.

Rückgabe
```

```
Ein Zeiger auf das Byte hinter der Kopie des Zeichens "c" im
Speicherbereich "dest" oder 0.
```

1.37 strsep

```
strsep
trennt Strings

Übersicht
#include <stormamiga.h>

r = strsep (s1, s2);

char *r;
char **s1;
char *s2;

Standard
(noch) keiner (BSD)

Erklärung
Die Funktion findet im String "*s1" (mit abschließendem Nullzeichen)
das erste Vorkommen eines Zeichens aus dem String "s2", ersetzt dieses
durch ein Nullzeichen und verzeichnet die Stelle des nächsten Zeichens
im String "*s1". Es wird der Originalwert des Strings "*s1" zurückgege-
ben. Wenn kein Zeichens aus dem String "s2" gefunden wird, wird 0 zurück-
gegeben.

Rückgabe
Der Originalwert des Strings "*s1" oder 0.
```

1.38 swab

```
swab
vertauschen angrenzender Bytes

Übersicht
#include <stormamiga.h>

r = swab (source, dest, n);

void r;
const void *source;
void *dest;
size_t n;

Standard
(noch) keiner (Version 7 AT&T UNIX)

Erklärung
Kopiert "n" Bytes von "source" nach "dest" und vertauscht die
angrenzenden Bytes. "n" muß eine gerade Zahl sein.
```

```
Rückgabe
keine
```

1.39 strnicmp

```
strnicmp
vergleichen zweier Strings mit Beachtung einer Maximallänge

Übersicht
#include <stormamiga.h>

r = strnicmp (s1, s2, n);

int r;
const char *s1;
const char *s2;
size_t n;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Vergleicht die beiden Strings "s1" und "s2", bis maximal zum Zeichen
mit Index "n", Zeichen für Zeichen, ohne die Groß- oder Kleinschreibung
der Buchstaben zu beachten.
Aus Portabilitätsgründen werden keine Umlaute unterstützt.

Rückgabe
< 0 wenn s1 < s2
= 0 wenn s1 = s2
> 0 wenn s1 > s2
```

1.40 strcasecmp

```
strcasecmp
vergleichen zweier Strings

Übersicht
#include <stormamiga.h>

r = strcasecmp (s1, s2);

int r;
const char *s1;
const char *s2;

Standard
(noch) keiner (BSD)

Erklärung
Vergleicht die beiden Strings "s1" und "s2" Zeichen für Zeichen,
```

```
ohne die Groß- oder Kleinschreibung der Buchstaben zu beachten.
Aus Portabilitätsgründen werden keine Umlaute unterstützt.

Rückgabe
< 0 wenn s1 < s2
= 0 wenn s1 = s2
> 0 wenn s1 > s2
```

1.41 strncasecmp

```
strncasecmp
vergleichen zweier Strings mit Beachtung einer Maximallänge

Übersicht
#include <stormamiga.h>

r = strncasecmp (s1, s2, n);

int r;
const char *s1;
const char *s2;
size_t n;

Standard
(noch) keiner (BSD)

Erklärung
Vergleicht die beiden Strings "s1" und "s2", bis maximal zum Zeichen
mit Index "n", Zeichen für Zeichen, ohne die Groß- oder Kleinschreibung
der Buchstaben zu beachten.
Aus Portabilitätsgründen werden keine Umlaute unterstützt.

Rückgabe
< 0 wenn s1 < s2
= 0 wenn s1 = s2
> 0 wenn s1 > s2
```

1.42 strlower

```
strlower
wandelt die Großbuchstaben eines Strings in Kleinbuchstaben um

Übersicht
#include <stormamiga.h>

r = strlower (s);

char *r;
char *s;

Standard
(noch) keiner (BSD)
```

```
Erklärung
Falls ein oder mehrere Zeichen des Strings "s" Großbuchstabe sind,
werden sie in Kleinbuchstaben umgewandelt, ansonsten bleiben sie
unverändert.
Aus Portabilitätsgründen werden keine Umlaute unterstützt.

Rückgabe
Der umgewandelte String "s".
```

1.43 strupper

```
strupper
wandelt die Kleinbuchstaben eines Strings in Großbuchstaben um

Übersicht
#include <stormamiga.h>

r = strupper (s);

char *r;
char *s;

Standard
(noch) keiner (BSD)

Erklärung
Falls ein oder mehrere Zeichen des Strings "s" Kleinbuchstaben sind,
werden sie in Großbuchstaben umgewandelt, ansonsten bleiben sie
unverändert.
Aus Portabilitätsgründen werden keine Umlaute unterstützt.

Rückgabe
Der umgewandelte String "s".
```

1.44 stricmp_d

```
                stricmp_d
vergleichen zweier Strings
deutsche Version von "stricmp"

Übersicht
#include <stormamiga.h>

r = stricmp_d (s1, s2);

int r;
const char *s1;
const char *s2;

Standard
(noch) keiner (Eigenentwicklung)
```

```
Erklärung
Vergleicht die beiden Strings "s1" und "s2" Zeichen für Zeichen,
ohne die Groß- oder Kleinschreibung der Buchstaben zu beachten.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt.
Da es "ß" nicht als Großbuchstabe gibt, muß dieser nicht extra
unterstützt werden.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
< 0 wenn s1 < s2
= 0 wenn s1 = s2
> 0 wenn s1 > s2
```

1.45 strnicmp_d

```
                strnicmp_d
vergleichen zweier Strings mit Beachtung einer Maximallänge
deutsche Version von "strnicmp"

Übersicht
#include <stormamiga.h>

r = strnicmp_d (s1, s2, n);

int r;
const char *s1;
const char *s2;
size_t n;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Vergleicht die beiden Strings "s1" und "s2", bis maximal zum Zeichen
mit Index "n", Zeichen für Zeichen, ohne die Groß- oder Kleinschreibung
der Buchstaben zu beachten.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt. Da es "ß"
nicht als Großbuchstabe gibt, muß dieser nicht extra unterstützt werden.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
< 0 wenn s1 < s2
= 0 wenn s1 = s2
> 0 wenn s1 > s2
```

1.46 strcasecmp_d

```
                strcasecmp_d
vergleichen zweier Strings
deutsche Version von "strcasecmp"

Übersicht
#include <stormamiga.h>

r = strcasecmp_d (s1, s2);

int r;
const char *s1;
const char *s2;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Vergleicht die beiden Strings "s1" und "s2" Zeichen für Zeichen,
ohne die Groß- oder Kleinschreibung der Buchstaben zu beachten.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt.
Da es "ß" nicht als Großbuchstabe gibt, muß dieser nicht extra
unterstützt werden.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
< 0 wenn s1 < s2
= 0 wenn s1 = s2
> 0 wenn s1 > s2
```

1.47 strncasecmp_d

```
                strncasecmp_d
vergleichen zweier Strings mit Beachtung einer Maximallänge
deutsche Version von "strncasecmp"

Übersicht
#include <stormamiga.h>

r = strncasecmp_d (s1, s2, n);

int r;
const char *s1;
const char *s2;
size_t n;

Standard
(noch) keiner (Eigenentwicklung)
```

```
Erklärung
Vergleicht die beiden Strings "s1" und "s2", bis maximal zum Zeichen
mit Index "n", Zeichen für Zeichen, ohne die Groß- oder Kleinschreibung
der Buchstaben zu beachten.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt. Da es "ß"
nicht als Großbuchstabe gibt, muß dieser nicht extra unterstützt werden.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
< 0 wenn s1 < s2
= 0 wenn s1 = s2
> 0 wenn s1 > s2
```

1.48 strlower_d

```
                strlower_d
wandelt die Großbuchstaben eines Strings in Kleinbuchstaben um
deutsche Version von "strlower"

Übersicht
#include <stormamiga.h>

r = strlower_d (s);

char *r;
char *s;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Falls ein oder mehrere Zeichen des Strings "s" Großbuchstabe sind,
werden sie in Kleinbuchstaben umgewandelt, ansonsten bleiben sie
unverändert.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt.
Da es "ß" nicht als Großbuchstabe gibt, muß dieser nicht extra
unterstützt werden.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
Der umgewandelte String "s".
```

1.49 strupper_d

```
                strupper_d
wandelt die Kleinbuchstaben eines Strings in Großbuchstaben um
deutsche Version von "strupper"

Übersicht
#include <stormamiga.h>

r = strupper_d (s);

char *r;
char *s;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Falls ein oder mehrere Zeichen des Strings "s" Kleinbuchstaben sind,
werden sie in Großbuchstaben umgewandelt, ansonsten bleiben sie
unverändert.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt.
Da es "ß" nicht als Großbuchstabe gibt, muß dieser nicht extra
unterstützt werden.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
Der umgewandelte String "s".
```

1.50 strlwr_d

```
                strlwr_d
wandelt die Großbuchstaben eines Strings in Kleinbuchstaben um
deutsche Version von "strlwr"

Übersicht
#include <stormamiga.h>

r = strlwr_d (s);

char *r;
char *s;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Falls ein oder mehrere Zeichen des Strings "s" Großbuchstabe sind,
werden sie in Kleinbuchstaben umgewandelt, ansonsten bleiben sie
unverändert.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt.
Da es "ß" nicht als Großbuchstabe gibt, muß dieser nicht extra
```

```
unterstützt werden.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
Der umgewandelte String "s".
```

1.51 strupr_d

```
                strupr_d
wandelt die Kleinbuchstaben eines Strings in Großbuchstaben um
deutsche Version von "strupr"

Übersicht
#include <stormamiga.h>

r = strupr_d (s);

char *r;
char *s;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Falls ein oder mehrere Zeichen des Strings "s" Kleinbuchstaben sind,
werden sie in Großbuchstaben umgewandelt, ansonsten bleiben sie
unverändert.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt.
Da es "ß" nicht als Großbuchstabe gibt, muß dieser nicht extra
unterstützt werden.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
Der umgewandelte String "s".
```

1.52 strdup

```
strdup
speichert die Kopie eines Strings

Übersicht
#include <stormamiga.h>

r = strdup (s);
```

```
char *r;
const char *s;

Standard
(noch) keiner (BSD)

Erklärung
"strdup" reserviert ausreichend Speicher für eine Kopie des Strings "s",
kopiert diesen, und gibt einen Zeiger auf die Kopie zurück. Dieser
Zeiger kann später als Argument für die Funktion  free  verwendet werden.

Rückgabe
Einen Zeiger auf die Kopie des Strings "s".
```

1.53 isalnum_d

```
                isalnum_d
testet, ob es sich um ein Buchstabe oder eine Ziffer handelt
deutsche Version von "isalnum"

Übersicht
#include <stormamiga.h>

r = isalnum_d (ch);

int r;
int ch;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Diese Funktion testet, ob das übergebene Zeichen ein Buchstabe oder
eine Ziffer ist.
Die Umlaute "ä", "ö", "ü", "ß", "Ä", "Ö" und "Ü" werden unterstützt.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
0 wenn das Zeichen kein Buchstabe und keine Ziffer ist, sonst ein
Wert ungleich 0
```

1.54 isalpha_d

```
                isalpha_d
testet, ob es sich um ein Buchstabe handelt
deutsche Version von "isalpha"
```

```
Übersicht
#include <stormamiga.h>

r = isalpha_d (ch);

int r;
int ch;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Diese Funktion testet, ob das übergebene Zeichen ein Buchstabe ist.
Die Umlaute "ä", "ö", "ü", "ß", "Ä", "Ö" und "Ü" werden unterstützt.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
0 wenn das Zeichen kein Buchstabe ist, sonst ein Wert ungleich 0
```

1.55 islower_d

```
                islower_d
testet, ob es sich um ein Kleinbuchstabe handelt
deutsche Version von "islower"

Übersicht
#include <stormamiga.h>

r = islower_d (ch);

int r;
int ch;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Diese Funktion testet, ob das übergebene Zeichen ein Kleinbuchstabe ist.
Die Umlaute "ä", "ö", "ü", "ß", "Ä", "Ö" und "Ü" werden unterstützt.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
0 wenn das Zeichen kein Kleinbuchstabe ist, sonst ein Wert ungleich 0
```

1.56 isupper_d

```
                isupper_d
testet, ob es sich um ein Großbuchstabe handelt
deutsche Version von "isupper"

Übersicht
#include <stormamiga.h>

r = isupper_d (ch);

int r;
int ch;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Diese Funktion testet, ob das übergebene Zeichen ein Großbuchstabe ist.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt. Da es "ß"
nicht als Großbuchstabe gibt, muß dieser nicht extra unterstützt werden.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
0 wenn das Zeichen kein Großbuchstabe ist, sonst ein Wert ungleich 0
```

1.57 tolower_d

```
                tolower_d
wandelt Großbuchstaben in Kleinbuchstaben um
deutsche Version von "tolower"

Übersicht
#include <stormamiga.h>

r = tolower_d (ch);

int r;
int ch;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Falls das Zeichen "ch" ein Großbuchstabe ist, wird es in einen
Kleinbuchstaben umgewandelt, ansonsten bleibt es unverändert.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt.
Da es "ß" nicht als Großbuchstabe gibt, muß dieser nicht extra
unterstützt werden.
```

```
Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
das umgewandelte Zeichen
```

1.58 toupper_d

```
                toupper_d
wandelt Kleinbuchstaben in Großbuchstaben um
deutsche Version von "toupper"

Übersicht
#include <stormamiga.h>

r = toupper_d (ch);

int r;
int ch;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Falls das Zeichen "ch" ein Kleinbuchstabe ist, wird es in einen
Großbuchstaben umgewandelt, ansonsten bleibt es unverändert.
Die Umlaute "ä", "ö", "ü", "Ä", "Ö" und "Ü" werden unterstützt.
Da es "ß" nicht als Großbuchstabe gibt, muß dieser nicht extra
unterstützt werden.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
das umgewandelte Zeichen
```

1.59 assert_

```
assert
testet eine Bedingung und unterbricht den Programmlauf

Übersicht
#include <stormamiga.h>

assert_ (x);

int x;
```

```
Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Siehe  assert .
Der einzige Unterschied zu "assert" besteht darin, daß zur Ausgabe
der Fehlermeldung "printf_" und nicht "printf" verwendet wird.
```

1.60 strftime

```
                strftime
Formatierte Ausgabe von Datum und Zeit in den Stringpuffer "s".
Erweiterte Version von "strftime" der "storm.lib".

Übersicht
#include <time.h>

r = strftime (s, size, format, tp);

size_t r;
char *s;
size_t size;
const char *format;
const struct tm *tp;

Standard
ANSI C3.159-1989 ("ANSI C")

Erklärung
Formatierte Ausgabe von Datum und Zeit in den
                 Stringpuffer
                 "s".
Der
                 Zeitformatstring
                 "format" beschreibt das Ausgabeformat.

Rückgabe
Die Anzahl der ausgegebenen Zeichen.
```

1.61 strftime_d

```
                strftime_d
Formatierte Ausgabe von Datum und Zeit in den Stringpuffer "s".
Erweiterte deutsche Version von "strftime" der "storm.lib".

Übersicht
#include <stormamiga.h>

r = strftime_d (s, size, format, tp);

size_t r;
```

```
char *s;
size_t size;
const char *format;
const struct tm *tp;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Formatierte Ausgabe von Datum und Zeit in den
                 Stringpuffer
                 "s".
Der
                 Zeitformatstring
                 "format" beschreibt das Ausgabeformat.
Die Namen der Monate und Wochentage werden in deutsch ausgegeben.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
Die Anzahl der ausgegebenen Zeichen.
```

1.62 asctime_d

```
                asctime_d
Erzeugt eine Zeichenkette aus Datum und Zeit
deutsche Version von "asctime"

Übersicht
#include <stormamiga.h>

tp = asctime_d (t);

char *tp;
const struct tm *t;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
"asctime_d" wandelt die Zeit aus "*t" in eine Zeichenkette der Form
"Mit Okt 07 01:03:42 1992" um. Diese Zeichenkette wird in einem
internen Puffer abgelegt und ein Zeiger darauf zurückgegeben.
Die Namen der Monate und Wochentage werden in deutsch ausgegeben.

Siehe auch bei
                 Deutsche Funktionen
                .

Rückgabe
Die Funktion liefert einen ASCII-Text mit der exakten Länge von 26 Zeichen.
Das Format des Textes ist:
```

```
  "DDD MMM dd hh:mm:ss YYYY\n\0"

DDD ist der Wochentag, MMM ist der Monat, dd ist der Tag im Monat,
hh:mm:ss sind Stunde:Minute:Sekunde und YYYY ist das Jahr. Zum Beispiel:
  "Mit Okt 25 12:05:43 1995\n\0"
```

1.63 ctime_d

```
                ctime_d
Konvertiert einen Zeit-Wert in einen ASCII-Text
deutsche Version von "ctime"

Übersicht
#include <stormamiga.h>

s = ctime_d (t);

char *s;
const time_t *t;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
"ctime_d" ist identisch mit "asctime_d (localtime (t))", wandelt also einen
"time_t"-Wert in eine Stringdarstellung um.
Die Namen der Monate und Wochentage werden in deutsch ausgegeben.

Siehe auch bei
                 Deutsche Funktionen
                .

Der Befehl "ctime_d" ist auch als
                 Inlinefunktionen
                 verfügbar.

Rückgabe
Die Funktion liefert einen ASCII-Text mit der exakten Länge von 26 Zeichen.
Das Format des Textes ist:
  "DDD MMM dd hh:mm:ss YYYY\n\0"
DDD ist der Wochentag, MMM ist der Monat, dd ist der Tag im Monat,
hh:mm:ss sind Stunde:Minute:Sekunde und YYYY ist das Jahr. Zum Beispiel:
  "Mit Okt 25 12:05:43 1995\n\0"
```

1.64 muls

```
                muls
signed 32 Bit mal 32 Bit Multiplikation mit 32 Bit Ergebnis

Übersicht
```

```
#include <stormamiga.h>

r = muls (arg1, arg2);

long r;
long arg1;
long arg2;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "muls" multipliziert "arg1" mit "arg2" und ermittelt
das Ergebnis "r".
Der Befehl "muls" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  SMult32  der "utility.library". Da die Funktion
und der Aufruf beider Befehle identisch ist, kann "SMult32"
einfach durch "muls" ersetzt werden.

Der Befehl "muls" ist auch als
                 Inlinefunktionen
                 verfügbar.

Rückgabe
signed 32 Bit Ergebnis "r"
```

1.65 mulu

```
                mulu
unsigned 32 Bit mal 32 Bit Multiplikation mit 32 Bit Ergebnis

Übersicht
#include <stormamiga.h>

r = mulu (arg1, arg2);

ulong r;
ulong arg1;
ulong arg2;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "mulu" multipliziert "arg1" mit "arg2" und ermittelt
das Ergebnis "r".
Der Befehl "mulu" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  UMult32  der "utility.library". Da die Funktion
und der Aufruf beider Befehle identisch ist, kann "UMult32"
einfach durch "mulu" ersetzt werden.

Der Befehl "mulu" ist auch als
                 Inlinefunktionen
                 verfügbar.
```

```
Rückgabe
unsigned 32 Bit Ergebnis "r"
```

1.66 divsl

```
divsl
signed 32 Bit durch 32 Bit Division mit 32 Bit Quotient und Rest

Übersicht
#include <stormamiga.h>

quotient : remainder = divsl (dividend, divisor);

long quotient;
long remainder
long dividend;
long divisor;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "divsl" teilt den Dividenden "dividend" durch den
Divisor "divisor" und ermittelt den Quotient "quotient" und den
Rest "remainder".
Der Befehl "divsl" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  SDivMod32  der "utility.library".
Da die Funktion und der Aufruf beider Befehle identisch ist, kann
"SDivMod32" einfach durch "divsl" ersetzt werden.

Rückgabe
signed 32 Bit Quotient "quotient"
signed 32 Bit Rest "remainder"
```

1.67 divul

```
divul
unsigned 32 Bit durch 32 Bit Division mit 32 Bit Quotient und Rest

Übersicht
#include <stormamiga.h>

quotient : remainder = divul (dividend, divisor);

ulong quotient;
ulong remainder
ulong dividend;
ulong divisor;

Standard
```

```
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "divul" teilt den Dividenden "dividend" durch den
Divisor "divisor" und ermittelt den Quotient "quotient" und den
Rest "remainder".
Der Befehl "divul" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  UDivMod32  der "utility.library".
Da die Funktion und der Aufruf beider Befehle identisch ist, kann
"UDivMod32" einfach durch "divul" ersetzt werden.

Rückgabe
unsigned 32 Bit Quotient "quotient"
unsigned 32 Bit Rest "remainder"
```

1.68 muls64

```
muls64
signed 32 Bit mal 32 Bit Multiplikation mit 64 Bit Ergebnis

Übersicht
#include <stormamiga.h>

r = muls64 (arg1, arg2);

long r;
long arg1;
long arg2;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "muls64" multipliziert "arg1" mit "arg2" und ermittelt
das Ergebnis "r".
Der Befehl "muls64" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  SMult64  der "utility.library". Da die Funktion
und der Aufruf beider Befehle identisch ist, kann "SMult64"
einfach durch "muls64" ersetzt werden.
Im Gegensatz zu "SMult64", benötigt "muls64" nicht AmigaOS 3.x,
sondern ist bereits ab AmigaOS 2.x lauffähig.

Rückgabe
signed 64 Bit Ergebnis "r"
```

1.69 mulu64

```
mulu64
unsigned 32 Bit mal 32 Bit Multiplikation mit 64 Bit Ergebnis

Übersicht
#include <stormamiga.h>
```

```
r = mulu64 (arg1, arg2);

ulong r;
ulong arg1;
ulong arg2;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "mulu64" multipliziert "arg1" mit "arg2" und ermittelt
das Ergebnis "r".
Der Befehl "mulu64" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  UMult64  der "utility.library". Da die Funktion
und der Aufruf beider Befehle identisch ist, kann "UMult64"
einfach durch "mulu64" ersetzt werden.
Im Gegensatz zu "UMult64", benötigt "mulu64" nicht AmigaOS 3.x,
sondern ist bereits ab AmigaOS 2.x lauffähig.

Rückgabe
unsigned 64 Bit Ergebnis "r"
```

1.70 divs64

```
divs64
signed 64 Bit durch 32 Bit Division mit 32 Bit Quotient und Rest

Übersicht
#include <stormamiga.h>

quotient : remainder = divs64 (dividend, divisor);

long quotient;
long remainder
long dividend;
long divisor;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "divs64" teilt den Dividenden "dividend" durch den
Divisor "divisor" und ermittelt den Quotient "quotient" und den
Rest "remainder".

Rückgabe
signed 32 Bit Quotient "quotient"
signed 32 Bit Rest "remainder"
```

1.71 divu64

```
divu64
unsigned 64 Bit durch 32 Bit Division mit 32 Bit Quotient und Rest

Übersicht
#include <stormamiga.h>

quotient : remainder = divu64 (dividend, divisor);

ulong quotient;
ulong remainder
ulong dividend;
ulong divisor;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "divu64" teilt den Dividenden "dividend" durch den
Divisor "divisor" und ermittelt den Quotient "quotient" und den
Rest "remainder".

Rückgabe
unsigned 32 Bit Quotient "quotient"
unsigned 32 Bit Rest "remainder"
```

1.72 muls_r

```
                muls_r
signed 32 Bit mal 32 Bit Multiplikation mit 32 Bit Ergebnis
Registerversion von "muls"

Übersicht
#include <stormamiga.h>

r = muls_r (register __d0 arg1, register __d1 arg2);

long r;
long arg1;
long arg2;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "muls_r" multipliziert "arg1" mit "arg2" und ermittelt
das Ergebnis "r".
Der Befehl "muls_r" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  SMult32  der "utility.library". Da die Funktion
und der Aufruf beider Befehle identisch ist, kann "SMult32"
einfach durch "muls_r" ersetzt werden.

Siehe auch bei
                 Registerfunktionen
                .
```

```
Rückgabe
signed 32 Bit Ergebnis "r"
```

1.73 mulu_r

```
                mulu_r
unsigned 32 Bit mal 32 Bit Multiplikation mit 32 Bit Ergebnis
Registerversion von "mulu"

Übersicht
#include <stormamiga.h>

r = mulu_r (register __d0 arg1, register __d1 arg2);

ulong r;
ulong arg1;
ulong arg2;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "mulu_r" multipliziert "arg1" mit "arg2" und ermittelt
das Ergebnis "r".
Der Befehl "mulu_r" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  UMult32  der "utility.library". Da die Funktion
und der Aufruf beider Befehle identisch ist, kann "UMult32"
einfach durch "mulu_r" ersetzt werden.

Siehe auch bei
                 Registerfunktionen
                .

Rückgabe
unsigned 32 Bit Ergebnis "r"
```

1.74 divsl_r

```
                divsl_r
signed 32 Bit durch 32 Bit Division mit 32 Bit Quotient und Rest
Registerversion von "divsl"

Übersicht
#include <stormamiga.h>

quotient : remainder = divsl_r (register __d0 dividend, register __d1 divisor);

long quotient;
long remainder
```

```
long dividend;
long divisor;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "divsl_r" teilt den Dividenden "dividend" durch den
Divisor "divisor" und ermittelt den Quotient "quotient" und den
Rest "remainder".
Der Befehl "divsl_r" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  SDivMod32  der "utility.library".
Da die Funktion und der Aufruf beider Befehle identisch ist, kann
"SDivMod32" einfach durch "divsl_r" ersetzt werden.

Siehe auch bei
                 Registerfunktionen
                .

Rückgabe
signed 32 Bit Quotient "quotient"
signed 32 Bit Rest "remainder"
```

1.75 divul_r

```
                divul_r
unsigned 32 Bit durch 32 Bit Division mit 32 Bit Quotient und Rest
Registerversion von "divul"

Übersicht
#include <stormamiga.h>

quotient : remainder = divul_r (register __d0 dividend, register __d1 divisor);

ulong quotient;
ulong remainder
ulong dividend;
ulong divisor;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "divul_r" teilt den Dividenden "dividend" durch den
Divisor "divisor" und ermittelt den Quotient "quotient" und den
Rest "remainder".
Der Befehl "divul_r" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  UDivMod32  der "utility.library".
Da die Funktion und der Aufruf beider Befehle identisch ist, kann
"UDivMod32" einfach durch "divul_r" ersetzt werden.

Siehe auch bei
                 Registerfunktionen
                .
```

```
Rückgabe
unsigned 32 Bit Quotient "quotient"
unsigned 32 Bit Rest "remainder"
```

1.76 muls64_r

```
                muls64_r
signed 32 Bit mal 32 Bit Multiplikation mit 64 Bit Ergebnis
Registerversion von "muls64"

Übersicht
#include <stormamiga.h>

r = muls64_r (register __d0 arg1, register __d1 arg2);

long r;
long arg1;
long arg2;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "muls64_r" multipliziert "arg1" mit "arg2" und ermittelt
das Ergebnis "r".
Der Befehl "muls64_r" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  SMult64  der "utility.library". Da die Funktion
und der Aufruf beider Befehle identisch ist, kann "SMult64"
einfach durch "muls64_r" ersetzt werden.
Im Gegensatz zu "SMult64", benötigt "muls64_r" nicht AmigaOS 3.x,
sondern ist bereits ab AmigaOS 2.x lauffähig.

Siehe auch bei
                 Registerfunktionen
                .

Rückgabe
signed 64 Bit Ergebnis "r"
```

1.77 mulu64_r

```
                mulu64_r
unsigned 32 Bit mal 32 Bit Multiplikation mit 64 Bit Ergebnis
Registerversion von "mulu64"

Übersicht
#include <stormamiga.h>

r = mulu64_r (register __d0 arg1, register __d1 arg2);
```

```
ulong r;
ulong arg1;
ulong arg2;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "mulu64_r" multipliziert "arg1" mit "arg2" und ermittelt
das Ergebnis "r".
Der Befehl "mulu64_r" ist ein sehr schneller und sehr kleiner Ersatz
für den Befehl  UMult64  der "utility.library". Da die Funktion
und der Aufruf beider Befehle identisch ist, kann "UMult64"
einfach durch "mulu64_r" ersetzt werden.
Im Gegensatz zu "UMult64", benötigt "mulu64_r" nicht AmigaOS 3.x,
sondern ist bereits ab AmigaOS 2.x lauffähig.

Siehe auch bei
                 Registerfunktionen
                .

Rückgabe
unsigned 64 Bit Ergebnis "r"
```

1.78 divs64_r

```
                divs64_r
signed 64 Bit durch 32 Bit Division mit 32 Bit Quotient und Rest
Registerversion von "divs64"

Übersicht
#include <stormamiga.h>

quotient : remainder = divs64_r (register __d0 dividend, register __d1 divisor);

long quotient;
long remainder
long dividend;
long divisor;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "divs64_r" teilt den Dividenden "dividend" durch den
Divisor "divisor" und ermittelt den Quotient "quotient" und den
Rest "remainder".

Siehe auch bei
                 Registerfunktionen
                .

Rückgabe
```

```
signed 32 Bit Quotient "quotient"
signed 32 Bit Rest "remainder"
```

1.79 divu64_r

```
                divu64_r
unsigned 64 Bit durch 32 Bit Division mit 32 Bit Quotient und Rest
Registerversion von "divu64"

Übersicht
#include <stormamiga.h>

quotient : remainder = divu64_r (register __d0 dividend, register __d1 divisor);

ulong quotient;
ulong remainder
ulong dividend;
ulong divisor;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "divu64_r" teilt den Dividenden "dividend" durch den
Divisor "divisor" und ermittelt den Quotient "quotient" und den
Rest "remainder".

Siehe auch bei
                 Registerfunktionen
                .

Rückgabe
unsigned 32 Bit Quotient "quotient"
unsigned 32 Bit Rest "remainder"
```

1.80 button_al

```
button_al
Abfrage der linken Maus- oder Joysticktaste an Port A

Übersicht
#include <stormamiga.h>

r = button_al ();

int r;

Standard
(noch) keiner (Eigenentwicklung)
```

```
Erklärung
Der Befehl "button_al" ist zur Abfrage der linken Maus- oder
Joysticktaste an Port A. Wenn zum Zeitpunkt der Abfrage die
entsprechende Taste betätigt ist, wird 1 zurückgegeben, sonst 0.

Rückgabe
1 wenn die entsprechende Taste betätigt ist, sonst 0

Beispiel
#include <stormamiga.h>

main()
{
start:
    if (!button_al ())
  goto start;
    printf ("Hello World!\n");
}
```

1.81 button_ar

```
button_ar
Abfrage der rechten Maustaste an Port A

Übersicht
#include <stormamiga.h>

r = button_ar ();

int r;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "button_ar" ist zur Abfrage der rechten Maustaste an Port A.
Wenn zum Zeitpunkt der Abfrage die entsprechende Taste betätigt ist,
wird 1 zurückgegeben, sonst 0.

Rückgabe
1 wenn die entsprechende Taste betätigt ist, sonst 0

Beispiel
#include <stormamiga.h>

main()
{
start:
    if (!button_ar ())
  goto start;
    printf ("Hello World!\n");
}
```

1.82 button_bl

```
button_bl
Abfrage der linken Maus- oder Joysticktaste an Port B

Übersicht
#include <stormamiga.h>

r = button_bl ();

int r;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "button_bl" ist zur Abfrage der linken Maus- oder
Joysticktaste an Port B. Wenn zum Zeitpunkt der Abfrage die
entsprechende Taste betätigt ist, wird 1 zurückgegeben, sonst 0.

Rückgabe
1 wenn die entsprechende Taste betätigt ist, sonst 0

Beispiel
#include <stormamiga.h>

main()
{
start:
    if (!button_bl ())
  goto start;
    printf ("Hello World!\n");
}
```

1.83 button_br

```
button_br
Abfrage der rechten Maustaste an Port B

Übersicht
#include <stormamiga.h>

r = button_br ();

int r;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "button_br" ist zur Abfrage der rechten Maustaste an Port B.
Wenn zum Zeitpunkt der Abfrage die entsprechende Taste betätigt ist,
wird 1 zurückgegeben, sonst 0.
```

```
Rückgabe
1 wenn die entsprechende Taste betätigt ist, sonst 0

Beispiel
#include <stormamiga.h>

main()
{
start:
    if (!button_br ())
  goto start;
    printf ("Hello World!\n");
}
```

1.84 button

```
button
Abfrage der Maus- und Joysticktasten

Übersicht
#include <stormamiga.h>

r = button (port, button);

int r;
int port;
int button;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "button" ist zur Abfrage der Maus- und Joysticktasten.
Für "port" kann 0, für Port A, oder 1, für Port B, angegeben werden.
Für "button" kann 0, für die linke Maustaste oder die Feuertaste am
Joystick, oder 1, für die rechte Maustaste, angegeben werden.
Wenn zum Zeitpunkt der Abfrage die entsprechende Taste betätigt ist,
wird 1 zurückgegeben, sonst 0.

Rückgabe
1 wenn die entsprechende Taste betätigt ist, sonst 0

Beispiel
#include <stormamiga.h>

main()
{
 int p = 0; // Port A
 int b = 0; // linke Maustaste oder Feuertaste

start:
    if (!button (p, b))
  goto start;
    printf ("Hello World!\n");
```

```
}
```

1.85 button_r

```
                button_r
Abfrage der Maus- und Joysticktasten
Registerversion von "button"

Übersicht
#include <stormamiga.h>

r = button_r (register __d0 port, register __d1 button);

int r;
int port;
int button;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "button_r" ist zur Abfrage der Maus- und Joysticktasten.
Für "port" kann 0, für Port A, oder 1, für Port B, angegeben werden.
Für "button" kann 0, für die linke Maustaste oder die Feuertaste am
Joystick, oder 1, für die rechte Maustaste, angegeben werden.
Wenn zum Zeitpunkt der Abfrage die entsprechende Taste betätigt ist,
wird 1 zurückgegeben, sonst 0.

Siehe auch bei
                 Registerfunktionen
                .

Rückgabe
1 wenn die entsprechende Taste betätigt ist, sonst 0

Beispiel
#include <stormamiga.h>

main()
{
 int p = 0; // Port A
 int b = 0; // linke Maustaste oder Feuertaste

start:
    if (!button_r (p, b))
  goto start;
    printf ("Hello World!\n");
}
```

1.86 waitbutton_al

```
waitbutton_al
Abfrage der linken Maus- oder Joysticktaste an Port A

Übersicht
#include <stormamiga.h>

r = waitbutton_al ();

void r;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "waitbutton_al" ist zur Abfrage der linken Maus- oder
Joysticktaste an Port A. Das Programm wartet solange, bis die
richtige Taste betätigt wird.

Rückgabe
keine

Beispiel
#include <stormamiga.h>

main()
{
    waitbutton_al ();
    printf ("Hello World!\n");
}
```

1.87 waitbutton_ar

```
waitbutton_ar
Abfrage der rechten Maustaste an Port A

Übersicht
#include <stormamiga.h>

r = waitbutton_ar ();

void r;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "waitbutton_ar" ist zur Abfrage der rechten Maustaste an Port A.
Das Programm wartet solange, bis die richtige Taste betätigt wird.

Rückgabe
keine

Beispiel
#include <stormamiga.h>
```

```
main()
{
    waitbutton_ar ();
    printf ("Hello World!\n");
}
```

1.88 waitbutton_bl

```
waitbutton_bl
Abfrage der linken Maus- oder Joysticktaste an Port B

Übersicht
#include <stormamiga.h>

r = waitbutton_bl ();

void r;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "waitbutton_bl" ist zur Abfrage der linken Maus- oder
Joysticktaste an Port B. Das Programm wartet solange, bis die
richtige Taste betätigt wird.

Rückgabe
keine

Beispiel
#include <stormamiga.h>

main()
{
    waitbutton_bl ();
    printf ("Hello World!\n");
}
```

1.89 waitbutton_br

```
waitbutton_br
Abfrage der rechten Maustaste an Port B

Übersicht
#include <stormamiga.h>

r = waitbutton_br ();

void r;

Standard
```

```
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "waitbutton_br" ist zur Abfrage der rechten Maustaste an Port B.
Das Programm wartet solange, bis die richtige Taste betätigt wird.

Rückgabe
keine

Beispiel
#include <stormamiga.h>

main()
{
    waitbutton_br ();
    printf ("Hello World!\n");
}
```

1.90 waitbutton

```
waitbutton
Abfrage der Maus- und Joysticktasten

Übersicht
#include <stormamiga.h>

r = waitbutton (port, button);

void r;
int port;
int button;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "waitbutton" ist zur Abfrage der Maus- und Joysticktasten.
Für "port" kann 0, für Port A, oder 1, für Port B, angegeben werden.
Für "button" kann 0, für die linke Maustaste oder die Feuertaste am
Joystick, oder 1, für die rechte Maustaste, angegeben werden.
Das Programm wartet solange, bis die richtige Taste betätigt wird.

Rückgabe
keine

Beispiel
#include <stormamiga.h>

main()
{
 int p = 0; // Port A
 int b = 0; // linke Maustaste oder Feuertaste

    waitbutton (p, b);
    printf ("Hello World!\n");
```

```
}
```

1.91 waitbutton_r

```
                waitbutton_r
Abfrage der Maus- und Joysticktasten
Registerversion von "waitbutton"

Übersicht
#include <stormamiga.h>

r = waitbutton_r (register __d0 port, register __d1 button);

void r;
int port;
int button;

Standard
(noch) keiner (Eigenentwicklung)

Erklärung
Der Befehl "waitbutton_r" ist zur Abfrage der Maus- und Joysticktasten.
Für "port" kann 0, für Port A, oder 1, für Port B, angegeben werden.
Für "button" kann 0, für die linke Maustaste oder die Feuertaste am
Joystick, oder 1, für die rechte Maustaste, angegeben werden.
Das Programm wartet solange, bis die richtige Taste betätigt wird.

Siehe auch bei
                 Registerfunktionen
                .

Rückgabe
keine

Beispiel
#include <stormamiga.h>

main()
{
 int p = 0; // Port A
 int b = 0; // linke Maustaste oder Feuertaste

    waitbutton_r (p, b);
    printf ("Hello World!\n");
}
```

1.92 stringpuffer

```
Der Stringpuffer "s"

Der Stringpuffer "s" muß mindestens so groß sein, daß die
```

```
Ausgabe mit abschließenden Nullzeichen hineinpaßt.
Die Funktion kann nicht feststellen ob der Stringpuffer groß
genug ist.
```

1.93 parameterliste

```
Die Parameterliste "vl"

Die Parameterliste "vl" muß vor dem Aufruf mit va_start initia-
lisiert werden und nach dem Aufruf mit va_end abgeschlossen wer-
den.
```

1.94 ausgabeformatstring_

```
Der Ausgabeformatstring_ "format" zur formatierten Ausgabe
Der Formatstring "format" zur formatierten Ausgabe besteht aus
Formatkommandos und Ausgabezeichen. Die Formatkommandos bestimmen
den Typ der Parameter der Ausgabefunktion und die Art der Konver-
tierung und Ausgabe. Alle Zeichen, die kein Formatkommando sind,
also nicht mit dem Zeichen "%" beginnen sind Ausgabezeichen und
werden unverändert ausgegeben.

Der Aufbau eines Formatkommandos:

% [flags] [width [.limit] ] [size] type

Die Angaben in den eckigen Klammern können optional angegeben werden.

flags
"-" zur Linksjustierung;
"+" zur Ausgabe auch eines positiven Vorzeichens bei Zahlen;
"0" zur Ausgabe führender Nullen bei Zahlen;
"#" zur Ausgabe von "0x" bei hexadezimalen Zahlen und "0" bei
    oktalen Zahlen

width
Feldbreite als dezimale Ziffernfolge oder "*", in diesem Fall wird
die Feldbreite als nächstes Argument des Typs int übergeben. Die
Feldbreite ist immer ein minimaler Wert, zu lange Ausgaben werden
nicht beschnitten.

limit
Die Genauigkeit als dezimale Ziffernfolge oder "*", in diesem Fall
wird die Genauigkeit als nächstes Argument des Typs int übergeben.
Der Wert beschreibt die maximale Anzahl von Zeichen bei Ausgabe
einer Zeichenkette oder die minimale Anzahl von Ziffern einer ganz-
zahligen Ausgabe.

size
Längenangabe des Arguments:
"h" für ein Argument des Typs short int oder unsigned short int;
"l" für ein Argument des Typs long int oder unsigned long int;
```

```
type
Typangabe des Arguments:

"d"
"i" zur Ausgabe einer vorzeichenbehafteten Dezimalzahl, das zuge-
    hörige Argument ist vom Typ int

"o" zur Ausgabe einer vorzeichenlosen Oktalzahl, das zugehörige
    Argument ist vom Typ int oder unsigned int

"x" zur Ausgabe einer vorzeichenlosen Hexadezimalzahl mit Kleinbuch-
    staben, das zugehörige Argument ist vom Typ int oder unsigned int

"X" zur Ausgabe einer vorzeichenlosen Hexadezimalzahl mit Großbuch-
    staben, das zugehörige Argument ist vom Typ int oder unsigned int

"u" zur Ausgabe einer vorzeichenlosen Dezimalzahl, das zugehörige
    Argument ist vom Typ unsigned int

"c" zur Ausgabe eines Zeichens, das zugehörige Argument ist vom Typ
    int und wird in unsigned char konvertiert

"s" zur Ausgabe einer Zeichenkette, die mit einem Nullzeichen abge-
    schlossen ist, das zugehörige Argument ist vom Typ char *

"p" zur Ausgabe einer hexadezimalen Speicheradresse, das zugehörige
    Argument ist vom Typ void *;

"n" zur Speicherung der Anzahl der bisher von diesem Funktionsaufruf
    ausgegebenen Zeichen in der Variablen, auf die das Argument vom
    Typ int * zeigt, es erfolgt keine Ausgabe

"%" zur Ausgabe eines Prozentzeichens

Alle anderen Typangaben führen zu undefinierten Ausgaben.
```

1.95 eingabeformatstring

```
Der Eingabeformatstring "format" zur formatierten Eingabe
Der Formatstring "format" zur formatierten Eingabe besteht aus den
Formatkommandos und Eingabezeichen. Die Formatkommandos bestimmen
den Typ der Parameter der Eingabefunktion und die Art der Konver-
tierung und Eingabe. Alle Zeichen, die kein Formatkommando sind,
also nicht mit dem Zeichen "%" beginnen und keine Trennzeichen (Leer-
zeichen, Tabulatoren und Zeilenvorschübe) sind, sind Eingabezeichen
und werden unverändert in der Eingabe erwartet.

Der Aufbau eines Formatkommandos:

% [width] [size] type

Die Angaben in den eckigen Klammern können optional angegeben werden.

width
```

```
Die Anzahl der zu lesenden Zeichen als dezimale Ziffernfolge oder "*",
in diesem Fall werden die Zeichen zwar gelesen, aber nicht in das
nächste Argument übertragen.

size
Längenangabe des Arguments:
"h" für ein Argument des Typs short int oder unsigned short int
"l" für ein Argument des Typs long int oder unsigned long int

type
Typangabe des Arguments:

"d" zur Eingabe einer vorzeichenbehafteten dezimalen Ganzzahl, das zuge-
    hörige Argument ist vom Typ int *

"i" zur Eingabe einer vorzeichenbehafteten Dezimalzahl, Oktalzahl (bei
    führender '0') oder Hexadezimalzahl (bei führendem '0x' oder '0X'),
    das zugehörige Argument ist vom Typ int *

"o" zur Eingabe einer Oktalzahl, das zugehörige Argument ist vom Typ int *

"x" zur Eingabe einer Hexadezimalzahl mit oder ohne '0x', das zugehörige
    Argument ist vom Typ int oder unsigned int

"c" zur Eingabe von "width" Zeichen, wobei Leerzeichen und Zeilentrenner
    nicht überlesen werden und kein Nullzeichen angehängt wird, das zuge-
    hörige Argument ist vom Typ char *

"s" zur Eingabe einer Zeichenkette, wobei führende Leerzeichen und Zeilen-
    trenner überlesen werden und ein Nullzeichen angehängt wird, das zuge-
    hörigen Argument ist vom Typ char *

"e"
"f"
"g" zur Eingabe einer Fließkommazahl in beliebiger Darstellung, das zuge-
    hörige Argument ist vom Typ float *

"p" zur Eingabe einer hexadezimalen Speicheradresse, das zugehörige Argument
    ist vom Typ int *

"n" zur Speicherung der Anzahl der bisher von diesem Funktionsaufruf gele-
    senen Zeichen in der Variablen, auf die das Argument vom Typ int * zeigt,
    es wird kein Zeichen gelesen.

"[...]"  zur Eingabe einer Zeichenkette, die nur aus den in den eckigen
   Klammern angegebenen Zeichen besteht, wobei ein Nullzeichen ange-
   hängt wird, das zugehörigen Argument ist vom Typ char *

"[^...]" zur Eingabe einer Zeichenkette, die nur aus Zeichen besteht, die
   nicht in den eckigen Klammern angegebenen sind, wobei ein Null-
   zeichen angehängt wird; das zugehörigen Argument ist vom Typ char *

"%%"     zur Eingabe eines Prozentzeichens

Alle anderen Typangaben führen zu undefinierten Eingaben.
```

1.96 eingabeformatstring_

```
Der Eingabeformatstring_ "format" zur formatierten Eingabe
Der Formatstring "format" zur formatierten Eingabe besteht aus den
Formatkommandos und Eingabezeichen. Die Formatkommandos bestimmen
den Typ der Parameter der Eingabefunktion und die Art der Konver-
tierung und Eingabe. Alle Zeichen, die kein Formatkommando sind,
also nicht mit dem Zeichen "%" beginnen und keine Trennzeichen (Leer-
zeichen, Tabulatoren und Zeilenvorschübe) sind, sind Eingabezeichen
und werden unverändert in der Eingabe erwartet.

Der Aufbau eines Formatkommandos:

% [width] [size] type

Die Angaben in den eckigen Klammern können optional angegeben werden.

width
Die Anzahl der zu lesenden Zeichen als dezimale Ziffernfolge oder "*",
in diesem Fall werden die Zeichen zwar gelesen, aber nicht in das
nächste Argument übertragen.

size
Längenangabe des Arguments:
"h" für ein Argument des Typs short int oder unsigned short int
"l" für ein Argument des Typs long int oder unsigned long int

type
Typangabe des Arguments:

"d" zur Eingabe einer vorzeichenbehafteten dezimalen Ganzzahl, das zuge-
    hörige Argument ist vom Typ int *

"i" zur Eingabe einer vorzeichenbehafteten Dezimalzahl, Oktalzahl (bei
    führender '0') oder Hexadezimalzahl (bei führendem '0x' oder '0X'),
    das zugehörige Argument ist vom Typ int *

"o" zur Eingabe einer Oktalzahl, das zugehörige Argument ist vom Typ int *

"x" zur Eingabe einer Hexadezimalzahl mit oder ohne '0x', das zugehörige
    Argument ist vom Typ int oder unsigned int

"c" zur Eingabe von "width" Zeichen, wobei Leerzeichen und Zeilentrenner
    nicht überlesen werden und kein Nullzeichen angehängt wird, das zuge-
    hörige Argument ist vom Typ char *

"s" zur Eingabe einer Zeichenkette, wobei führende Leerzeichen und Zeilen-
    trenner überlesen werden und ein Nullzeichen angehängt wird, das zuge-
    hörigen Argument ist vom Typ char *

"p" zur Eingabe einer hexadezimalen Speicheradresse, das zugehörige Argument
    ist vom Typ int *

"n" zur Speicherung der Anzahl der bisher von diesem Funktionsaufruf gele-
    senen Zeichen in der Variablen, auf die das Argument vom Typ int * zeigt,
    es wird kein Zeichen gelesen.
```

```
"[...]"  zur Eingabe einer Zeichenkette, die nur aus den in den eckigen
   Klammern angegebenen Zeichen besteht, wobei ein Nullzeichen ange-
   hängt wird, das zugehörigen Argument ist vom Typ char *

"[^...]" zur Eingabe einer Zeichenkette, die nur aus Zeichen besteht, die
   nicht in den eckigen Klammern angegebenen sind, wobei ein Null-
   zeichen angehängt wird; das zugehörigen Argument ist vom Typ char *

"%%"     zur Eingabe eines Prozentzeichens

Alle anderen Typangaben führen zu undefinierten Eingaben.
```

1.97 zeitformatstring

```
Zeitformatstring zur Konvertierung einer Zeitangabe.
Der Zeitformatstring "format" zur formatierten Ausgabe von Datum und Zeit besteht
aus den Formatkommandos und Eingabezeichen. Die Formatkommandos bestimmen den
Typ der Parameter der Ausgabefunktion und die Art der Konvertierung und Ausgabe.
Alle Zeichen, die kein Formatkommando sind, also nicht mit dem Zeichen "%"  ↩
   beginnen
sind Ausgabezeichen und werden unverändert ausgegeben.

Der Aufbau eines Formatkommandos:

%type

type
Typangabe des Arguments:

"a" für den abgekürzten Namen des Wochentages (Mon, Tue, ...)

"A" für den vollständigen Namen des Wochentages

"b"
"h" für den abgekürzten Namen des Monats (Jan, Feb, ...)

"B" für den vollständigen Namen des Monats

"c" für die Kurzdarstellung von Datum und Uhrzeit ("%m/%d/%y %I:%M:%S")

"C" für die Darstellung im Format "%a %b %e %I:%M:%S %Y"

"d" für die Nummer des Tages des Monats (01 bis 31)

"e" für die Nummer des Tages des Monats ( 1 bis 31)

"H" für die amerikanische Stundendarstellung (01 bis 12)

"I" für die europäische Stundendarstellung (00 bis 23)

"j" für die Nummer des Tages im Jahr (001 bis 366)

"k" für die europäische Stundendarstellung ( 0 bis 23)
```

"l" für die amerikanische Stundendarstellung (1 bis 12)

"m" für die Nummer des Monats (01 bis 12)

"M" für die Anzahl der Minuten (00 bis 59)

"n" für eine neue Zeile

"p" für die Tageshälfte ("AM" oder "PM")

"r" für die Darstellung im Format "%H:%M:%S %p"

"R" für die Darstellung im Format "%I:%M"

"S" für die Anzahl von Sekunden (00 bis 60)

"t" für einen Tabulator

"u" für die Nummer des Wochentages (1 bis 7); Montag ist erster Wochentag

"U" für die Nummer der Woche im Jahr (00 bis 53); Sonntag ist erster Wochentag

"V" für die Nummer der Woche im Jahr (01 bis 53); Montag ist erster Wochentag

"w" für die Nummer des Wochentages (0 bis 6); Sonntag ist erster Wochentag

"W" für die Nummer der Woche im Jahr (00 bis 53); Montag ist erster Wochentag

"x"
"D" für die Kurzdarstellung des Datums ("%m/%d/%y")

"X"
"T" für die Kurzdarstellung der Uhrzeit ("%I:%M:%S")

"y" für die Jahreszahl ohne Jahrhundert

"Y" für die vollständige Jahreszahl mit Jahrhundert

"Z" für den Namen der Zeitzone

"%" für ein Prozentzeichen

Alle anderen Typausgaben führen zu undefinierten Ausgaben.

1.98 anwendungshinweise

Anwendungshinweise:
~~~~~~~~~~~~~~~~~~~

Obwohl die Anwendung der "stormamiga.lib" sehr einfach ist, möchte
ich in diesem Abschnitt einige Hinweise geben.

Allgemeine Hinweise
~~~~~~~~~~~~~~~~~~~

Nützliche Hinweise zur Anwendung.

Inlinefunktionen
Beschreibung der Inlinefunktionen.

Registerfunktionen
Beschreibung der Registerfunktionen.

Deutsche Funktionen
Beschreibung der deutschen Funktionen.

1.99 allgemeine hinweise

```
                Allgemeine Hinweise:
~~~~~~~~~~~~~~~~~~~~
```

Die "stormamiga.lib" enthält die Funktionen für das große
Codemodell in Verbindung mit dem großen Datenmodell und dem
kleinen Datenmodell A4.

Die "stormamiga_nc.lib" enthält die Funktionen für das kleine
Codemodell in Verbindung mit dem großen Datenmodell und dem
kleinen Datenmodell A4.

Die Includedatei "stormamiga.h" enthält die Deklaration für
alle Spezialfunktionen. Die Includedateien "stdio.h" und "time.h"
werden automatisch geöffnet.
Um nicht ständig unendliche Verse wie "unsigned long long int"
schreiben zu müssen, habe ich kurze Worte wie "ullint" definiert.

Definitionen

```
lint   für long int
llint  für long long int
uint   für unsigned int
ulint  für unsigned long int
ullint für unsigned long long int
llong  für long long
ulong  für unsigned long
ullong für unsigned long long
```

Die Includedatei "stormamigainline.h" enthält alle verfügbaren

Inlinefunktionen
.

Wenn Sie fragen zum Startupcode haben, dann sehen Sie bitte bei

Der Startupcode
nach.

```
Alle Funktionen mit dem Namen "..._r" (z.B.: "mulu_r") sind

                 Registerfunktionen
                .

Alle Funktionen mit dem Namen "..._d" (z.B.: "ctime_d") sind

                 Deutsche Funktionen
                .
```

Wenn Ihre Quelltexte auch mit anderen Compilern problemlos funktionieren sollen, dann sollten Sie Ihre Quelltexte wie dieses Beispiel gestalten.

Beispiel

```
#ifdef __STORM__
  #define STORMAMIGA_INLINE
  #define STORMAMIGA_DEUTSCH
  #include <stormamiga.h>
  #define printf printf_
  #define main main__
#else
#include <stdio.h>
#include <time.h>
#endif

void main (void)
{
  struct tm *tp;
  time_t t;

  time (&t);
  tp = localtime (&t);
  printf ("Die aktuelle Zeit ist %s", asctime (tp));
}
```

1.100 inlinefunktionen

Die Inlinefunktionen:
~~~~~~~~~~~~~~~~~~~~

Die Inlinefunktionen werden, wie der Name schon sagt, an die Stelle (in die Zeile oder Linie) des Funktionsaufrufes eingefügt. Dadurch werden die Programme meistens etwas kürzer und schneller. Bei keiner oder geringer Optimierung kann es sein, daß die Programme wesentlich größer und langsamer werden.

Wenn Sie die Inlinefunktionen nutzen wollen, müssen Sie die Zeile "#define STORMAMIGA_INLINE", vor dem Aufruf der Includedatei "stormamiga.h", in Ihr Programm einbinden. Sie können "#define STORMAMIGA_INLINE" aber auch bei den Compileroptionen (Preprozessor)
~~~~~~~~~~~~~~~~~~~~

des Menüs Einstellungen eintragen.

Alle verfügbaren Funktionen:

stdio Funktionen

fgetc()
getc()
getchar()
ungetc()
fputc()
putc()
putchar()
feof()
ferror()
clearerr()
perror()
remove()
rename()
setbuf()
setbuffer()
setlinebuf()
vprintf()
vprintf_()
vscanf()
vscanf_()

stdlib Funktionen

atoi()
atol()
atoll()
atof()
abort()

time Funktionen

localtime()
ctime()
difftime()
ctime_d()

Spezial Funktionen

muls()
mulu()

1.101 registerfunktionen

Die Registerfunktionen:
~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~

Bei den Registerfunktionen werden die Daten und Parameter nicht über
den Stackpointer, sondern über festgelegte Register übergeben. Dadurch
werden die Programme manchmal etwas kürzer und schneller.

Wenn Sie die Registerfunktionen nutzen wollen, müssen Sie die Zeile
"#define STORMAMIGA_REGISTER", vor dem Aufruf der Includedatei
"stormamiga.h", in Ihr Programm einbinden. Sie können "#define
STORMAMIGA_REGISTER" aber auch bei den Compileroptionen (Preprozessor)
des Menüs Einstellungen eintragen.

Alle verfügbaren Funktionen:

Spezial Funktionen

muls_r()
mulu_r()
divsl_r()
divul_r()
muls64_r()
mulu64_r()
divs64_r()
divu64_r()
button_r()
waitbutton_r()

1.102 deutsche funktionen

Die deutschen Funktionen:
~~~~~~~~~~~~~~~~~~~~~~~~~

Da in "ANSI C" und "C++" keine Umlaute unterstützt werden und alle
Texte auf englisch ausgegeben werden, habe ich einige deutsche Funktionen
geschrieben, die alle Texte auf deutsch ausgeben und alle Umlaute
unterstützen.
Wenn Sie alle deutschen Funktionen verwenden wollen, müssen Sie die Zeile
"#define STORMAMIGA_DEUTSCH", vor dem Aufruf der Includedatei
"stormamiga.h", in Ihr Programm einbinden. Sie können "#define
STORMAMIGA_DEUTSCH" aber auch bei den Compileroptionen (Preprozessor)
des Menüs Einstellungen eintragen.
Wenn Sie nur einige deutsche Funktionen verwenden wollen, müssen Sie an
die Namen der entsprechenden Funktionen "_d" anhängen (aus "ctime" wird
also "ctime_d").

Alle verfügbaren Funktionen:

string Funktionen

stricmp_d()
strnicmp_d()
strcasecmp_d()
strncasecmp_d()
strlwr_d()
strupr_d()
~~~~~~~~~~~~~~~~~~~~~~~~~

```
strlower_d()
strupper_d()


ctype Funktionen

isalnum_d()
isalpha_d()
islower_d()
isupper_d()
tolower_d()
toupper_d()


time Funktionen

strftime_d()
asctime_d()
ctime_d()
```

1.103 beispiele

```
Beispiele:
~~~~~~~~~~

Um die Vorteile und Anwendungsmöglichkeiten der "stormamiga.lib"
etwas zu verdeutlichen, habe ich ein paar Beispiele beigelegt.

Die Beispiele mit dem Namen "...-storm" werden mit der "storm.lib"
und dem Startupcode "startup.o" gelinkt.

Die Beispiele mit dem Namen "...-stormamiga" werden mit der
"stormamiga.lib" und dem Startupcode "stormamiga_startups.o" gelinkt.

Die Beispiele mit dem Namen "...-stormamiga-2" werden mit der
"stormamiga.lib" und dem Startupcode "stormamiga_startups.o" gelinkt.
Außerdem wurde der Quelltext für die "stormamiga.lib" optimiert.
```

1.104 bekannte fehler

```
Bekannte Fehler:
~~~~~~~~~~~~~~~~

- K E I N E
```

1.105 updates

```
                Updates:
~~~~~~~~
```

Wenn Sie Zugang zum Internet haben, dann können Sie die Updates auf
der HomePage "http://home.pages.de/~haage" der Haage & Partner
Computer GmbH bekommen.
Wer ein CD-Laufwerk besitzt, kann die Updates auf den CDs der "AMIGA plus"
und vom "AMIGA MAGAZIN" finden.
Sie können die neuste Version natürlich auch direkt von
mir
bekommen.
Allerdings müssen Sie mir dann einen frankierten Briefumschlag und
eine Diskette (HD oder DD) zuschicken.

1.106 kopierrecht

Kopierrecht:
~~~~~~~~~~~~

Die "stormamiga.lib" ist FREEWARE. Sie darf frei kopiert werden, solange
sie in KEINSTER Weise verändert wird und solange ALLE dazugehörigen
Dateien UNVERÄNDERT mitkopiert werden.

Die "stormamiga.lib" darf auch im Zusammenhang mit anderen Programmen
verwendet und vertrieben werden, solange KLARGESTELLT ist, daß es sich um
FREEWARE handelt UND solange ALLE Dateien UNVERÄNDERT mitkopiert werden.
Mit der Weitergabe der "stormamiga.lib" darf kein Gewinn erzielt werden.
Der Verkaufspreis einer Diskette, die die "stormamiga.lib" enthält, darf
nicht mehr als 5,- DM betragen. Ausgenommen davon sind Disketten, die es
zu Zeitschriften gibt.

Eine Reassemblierung der "stormamiga.lib" ist selbstverständlich NICHT
gestattet.

AM WICHTIGSTEN:

Die Benutzung der "stormamiga.lib" erfolgt AUSSCHLIEßLICH auf eigenes Risiko.

Der Autor kann auf KEINEN FALL für einen Schaden oder Datenverlust der direkt
oder indirekt mit dem Gebrauch der "stormamiga.lib" entstehen sollte verant-
wortlich gemacht werden.

Alle Rechte vorbehalten. Für Fehlermitteilungen oder Verbesserungsvorschläge
bin ich jederzeit dankbar.

1.107 geschichte

Geschichte:
~~~~~~~~~~~

V41.000 alpha - V41.031 beta

```
                V41.032 beta  - V41.035
               V42.00 (14.06. - 27.10.1996):
------------------------------
- die ctype-Funktionen "which_xdigit" "isalnum_d", "isalpha_d",
  "islower_d", "isupper_d", "tolower_d" und "toupper_d"  geschrieben
- die ctype-Funktionen "isalnum", "isalpha", "iscntrl", "isdigit",
  "isgraph", "islower", "isprint", "ispunct", isspace", "isupper",
  "isxdigit", "tolower" und "toupper" neu geschrieben
- die string-Funktionen "strnicmp", "strcasecmp", "strncasecmp",
  "stricmp_d", "strnicmp_d", "strcasecmp_d", "strncasecmp_d",
  "strlower", "strupper", "strlower_d", "strupper_d", "strlwr_d",
  "strupr_d" und "strdup" geschrieben
- die string-Funktionen "strcoll", "strxfrm", "memcpy", "strcpy",
  "strcmp", "strncmp" und "stricmp" neu geschrieben
- die stdlib-Funktionen "abort", "atexit", "getenv", "system", "atof"
  und "strtod" geschrieben
- die stdlib-Funktionen "bsearch" und "qsort" der "storm.lib" für den
  MC68EC020+ optimiert
- die stdio-Funktionen "setbuf", "setbuffer", "setlinebuf", "perror",
  "remove", "rename", "fopen", "freopen", "fclose", "tmpnam" und
  "tmpfile" geschrieben
- die stdio-Funktionen "vfprintf" und "vfprintf_" (mit den Routinen
  zur formatierten Ausgabe) neu geschrieben
- die stdio-Funktionen "setvbuf", "vprintf", "vprintf_", "vscanf",
  "vscanf_", "vsscanf" und  "vsscanf_" optimiert
- die time-Funktionen "ctime", "asctime", "localtime", "gmtime",
  "strftime", "ctime_d", "asctime_d", "strftime_d", "difftime" und
  "mktime" geschrieben
- die time-Funktionen "clock" und "time" neu geschrieben
- die signal-Funktionen "raise" und "signal" geschrieben
- die setjmp-Funktionen "longjmp" und "setjmp" der "storm.lib"
  integriert
- die math-Funktionen "atan2", "fmod", "frexp", "ldexp" und "modf"
  geschrieben
- die amiga-Funktionen "ArgArrayInit", "ArgArrayDone", "ArgInt" und
  "ArgString" geschrieben
- die Spezial-Funktionen "divs64", "divu64", "button", "button_al",
  "button_ar", "button_bl", "button_br", "waitbutton", "waitbutton_al",
  "waitbutton_ar", "waitbutton_bl", "waitbutton_br", "muls_r", "mulu_r",
  "divsl_r", "divul_r", "muls64_r", "mulu64_r", "divs64_r", "divu64_r",
  "button_r" und "waitbutton_r" geschrieben
- die internen Funktionen "INIT_0_NEAR_CODE_StdioFiles", "intern__time_",
  "userbreak", "intern__signal_", "__ctypetable", "_do_assert" und
  "_do_assert_" geschrieben
- die internen Funktionen "intern__ctype_", "intern__form_in" und
  "intern__form_in_" neu geschrieben
- die internen Funktionen "UMult64", "INIT_5_InitStdIOFiles", "EXIT_5_Init-
  StdIOFiles", "amigaread", "amigawrite", "amigaputc", "amigaflush",
  "amigaclose", "amigaseek", "char_in", "string_in", "int_in" und "double_in"
  optimiert
- die interne Funktion "_floattostr" der "storm.lib" für den MC68EC020+
  optimiert
- die internen Funktionen "blocksize_a2_d2", "lib_destruct", "lib_throw",
  "lib_destruct_a0", "lib_rethrow", "lib_catch", "set_unexpected__PFvp",
  "terminate_", "set_terminate__PFvp", "unexpected_", "lib_catchclass",
  "LibInit", "LibOpen", "LibClose", "LibExpunge", "LibNull", "_fpwr10" und
  alle internen mathe-Funktionen der "storm.lib" integriert
```

```
- Funktionen zum automatischen Öffnen und Schließen der "realtime.library",
  "reqtools.library" und "muimaster.library" geschrieben
- Fehlerkorrektur der stdlib-Funktion "free"
- Unterstützung des kleinen Datenmodelles a4 und des kleinen Codemodelles
  integriert
- Startupcodes für Ansi-C und C++ (für das kleine und große Codemodell
  geschrieben
- Inline-Funktionen geschrieben
- Includedatei "stormamiga.h" erweitert
- Benutzer-Lexikon erweitert
- Installerskript überarbeitet und erweitert
- Anleitung überarbeitet und erweitert (Beschreibung der neuen Funktionen
  hinzugefügt)


V42.01 (29.10. - 01.11.1996):
-----------------------------
- Fehlerkorrektur der stdlib-Funktionen "malloc" und "free" (Im großen Daten-
  modell fehlte die Kennung "FAR". Dadurch wurden diese Funktionen auch
  für das kleine Datenmodell verwendet.)
- Fehlerkorrektur der stdio-Funktionen "printf" und "printf_"
- Fehlerkorrektur der internen Funktion "amigawrite"
```

1.108 Geschichte

```
V41.000 alpha - V41.002 alpha (18.03. - 21.03.1996):
----------------------------------------------------
  interne Entwicklungsphase

- Funktionen der "amiga.lib" für MC68000 oder höher geschrieben


V41.003 alpha - V41.021 alpha (22.03. - 16.05.1996):
----------------------------------------------------
  interne Entwicklungsphase

- Funktionen der "amiga.lib" für MC68EC020 oder höher umgeschrieben (ab
  V41.003 alpha wird ein MC68EC020 oder höher benötigt)
- Ein- und Ausgaberoutinen geschrieben (GCC-kompatibel)
- Funktionen zum automatischen Öffnen und Schließen der Libraries ge-
  schrieben
- die meisten ctype-Funktionen geschrieben
- einige stdio-, string- und stdlib-Funktionen geschrieben
- Startupcode für die "stormamiga.lib" optimiert
- einige andere Funktionen geschrieben
- einige Optimierungen und Fehlerkorrekturen
- Includedatei "stormamiga.h" geschrieben


V41.022 alpha - V41.029 alpha (20.05. - 11.06.1996):
----------------------------------------------------
  interne Entwicklungsphase

- Ein- und Ausgaberoutinen komplett neu geschrieben (StormC-kompatibel)
```

```
- einige stdio-Funktionen geschrieben
- 64Bit Befehle der "storm.lib" für MC68EC020+ optimiert
- einige andere Funktionen geschrieben
- einige Optimierungen und Fehlerkorrekturen
- Includedatei "stormamiga.h" erweitert
- Anleitung geschrieben


Hinweis:

Durch ein Versehen wurde die Version 41.028 alpha, einige uralte Beispiel-
programme und Teile der Anleitung, mit der Version 1.1 von StormC, veröffent-
licht. Diese Version hat aber noch einige größere Fehler und funktioniert
nicht mit den alten Beispielprogrammen.
Die erste, zur Veröffentlichung gedachte Version, ist die Version 41.032 beta.


V41.030 beta - V41.031 beta (13.06. - 15.06.1996):
--------------------------------------------------
  interne Entwicklungs- und Testphase

- Optimierung der Ein- und Ausgaberoutinen
- Fehlerkorrekturen
- Anleitung überarbeitet
```

1.109 Geschichte

```
V41.032 beta (16.06.1996):
--------------------------
  Erste öffentliche Version


V41.033 beta (17.06. - 15.07.1996):
-----------------------------------
- umfangreiches Betatesting
- Fehlerkorrektur der string-Funktion "memcpy" (Die Benutzung von "memcpy"
  führte zu einem Fehler oder zum Systemabsturz. Die Ursache war ein Tipp-
  fehler. Ich hatte "a2" statt "a1" geschrieben.)
- die mathematischen Funktionen "acos", "asin", "atan", "ceil", "cos",
  "cosh", "exp", "fabs", "floor", "log", "log10", "pow", "sin", "sinh",
  "sqrt", "tan" und "tanh" geschrieben
- die string-Funktionen "memccpy", "strcoll", "strlwr", "strsep", "strupr",
  "strxfrm" und "swab" geschrieben
- die stdlib-Funktionen "abs", "labs", "atoi" und "atol" geschrieben
- die stdlib-Funktion "rand" optimiert
- die string-Funktionen "memchr", "memmove", "memset", "strcspn", "strpbrk",
  "strspn" und "strtok" optimiert
- Includedatei "stormamiga.h" überarbeitet und erweitert


V41.034 (16.07. - 31.07.1996):
------------------------------
- Fehlerkorrektur der internen Funktion "amigawrite" (Der auszugebende Text
  wurde erst nach einigen Sekunden angezeigt. Bei dem Programm "GadTools"
  wurde der Text erst ausgegeben, wenn 5 Schalter gedrückt wurden.)
```

```
- die string-Funktionen "strerror" und "stricmp" geschrieben
- die string-Funktionen "memcmp", "strcat", "strcmp", "strcpy", "strncat",
  "strncmp", "strncpy" und "strstr" optimiert
- die stdlib-Funktionen "malloc" und "free" optimiert
- die stdio-Funktionen "vfprintf", "vfprintf_", "vfprintf__", "vfscanf",
  "vfscanf_", "vfscanf__", "fflush" und "setvbuf" optimiert
- die internen Funktionen "amigaread", "amigareadunget", "amigawrite", "amiga-
  eof", "amigaseek", "amigagetc", "amigagetcunget", "amigaungetc", "amigaputc",
  "amigaflush", "amigaclose", "SMult64", "UMult64", "SDiv64", "INIT_0_InitFiles",
  "EXIT_5_InitFiles", "INIT_5_InitStdIOFiles", "EXIT_5_InitStdIOFiles" und
  "EXIT_4_free" optimiert
- "aufräumen" der "stormamiga.lib" (dadurch wird das Linken beschleunigt,
  die Programme etwas schneller und die Map-Datei übersichtlicher)
- Anleitung überarbeitet


V41.035 (02.08. - 17.08.1996):
------------------------------
- die string-Funktionen "strcmp", "strcoll" und "strxfrm" neu geschrieben
- die stdlib-Funktionen "llabs",  "atoll", "strtol", "strtoll", "strtoul",
  "strtoull", "inttostr", "llongtostr", "uinttostr" und "ullongtostr"
  geschrieben
- die stdio-Funktion "puts" geschrieben
- die Spezial-Funktionen "muls64" und "mulu64" geschrieben
- Includedatei "stormamiga.h" erweitert
- Benutzer-Lexikon mit allen Sonder-Funktionen der "stormamiga.lib" geschrieben
- Installerskript geschrieben
- Anleitung überarbeitet (Beschreibung der Funktionen neu geschrieben, Index
  hinzugefügt und an AmigaOS 3.0 angepaßt); Anleitung als ASCII-Text beigelegt
```

1.110 In Zukunft

```
In Zukunft:
~~~~~~~~~~~

Die folgenden Punkte habe ich mir für die nächsten Versionen der
"stormamiga.lib" vorgenommen.

- alle, noch fehlenden, Funktionen der "storm.lib" und der "amiga.lib"
  in die "stormamiga.lib" integrieren

- spezielle Version der "stormamiga.lib" für die Koprozessoren
  MC68881 und MC68882 (mit "68040.library" bzw. "68060.library"
  auch für den MC68040 und MC68060)

- spezielle Version der "stormamiga.lib" für den MC68040 (mit
  "68060.library" auch für den MC68060)

- Unterstützung des kleinen Datenmodelles a6 (wenn Interesse besteht)
```

1.111 danksagungen

Danksagungen:
~~~~~~~~~~~~~

Als erstes möchte ich mich bei der Haage & Partner Computer GmbH
bedanken, weil sie mir ihre Entwicklerunterlagen und Quelltexte
kostenlos überlassen haben. Ohne diese Unterlagen währe ich wahr-
scheinlich an den Ein- und Ausgaberoutinen für die Befehle printf
und scanf verzweifelt.
Besonderen Dank an Jochen Becher, der auch an Sonntagen zu später
Stunde Zeit für meine Probleme hatte.

Ohne die Unterstützung der Haage & Partner Computer GmbH würde
es die "stormamiga.lib", in dieser Form, nicht geben.

Außerdem möchte ich mich bei folgenden Leuten bedanken:

- Uwe Schienbein, für Betatesting, Bugreports und neue Ideen
- Carsten Bornholz, für Betatesting
- Dietmar Heidrich, für seinen "OMA"
- Frank Wille, für seinen "FreePhxAss"

1.112 autor

Autor:
~~~~~~

Matthias Henze
Gorkistraße 119
04347 Leipzig
Deutschland

Telefon: 0341/2326414

Für Fehlerberichte und Verbesserungsvorschläge bin ich jederzeit
dankbar. Es wäre auch sehr schön, wenn Sie mir Ihre Meinung zur
"stormamiga.lib" mitteilen würden.

An alle Anwender, die für ein "Dankeschön" arbeiten

Ich suche dringend einige Anwender, die die Anleitung und den Installer-
script in andere Sprachen (Englisch, Italienisch, Französisch und andere)
übersetzen.
Außerdem brauche ich noch einige Betatester.
Wenn Sie Interesse haben, können Sie mir schreiben oder mich anrufen.

Für Ihre Mühe danke ich im Voraus.

1.113 Index

Index:
~~~~~~
~~~~~~

```
main__()

memccpy

muls

muls_r

muls64

muls64_r

mulu

mulu_r

mulu64

mulu64_r
P


Parameterliste

printf_
R


Registerfunktionen

Registrierung

rindex
S


scanf_

setbuffer

setlinebuf

SPRINTF

sprintf_

sscanf_

strcasecmp

strcasecmp_d

strcoll

strdup
```

strftime

strftime_d

stricmp_d

Stringpuffer

strlower

strlower_d

strlwr_d

strncasecmp

strncasecmp_d

strnicmp

strnicmp_d

strsep

strupper

strupper_d

strupr_d

strxfrm

swab

Systemanforderungen
T

tolower_d

toupper_d
U

Updates
V

vfprintf_

vfscanf

vfscanf_

vprintf_

vscanf

vscanf_

VSPRINTF

vsprintf_

vsscanf

vsscanf_
W

waitbutton

waitbutton_r

waitbutton_al

waitbutton_ar

waitbutton_bl

waitbutton_br
Z

Zeitformatstring