

# **GNU C/C++**

Christian Felsch

**COLLABORATORS**

	<i>TITLE :</i> GNU C/C++		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Christian Felsch	January 16, 2023	

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>GNU C/C++</b>	<b>1</b>
1.1	titel . . . . .	1
1.2	vorwort . . . . .	1
1.3	installation . . . . .	2
1.4	compiler . . . . .	3
1.5	tools . . . . .	3
1.6	fixstk.ttp . . . . .	4
1.7	nm.ttp . . . . .	4
1.8	strip.ttp . . . . .	6
1.9	toglclr.ttp . . . . .	6
1.10	bibliotheken . . . . .	6
1.11	dokumentation . . . . .	7
1.12	installation, minix . . . . .	8
1.13	installation, tos/magic . . . . .	9
1.14	installation, prfen . . . . .	11
1.15	installation, archive . . . . .	12
1.16	tips+tricks . . . . .	12
1.17	stacksize . . . . .	13
1.18	-pipe . . . . .	14
1.19	debugger . . . . .	14
1.20	nderungen . . . . .	15

---

# Chapter 1

## GNU C/C++

### 1.1 titel

Titel  
GNU C/C++

↔

GNU C/C++ auf dem Atari

von

Christian Felsch

Inhalt:

Vorwort

Installation

Tips & Tricks zu GNU C auf dem Atari  
nderungen

### 1.2 vorwort

Vorwort

GNU C/C++

Da ich im MausNet immer mal wieder Anfragen dem GNU-Compiler und zu dessen Installation lese, ist es wohl Zeit, mal ein wenig darber aufzuschreiben.

Dieser Hypertext soll bei der Installation des GNU Compilers auf dem Atari helfen. Er ist keine generelle Anleitung zum Thema GNU C, es wird nur auf die Atari-spezifischen Dinge eingegangen. Wer mehr ber die Benutzung von GNU C erfahren will, den verweise ich auf die entsprechenden Dokumentationen.

Durch die Herkunft des GNU Compilers aus der UNIX-Welt sind für dessen Installation einige Schritte nötig, die man sonst vom Atari her nicht kennt. Unter MiNT mit einem MinixFS ist die Installation vergleichbar mit der unter UNIX.

Immer mehr User wollen den Compiler aber unter anderen Systemen (z.B. Magic) benutzen wollen und da dort vieles der UNIX-Umgebung fehlt, sind bei der Installation des Compilers einige Dinge zu beachten.

Im Vorfeld zu dieser Dokumentation habe ich den Compiler in der aktuellen Version 2.8.1 auf meinem TT selbst übersetzt. Wichtigster Grund dafür war folgender:

Die letzten Versionen (ab 2.6) wurden von verschiedenen Leuten für den Atari portiert, allerdings immer mit der Einschränkung, da MiNT mit einem MinixFS vorausgesetzt wurde.

Die Version 2.5.8 war die letzte, die unter allen Systemen lief und sich auch auf dem normalen TOS-Dateisystem installieren lie.

Ich habe die wesentlichen Portierungen (2.5.8 und 2.7.2) genommen und so in die aktuellen Quellen integriert, da am Ende wieder ein Compiler heraus kam, der wieder auf allen Systemen benutzbar ist.

Alle in diesem Dokument benutzen Dateinamen (der Programme, Archive usw.) beziehen sich auf das von mir zusammengestellte Paket, da im InterNet zu finden ist:

`ftp://ftp.rz.tu-harburg.de/pub/software/systems/atari/gnu/gcc`

## 1.3 installation

Die Installation  
GNU C/C++

↔

Bevor man sich an die Installation macht, muss sichergestellt sein, dass alle benötigten Komponenten des Compilers verfügbar sind:

Compiler

Tools

Bibliotheken

Dokumentation

übersicht der Archive

Die Installation lässt sich in zwei Teile aufteilen:

Installation auf einem Minix-FS

Installation auf einem TOS-FS

überprüfen der Installation

## 1.4 compiler

Komponenten: der Compiler

GNU C/C++

Der eigentliche Compiler besteht aus fnf einzelnen Programmen:

gcc.ttp	Treiberprogramm, welches vom Anwender gestartet wird und nach einander alle ntigen Programme aufruft. Mit ihm werden C Quelltexte bersetzt.
cpp.ttp	Der C/C++ Prprozessor. Wird von gcc.ttp gestartet, so da der Anweder im Normalfall nichts davon sieht.
ccl.ttp	Der eigentlche C-Compiler. Wird ebenfalls von gcc.ttp gestartet.
g++.ttp	Treiberprogramm fr C++ Quelltexte. Es startet gcc.ttp mit einigen C++ spezifischen Parametern.
cclplus.ttp	Der eigentliche C++ Compiler.

## 1.5 tools

Komponenten: die Tools

GNU C/C++

Neben dem Compiler bentigt man noch einige andere Tools. Zwei sind unverzichtbar:

as.ttp	Der Assembler, der aus den Assemblerdateien die Objektdateien erzeugt.
ld.ttp	Der Linker, der aus Objektdateien und Bibiliotheken das ausfhrbare Programm erzeugt.

Neben Assembler und Linker gibt es noch diverse andere Tools, die nicht unbedingt erforderlich sind, aber zustzliche Funktionen zur Verfgung stellen:

ar.ttp	Der Archivator. Er wird bentigt, wenn man mehrere Objektdateien zu einer Bibliothek zusammenfassen mchte.
nm.ttp	Zeigt die Symbol-Namen an, die in einem ausfhrbaren Programm enthalten sind.
fixstk.ttp	Stellt die Stacksize eines Programms ein. Nheres dazu unter Tips+Tricks.
nm.ttp	Zeigt die Symbol-Name in Objektdateien oder Bibliotheken

	an. Damit kann man z.B. nach bestimmten Funktionen in einer Bibliothek suchen.
printstk.ttp	Zeigt die eingestellte Stacksize an. Nheres dazu unter Tips+Tricks.
size.ttp	Zeigt die Gre der Programmbereiche (Text, Data, Bss) in Objektdateien oder Bibliotheken an.
size68.ttp	Zeigt Informationen ber ein aufrhrbares Programm an.
strings.ttp	Zeigt alle 'lesbaren' Zeichenketten (also ASCII-Werte zwischen 32 und 127) an. Damit kann man z.B. fest in ein programm eingebaute Pfade finden.
strip.ttp	Entfernt die Symboltabelle aus einem Programm.
sym-ld.ttp	Symbolischer Linker, wird zum bentigt.
toglclr.ttp	Dient zum Anzeigen bzw. Einstellen der 'Programm-Header-Bits'.

## 1.6 fixstk.ttp

fixstk - einstellen GNU C/C++

Mit dem Programm fixstk.ttp kann die eines Programms verndert werden.

Aufruf: `fixstk size file ...`

Der Wert von <size> gibt dabei die Anzahl der Bytes an. Einige Werte haben eine besondere Funktion, nheres dazu unter Stacksize. Bei der Angabe der Gre kann eine Abkrzung angegeben werden. Dabei steht ein 'k' hinter der Zahl fr Kilobyte und ein 'm' fr Megabyte. Gltige Werte fr <size> sind also z.B. 4096, 4k oder 2m. Das Programm zeigt nach dem Aufruf die vorher eingestellte und die neue an:

```
/tmp > fixstk 4k test.app
test.app: _stksize was 0 (0K)
test.app: _stksize now is 4096 (4K)
```

Sollte fixstk einen Fehler der Art 'no symbol table' melden, kann es das Symbol \_stksize nicht mehr finden (es wurde wegge'stripped!) und man kann nichts mehr verndern.

## 1.7 nm.ttp

nm.ttp - Symbole anzeigen GNU C/C++

Das Programm listet die in einer Objektdatei oder Bibliothek enthaltenen

Symbole auf.

Aufruf: `nm [option] file`

Folgende Optionen stehen zur Verfügung:

- a Wenn die Objektdatei mit `-g` erzeugt wurde, werden spezielle Debug-Infos angezeigt.
- g Nur globale Symbole listen.
- n Nach Adressen (erste Spalte) sortiert, sonst alphabetisch.
- o Dateiname der Objektdatei mit ausgeben. Sinnvoll z.B. bei Bibliotheken.
- r Ausgabe umgekehrt sortiert.
- u Nur undefinierte Symbole, also solche die nicht in der Lib deklariert sind, anzeigen.

Die Ausgabe von `nm` sieht so aus:

```
00000000 t __gnu_compiled_c
          U __mint
          U __open_stat
00000684 d __umask
          U __enoent
00000000 t __get_umask
          U __unix2dos
00000630 T _creat
          U _errno
          U _fstat
          U _isatty
00000046 T _open
          U _strncmp
          U _strtoul
0000064a T _umask
00000000 t gcc2_compiled.
```

In der ersten Spalte wird die relative Adresse ausgegeben. Der einzelne Buchstabe klassifiziert den Typ des Symbols. Folgende Typen gibt es:

- C Eine Variable, die definiert aber nicht initialisiert ist.
  - b Eine lokal (static) definierte Variable.
  - D Eine Variable, die bei ihrer Definition initialisiert wird.
  - d Eine lokal (static) definierte Variable mit Initialisierung.
  - t,  
T Funktionsnamen. 't' bei static sonst 'T'.
  - U Undefinierte (externe) Symbole.
-

## 1.8 strip.ttp

strip.ttp - Symboletabelle lschen GNU C/C++

Mit dem Programm kann die Symboltabelle eines Programms entfernt werden. Das Programm belegt dadurch weniger Platz auf der Platte.

Aufruf: strip [option] file ...

Folgende Optionen stehen zur Verfügung:

- a Symboltabelle komplet lschen. Sollte man nicht verwenden, da dann die Stacksize nicht mehr einstellbar ist.
- g Alle Symbole bis auf die globalen lschen.
- k Erhlt das Symbol fr die Stacksize (\_stksize).
- l names Alles Symbole bis auf die angegebenen lschen.
- t TurboC/PureC-Programm stripfen.

## 1.9 toglclr.ttp

toglclr.ttp - Programheader manipulieren GNU C/C++

Mit dem Programm knnen die Bits im Programheader manipuliert werden.

Aufruf: toglcr [option] file ...

Folgende Optionen stehen zur Verfügung:

- fload Fast-Load Bit.
- frun Programm in TT-RAM legen.
- fram Programm bekommt TT-RAM.
- fshare Shared-Text Bit.
- private MiNT MemoryProtection: Speicher privat.
- global MiNT MemoryProtection: Speicher frei fr alle.
- super MiNT MemoryProtection: Speicher frei fr alle im Supermode.
- readable MiNT MemoryProtection: Speicher lesbar fr alle.

## 1.10 bibliotheken

Komponenten: die Bibliotheken GNU C/C++

---

Zu jedem C-Compiler gehen verschiedene Bibliotheken. Auf jedenfall bentigt wird die Standard C Bibliothek. Fr den Atari gibt es zwei verschiedene Versionen dieser Standardbibliothek. Zum einen ist das die GNU-Lib in der Version PL 98, die schon relativ alt ist und nicht mehr weiterentwickelt wird. Zum anderen ist da die sogenannte MiNT-Lib, die stmliche MiNT-Funktionen untersttzt und inzwischen auch wieder weiterentwickelt wird. Ich werde mich hier auf die Beschreibung der MiNT-Lib als Standard-bibliothek beschrnken.

Einige Bibliotheken liegen in verschiedenen Ausfhrungen vor. Wenn ein Bibliotheksname auf '16' endet, bedeutet das, da der verwendete Integertyp nicht wie blich bei GNU 32bit sondern nur 16bit gro ist. Es gibt einige Programme, die sich nur mit 16bit Integer bersetzen lassen (bei PureC ist 16bit Standard), die dann diese Bibliotheken bentigen. Eine Mischung von 16bit und 32bit Objektdateien und Bibliotheken frht zu nicht funktionstchtigen Programmen!

Auerdem gibt es noch Bibliotheken, deren Name mit einem 'b' beginnen. Dies sind Bibliotheken, die das Text-Sharing (ein mehrfach gestartetes Programm belegt mit seinem Textsegment nur einmal RAM) von MiNT untersttzen.

Folgende Bibliotheken sind verfgbar:

crt.o	Der Startupcode, der vom Linker zu jedem Programm hinzugelinkt wird.
gnu.olb	
gnu16.olb	Die Standard C Bibliothek (MiNT-Lib).
gem.olb	
gem16.olb	Die Schnittstelle zum AES/VDI
iiio.olb	
iiio16.olb	Enthlt Funktionen fr Integer-IO.
pml.olb	
pml16.olb	Mathematische Standardfunktionen.
stdc++.olb	
g++.olb	Die beiden Standard-Bibliotheken fr C++.

Neben den genannten gibt es noch eine Reihe weiterer Bibliotheken, die nicht zu meinem GNU-Paket gehen aber wenigstens genannt werden sollen:

- curses
- portlib
- socket
- gem++

## 1.11 dokumentation

Zum GNU Compiler gibt es jede Menge Dokumentation, die in den Quellen des Compiler zu finden ist.  
 In meinem Doku-Archiv sind neben diesem Hypertext noch ein paar weitere Dokumentationen, die sich speziell mit GNU C auf dem Atari beschäftigen (alle auf Englisch):

gcc-st.tex	Die ursprüngliche Doku von Frank Riddebusch. Zu groen Teilen ist sie noch gltig.
lib-card.tex	Eine Doku zur MiNT-Lib, als RefCard zusammengefasst. Sie bietet einen guten berblick ber die Lib, ist aber leider nicht vollstndig.
var-card.tex	Eine bersicht der Betriebssystem-Struturen, die in der MiNT-Lib benutzt und definiert sind.

Fr alle, die kein TeX haben, sind die Dokus auch als Postscript verfgbar. Und wer auch kein Ghostscript zum Anschauen/Drucken hat, hat selber schuld ;-)

## 1.12 installation, minix

Installation auf einem MinixFS

GNU C/C++

Die Installation unter MiNT auf einem MinxFS stellt sich genau so dar, wie GNU C es von -Systemen her gewhnt ist.

Da ich nicht zwei verschiedene Archive erstellen wollte, mssen alle, die den Compiler in dieser Art installieren woll, die Dateien in die UNIX-Namen umbenennen :-)

Bei der unten angegebenen Verzeichnisstruktur wird der jeweilige Name im Archiv in () angegeben.

```

/usr/lib/gcc-lib/m68k-atari-mint/2.8.1
  Enthlt den eigentlichen Compiler:
    ccl          (bin/ccl.ttp)
    cclplus     (bin/cclplus.ttp)
    cpp         (bin/cpp.ttp)
    specs      (lib/specs)

/usr/m68k-atari-mint/bin
  Enthlt die Programme, die vom gcc gestartet werden:
    as          (bin/as.ttp)
    ld          (bin/ld.ttp)
    sym-ld     (bin/ld.ttp)

/usr/bin
  Alle Programme, die der User benutzen will, mssen so
  installiert sein, das sie gefunden werden. Das Verzeichnis
  mu in der Environmentvariable PATH enthalten sein.
    gcc        (bin/gcc.ttp)
    g++       (bin/g++.ttp)
    ar        (bin/ar.ttp)
    ...

/usr/lib
```

```

/usr/local/lib
/usr/m68k-atari-mint/lib
/usr/lib/gcc-lib/m68k-atari-mint/2.8.1/lib
  Hier sucht der Linker nach den Bibliotheken.
  Dabei geht er von den UNIX-Namen aus, die mit 'lib' beginnen
  und auf '.a' enden.
    libgnu.a      (lib/gnu.olb)
    libgem.a     (lib/gem.olb)
    crt0.o       (lib/crt0.o)
    ...
  Achtung: Der Linker entscheidet anhand des Vorhandenseins
  der Environmentvariable GNULIB, ob er die UNIX- oder die TOS-
  Namen verwendet. Wer also die Minix-Installation verwendet,
  sollte sicherstellen, die diese Variable nicht gesetzt ist!

```

```

/usr/local/include
/usr/m68k-atari-mint/include
/usr/lib/gcc-lib/m68k-atari-mint/2.8.1/include
/usr/include
/usr/lib/g++-include
  Hier sucht der Preprozessor nach den Header-Dateien.
  Der letzte Pfad wird nur bei C++ benutzt.

```

Für den Compiler selbst sind keine speziell Environmentvariablen nötig.  
Er beachtet aber folgende Standardvariablen:

```

TMPDIR
TEMP
TMP      Der temporäre Bereich, in dem einige Dateien zwischen-
         gelagert werden.

PATH     Wer eine Kommandoshell benutzt, sollte den Pfad, wo die
         ausführbaren Programme (*.ttp) der Compilers liegen,
         darin mit angeben.

```

## 1.13 installation, tos/magic

Installation auf einem TosFS

GNU C/C++

Wenn der Compiler nicht wie in der UNIX-Welt blich installiert wird, weil z.B. kein Minix-Dateisystem zur Verfügung steht oder man den Compiler mit einer anderen Struktur installieren möchte, können die benutzten Suchpfade von außen eingestellt werden. Diese Möglichkeit ist für alle diejenigen interessant, die den Compiler unter normalem TOS oder aber Magic benutzen möchten. Durch das VFAT-System und dem Vorhandensein des Laufwerks U: unter Magic ist es vielleicht möglich, die UNIX-Struktur nachzubilden, ich möchte hier aber keine besonderen Möglichkeiten von Magic bzw. VFAT ausnutzen und nur den allgemeinen Weg der Installation beschreiben.

Im Gegensatz zur Installation auf einem MinixFS können die Dateien so installiert werden, wie sie in den Archiven enthalten sind, brauchen also nicht umbenannt werden. Auch die in einigen Archiven längeren Namen, die auf einem TOSFS auf 8.3 verkürzt werden, sollten kein Problem darstellen.

Als zweckm ig hat sich die folgende Struktur auf einer TOS-Partition X: erwiesen:

X:\gnu\bin

Hier werden alle ausfhrbaren Programme (\*.ttp) abgelegt.

X:\gnu\lib

Hier liegen alle Bibliotheken (\*.olb, \*.o) und, ganz wichtig, die Datei 'specs'.

X:\gnu\include

X:\gnu\include\g++

Hier liegen alle Header-Dateien (C und C++).

Um den Compiler mitzuteilen, wo er die Dateien findet, gibt es vier Environment-Variablen. Die Pfade, die in diese Variablen eingetragen werden, drfen sowohl '\' als auch '/' als Verzeichnistrenner enthalten. Sollen mehrere Pfade in einer Variablen eingetragen werden, sind diese durch ',' zu trennen.

**GNUEXEC** ber diese Variable wird dem gcc.ttp mitgeteilt, wo Compiler, Assembler und Linker liegen. Der eingestellte Pfad mu mit einem '/' bzw. '\' enden, da er den einzelnen Programmen einfach nur vorangestellt wird und dabei ein gltiger Pfad entstehen mu.

Im Beispiel:

GNUEXEC=X:/gnu/bin/

**GNULIB** Auf dem in dieser Variablen eingestellten Pfad werden die Bibliotheken und die 'specs'-Datei gesucht. Hier ist der abschließende Verzeichnistrenner nicht ntig.

Im Beispiel:

GNULIB=X:/gnu/lib

**GNUINC** Auf dem in dieser Variablen eingestellten Pfad werden die Standard C Header-Dateien gesucht.

Im Beispiel:

GNUINC=X:/gnu/include

**GXXINC** Wer C++ machen mchte, mu in dieser Variablen den Pfad zu den C++ Header eintragen. Auerdem mu zustzlich noch der Pfad zu den Standard C Headern eingetragen werden.

Im Beispiel:

GXXINC=X:/gnu/include/g++,X:/gnu/include

Neben diesen absolut notwendigen Variablen, beachtet der Compiler noch folgende andere Variablen:

**TMPDIR**

**TEMP**

**TMP** Der temporre Bereich, in dem einige Dateien zwischen- gelagert werden.

**PATH**

Wer eine Kommandoshell benutzt, sollte den Pfad, wo die ausfhrbaren Programme (\*.ttp) der Compilers liegen, darin mit angeben.

## 1.14 installation, prfen

berprfung der Installation

GNU C/C++

Eine berprfung der Konfiguration kann durch die Option '-v' beim Aufruf von gcc.tpp erfolgen.

Wenn man ein C-Testprogramm mit 'gcc -v test.c' bersetzt und die Installation erfolgreich war, gibt es folgende Ausgaben:

installiert auf MinixFS

```
/tmp > gcc -v test.c
Reading specs from /usr/lib/gcc-lib/m68k-atari-mint/2.8.1/specs
gcc version 2.8.1
/usr/lib/gcc-lib/m68k-atari-mint/2.8.1/cpp -lang-c -v -undef
-D__GNUC__=2 -D__GNUC_MINOR__=8 -Dmc68000 -Datarist -Dgem -D__mc68000__
-D__atarist__ -D__gem__ -D__mc68000 -D__atarist -D__gem -Asystem(tos)
-Asystem(gem) -Acpu(m68k) -Amachine(m68k) test.c /tmp/cc482000.i
GNU CPP version 2.8.1 (68k, MIT syntax)
#include "... " search starts here:
#include <...> search starts here:
/usr/include
End of search list.
/usr/lib/gcc-lib/m68k-atari-mint/2.8.1/cc1 /tmp/cc482000.i -quiet
-dumpbase test.c -version -o /tmp/cc482000.s
GNU C version 2.8.1 (m68k-atari-mint) compiled by GNU C version 2.8.1.
/usr/m68k-atari-mint/bin/as -v -mc68000 -o /tmp/cc4820001.o /tmp/cc482000.s
GNU assembler version 2.9 (m68k-atari-mint)
/usr/m68k-atari-mint/bin/ld -v /usr/lib/crt0.o -L/usr/lib/gcc-lib/
m68k-atari-mint/2.8.1/tmp/cc4820001.o -lgnu
/usr/m68k-atari-mint/bin/ld Patchlevel 40 (AtariST) for GNU
```

installiert auf TosFS

```
i:\ > gcc -v test.c
Reading specs from g:/gnu/lib/specs
gcc version 2.8.1
g:/gnu/bin/cpp.ttp -lang-c -v -iprefix g:/gnu/bin/m68k-atari-mint/2.8.1/
-undef -D__GNUC__=2 -D__GNUC_MINOR__=8 -Dmc68000 -Datarist -Dgem
-D__mc68000__ -D__atarist__ -D__gem__ -D__mc68000 -D__atarist -D__gem
-Asystem(tos) -Asystem(gem) -Acpu(m68k) -Amachine(m68k) test.c
/dev/p/cc320000.i
GNU CPP version 2.8.1 (68k, MIT syntax)
#include "... " search starts here:
#include <...> search starts here:
g:/gnu/include
End of search list.
g:/gnu/bin/cc1.ttp /dev/p/cc320000.i -quiet -dumpbase test.c -version -o
/dev/p/cc320000.s
GNU C version 2.8.1 (m68k-atari-mint) compiled by GNU C version 2.8.1.
g:/gnu/bin/as.ttp -v -mc68000 -o /dev/p/cc3200001.o /dev/p/cc320000.s
GNU assembler version 2.9 (m68k-atari-mint)
g:/gnu/bin/ld.ttp -v g:/gnu/lib/crt0.o -Lg:/gnu/bin -Lg:/gnu/lib
/dev/p/cc3200001.o -lgnu
g:/gnu/bin/ld.ttp Patchlevel 40 (AtariST) for GNU (Oct 21 1997 13:35:50)
```

In beiden Fällen erkennt man, wie nacheinander Präprozessor, Compiler, Assembler und Linker gestartet werden. Dabei entsteht ein Programm mit dem Namen 'a.out'. Dies ist der Defaultname, der vom Linker erzeugt wird, wenn kein anderer angegeben wurde. Um z.B. 'test.tos' zu erzeugen, muss man den Aufruf entsprechend erweitern:

```
gcc -v test.c -o test.tos
```

## 1.15 installation, archive

Alle Archive auf einen Blick

GNU C/C++

Hier eine Übersicht über die von mir zusammengestellten Archive:

Name	Inhalt
gcc-281.lzh	Treiber, Präprozessor und Compiler für C.
g++-281.lzh	Treiber und Compiler für C++.
gas-29.lzh	Der neue Assembler.
util-40b.lzh	Utilities wie Assembler, Linker usw.
mntlib48.lzh	MiNT-Lib, die Bibliotheken.
mntinc48.lzh	MiNT-Lib, die Includes.
gemlib36.lzh	GEM-Bibliothek. Die in dem Archiv enthaltenen Header-Dateien sind ein Ersatz für die in der MiNT-Lib!
pmlib23.lzh	Mathe-Bibliothek.
lib++272.lzh	C++ Bibliothek (von Frank Naumann übersetzt).
doku-st.lzh	Dieser Hypertext und einige andere Dokus zum Thema 'GNU C auf dem Atari'.

Weitere nützliche Programme, die nicht zu diesem Paket gehören:

```
gdb36p4b.zoo  GNU Debugger.
make3761.zoo  GNU Make.
```

## 1.16 tips+tricks

Tips & Tricks zu GNU C  
GNU C/C++

↔

GNU-Programme und ihre Stacksize

Die Option `-pipe` des `gcc.ttp`

Debuggen mit dem `gdb`

## 1.17 stacksize

GNU-Programme und die Stacksize

GNU C/C++

Sehr viele GNU-Programme, nicht nur der GNU C Compiler, nutzen intensiv eine besondere Speicherverwaltung. Normalerweise fordert man Speicher mit dem C-Befehl `malloc()` an. Der Nachteil dieser Methode ist der, da man so angeforderten Speicher explizit mit `free()` wieder freigeben mu. Viel praktischer ist es, wenn man in einer Prozedur lokal Speicher anfordern knnte, der dann, beim Verlassen der Prozedur automatisch wieder freigegeben wird. Im GNU-Compiler gibt es die Funktion `alloca()`, die genau diese Funktionalitt zur Verfugung stellt.

Das Problem bei dieser methode: der Speicher, der lokal angefordert wird, mu irgendwo herkommen - nmlich vom Stack. Unter UNIX hat jedes Programm einen quasi unendlich groen Stack, da man dort virtuellen Speicher hat. Auf dem Atari mu einem Programm aber der Stack explizit zugewiesen werden. Sollte die als Stack zugewiesene Speichermenge zu klein sein, strzen die meisten Programme einfach mit Buserror ab.

Die MiNT-Lib bietet nun die Mglichkeit, von auen, nachtrglich diese Stacksize einzustellen. Dazu gibt es in den Programmen die globalbe Variable `_stksize`, deren Wert normalerweise 0 enthlt. Bei Programmen, die von vornherein einen groen Stack bentigen (z.B. der Compiler), kann der Wert entweder direkt im Quelltext gesetzt oder nachtrglich mit dem Programm eingestellt werden.

Wem also mal ein GNU-Programm pltzlich mit Buserror abstrzt, sollte als aller erstes mal die Stacksize etwas erhhen.

Die aktuell eingestellte Gre kann man sich mit dem Programm `printstk.ttp` anzeigen lassen.

Da der Speicher, den man als Stacksize anmeldet, beim Start des Programms sofort belegt wird, obwohl das Programm gar nicht soviel bentigt (z.B. Stacksize des `ccl.ttp` 4MB -> die 4MB sind beim Start von `ccl.ttp` weg!), haben sich die Programmierer der MiNT-Lib einige Spezialffle ausgedacht. Normalerweise enthlt `_stksize` einen Wert, der die Gre des Stacks in Bytes angibt. Folgende Werte haben eine besondere Bedeutung:

- 1 Benutze smtlichen freien Speicher als Stack. Dieser Schalter ist auf (Multitasking-)Systemen mit viel RAM sehr unpraktisch, da dann stmlicher RAM von einem Programm belegt wird.
- 0 Minimale Stackgre (8k), Standardeinstellung.
- 1 Gibt ein viertel des freien RAMs als Stack an das Programm, die restlichen 3/4 bleiben frei.
- 2 Gibt die Hlfte des freien RAMs als Stack.
- 3 Gibt drei viertel des RAMs als Stack und lt dem System noch ein viertel.

Bei dem von mir verteilten Compiler ist für `cpp.ttp`, `ccl.ttp` und `cclplus.ttp` der Wert 1 eingestellt, da bei mir 1/4 des RAMs normalerweise ausreicht.

Für Leute mit wenig RAM (z.B. 4MB insgesamt), sollte für die Compiler `_stksize` auf -1 gesetzt werden, damit sie allen verfügbaren Speicher bekommen.

Achtung:

Das zur Einstellung der Stacksize benötigte Symbol `_stksize` kann durch das Programm `strip.ttp` gelöscht werden, so dass danach die Änderung der Stacksize nicht mehr möglich ist!

Wenn also selbstübersetzte Programme mit `strip.ttp` verkleinert werden sollen, darf auf keinen Fall vollständig ge' stripped werden!

## 1.18 -pipe

Die Option `-pipe` des `gcc.ttp`

GNU C/C++

Normalerweise startet `gcc.ttp` Prozessor, Compiler, Assembler und Linker nacheinander. Die bei jeder Schritt erzeugte Datei wird im Temp-Verzeichnis zwischengelagert und dem nächsten Programm übergeben.

Wird die Option `'-pipe'` beim Aufruf des `gcc.ttp` gesetzt, werden alle nötigen Programme gleichzeitig gestartet und das Ergebnis des einen wird über eine Pipe zum nächsten gereicht.

Die Option ist auch auf dem Atari verfügbar, es gibt aber ein paar Dinge zu beachten:

1. Sie funktioniert nur unter MiNT, da den Pipes in MagiC dafür scheinbar Funktionalität fehlt. Es mag auch sein, da es an der Mupfel liegt, mit der ich es unter MagiC probiert habe.
2. Wird die Option unter MiNT benutzt, muss man daran denken, da alle Programme gleichzeitig im Speicher liegen und damit auch gleichzeitig ihre Anforderungen fordern. Sollte eines der Programme sämtlichen Speicher belegen (`Stacksize -1`), bleibt natürlich nichts mehr für die anderen übrig.

Wenn man diese beiden Dinge beachtet, kann man die Option unter MiNT einsetzen. Unter MagiC konnte ich die Option noch nicht erfolgreich einsetzen. Entweder liegt das an der als Kommandoshell verwendeten Mupfel oder aber an nicht korrekt funktionierenden Pipes in MagiC.

Worin der eigentliche Sinn besteht, alle Programme gleichzeitig laufen zu lassen, weiß ich jedenfalls nicht. Subjektiv erscheint einem der Compilerlauf schneller, was aber ja eigentlich nicht sein kann, da zwar alle Programme gleichzeitig laufen, aber jedes ja nur etwa ein Viertel der Rechenzeit bekommt.

## 1.19 debugger

Debugging mit dem gdb

Ja, auch das gibts fr Atari: den GNU Debugger.

Die letzte Version, die portiert wurde, ist zwar schon etwas lter, lt sich aber trotzdem gut zum Debugging einsetzen.

Den gdb habe ich aber bisher nur unter MiNT zum Laufen bekommen. Dort luft er aber sehr gut, unter N.AES kann man sogar erfolgreich GEM-Programme im Single-Step Mode entwanzen.

Um den Debugger benutzen zu knnen, mu vom Compiler zum einen das Programm, zum anderen eine erweiterte Symbolliste erstellt werden, damit der Debugger auf Variablen und Quelltext zurckgreifen kann. Da diese Symbolliste nicht, wie unter UNIX blich innerhalb des ausfhrbaren Programms abgelegt werden kann, gibt es einen speziellen Symbol-Linker (sym-ld.ttp), der eine extra Datei mit den Symbolen erzeugt.

Die Erzeugung einer debug-fhige Programm-Version luft also ber drei Stufen:

1. gcc -g test.c -o test.o  
Programmdatei mit Debugcode erzeugen
2. gcc test.o -o test.tos  
Ausfhrbares Programm linken
3. gcc test.o -o test.sym -B/X/gnu/bin/sym-  
Symboldatei erzeugen. Dafr mu bei der Option -B der vollstndige Pfad zum Symbollinker inklusive der Zeichen 'sym-' angegeben werden. Der gcc versucht nun, den normalen Linker ('ld') zu starten, fgt vor 'ld' aber den angegebenen Prfix ein, wobei dann ein gltiger Pfad auf den Symbollinker entsteht und gcc startet selbigen.

Die beiden entstandenen Dateien knnen dann mit

```
gdb test.tos test.sym
```

dem Debugger bergeben werden.

Zur Bedienung des gdb verweise ich auf die beraus informative Quick Reference Card des gdb.

## 1.20 nderungen

nderungen

Neu in Version 2.8.1

ù Bei '-g' wird nicht mehr automatisch der Symbollinker gestartet. Es hat sich doch als unpraktisch erwiesen, da die Symboldateien automatisch erzeugt wurden, da z.B. bei normalen GNU-Makefiles '-g' immer gesetzt ist.

Wer Debugcode braucht, sollte eigentlich so fit sein, das Makefile um den entsprechenden Aufruf zu erweitern :-)

ù Ab Version 2.8.0 erzeugt der Compiler Assemblercode, den der gute alte gas 1.38 nicht mehr verarbeiten kann. Durch geschickte 'Manipulation' gelang es, den aktuelle GNU Assembler gas 2.9 fr MiNT nutzbar zu machen. Es wurde aber lediglich der Assembler portiert, alle anderen Bestandteile der GNU binutils (ld, ar usw.) sind nicht portiert worden, so da die alten Utils PL40 weiterhin verwendet werden.

ù gcc 2.8.1 sollte 'weak symbols' untersttzten (nicht getestet).

Version 2.7.2.3

Erste ffentliche Version.