GCC for the Atari ST, TT & Falcon

Using the GNU C-Compiler on the Atari ST, TT & Falcon computers
25 October 1997

by Frank Ridderbusch

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore,

by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

12. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
13. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) 19yy  name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Contributors to GNU CC

In addition to Richard Stallman, several people have written parts of GNU CC.

- The idea of using RTL and some of the optimization ideas came from the U. of Arizona Portable Optimizer, written by Jack Davidson and Christopher Fraser. See "Register Allocation and Exhaustive Peephole Optimization", Software Practice and Experience 14 (9), Sept. 1984, 857-866.
- Paul Rubin wrote most of the preprocessor.
- Leonard Tower wrote parts of the parser, RTL generator, RTL definitions, and of the Vax machine description.
- Ted Lemon wrote parts of the RTL reader and printer.
- Jim Wilson implemented loop strength reduction and some other loop optimizations.
- Nobuyuki Hikichi of Software Research Associates, Tokyo, contributed the support for the SONY NEWS machine.
- Charles LaBrec contributed the support for the Integrated Solutions 68020 system.
- Michael Tiemann of MCC wrote most of the description of the National Semiconductor 32000 series cpu. He also wrote the code for inline function integration and for the SPARC cpu and Motorola 88000 cpu and part of the Sun FPA support.
- Jan Stein of the Chalmers Computer Society provided support for Genix, as well as part of the 32000 machine description.
- Randy Smith finished the Sun FPA support.
- Robert Brown implemented the support for Encore 32000 systems.
- David Kashtan of SRI adapted GNU CC to the Vomit-Making System.
- Alex Crain provided changes for the 3b1.
- Greg Satz and Chris Hanson assisted in making GNU CC work on HP-UX for the 9000 series 300.
- William Schelter did most of the work on the Intel 80386 support.
- Christopher Smith did the port for Convex machines.
- Paul Petersen wrote the machine description for the Alliant FX/8.

Aside from Michael Tiemann, who worked out the front end for GNU C++, and Richard Stallman, who worked out the back end, the following people (not including those who have made their contributions to GNU CC) should not go unmentioned.

- Doug Lea contributed the GNU C++ library. This includes support for streams, obstacks, structured files, and other public service objects.
- Doug Schmidt has spent countless hours pursuing bugs in this compiler for sport. He also wrote a perfect hash function generator in GNU C++ which was used to generate

a replacement for the keyword recognizer in the lexical analyzer for both GNU CC and GNU C++.

- Marc Shapiro and Phillipe Gautron helped me implement features needed for the SOR distributed object management environment.
- Dirk Grunwald made the collect program usable under COFF.
- Angel Li adapted GNU C++ to VMS.
- Ron Cole provided additional help getting GNU C++ working on COFF-based systems.
- James Clark wrote a name demangler for the GNU C++ naming scheme, and integrated it with the linker.
- Michael Powell and Jim Mitchell helped design the GNU C++ exception handling mechanism.

The following people contributed specially to the version for the Atari ST & TT.

- John R. Dunning did the original port to the Atari ST.
- Jwahar R. Bammi improved the port and the libraries.
- Eric R. Smith wrote lots of code for the libraries.
- David Boyce ported G++ 1.39.1 and the libg++ 1.39.0 to the ST.
- The following is a not necessarily complete list of people who either contributed code or bugfixes to the libraries: Michal Jaegermann, Scott Kolodzieski, Andreas Schwab, Frank Celler, Edgar Roeder, Kai-Uwe Bloem, Allan Pratt, Jens Tingleff, Thomas Koenig, Markus Nick. Apologies to those I forgot to mention.
- Frank Ridderbusch compiled this manual specially for the Atari ST.

Introduction

This manual documents how to install and run the GNU C compiler on the Atari ST & TT computers. It does not give an introduction in C or M68000 assembler. There is enough material on both subjects available. The user, who is familiar with a C compiler, that runs on a U**x system, should have no trouble at all to get GNU C running on the Atari ST. This manual was compiled from existing GNU manuals and various bits and pieces from John R. Dunning and Jwahar R. Bammi.

The sections, which describe the compiler driver, the preprocessor and the G++ compiler driver are nearly verbatim copies of sections in the respective manuals. The original manuals (*Using and Porting GNU CC* and *The C Preprocessor*), were written by Richard M. Stallman and Michal D. Tieman (*User's Guide to GNU C++*). All of these three documents are copyright © The Free Software Foundation. I modified these sections by removing material, which described features of GNU C for systems like Vaxen or Suns. To keep this manual reasonably compact, I extracted only the sections, which describe the supported command options (and predefined macros in case of the preprocessor). If the user is interested in the extensions and details, which are implemented in GNU C, he has to refer to the original manuals. Whether all described options are useful on the Atari has to be decided.

The facts, which are presented in the assembler and utility sections are mostly derived from the sources of the respective programs (from a cross compiler kit by J. R. Bammi based on GNU C 1.31), which were available to me. Other facts were gathered by try and error. So, these sections may be a bit shaky.

The first version of this manual was based on GCC 1.37.1. Then, GCC 1.40 and G++ 1.39.1 became available. The most noticeable differences were some new options (`-mint`, `-G`, `-z`) and the extended symbol table format. In the beginning of 1992 the FSF released GCC 2.x. In this release GCC and G++ were merged into one large package. At the time of this writing the current version for the Atari is 2.4.5 with patchlevel 1. This manual doesn't cover all new command line options from GCC 2.3.3, but only the most valuable (in the authors opinion). *The coverage of GCC 2.x is not yet complete. Also, GAS 1.92 is pending.* Patchlevel 4 of GCC 2.2.2 and later versions in connection with MiNT 0.96 pl14 allow the creation of executables, whose text sections can be shared. This means, if multiple copies of the same program are running, the actual program code is memory only once, while each running program has it private data section.

Additionally two flavours of libraries are present. One version, which is now mostly maintained by J.R.Bammi, is for the ST running the native TOS operating system. The other flavour was originally maintained by E.R.Smith, since Eric moved to Atari to continue development of MultiTOS Nick Castellano (Email: entropy@terminator.rs.itd.umich.edu) took over. This version is specially modified to support MiNT, the multitasking TOS extension, also from E.R.Smith. BTW, MiNT has now become the base of Atari's multitasking TOS. It is the aim of both maintainers to keep the libraries in sync as much as possible and possibly merge them together in the future.

The best place to look for all the components (binaries and sources) is at the moment the Atari archive at terminator. The internet address for anonymous ftp is atari.archive.umich.edu. Also a mail server called BART is active. Send a message with the word `help` in it to the address atari@atari.archive.umich.edu and BART will explain himself. The maintainers of this archive post a monthly message to the USENET newsgroup '`comp.sys.atari.st`', which explains, how to get things from the archive. The packaging of the files may be different as it is explained below.

If you find any errors or typos in this manual or have any other comments, please let me know. My email address is (the SNI addresses are prefered):

ridderbusch.pad@sni-usa.com
(Amerika (North & South))
ridderbusch.pad@sni.de
(Rest of world)
Frank_Ridderbusch@pb.maus.de
(MausNet, a FIDO like network in Germany)

1 Installing GCC

There are basically three components, which make up a basic compiling system and which have to be installed. Each component is accessed via an environment variable. This three components are:

The executables
These are accessed via the normal `PATH` variable, by which all other programs are found and the variable `GCCEXEC`.

The header files
The preprocessor accesses the header files via the variable `GNUINC`. Any C++ header files are accessed via `GXXINC`

The libraries
The linker finds the startup file and the required libraries via the variable `GNULIB`. The C++ library also belongs into this directory.

All this stuff basically assumes that you're using a CLI (command line interpreter). A really good choice is Gulam, which has very nice set of features, but there are quite a number of other CLI's around, which also might do the job. If want as much U**x feeling as possible, you might consider either '`ash`', which is compatible to the Bourne shell (ported by Stefan Neuhaus), or E.R.Smiths port of '`tcsh`' or Scott Kolodzieskis port of BASH 1.12.

Apart from the CLI you definitely should get yourself a *make* utility. Again, good choices here are either the GNU Make, which offers nearly the complete U**x make functionality on the ST or the PDMAKE, which has only the core make functionality, but has on the other hand the advantage, that it requieres fewer system resources.

I suggest the following directory structure on your disk partition:

'`\gnu\bin`'
for all executable programs. The compiler driver finds the executables in this directory by looking up the environment variable `GCCEXEC`.

'`\gnu\lib`'
for the startup object modules and the libraries. The linker find the startup code and the libraries in this directory by looking up the variable `GNULIB`.

'`\gnu\include`'
for the header files. The preprocessor finds the include file in this directory by looking up the environment variable `GNUINC`.

With earlier versions of GNU CC it was only allowed to put one path into the variables `GNULIB` and `GNUINC`. GCC 1.37 and later allows you to put several paths into these variables, which are separated by either a '`,`' or a '`;`'. All the mentioned paths are searched in order to locate a specific file. However the startup module '`crt0.o`' is **only** looked for in the first directory specified in `GNULIB`. If the preprocessor can't find a include file in one of the directories specified by `GNUINC`, it will also search the paths listed in `GNULIB`.

1.1 Installing the Executables

The compressed archive of the GNU C compiler binary distribution contains the 'common' executables of the GNU compiler. That means the compiler driver (`gcc.ttp`), the preprocessor (`gcc-cpp.ttp`), the main body (`gcc-cc1.ttp`), the assembler (`gcc-as.ttp`) and the linker (`gcc-ld.ttp`), but depending from where you got your GCC the packaging might be different. The just mentioned programs are the absolute minimum, if you want to write C programs. To be comfortable, you should get the following support programs:

`gcc-ar.ttp`
: is the object library maintainer.

`gdb.ttp`
: is the GNU debugger 2.6 modified for the Atari ST. John Dunning did the original port to the Atari. Since then Jwahar Bammi has extensively hacked it. GDB now uses DBX debugging information in the object files. This requires an assembler with version 1.36 or greater. The latest version is patchlevel 18.

  In the meantime a new gdb port is available. This port is based on GDB 3.5 and requires MiNT 0.96 pl14 or later or MultiTOS. This port was done by Andreas Schwab (MiNT 0.96 needs a little patch, which is included).

  Some time later Andreas announced the port of GDB 3.6, which is only available in source form at the time of this writing.

`sym-ld.ttp`
: creates the symbol file needed with GDB.

`gcc-nm.ttp`
: prints the symbols of a GNU object library or an object file.

`cnm.ttp`
: prints the symbol table of a GEMDOS executable.

`fixstk.ttp`
`printstk.ttp`
: are used to modify and print the current stack size of an executable.

`toglclr.ttp`
: TOS 1.4 users can toggle the clear above BSS to end of TPA flag for the GEMDOS loader. A newer version of `toglclr.ttp` also allows to toggle the loader bits, that were introduced with TOS versions 2.x and 3.x.

`size68.ttp`
: This program list the values of the TEXT, DATA, and BSS sections of a ready to run executable.

`xstrip.ttp`
: removes the symbol table from an executable.

All this files should go in `\gnu\bin\` directory on your gnu disk. I personally keep my executables in the directory `e:\gnu\bin`. You should than extend the search path of your CLI to include this directory or you move the compiler driver `gcc.ttp` and the files, which are **not** invoked by `gcc.ttp` (`gcc.ttp` calls `gcc-cpp.ttp`, `gcc-cc1.ttp`, `gcc-as.ttp` and `gcc-ld.ttp`) into the directory, where you keep your other executables. The next step is to actually define `GCCEXEC`. `gcc.ttp` uses this variable to locate the preprocessor, compiler, assembler and the linker. `GCCEXEC` contains a device/dir/partial-pathname, which not only consists of the directory, where the executables are kept, but also a common prefix, which is '`gcc-`'. Assuming you also put the executables in the directory as described above, `GCCEXEC` would contain '`e:\gnu\bin\gcc-`'. The value is the same, you would give the compiler driver with the '`-B`' option.

Then you should define a variable called `TEMP`. During compilation the output of the various intermediate stages is kept here. The variable must **not** contain a trailing backslash. If you have enough memory, `TEMP` should point to a ramdisk.

There is another set of executables. They are definitely available from atari.archive.umich.edu, somewhere in the '`Mint`' directory. Other ftp archive in Europe also have them. This set is compiled with the '`-mbaserel`' option and linked with the MiNT libraries. They functional identical to the above described programs, except they behave a little better when MiNT is running (they can be interupted better and can work on a Minix file system).

1.2 Installing the libraries

The next thing to do is to install the libraries. The distributed archive contains the following libraries (again, the packaging may vary):

'`crt0.o`'
'`gcrt0.o`' are the startup object modules. The file '`gcrt0.o`' instead of '`crt0.o`' is used, if the sources files are compiled for execution profiling (the '`-pg`' option).

'`gnu.olb`'
'`gnu16.olb`'
are the standard libraries, the usual '`libc`' on other systems.

'`curses.olb`'
'`curses16.olb`'
are ports of the BSD curses.

'`gem.olb`'
'`gem16.olb`'
contain the Atari ST Aes/Vdi/FSM-GDOS bindings.

'`iio.olb`'
'`iio16.olb`'
contain the integer only '`printf`' and '`scanf`' functions.

'`pml.olb`'
'`pml16.olb`'
are the portable math libraries.

'`termcap.olb`'
'`termcap16.olb`'
are for the pure '`termcap`' support.

'`widget.olb`'
'`widget16.olb`'
are a small widget based on '`curses`'

All these libraries go to a place described by the environment variable GNULIB. Again this variable must **not** contain a trailing backslash. Staying with the above example, I've set the variable to 'e:\gnu\lib'. The libraries, which have a 16 in their names were compiled with the '-mshort' option. This makes integers the same size as shorts.

If you like to write programs for MiNT, the TOS multitasking extension from E.R.Smith, you might consider to replace 'gnu.olb', 'gnu16.olb', 'iio.olb' and 'iio16.olb' with the libraries supplied by Eric Smith. The source and the binaries of these libraries can also be retrieved from the Atari archive at terminator. The files are 'mntlib*xx*.zoo' for the sources and 'mntolb*xx*.zoo' for the binaries. They are found in the 'mint' directory. *xx* is the version number, currently 24. Programs written with this libraries will also run under TOS, as long no MiNT specific features have been used. (See Chapter 4 [The C-Compiler Driver], page 21, for more info on compiling programs for MiNT (the '-mint' option))

Another option is to have both sets of libraries installed. For this you have to rename the MiNT libraries according to the following scheme:

- 'crt0.o' ⇒ 'mcrt0.o'
- 'gcrt0.o' ⇒ 'mgcrt0.o'
- 'gnu.olb' ⇒ 'mint.olb'
- 'gnu16.olb' ⇒ 'mint16.olb'

To select these files instead of the standard TOS versions and to activate the MiNT specific portions of the header files you have to include the '-mint' option in the 'gcc.ttp' command line.

With GCC 2.2.2 pl4 and later it is possible to have another set of libraries of either the standard TOS or the MiNT libs. These new sets are named as above plus a leading 'b'. The 'b' indicates, that these files have been compiled with the '-mbaserel' option. To actually make use of these new libraries, you have to be running MiNT 0.96 patchlevel 14. They are totally useless under plain TOS. Additionally you need a linker with at least patchlevel 32.

With the option '-mbaserel' it is possible under MiNT to create executables, whose text (program code) section may be shared. (See Chapter 4 [The C-Compiler Driver], page 21, for more info on compiling programs for with shared text section (the '-mbaserel' option).) The following table gives an overview over the various files, which are used for linking depending on the compiler options.

The following startup modules used:

'bcrt0.o' if '-mbaserel' and no '-mint'.

'bmcrt0.o'
: if '-mbaserel' and '-mint'

'bgcrt0.o'
: if '-mbaserel' and no '-mint' and '-pg'

'bmgcrt0.o'
: if '-mbaserel' and '-mint' and '-pg'

The library files are used as follows (from the linker):

'-lbgnu16'
: if '-mbaserel' and '-mshort' and no '-mint'.

'-lbgnu' if '-mbaserel' and no '-mshort' and no '-mint'

'-lbmint16'
'-lbgnu16'
if '-mbaserel' and '-mshort' and '-mint'

'-lbmint'
'-lbgnu' if '-mbaserel' and no '-mshort' and '-mint'

If you have trouble with the correct setup run 'gcc -v' and see, what GCC is trying to do.

1.3 Installing the Header Files

The last bit to install are the header files. They are contained in an archive of their own. The preprocessor now knows about the variable GNUINC. Earlier version had to use the '-I*prefix*' option, to get to the header files. According to the above examples, the files would be put in the directory 'e:\gnu\include'. GNUINC has to be set accordingly.

If you like to write programs for MiNT, apart from the libraries you also need the MiNT specific include files (also from the Atari archive). These are found in the archive 'mntinc*xx*.zoo' in the 'mint' directory. *xx* matches the version number of the library. (See Chapter 4 [The C-Compiler Driver], page 21, for more info on compiling programs for MiNT (the '-mint' option)).

If you choose to have both sets of libraries installed you can keep the TOS specific header files since they are compatible with the MiNT ones.

1.4 Gulam Notes

The programs, which come with the GCC distribution also understand filenames, which use the slash ('/') as a separator. When Gulam is your favorite CLI you will stick to the backslashes, since you otherwise lose the feature of command line completition.

If you are using Gulam, you can define 'aliases' to reach the executables under more common names.

```
alias cc e:\gnu\bin\gcc.ttp
alias ar e:\gnu\bin\gcc-ar.ttp
alias as e:\gnu\bin\gcc-as.ttp
alias ld e:\gnu\bin\gcc-ld.ttp
...
```

Now you should be able to say 'cc foo.c -o foo.ttp' and the obvious thing should happen. If you still have trouble, compare your settings with the ones from the sample file 'gulam.g'. That should give you the right idea.

One additional note to Gulam. 'crt0.o' is currently set up to understand the MWC/Atari convention of passing long command lines (except it doesn't look into the _io_vector part). Gulam users should set 'env_style mw', if you want to give long args lines to 'gcc.ttp'.

To summarize the above, here are the settings from my 'gulam.g' initialization file. The usage of UNIXMODE environment variable is explained in the file 'unixmode.doc', which is part of the library sources. The GXXINC variable is for G++.

```
set env_style mw
setenv TEMP i:
setenv PATH e:\gnu\bin;<your other search paths here>
setenv GCCEXEC e:\gnu\bin\gcc-
setenv GNULIB e:\gnu\lib
setenv GNUINC e:\gnu\include
setenv GXXINC e:\gnu\g++-inc
setenv UNIXMODE 'd/brG'
```

2 Installing G++

For the G++ installation apply the same rules as for the GCC installation. The G++ compiler driver 'g++.ttp' and the actual compiler 'gcc-cc1+.ttp' belong into the same directory as the GCC executables. The preprocessor is shared between G++ and GCC. The library 'g++.olb' goes into the same directory as all the other libraries. Since G++ has a complete set of include files of it's own, they all should be copied into the directory '\gnu\g++-inc'. To let the preprocessor know, where it can find the include files, the variable GXXINC is used.

The above is valid for G++ 1.xx. Since GCC and G++ were merge into one large package with version 2.x, there is no longer a special compiler driver for G++. 'gcc.ttp' determines from the file extension, whether the C or C++ compiler should be invoked. The file 'gcc-cc1plus.ttp' is the actual C++ compiler. This file name is usually truncated to 'gcc-cc1p.ttp' in the 8+3 TOS file system. What is said about the include files and the variable GXXINC is also true for G++ 2.x.

To actually use G++, some requirements have to be fulfilled. You need the GCC include files and libraries with at least patch level 72. Additionally the linker 'gcc-ld.ttp' must have at least patch level 22.

The library 'g++.olb' is at the moment not 100% 16bit clean. That means, there is at the moment no version, which is compiled with the '-mshort' option.

3 Memory Requirements

GCC loves memory. A lot. It loves to cons structures. Lots of them. Earlier versions probably won't run at all in less than 1 Meg; the version 1.36 of GCC will probably need 2 Meg. The '`gcc-cc1.ttp`' had 1/2 meg stack, and needs it for compiling large files with optimization turned on. Happily, it doesn't need all that stack for smaller files, or even big files without the '`-O`' option, so it should be feasible to make a compiler with a smaller stack (with '`fixstk.ttp`').

GCC versions 1.37 and later uses another scheme for memory allocation. The programs '`gcc-cpp.ttp`' and '`gcc-cc1.ttp`' are setup for `_stksize == -1L`. This means, that an executable will use all available memory, doing mallocs from internal heap (as opposed to the system heap via `Malloc`), with `SP` initially set at the top, and heap starting just above the `BSS`. So if the compiler runs out of memory, you probably need more memory (or get rid of accessories, tsr's etc and try).

During my compilation of TeX 3.1 on my ST, I found that the size of a source file is not main the limiting factor, but the size of a function. At that time my ST was equipped with 2.5 megs of memory. About 512 Kb was used for ramdisk, cache and some auto folder programs. With this configuration the maximum size of a function, which could be compiled, was about 14-20 KB depending on how much code was inlined. Additionally I was able to compile GCC 1.40 and GAS 1.38 on my ST, but for this I had to disable nearly every program in the auto folder. So, with GCC 1.40 you're doing fine with 2 or 2,5 Mb.

With GCC 2.2 and later you definitly need more memory. The compiler executable itself is about 850 Kb in size. The C++ compiler is even larger (about 1.1 Mb). So, with GCC 2 you should have atleast 4 Mb.

The above described scheme, how GCC allocates memory from the heap is not so nice in a multitasking environment (MiNT or MultiTOS). Once the comiler has grabed all the available memory, it is practically impossible to start another process. You can the improve the situation by changeing the value of `_stksize`, when you compile under MiNT. You can set `_stksize` for '`gcc-cc1.ttp`' to approx 350 - 512 Kb. This gives the compiler enough stack space to run comfortably. For the preprocessor the situation is slightly more complicated. Since '`gcc-cpp.ttp`' allocates nearly all memory through the routine `alloca()`, which in turn allocates it's memory from the stack, you have to set `_stksize` to a pretty high value. The value directly corresponds to size of the source file, you want to preproccess, since the whole file is slurped into memory and then processed. In my opinion a good rule of thumb would be to multiply the size of the largest file you want to compile by 2 or 2.5 and set `_stksize` to this value.

Below is a snapshot from my system (SST68030 with 8MB), while compiling the file '`makeinfo.c`' from the Texinfo 2.16 distribution. This file has a size of about 207Kb. '`_stksize`' of '`gcc-cc1.ttp`' is set to 512 Kb. The preprocessor ('`_stksize`' set to 768Kb) used about 1.5 Mb of memory.

```
PID PPID STATUS   SIZE  COMMAND
... ... ....
000 000 Wait    724480  (idle)
002 000 Wait      1024  GEM
003 002 Run     557056  GEM
```

```
005 000 Sleep   359936  TCSH
149 000 Sleep  1557760  EMACS
150 149 Sleep    29184  wakeup 60
158 149 Wait     99072  sh -c gmake -f makefile.st CC=gcc makeinfo
159 158 Wait    175616  gmake -f makefile.st CC=gcc makeinfo.o
160 159 Wait     69632  gcc -c -O2 -Wall -DDOTS -fomit-frame-point
161 005 Run      78336  top
163 160 Run    1981184  gcc-cc1 c:/clipbr/cc1600.i -quiet -dumpbas
```

Another topic, which frequently comes up on comp.sys.atari.st and which also loosely fits into this section is "Why are GCC compiled binaries so big". The reason is simply maximum U**x compatiblity and *not* bad code quality. There is quite a bit of code in the libraries, which emulates certain U**x features, which are alway automatically linked into the executable. Apart from other minor compatiblity functions, the features, which implement most of the overhead, are the `signal()` emulation and the functionality of the extended file system, directed through the `UNIXMODE` environment variable.

The extended filesystem in the TOS libraries is a layer in between the `open()` and all the other file descriptor based i/o routines and the actual systemcalls in the TOS kernel (`fopen()`, `fwrite()`, etc.). This extended file system allows file names with upto 32 characters and symbolic links. The just mentioned layer takes care of the mapping from the long to a unique file name in the standard 8+3 TOS file system, by maintaining an additional file (‘`.dir`’), which holds the mapping from long to short name. The setting of the `UNIXMODE` environment variable determines, which of the available features in extended file system layer are used at runtime.

```
user land        +-----------------+
  -+-            | Long file name |
   ^             +-----------------+
   |                     |
   |                     V
   |             +-----------------+
                 | STDIO fopen()  |
   L             +-----------------+
   i                     |
   b                     V
   r             +-----------------+
   a             |    open()       |
   r             +-----------------+
   y                     |
                         V
   |          +--------------------------+
   |          |extended file system layer|-->-+
   |          +--------------------------+    |
   |                     |                    |
  -+-                    V                    |
                 +-----------------+          |
kernel land      |  TOS fopen()   |          |
                 +-----------------+          |
                         |                    |
                         V                    V
                 +-----------------+      +------+
TOS file system  | 8+3 file name  |      | .dir |
                 +-----------------+      +------+
```

Now, with that in mind you will probably understand, why executables, which are linked with the TOS libs are so much bigger compared to executables from other Atari compilers. All this compatility stuff adds up. What you get for this increased executable size is maximum U**x compatiblity. You often have to change only a hand full of lines to succesfully compile a source file from an U**x environment.

If you know a bit about operating systems in general, you will know, that stuff like the above doesn't belong into a library into the hands of a user. This stuff belongs into the kernel. All this is implemented in the MiNT kernel (through the loadable file system). The executables, which are linked with the MiNT libs are therefore smaller, than their TOS equivalents, since a lot of compatibily hacks are no longer present.

4 Controlling the C-Compiler Driver (`gcc.ttp`)

The GNU C compiler uses a command syntax much like the U**x C compiler. The `gcc.ttp` program accepts options and file names as operands. Multiple single-letter options may *not* be grouped: `-dr` is very different from `-d -r`.

When you invoke GNU CC, it normally does preprocessing, compilation, assembly and linking. File names which end in `.c` are taken as C source to be preprocessed and compiled; file names ending in `.i` are taken as preprocessor output to be compiled; compiler output files plus any input files with names ending in `.s` are assembled; then the resulting object files, plus any other input files, are linked together to produce an executable.

Command options allow you to stop this process at an intermediate stage.

For example, the `-c` option says not to run the linker. Then the output consists of object files output by the assembler.

Other command options are passed on to one stage of processing. Some options control the preprocessor and others the compiler itself. Yet other options control the assembler and linker; these are not documented here, but you rarely need to use any of them.

Here are the options to control the overall compilation process, including those that say whether to link, whether to assemble, and so on.

`-o` *file*
: Place output in file *file*. This applies regardless to whatever sort of output is being produced, whether it be an executable file, an object file, an assembler file or preprocessed C code.

  If `-o` is not specified, the default is to put an executable file in `a.out`, the object file `*source*.c` in `*source*.o`, an assembler file in `*source*.s`, and preprocessed C on standard output.

`-c`
: Compile or assemble the source files, but do not link. Produce object files with names made by replacing `.c` or `.s` with `.o` at the end of the input file names. Do nothing at all for object files specified as input.

`-S`
: Compile into assembler code but do not assemble. The assembler output file name is made by replacing `.c` with `.s` at the end of the input file name. Do nothing at all for assembler source files or object files specified as input.

`-E`
: Run only the C preprocessor. Preprocess all the C source files specified and output the results to standard output.

`-v`
: Compiler driver program prints the commands it executes as it runs the preprocessor, compiler proper, assembler and linker. Some of these are directed to print their own version numbers.

`-s`
: The executable is stripped from the DRI compatible or extended symbol table. Certain symbolic debuggers like `sid.prg` work with this symbol table. Also the programs `printstk.ttp` and `fixstk.ttp` (See See Chapter 8 [The Utilities], page 53, for more info) lookup the symbol `_stksize` in this table.

`-x`
: This option directs the linker to discard all local labels while creating the symbol table and write only those labels, which are marked global.

‘-G’ Instead of the standard DRI compatible symbol table, an extended symbol table is written, which allows symbol names to be up to 22 characters long. Most of the utility programs have been updated to work with this format. The most benefit you get with ‘gprof.ttp’ and ‘adb’ (the adb-like debugger, originally written for the Sozobon C compiler by Johann Rueg and Don Dugger and later improved by Michal Jaegermann (See See Chapter 9 [Debugging], page 62, for additional info about debugging)).

‘-B*prefix*’ The compiler driver program tries *prefix* as a prefix for each program it tries to run. These programs are ‘gcc-cpp.ttp’, ‘gcc-cc1.ttp’, ‘gcc-as.ttp’ and ‘gcc-ld.ttp’.

For each subprogram to be run, the compiler driver first tries the ‘-B’ prefix, if any. If that name is not found, or if ‘-B’ was not specified, the driver tries two standard prefixes, which are ‘/usr/lib/gcc-’ and ‘/usr/local/lib/gcc-’. If neither of those results in a file name that is found, the unmodified program name is searched for using the directories specified in your ‘PATH’ environment variable.

You can get a similar result from the environment variable `GCCEXEC`. If it is defined, its value is used as a prefix in the same way. If both the ‘-B’ option and the `GCCEXEC` variable are present, the ‘-B’ option is used first and the environment variable value second.

‘-z’ This option directs all output from ‘stderr’ to the file ‘compile.err’. So, all error messages and warnings, which are printed during a compile run are written to this file. The redirection is done by the compiler driver and is therefore only valid for those programs, which are subsequently invoked by ‘gcc.ttp’. The ‘-z’ option was introduced only very lately, so not every executable floating around might have it.

These options control the details of C compilation itself.

‘-ansi’ Support all ANSI standard C programs.

This turns off certain features of GNU C that are incompatible with ANSI C, such as the `asm`, `inline` and `typeof` keywords, and predefined macros such as `unix` and `vax` that identify the type of system you are using. It also enables the undesirable and rarely used ANSI trigraph feature.

The ‘-ansi’ option does not cause non-ANSI programs to be rejected gratuitously. For that, ‘-pedantic’ is required in addition to ‘-ansi’.

The macro `__STRICT_ANSI__` is predefined when the ‘-ansi’ option is used. Some header files may notice this macro and refrain from declaring certain functions or defining certain macros that the ANSI standard doesn't call for; this is to avoid interfering with any programs that might use these names for other things.

‘-traditional’

Attempt to support some aspects of traditional C compilers. Specifically:

- All `extern` declarations take effect globally even if they are written inside of a function definition. This includes implicit declarations of functions.

- The keywords `typeof`, `inline`, `signed`, `const` and `volatile` are not recognized.
- Comparisons between pointers and integers are always allowed.
- Integer types `unsigned short` and `unsigned char` promote to `unsigned int`.
- Out-of-range floating point literals are not an error.
- All automatic variables not declared `register` are preserved by `longjmp`. Ordinarily, GNU C follows ANSI C: automatic variables not declared `volatile` may be clobbered.
- In the preprocessor, comments convert to nothing at all, rather than to a space. This allows traditional token concatenation.
- In the preprocessor, macro arguments are recognized within string constants in a macro definition (and their values are stringified, though without additional quote marks, when they appear in such a context). The preprocessor always considers a string constant to end at a newline.
- The predefined macro `__STDC__` is not defined when you use ‘`-traditional`’, but `__GNUC__` is (since the GNU extensions which `__GNUC__` indicates are not affected by ‘`-traditional`’). If you need to write header files that work differently depending on whether ‘`-traditional`’ is in use, by testing both of these predefined macros you can distinguish four situations: GNU C, traditional GNU C, other ANSI C compilers, and other old C compilers.

‘`-O`’
: Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. Without ‘`-O`’, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.

Without ‘`-O`’, only variables declared `register` are allocated in registers. The resulting compiled code is a little worse than produced by PCC without ‘`-O`’.

With ‘`-O`’, the compiler tries to reduce code size and execution time.Some of the ‘`-f`’ options described below turn specific kinds of optimization on or off.

‘`-g`’
: Produce debugging information in the operating system's native format (for DBX or SDB). GCC on the Atari produces the DBX debugging format. GDB also works with this debugging information.

Unlike most other C compilers, GNU CC allows you to use ‘`-g`’ with ‘`-O`’. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops. Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

‘-gg’ Produce debugging information in GDB’s own format. This option is no longer supported. Do not use it.

‘-w’ Inhibit all warning messages.

‘-W’ Print extra warning messages for these events:

- An automatic variable is used without first being initialized.

  These warnings are possible only in optimizing compilation, because they require data flow information that is computed only when optimizing. They occur only for variables that are candidates for register allocation. Therefore, they do not occur for a variable that is declared `volatile`, or whose address is taken, or whose size is other than 1, 2, 4 or 8 bytes. Also, they do not occur for structures, unions or arrays, even when they are in registers.

  Note that there may be no warning about a variable that is used only to compute a value that itself is never used, because such computations may be deleted by the flow analysis pass before the warnings are printed.

  These warnings are made optional because GNU CC is not smart enough to see all the reasons why the code might be correct despite appearing to have an error. Here is one example of how this can happen:

  ```
  {
    int x;
    switch (y)
      {
      case 1: x = 1;
        break;
      case 2: x = 4;
        break;
      case 3: x = 5;
      }
    foo (x);
  }
  ```

  If the value of `y` is always 1, 2 or 3, then `x` is always initialized, but GNU CC doesn’t know this. Here is another common case:

  ```
  {
    int save_y;
    if (change_y) save_y = y, y = new_y;
    ...
    if (change_y) y = save_y;
  }
  ```

  This has no bug because `save_y` is used only if it is set.

  Some spurious warnings can be avoided if you declare as `volatile` all the functions you use that never return.

- A nonvolatile automatic variable might be changed by a call to `longjmp`. These warnings as well are possible only in optimizing compilation.

  The compiler sees only the calls to `setjmp`. It cannot know where `longjmp` will be called; in fact, a signal handler could call it at any point in the code.

As a result, you may get a warning even when there is in fact no problem because `longjmp` cannot in fact be called at the place which would cause a problem.

- A function can return either with or without a value. (Falling off the end of the function body is considered returning without a value.) For example, this function would inspire such a warning:

  ```
  foo (a)
  {
    if (a > 0)
      return a;
  }
  ```

  Spurious warnings can occur because GNU CC does not realize that certain functions (including `abort` and `longjmp`) will never return.

- An expression-statement contains no side effects.

In the future, other useful warnings may also be enabled by this option.

‘`-Wimplicit`’
: Warn whenever a function is implicitly declared.

‘`-Wreturn-type`’
: Warn whenever a function is defined with a return-type that defaults to `int`. Also warn about any `return` statement with no return-value in a function whose return-type is not `void`.

‘`-Wunused`’
: Warn whenever a local variable is unused aside from its declaration, and whenever a function is declared static but never defined.

‘`-Wswitch`’
: Warn whenever a `switch` statement has an index of enumeral type and lacks a `case` for one or more of the named codes of that enumeration. (The presence of a `default` label prevents this warning.) `case` labels outside the enumeration range also provoke warnings when this option is used.

‘`-Wcomment`’
: Warn whenever a comment-start sequence ‘`/*`’ appears in a comment.

‘`-Wtrigraphs`’
: Warn if any trigraphs are encountered (assuming they are enabled).

‘`-Wall`’
: All of the above ‘`-W`’ options combined. These are all the options which pertain to usage that we recommend avoiding and that we believe is easy to avoid, even in conjunction with macros.

  The other ‘`-W...`’ options below are not implied by ‘`-Wall`’ because certain kinds of useful macros are almost impossible to write without causing those warnings.

‘`-Wshadow`’
: Warn whenever a local variable shadows another local variable.

`-Wid-clash-`*len*
: Warn whenever two distinct identifiers match in the first *len* characters. This may help you prepare a program that will compile with certain obsolete, brain-damaged compilers.

`-Wpointer-arith`
: Warn about anything that depends on the "size of" a function type or of `void`. GNU C assigns these types a size of 1, for convenience in calculations with `void *` pointers and pointers to functions.

`-Wcast-qual`
: Warn whenever a pointer is cast so as to remove a type qualifier from the target type. For example, warn if a `const char *` is cast to an ordinary `char *`.

`-Wwrite-strings`
: Give string constants the type `const char[`*length*`]` so that copying the address of one into a non-`const char *` pointer will get a warning. These warnings will help you find at compile time code that can try to write into a string constant, but only if you have been very careful about using `const` in declarations and prototypes. Otherwise, it will just be a nuisance; this is why we did not make `-Wall` request these warnings.

`-p`
: Generate extra code to write profile information suitable for the analysis program `prof`. This is useless on the Atari ST. Use `-pg` instead.

`-pg`
: Generate extra code to write profile information suitable for the analysis program `gprof`.

`-l`*library*
: Search a standard list of directories for a library named *library*, which is actually a file named `$GNULIB\`*library*`.olb`. The linker uses this file as if it had been specified precisely by name.

  The directories searched include several standard system directories plus any that you specify with `-L`.

  Normally the files found this way are library files—archive files whose members are object files. The linker handles an archive file by scanning through it for members which define symbols that have so far been referenced but not defined. But if the file that is found is an ordinary object file, it is linked in the usual fashion. The only difference between using an `-l` option and specifying a file name is that `-l` searches several directories.

`-L`*dir*
: Add directory *dir* to the list of directories to be searched for `-l`.

`-nostdlib`
: Don't use the standard system libraries and startup files when linking. Only the files you specify (plus `gnulib`) will be passed to the linker.

`-m`*machinespec*
: Machine-dependent option specifying something about the type of target machine. These options are defined by the macro `TARGET_SWITCHES` in the machine description. The default for the options is also defined by that macro, which enables you to change the defaults.

These are the ‘-m’ options defined in the 68000 machine description:

‘-m68000’

‘-mc68000’
: Generate output for a 68000. This is the default on the Atari ST.

‘-m68020’
‘-mc68020’
: Generate output for a 68020 (rather than a 68000).

‘-m68881’
: Generate output containing 68881 instructions for floating point.

‘-msoft-float’
: Generate output containing library calls for floating point.

‘-mshort’
: Consider type int to be 16 bits wide, like short int and causes the macro __MSHORT__ to be defined. Using this option also causes the library ‘*library*16.olb’ to be linked. (Also See Section 6.2 [Predefined Macros], page 41, for more info)

‘-mint’
: Compile for MiNT (MiNT is not TOS). The macro __MINT__ is defined and the linker links with the mint library ‘-lmint’ before linking with the normal C library ‘-lgnu’. Also, the linker uses the startup file ‘mcrt0.o’ instead of the normal ‘crt0.o’. If ‘-mshort’ is also specified, then both the macros __MSHORT__ and __MINT__ are defined and the linker links with ‘-lmint16 -lgnu16’.

‘-mbaserel’

‘-mpcrel’
: This is no op at the moment.

‘-mnobitfield’
: Do not use the bit-field instructions. ‘-m68000’ implies ‘-mnobitfield’.

‘-mbitfield’
: Do use the bit-field instructions. ‘-m68020’ implies ‘-mbitfield’. This is the default if you use the unmodified sources.

‘-mrtd’
: Use a different function-calling convention, in which functions that take a fixed number of arguments return with the rtd instruction, which pops their arguments while returning. This saves one instruction in the caller since there is no need to pop the arguments there.

  This calling convention is incompatible with the one normally used on U**x, so you cannot use it if you need to call libraries compiled with the U**x compiler.

  Also, you must provide function prototypes for all functions that take variable numbers of arguments (including printf); otherwise incorrect code will be generated for calls to those functions.

  In addition, seriously incorrect code will result if you call a function with too many arguments. (Normally, extra arguments are harmlessly ignored.)

The `rtd` instruction is supported by the 68010 and 68020 processors, but not by the 68000.

'-f*flag*'
Specify machine-independent flags. Most flags have both positive and negative forms; the negative form of '-ffoo' would be '-fno-foo'. In the table below, only one of the forms is listed—the one which is not the default. You can figure out the other form by either removing 'no-' or adding it.

'-ffloat-store'
: Do not store floating-point variables in registers. This prevents undesirable excess precision on machines such as the 68000 where the floating registers (of the 68881) keep more precision than a `double` is supposed to have.

  For most programs, the excess precision does only good, but a few programs rely on the precise definition of IEEE floating point. Use '-ffloat-store' for such programs.

'-fno-asm'
: Do not recognize `asm`, `inline` or `typeof` as a keyword. These words may then be used as identifiers.

'-fno-defer-pop'
: Always pop the arguments to each function call as soon as that function returns. Normally the compiler (when optimizing) lets arguments accumulate on the stack for several function calls and pops them all at once.

'-fstrength-reduce'
: Perform the optimizations of loop strength reduction and elimination of iteration variables.

'-fcombine-regs'
: Allow the combine pass to combine an instruction that copies one register into another. This might or might not produce better code when used in addition to '-O'. I am interested in hearing about the difference this makes. (Only GCC and G++ 1.40).

'-fforce-mem'
: Force memory operands to be copied into registers before doing arithmetic on them. This may produce better code by making all memory references potential common subexpressions. When they are not common subexpressions, instruction combination should eliminate the separate register-load. I am interested in hearing about the difference this makes.

'-fforce-addr'
: Force memory address constants to be copied into registers before doing arithmetic on them. This may produce better code just as '-fforce-mem' may.

‘-fomit-frame-pointer’
: Don’t keep the frame pointer in a register for functions that don’t need one. This avoids the instructions to save, set up and restore frame pointers; it also makes an extra register available in many functions. **It also makes debugging impossible.**

  On some machines, such as the Vax, this flag has no effect, because the standard calling sequence automatically handles the frame pointer and nothing is saved by pretending it doesn’t exist. The machine-description macro `FRAME_POINTER_REQUIRED` controls whether a target machine supports this flag.

‘-finline-functions’
: Integrate all simple functions into their callers. The compiler heuristically decides which functions are simple enough to be worth integrating in this way.

  If all calls to a given function are integrated, and the function is declared `static`, then the function is normally not output as assembler code in its own right.

‘-fcaller-saves’
: Enable values to be allocated in registers that will be clobbered by function calls, by emitting extra instructions to save and restore the registers around such calls. Such allocation is done only when it seems to result in better code than would otherwise be produced.

  This option is enabled by default on certain machines, usually those which have no call-preserved registers to use instead.

‘-fkeep-inline-functions’
: Even if all calls to a given function are integrated, and the function is declared `static`, nevertheless output a separate run-time callable version of the function.

‘-fwritable-strings’
: Store string constants in the writable data segment and don’t uniquize them. This is for compatibility with old programs which assume they can write into string constants. Writing into string constants is a very bad idea; “constants” should be constant.

‘-fcond-mismatch’
: Allow conditional expressions with mismatched types in the second and third arguments. The value of such an expression is void.

‘-fno-function-cse’
: Do not put function addresses in registers; make each instruction that calls a constant function contain the function’s address explicitly.

  This option results in less efficient code, but some strange hacks that alter the assembler output may be confused by the optimizations performed when this option is not used.

‘-fvolatile’
: Consider all memory references through pointers to be volatile.

‘-fshared-data’
: Requests that the data and non-`const` variables of this compilation be shared data rather than private data. The distinction makes sense only on certain operating systems, where shared data is shared between processes running the same program, while private data exists in one copy per process.

‘-funsigned-char’
: Let the type `char` be the unsigned, like `unsigned char`.

  Each kind of machine has a default for what `char` should be. It is either like `unsigned char` by default or like `signed char` by default. (Actually, at present, the default is always signed.)

  The type `char` is always a distinct type from either `signed char` or `unsigned char`, even though its behavior is always just like one of those two.

  Note that this is equivalent to ‘-fno-signed-char’, which is the negative form of ‘-fsigned-char’.

‘-fsigned-char’
: Let the type `char` be signed, like `signed char`.

  Note that this is equivalent to ‘-fno-unsigned-char’, which is the negative form of ‘-funsigned-char’.

‘-ffixed-*reg*’
: Treat the register named *reg* as a fixed register; generated code should never refer to it (except perhaps as a stack pointer, frame pointer or in some other fixed role).

  reg must be the name of a register. The register names accepted are machine-specific and are defined in the `REGISTER_NAMES` macro in the machine description macro file.

  This flag does not have a negative form, because it specifies a three-way choice.

‘-fcall-used-*reg*’
: Treat the register named *reg* as an allocatable register that is clobbered by function calls. It may be allocated for temporaries or variables that do not live across a call. Functions compiled this way will not save and restore the register *reg*.

  Use of this flag for a register that has a fixed pervasive role in the machine's execution model, such as the stack pointer or frame pointer, will produce disastrous results.

  This flag does not have a negative form, because it specifies a three-way choice.

`-fcall-saved-`*reg*

Treat the register named *reg* as an allocatable register saved by functions. It may be allocated even for temporaries or variables that live across a call. Functions compiled this way will save and restore the register *reg* if they use it.

Use of this flag for a register that has a fixed pervasive role in the machine's execution model, such as the stack pointer or frame pointer, will produce disastrous results.

A different sort of disaster will result from the use of this flag for a register in which function values may be returned.

This flag does not have a negative form, because it specifies a three-way choice.

`-pedantic`

Issue all the warnings demanded by strict ANSI standard C; reject all programs that use forbidden extensions.

Valid ANSI standard C programs should compile properly with or without this option (though a rare few will require `-ansi`). However, without this option, certain GNU extensions and traditional C features are supported as well. With this option, they are rejected. There is no reason to *use* this option; it exists only to satisfy pedants.

These options control the C preprocessor, which is run on each C source file before actual compilation. If you use the `-E` option, nothing is done except C preprocessing. Some of these options make sense only together with `-E` because they request preprocessor output that is not suitable for actual compilation.

`-C` Tell the preprocessor not to discard comments. Used with the `-E` option.

`-I`*dir* Search directory *dir* for include files.

`-I-` Any directories specified with `-I` options before the `-I-` option are searched only for the case of `#include "`*file*`"`; they are not searched for `#include <`*file*`>`.

If additional directories are specified with `-I` options after the `-I-`, these directories are searched for all `#include` directives. (Ordinarily *all* `-I` directories are used this way.)

In addition, the `-I-` option inhibits the use of the current directory as the first search directory for `#include "`*file*`"`. Therefore, the current directory is searched only if it is requested explicitly with `-I.`. Specifying both `-I-` and `-I.` allows you to control precisely which directories are searched before the current one and which are searched after.

`-nostdinc`

Do not search the standard system directories for header files. Only the directories you have specified with `-I` options (and the current directory, if appropriate) are searched.

Between `-nostdinc` and `-I-`, you can eliminate all directories from the search path except those you specify.

‘-M’ Tell the preprocessor to output a rule suitable for make describing the dependencies of each source file. For each source file, the preprocessor outputs one make-rule whose target is the object file name for that source file and whose dependencies are all the files ‘#include’d in it. This rule may be a single line or may be continued with ‘\’-newline if it is long.

‘-M’ implies ‘-E’.

‘-MM’ Like ‘-M’ but the output mentions only the user-header files included with ‘#include "*file*"’. System header files included with ‘#include <*file*>’ are omitted.

‘-MM’ implies ‘-E’.

‘-D*macro*’ Define macro *macro* with the empty string as its definition.

‘-D*macro*=*defn*’
Define macro *macro* as *defn*.

‘-U*macro*’ Undefine macro *macro*.

‘-T’ Support ANSI C trigraphs. You don’t want to know about this brain-damage. The ‘-ansi’ option also has this effect.

5 Controlling the C++-Compiler Driver (‘g++.ttp’)

The GNU C++ compiler uses a command syntax much like the AT&T C++ compiler. The `g++.ttp` program accepts options and file names as operands. Multiple single-letter options may *not* be grouped: ‘`-dr`’ is very different from ‘`-d -r`’.

When you invoke GNU C++, it normally does preprocessing, compilation, assembly and linking. File names which end in ‘`.c`’, ‘`.cc`’, or ‘`.C`’ are taken as GNU C++ source to be preprocessed and compiled; compiler output files plus any input files with names ending in ‘`.s`’ are assembled; then the resulting object files, plus any other input files, are linked together to produce an executable.

Unlike C++, there is no ‘`-F`’ option. This is because GNU C++ is a native-code C++ compiler, not a front-end pre-processor. The advantages of this organization are faster compilation speed, better error-reporting capabilities, better opportunity for compiler optimization, and true source-level debuggability with the GDB debugger (version 3.4 or higher).

Command options allow you to stop this process at an intermediate stage. For example, the ‘`-c`’ option says not to run the linker. Then the output consists of object files output by the assembler.

Other command options are passed on to one stage. Some options control the preprocessor and others the compiler itself. Yet other options control the assembler and linker; these are not documented here because the GNU assembler and linker are not yet released.

Here are the options to control the overall compilation process, including those that say whether to link, whether to assemble, and so on. The options, which don't have any text, behave exactly as their GCC counterparts.

With GCC 2.x there is no independend compiler driver for C++. ‘`gcc.ttp`’ handles both cases. One major difference between ‘`g++.ttp`’ from version 1.xx and ‘`gcc.ttp`’ from version 2.x is, that you have to explicitly link with ‘`g++.olb`’. Therefore when you compile C++ programs with GCC 2.x you alway have to include `-lg++` on the command line, when you create the final executable.

‘`-o` *file*’

‘`-c`’

It is intended that the compiler driver of GNU C++ will invoke the appropriate translator (or series of translators) for a given source file. Currently, the translators are selected on the basis of their file extension. So that one driver can be used for many different translators, it is important that these extensions be distinct. It is strongly suggested that users become accustomed to using a ‘`.cc`’ file extension for GNU C++ code, to distinguish it from the ‘`.c`’ file extension already used for GNU CC code.

‘`-S`’

‘`-E`’

‘`-v`’

‘`-s`’

‘-x’

‘-G’

These options control the details of GNU C++ compilation itself.

‘-ansi’

With this option enabled, differences between GNU C++ and AT&T C++ are also flagged. Because the C++ language definition and the ANSI draft differ on the interpretation of syntactically identical constructs, it is unlikely that this flag could possibly be of any real use. (For this reason, this flag is currently not fully implemented).

‘-traditional’

- The other aspects of ‘-traditional’ are equivalent to GCC.
- The predefined macro `__cplusplus` is defined to identify compilation for C++ 2.0. C++ version 1.2 uses `c_plusplus` as its identifying macro. Since GNU C++ implements version 2.0 semantics, the former is defined, while the latter is not. The macro `__GNUG__` is also defined, so that features specific to GNU C++ can be used conditionally.

‘-O’

‘-g’

‘-w’

‘-W’

‘-Wimplicit’

‘-Wreturn-type’

‘-Wunused’

‘-Wswitch’

‘-Wcomment’

‘-Wtrigraphs’

‘-Wall’

‘-Wshadow’

‘-Wid-clash-*len*’

‘-Wpointer-arith’

‘-Wcast-qual’

‘-Wwrite-strings’

‘-p’

‘-pg’

‘-l*library*’

‘-L*dir*’

‘-nostdlib’

`-m`*machinespec*

`-m68020`
`-mc68020`

`-m68000`

`-mc68000`

`-m68881`

`-msoft-float`

`-mshort`

`-mint`

`-mnobitfield`

`-mbitfield`

`-mrtd`

`-f`*flag*

`-ffloat-store`

`-fno-asm`

`-fno-defer-pop`

`-fstrength-reduce`

`-fcombine-regs`

`-fforce-mem`

`-fforce-addr`

`-fomit-frame-pointer`

`-finline-functions`

`-fdefault-inline`

If this option is enabled then member functions defined inside class scope are compiled inline by default, *i.e.*, you don't need to add *inline* in front of the member function name. By popular demand, this option is now the default. To keep GNU C++ from inlining these member functions, specify `-fno-default-inline`.

`-fcaller-saves`

`-fkeep-inline-functions`

`-fwritable-strings`

`-fcond-mismatch`

`-fno-function-cse`

`-fvolatile`

`-fshared-data`

`-funsigned-char`

`-fsigned-char`

`-ffixed-`*reg*

`-fcall-used-`*reg*

`-fcall-saved-`*reg*

`-fstrict-prototype`

Consider the declaration `int foo ();`. In C++, this means that the function `foo` takes no arguments. In ANSI C, this is declared `int foo(void);`. With the flag `-fno-strict-prototype`, declaring functions with no arguments is equivalent to declaring its argument list to be untyped, i.e., `int foo ();` is equivalent to saying `int foo (...);`.

`-felide-constructors`

Using this option instructs the compiler to be smarter about when it can elide constructors. With out this flag, GNU C++ and cfront both generate effectively the same code for:

```
A foo ();
A x (foo ());   // x is initialized by `foo ()', no ctor called here
A y = foo ();   // call to `foo ()' heads to temporary,
                // y is initialized from the temporary.
```

Note the difference! With this flag, GNU C++ initializes 'y' directly from the call to 'foo ()' without going through a temporary.

`-fall-virtual`

When the `-fall-virtual` option is used, all member functions (except for constructor functions and new/delete member operators) declared in the same class with a "method-call" operator method have entries made for them in the vtable for the given class. In effect, all of these methods become "implicitly virtual."

This does *not* mean that all calls to these methods will be made through the vtable. There are some circumstances under which it is obvious that a call to a given virtual function can be made directly, and in these cases the calls still go direct.

The effect of making all methods of a class with a declared 'operator->()()' implicitly virtual using `-fall-virtual` extends also to all non-constructor methods of any class derived from such a class.

`-fthis-is-variable`

The incorporation of user-defined free store management into C++ has made assignment to *this* an anachronism. Therefore, by default GNU C++ treats the type of *this* in a member function of *class X* to be *X *const*. In other words, it is illegal to assign to *this* within a class member function. However, for backwards compatibility, you can invoke the old behavior by using `-fthis-is-variable`.

`-fsave-memoized`

`-fmemoize-lookups`

These flags are of use to get the compiler to compile programs faster using heuristics. They are not on by default since they only do so about half the time. They other half of the time programs compile more slowly (and take more memory).

The first time the compiler must build a call to a member function (or reference to a data member), it must (1) determine whether the class implements member functions of that name (2) resolve which member function to call (which involves figuring out what sorts of type conversions need to be made), and (3) check the visibility of the member function to the caller. All of this adds up to slower compilation. Normally, the second time a call is made to that member function (or reference to that data member), it must go through the same lengthy process again. This means that code like this

```
cout << "This " << p << " has " << n << " legs.\n";
```

makes six passes through all three steps. By using a software cache, a "hit" significantly reduces this cost. Unfortunately, using the cache introduces another layer of mechanisms which must be implemented, and so incurrs its own overhead. The '`-fmemoize-lookups`' enables the software cache.

Because access privileges (visibility) to members and member functions may differ from one function context to the next, may need to be flushed. With the '`-fmemoize-lookups`' flag, the cache is flushed after every function that is compiled. With the '`-fsave-memoized`' flag, when the compiler determines that the context of the last function compiled would yield the same access privileges of the next function to compile, it preserves the cache. This really helps when defining many member functions for the same class: with the exception of member functions which are friends of other classes, each member function has exactly the same access privileges as every other, and the cache need not be flushed.

'`-pedantic`'

Attempt to support strict ANSI standard C. Since C++ invalidates a number of ANSI constructions, this switch is of dubious value. Some attempt has been made to warn about non-standard C++ features, however, even this is of uncertain value, as there are two C++ standards currently in existence: the standard as documented by AT&T, and the standard as implemented by the AT&T C++ compiler. Valid C++ programs should compile properly with or without this switch. However, without this switch, certain useful or traditional constructs banned by the standard are supported. With this switch, they are rejected. There is no reason to use this switch; it exists only to satisfy curious pedants.

The options, which control the behaviour of the C preprocessor are the same as for GCC (See Chapter 4 [The C-Compiler Driver], page 21, last section).

6 The Preprocessor

6.1 Invoking the C Preprocessor

Most often when you use the C preprocessor you will not have to invoke it explicitly: the C compiler will do so automatically. However, the preprocessor is sometimes useful individually.

The C preprocessor expects two file names as arguments, *infile* and *outfile*. The preprocessor reads *infile* together with any other files it specifies with '`#include`'. All the output generated by the combined input files is written in *outfile*.

Either *infile* or *outfile* may be '`-`', which as *infile* means to read from standard input and as *outfile* means to write to standard output. Also, if *outfile* or both file names are omitted, the standard output and standard input are used for the omitted file names.

Here is a table of command options accepted by the C preprocessor. Most of them can also be given when compiling a C program; they are passed along automatically to the preprocessor when it is invoked by the compiler.

'`-P`'
: Inhibit generation of '`#`'-lines with line-number information in the output from the preprocessor. This might be useful when running the preprocessor on something that is not C code and will be sent to a program which might be confused by the '`#`'-lines

'`-C`'
: Do not discard comments: pass them through to the output file. Comments appearing in arguments of a macro call will be copied to the output before the expansion of the macro call.

'`-T`'
: Process ANSI standard trigraph sequences. These are three-character sequences, all starting with '`??`', that are defined by ANSI C to stand for single characters. For example, '`??/`' stands for '`\`', so '`'??/n'`' is a character constant for Newline. Strictly speaking, the GNU C preprocessor does not support all programs in ANSI Standard C unless '`-T`' is used, but if you ever notice the difference it will be with relief.

  You don't want to know any more about trigraphs.

'`-pedantic`'
: Issue warnings required by the ANSI C standard in certain cases such as when text other than a comment follows '`#else`' or '`#endif`'.

'`-pedantic-errors`'
: Like '`-pedantic`', except that errors are produced rather than warnings.

'`-Wtrigraphs`'
: Warn if any trigraphs are encountered (assuming they are enabled).

'`-Wcomment`'
: Warn whenever a comment-start sequence '`/*`' appears in a comment.

'`-Wall`'
: Requests both '`-Wtrigraphs`' and '`-Wcomment`' (but not '`-Wtraditional`').

'`-Wtraditional`'
: Warn about certain constructs that behave differently in traditional and ANSI C.

'`-I` *directory*'
: Add the directory *directory* to the end of the list of directories to be searched for header files. This can be used to override a system header file, substituting your own version, since these directories are searched before the system header file directories. If you use more than one '`-I`' option, the directories are scanned in left-to-right order; the standard system directories come after.

'`-I-`'
: Any directories specified with '`-I`' options before the '`-I-`' option are searched only for the case of '`#include "`*file*`"`'; they are not searched for '`#include <`*file*`>`'.

  If additional directories are specified with '`-I`' options after the '`-I-`', these directories are searched for all '`#include`' directives.

  In addition, the '`-I-`' option inhibits the use of the current directory as the first search directory for '`#include "`*file*`"`'. Therefore, the current directory is searched only if it is requested explicitly with '`-I.`'. Specifying both '`-I-`' and '`-I.`' allows you to control precisely which directories are searched before the current one and which are searched after.

'`-nostdinc`'
: Do not search the standard system directories for header files. Only the directories you have specified with '`-I`' options (and the current directory, if appropriate) are searched.

'`-nostdinc++`'
: Do not search for header files in the C++-specific standard directories, but do still search the other standard directories. (This option is used when building '`libg++`'.)

'`-D` *name*'
: Predefine *name* as a macro, with definition '`1`'.

'`-D` *name*`=`*definition*'
: Predefine *name* as a macro, with definition *definition*. There are no restrictions on the contents of *definition*, but if you are invoking the preprocessor from a shell or shell-like program you may need to use the shell's quoting syntax to protect characters such as spaces that have a meaning in the shell syntax.

'`-U` *name*'
: Do not predefine *name*. If both '`-U`' and '`-D`' are specified for one name, the '`-U`' beats the '`-D`' and the name is not predefined.

'`-undef`'
: Do not predefine any nonstandard macros.

'`-d`'
: Instead of outputting the result of preprocessing, output a list of '`#define`' commands for all the macros defined during the execution of the preprocessor.

'`-dM`'
: Instead of outputting the result of preprocessing, output a list of '`#define`' commands for all the macros defined during the execution of the preprocessor, including predefined macros. This gives you a way of finding out what is predefined in your version of the preprocessor; assuming you have no file '`foo.h`', the command

```
touch foo.h; cpp -dM foo.h
```

will show the values of any predefined macros.

'-dD'
: Like '-dM' except in two respects: it does *not* include the predefined macros, and it outputs *both* the '#define' commands and the result of preprocessing. Both kinds of output go to the standard output file.

'-M'
: Instead of outputting the result of preprocessing, output a rule suitable for `make` describing the dependencies of the main source file. The preprocessor outputs one `make` rule containing the object file name for that source file, a colon, and the names of all the included files. If there are many included files then the rule is split into several lines using '\'-newline.

  This feature is used in automatic updating of makefiles.

'-MM'
: Like '-M' but mention only the files included with '#include "*file*"'. System header files included with '#include <*file*>' are omitted.

'-i *file*'
: Process *file* as input, discarding the resulting output, before processing the regular input file. Because the output generated from *file* is discarded, the only effect of '-i *file*' is to make the macros defined in *file* available for use in the main input.

'-MD'
: Like '-M' but the dependency information is written to files with names made by replacing '.c' with '.d' at the end of the input file names. This is in addition to compiling the file as specified—'-MD' does not inhibit ordinary compilation the way '-M' does.

  In Mach, you can use the utility `md` to merge the '.d' files into a single dependency file suitable for using with the 'make' command.

'-MMD'
: Like '-MD' except mention only user header files, not system header files.

'-H'
: Print the name of each header file used, in addition to other normal activities.

'-imacros *file*'
: Process *file* as input, discarding the resulting output, before processing the regular input file. Because the output generated from *file* is discarded, the only effect of '-imacros *file*' is to make the macros defined in *file* available for use in the main input.

'-include *file*'
: Process *file* as input, and include all the resulting output, before processing the regular input file.

'-lang-c'
'-lang-c++'
'-lang-objc'
'-lang-objc++'
: Specify the source language. '-lang-c++' makes the preprocessor handle C++ comment syntax, and includes extra default include directories for C++, and '-lang-objc' enables the Objective C '#import' directive. '-lang-c' explicitly turns off both of these extensions, and '-lang-objc++' enables both.

These options are generated by the compiler driver gcc, but not passed from the 'gcc' command line.

'-lint' Look for commands to the program checker lint embedded in comments, and emit them preceded by '#pragma lint'. For example, the comment '/* NOTREACHED */' becomes '#pragma lint NOTREACHED'.

This option is available only when you call cpp directly; gcc will not pass it from its command line.

'-$' Forbid the use of '$' in identifiers. This is required for ANSI conformance. gcc automatically supplies this option to the preprocessor if you specify '-ansi', but gcc doesn't recognize the '-$' option itself—to use it without the other effects of '-ansi', you must call the preprocessor directly.

6.2 Predefined Macros

The standard predefined macros are available with the same meanings regardless of the machine or operating system on which you are using GNU C. Their names all start and end with double underscores. Those preceding __GNUC__ in this table are standardized by ANSI C; the rest are GNU C extensions.

__FILE__ This macro expands to the name of the current input file, in the form of a C string constant. The precise name returned is the one that was specified in '#include' or as the input file name argument.

__LINE__ This macro expands to the current input line number, in the form of a decimal integer constant. While we call it a predefined macro, it's a pretty strange macro, since its "definition" changes with each new line of source code.

This and '__FILE__' are useful in generating an error message to report an inconsistency detected by the program; the message can state the source line at which the inconsistency was detected. For example,

```
fprintf (stderr,
 "Internal error: negative string length "
         "%d at %s, line %d.",
         length, __FILE__, __LINE__);
```

A '#include' command changes the expansions of '__FILE__' and '__LINE__' to correspond to the included file. At the end of that file, when processing resumes on the input file that contained the '#include' command, the expansions of '__FILE__' and '__LINE__' revert to the values they had before the '#include' (but '__LINE__' is then incremented by one as processing moves to the line after the '#include').

The expansions of both '__FILE__' and '__LINE__' are altered if a '#line' command is used.

__INCLUDE_LEVEL__
This macro expands to a decimal integer constant that represents the depth of nesting in include files. The value of this macro is incremented on every '#include' command and decremented at every end of file.

`__DATE__` This macro expands to a string constant that describes the date on which the preprocessor is being run. The string constant contains eleven characters and looks like '`"Jan 29 1987"`' or '`"Apr 1 1905"`'.

`__TIME__` This macro expands to a string constant that describes the time at which the preprocessor is being run. The string constant contains eight characters and looks like '`"23:59:01"`'.

`__STDC__` This macro expands to the constant 1, to signify that this is ANSI Standard C. (Whether that is actually true depends on what C compiler will operate on the output from the preprocessor.)

`__GNUC__` This macro is defined if and only if this is GNU C. This macro is defined only when the entire GNU C compiler is in use; if you invoke the preprocessor directly, '`__GNUC__`' is undefined.

`__STRICT_ANSI__`
This macro is defined if and only if the '`-ansi`' switch was specified when GNU C was invoked. Its definition is the null string. This macro exists primarily to direct certain GNU header files not to define certain traditional U**x constructs which are incompatible with ANSI C.

`__BASE_FILE__`
This macro expands to the name of the main input file, in the form of a C string constant. This is the source file that was specified as an argument when the C compiler was invoked.

`__VERSION__`
This macro expands to a string which describes the version number of GNU C. The string is normally a sequence of decimal numbers separated by periods, such as '`"1.18"`'. The only reasonable use of this macro is to incorporate it into a string constant.

`__OPTIMIZE__`
This macro is defined in optimizing compilations. It causes certain GNU header files to define alternative macro definitions for some system library functions. It is unwise to refer to or test the definition of this macro unless you make very sure that programs will execute with the same effect regardless.

`__CHAR_UNSIGNED__`
This macro is defined if and only if the data type `char` is unsigned on the target machine. It exists to cause the standard header file '`limit.h`' to work correctly. It is bad practice to refer to this macro yourself; instead, refer to the standard macros defined in '`limit.h`'.

`__MSHORT__`
This macro is defined, if '`gcc.ttp`' is invoked with the '`-mshort`' option, which causes integers to be 16 bit. Please carefully examine the prototypes in the '`#include <>`' headers for types before using '`-mshort`'.

`__MINT__` This macros is defined, if '`gcc.ttp`' is invoked with the '`-mint`' option. This macros activates some portions of the header files, which are MiNT specific. Up to version 8 of the MiNT libraries and headers the header files of J.R.Bammi's libraries are compatible with the ones from Eric Smith's library. Therefore if you were writing programs for MiNT you could stick to Bammi's headers and use the '-mint' option. I don't know, if header files are still compatible with version 10 of the MiNT libraries.

`__MBASE__`
`__MBASESTR__`
hudli wirz

Apart from the above listed macros, there are usually some more to to indicate what type of system and machine is in use. For example 'unix' is normally defined on all U**x systems. Other macros describe more or less the type of CPU the system runs on. GNU CC for the Atari ST has the following macros predefined.

- 'atarist'
- 'gem'
- 'm68k'

Please keep in mind, that these macros are only defined, if the preprocessor is invoked from the compiler driver 'gcc.ttp'.

These predefined symbols are not only nonstandard, they are contrary to the ANSI standard because their names do not start with underscores. However, the GNU C preprocessor would be useless if it did not predefine the same names that are normally predefined on the system and machine you are using. Even system header files check the predefined names and will generate incorrect declarations if they do not find the names that are expected.

The '-ansi' option which requests complete support for ANSI C inhibits the definition of these predefined symbols.

7 The GNU Assembler (GAS)

Most of the time you will be programming in C. But there may certain situations, where it is feasible to write in assembler. Time is usually a main reason to dive into assembler programming, when you have to squeeze the last redundant machine cycle out of your routine, to meet certain time limits. Another reason might be, that you have to do very low level stuff like fiddling with bits in the registers of a peripheral chip. An example for low level stuff is the startup module '`crt0.o`', which is written in assembler.

If you already have some experience in assembler programming, you might miss the feature of creating macros. This is not really a lack given the fact, that the assembler originated from an U**x environment. Under this operating system there is a tools for nearly every purpose. If you were in the need of an extensive macros facility, you would use the M4 macro processor. A GNU version of the M4 macro processor exists. It should be no problem to port it to the Atari with GCC. For some macro processing tasks you just as well use the C preprocessor. What I personally miss is the ability to produce a listing, but this will be fixed with GAS 1.92.

One command line option was introduced only very lately. The changes for the option '`-m68040`' were part of update 20, which Bammi released around end of April 1992. The assembler identifies itself, when invoked with the '`-v`' option with the string '`GNU assembler version 1.38 atariST PatchLevel 2`'.

7.1 Invoking the Assembler

'`gcc-as.ttp`' supports the following command line options. The output is written to '`a.out`' by default.

'`-G`'
: assembles the debugging information the C compiler included into the output. Without this flag the debugging information is otherwise discarded.

'`-L`'
: Normally all labels, that start with a '`L`' are discarded and don't show up as symbols in the object code module. They are local to that assembler module. If the '`-L`' option is given, all local labels will be included in the object code module.

'`-m68000`'
'`-m68010`'
'`-m68020`'
'`-m68040`'
: These options modify the behavior of assembler in respect of the used CPU. The M68020, for example, allows relative branches with 32-bit offset.

'`-o`*filename*'
: writes the output to *filename* instead of '`a.out`'.

'`-R`'
: The information, which normally would be assembled into the data section of the program, is moved into the text section.

'`-v`'
: displays the version of the assembler.

'`-W`'
: suppresses all warning messages.

7.2 Syntax

The assembler uses a slightly modified syntax from the one you might know from other 68000 assemblers, which use the original Motorola syntax. The next sections trys to describe the syntax, GAS uses.

The most obvious differences are the missing '`.`' and the usage of the at sign ('`@`'). The original Motorola syntax uses the '`.`' to separate the size modifier (`b`, `w`, `l`) from the main instruction. In Motorola syntax one would write '`move.l #1,d0`' to move a long word with value 1 into register `d0`. With GAS you simple write '`movel #1,d0`'. The '`@`' is used to mark an indirection equivalent to the Motorola parentheses. To move a long word of value 1 to the location addressed by `a0`, you have to write '`movel #1,a0@`'. The equivalent instruction expressed in Motorola syntax is '`move.l #1,(a0)`'. The '`#`' indicates immediate data in both cases.

7.2.1 Register Names and Addressing Modes

The register mnemonics are `d0`...`d7` for the data registers and `a0`...`a7` or `sp` for address register and the stack pointer. `pc` is the program counter, `sr` the status register, `ccr` the condition code register and `usp` the user stack pointer.

The following table shows the operands GAS can parse. (The first part part describe the used abbreviations. The second part show the addressing modes with a equivalent C expression.)

`numb:` a 8 bit number

`numw:` a 16 bit number

`numl:` a 32 bit number

`dreg:` data register 0...7

`reg:` address or data register

`areg:` address register 0...7

`apc:` address register or PC

`num:` a 16 or 32 bit number

`num2:` a 16 or 32 bit number

`sz:` `w` or `l`; if omitted, `l` is assumed.

`scale:` 1 2 4 or 8. If omitted, 1 is assumed.

Addressing Modes:

```
Immediate Data
                #num                          --> NUM

Data- or Address Register Direct
                dreg                          --> dreg
                areg                          --> areg

Address Register Indirect
```

```
               areg@                          --> *(areg)

Address Register Indirect with Postincrement or Predecrement
               areg@+                         --> *(areg++)
               areg@-                         --> *(--areg)

Address Register (or PC) Indirect with Displacement
               apc@(numw)                     --> *(apc+numw)

Address Register (or PC) Indirect with Index (8-Bit Displacement)
(M68020 only)
               apc@(num,reg:sz:scale)         --> *(apc+num+reg*scale)
               apc@(reg:sz:scale)             --> same, with num=0

Memory Indirect Postindexed
(M68020 only)
               apc@(num)@(num2,reg:sz:scale) --> *(*(apc+num)+num2+reg*scale)
               apc@(num)@(reg:sz:scale)       --> same, with num2=0
               apc@(num)@(num2)               --> *(*(apc+num)+num2)
                                                 (previous mode without an index reg)

Memory Indirect Preindexed
(M68020 only)
               apc@(num,reg:sz:scale)@(num2) --> *(*(apc+num+reg*scale)+num2)
               apc@(reg:sz:scale)@(num2)      --> same, with num=0

Absolute Address
               num:sz                         --> *(num)
               num                            --> *(num) (sz L assumed)
```

7.2.2 Labels and Identifiers

User defined identifiers are basically defined by the same rules as C identifier. They may contain the digits 0...9, the letters A...z and the underscore and must not start with a digit. Identifier, which end with a ':' are labels. A special form of labels starts with a 'L' or consists of only a digit. Both types are local labels, which disappear, when the assembly is complete (unless the '-L' option was specified). They can't be used to resolve external references. The 'L' type label are referenced by their name, just as any other label. The digit type labels form a special kind of local labels. You might also call them temporary labels. They are especially useful when you have to create small loops, which poll a peripheral or fill a memory area. They are referenced by appending either a 'f', for a forward reference, or a 'b', for a backward reference, to the digit. Lets look at the following example, which is used to split a memory area starting at 0x80000. All data on an even addresses is copied to the area starting at 0x70000; all data from odd addresses goes to the area starting at 0x78000.

```
start:
        lea     0x80000,a0
        lea     0x70000,a1
        lea     0x78000,a2
        movel   #0x7fff,d5
0:                                | label '0' is defined
```

```
        moveb   a0@+,a1@+
        moveb   a0@+,a2@+
        dbra    d5,0b           | reference of label '0'
        ...
```

The label '0' is referenced 3 lines later by '0b', since the reference is backward. You can use the label '0' again at a later time to construct more such loops. Since this temporary labels are restricted to one digit in length, you can only build constructs, which use 10 temporary labels at the same time.

7.2.3 Comments

The above example also shows, that comments start with a '|'. '#' is also used to mark a comments. The C compiler and the preprocessor generate lines, that start with a '#'.

7.2.4 Numerical and String Constants

Numerical values are given the same way as in a C programs. By default number are taken to be decimal. A leading '0' denotes an octal and a '0x' a hexadecimal value. Floating point numbers start with a '0f'. The optional exponent starts with a 'e' or 'E'.

String constants are equivalent to C defined. They are enclosed in '"'. Some special character constants are defined by '\' and a following letter. These characters are possible:

`\b` Backspace, Code 0x08

`\t` Tab, Code 0x09

`\n` Line Feed, Code 0x0a

`\f` Form Feed, Code 0x0c

`\r` Carriage Return, Code 0x0d

`\\` Backslash itself

`\"` Double Quote itself

number were *number* is a octal number with up to 3 digits specifying the character code.

7.2.5 Assignments and Operators

A '=' is used to assign a value to a Symbol.

```
Lexp_frame = 8
```

This is equivalent to the 'equ' directive other assemblers use.

GAS supports addition (+), subtraction (-), multiplication(*), division (/), right shift (>), left shift (<), and (&), or (|), not (!), xor (^) and modulo (%) in expressions. The order of precedence is

```
          Rank    Examples
lowest     0       operand, (expression)
           1       + -
```

```
2       & ^ ! |
3       * / % < >
```

Parentheses are used to coerce the order of evaluation.

7.2.6 Segments, Location Counters and Labels

A program written in assembler language may be broken into three different segments; the TEXT, DATA and BSS sections. Pseudo opcodes are used to switch between the sections. The assembler maintains a location counter for each segment. When a label is used in the assembler input, it is assigned the current value of the active location counter. The location counter is incremented with every byte, that the assembler outputs. GAS actually allows you to have more than one TEXT or DATA segment. This is so to ease code generation by high level compilers. The assembler concatenates the different sections in the end to form continuous regions of TEXT and/or DATA. When you do assembly programming by hand you would stick to the pseudo opcodes '`.text`' or '`.data`', which use text or data segment with number 0 by default.

7.2.7 Types

Symbol and Labels can be of one of three type. A Symbol is *absolute*; when it's values is known at assembly time. A assignment like '`Lexp_frame = 8`' gives the symbol '`Lexp_frame`' the absolute value 8. A symbol or label, which contains an offset from the beginning of a section, is called *relocatable*. The actual value of this symbol can only be determined after the linking process or when the program is running in memory. The third type of symbols are *undefined externals*. The actual value of this symbol is defined in an other program.

When different types of symbols are combined to form expressions the following rules apply: (`abs` = absolute, `rel` = relocatable, `ext` = undefined external)

```
abs + abs => abs
abs + rel = rel + abs => rel
abs + ext = ext + abs => ext

abs - abs => abs
rel - abs => rel
ext - abs => ext
rel - rel => abs
(makes only sense, when both relocatable expression are relative to
same segment)
```

All other possible operators are only useful to form expressions with *absolute* values or symbols.

7.3 Supported Pseudo Opcodes (Directives)

All pseudo opcodes start with a '.'. They are followed by 0, 1 or more expressions separated by commas (depending on the directive). The following table omits the pseudo opcodes, which include special information for debugging purposes (for GDB).

`.abort` aborts the assembly on the point.

`.align` *integer*
: aligns the current segment in size to *integer* power of 2. The maximum value of *integer* is 15. The lines

```
.text
some code ...
.align 10                 | 2^10 = 1024
.data
some more code ...
.align 10                 | 2^10 = 1024
```

will create text and data sections, which both have the size 1024, although the actual code, that goes into the sections may be smaller.

`.ascii` *string* `[,`*string*`,...]`
: includes the *string*('s) in the assembly output.

`.asciz` *string* `[,`*string*`,...]`
: This directive is the same as above, but additionally appends a '`\0`' character to the string.

`.byte` *expr* `[,`*expr*`,...]`
: puts consecutive bytes with value *expr* into the output.

`.comm` *identifier*`,`*integer*
: creates a common area of *integer* bytes in the current segment, which is referenced by *identifier*. The *identifier* is visible from the outside of the module. It can therefore be used to resolve external reference from other modules.

`.data [`*integer*`]`
: switches to DATA section *integer*. If *integer* is omitted, data section 0 is selected.

`.desc`
: Whatsit good for ???

`.double` *double* `[,`*double*`,...]`
: puts consecutive doubles with value *double* into the output.

`.even`
: sets the location counter of the current segment to the next even value.

`.file`
`.line`
: If a file is assembled, which was generated by a compiler or preprocessed by the C preprocessor, the input may contain lines like '`# 132 stdio.h`'. These lines are change by the assembler to the form

```
.line 132
.file stdio.h
```

`.fill` *count*`,`*size*`,`*expr*
: puts *count* areas with *size* into the output. Each area contains the value *expr*. *size* may be an even number up to or equal to 8. The line

```
.fill 3, 4, 0xa5a
```

would put the following byte sequence in the output ('`|`' is only used to mark the size of the area.)

```
00 00 0a 5a | 00 00 0a 5a | 00 00 0a 5a
```

`.float` *float*`[,`*float*`,...]`
: puts consecutive floats with value *float* into the output.

`.globl` *identifier*`[,`*identifier*`,...]`
: When labels or identifiers are assigned, they are only locally defined. The `.globl` directive gives *identifier* external scope. The label can therefore be used to resolve external references from other modules. *identifier* don't have to be assigned in the current module, but can be defined in another module.

`.int` *expr*`[,`*expr*`,...]`
: puts consecutive integers (32 bit) with value *expr* into the output.

`.lcomm` *identifier*`,`*integer*
: is basically the same as `.comm`, except that area is allocated in the BSS segment. The scope of *identifier* is only local (only visible in the module, where it is defined).

`.long` *expr*`[,`*expr*`,...]`
: same as `int`.

`.lsym` *identifier*`,`*expr*
: sets the local *identifier* to the value of *expr*. The *identifier* is referenced by preceding it with a '`L`'. (`L`*identifier*) (When I tried this, the linker threw a bomb. Trying again crashed the system.)

`.octa`
: Whatsit good for ???

`.org` *expr*
: sets the location counter of the current segment to *expr*.

`.quad`
: Whatsit good for ???

`.set` *identifier*`,`*expr*
: sets *identifier* to the value of *expr*. If *identifier* is not explicitly marked external by the `.globl` directive, is has only local scope.

`.short` *expr*`[,`*expr*`,...]`
: puts consecutive shorts (16 bit) with value *expr* into the output.

`.space` *count*`,` *expr*
: puts *count* consecutive number of bytes with value *expr* into the output. The line

```
.space 5,3
```

is equivalent to

```
.byte 3, 3, 3, 3, 3
```

The `space` directive is a special form of the `fill` directive.

`.text [`*integer*`]`
: switches to TEXT section *integer*. If *integer* is omitted, text section 0 is selected.

`.word` *expr*`[,`*expr*`,...]`
: same as `.short`.

7.4 Converting Motorola Assembler Syntax

There is a utility, which is part of the MiNT source distribution since around 0.96 pl14. It is called '`asmtrans.ttp`'. This utility is used to convert some of MiNT's source files, which are written in standard Motorola assembler syntax into GAS syntax. Eric R. Smith wrote this little utility, so that he doesn't have to maintain the assembler files in more than on syntax versions.

The general syntax to invoke '`asmtrans.ttp`' is:

`asmtrans [`*options*`]` *infile*

The following options are possible:

'`-gas`' converts the input file into GAS syntax, also known as MIT syntax. Apart from the modification of the standard assembler intructions the following pseude opcodes or assembler directives are mapped:

- '`XREF`' ⇒ '`.globl`'
- '`XDEF`' ⇒ '`.globl`'
- '`TEXT`' ⇒ '`.text`'
- '`DATA`' ⇒ '`.data`'
- '`BSS`' ⇒ '`.data`'
- '`END`' ⇒ '`| END`'
- '`dc.l`' ⇒ '`.long`'
- '`dc.w`' ⇒ '`.word`'
- '`dc.b`' ⇒ '`.byte`'

'`-asm`' converts the input file into ASM syntax (I think the assembler of the Lattice compiler is meant). This option only causes some assembler directives to be changed.

- '`TEXT`' ⇒ '`SECTION TEXT`'
- '`DATA`' ⇒ '`SECTION DATA`'
- '`BSS`' ⇒ '`SECTION BSS`'

'`-purec`' converts the input file into Pure C syntax for the assembler from the Pure C compiler. As far as I can tell, the only effect, that this option has is, that a leading underscore is removed from every label.

'`-o` *outfile*'
writes the output into the file *outfile*.

'`asmtrans.ttp`' supports some simple directives to allow conditional processing. The directives are very similar to the ones of standard C preprocessor except that they don't start with '`#`' but with '`%`'. The following directives are supported.

`%define` *name*

`%include` *file*

`%ifdef` *name*

`%ifndef` *name*

`%else`

`%endif`

The directives behave equivalent to their cpp counter parts. The definition *name* may consist of the same characters as any C symbol. It may begin with a '_' or a letter and continue with either '_', letter or digit.

8 The Utilities

This chapter describes the programs, which don't actually convert the source code into object code, but instead combine several object code modules to a runnable program or an object code library. Other programs can be used to print symbol information from either the object code or the executable. The last group of utility programs modify the executables in terms of memory usage and startup time.

8.1 The Linker '`gcc-ld.ttp`'

A linker combines several object modules and extracts modules from a library to produce a runnable program. During this process all undefined symbol references are resolved. Additionally all sections from the object modules, which belong to either the TEXT, DATA or BSS are moved to the correct program segment. For example, all areas of all the object code modules, which have the type TEXT, are moved to form one large TEXT section. The same applies to the DATA and BSS sections.

For the most time you don't have invoke the linker explicitly. The compiler driver does the job for you. But in case you have to, the general syntax is:

`gcc-ld [`*options*`] $GNULIB\crt0.o` *file*`.o -l`*library*

The above syntax assumes, that the executable is produced from C source code, which normally makes is necessary to link a startup module and a library. If an executable from a self contained assembler text is to be created, the startup module '`crt0.o`' and the library might be missing. '`gcc-ld.ttp`' creates a file '`a.out`' by default. The linker can also append a DRI compatible or an extended symbol table to the executable.

'`gcc-ld.ttp`' supports the following command line options. The options '`-f`' and '`-h`' were introduced with update 29, which brought the utilites to patchlevel 34 (March 1993).

'`-f`*load flags*'
: Set the program load flags to *load flags*. The default program load flags is 7 (run and malloc in/from TT ram, fastload). (See Section 8.3.6 [toglclr.ttp], page 58 for additional info)

'`-h`*altheap size*'
: Set the minalt size in the executable header to *altheap size*. The default value is zero. Remember that value is specified in 128k units. What this means is (quoting mintsrc/mem.c): If (flags & F_ALTLOAD == 1), then we might decide to load in alternate RAM if enough is available. "enough" is: if more alt ram than ST ram, load there; otherwise, if more than (minalt+1)*128K alt ram available for heap space, load in alt ram ("minalt" is the high byte of flags).

'`-l`*library*'
: Search *library* to satisfy unresolved references. The environment variable `GNULIB` is used to locate the library. `GNULIB` contains a '`,`' or '`;`' separated list of paths, each path without a trailing slash or backslash.

'`-L`*directory*'
: Includes *directory* in the search path to locate a library.

'-M' During the linking process extensive information about the encountered symbols is displayed.

'-n'

'-G' Instead of the standard DRI compatible symbol table, an extended symbol table is written, which allows symbol names to be up to 22 characters long. Most of the other utility programs have been updated to work with this format. The most benefit you get with 'gprof.ttp' and 'szadb' (the adb-like debugger, originally written for the Sozobon C compiler by Johann Rueg and Don Dugger and later significantly improved by Michal Jaegermann).

'-o*filename*'
The resulting output of the linking process is written to *filename* instead to 'a.out'.

'-s' prevents the linker from attaching a symbol table to the executable.

'-t' During the linking process the files loaded and the modules extracted from a library are displayed.

'-x' This option discards all local symbols from the DRI symbol table. All global symbols are left in place.

'sym-ld.ttp'

'sym-ld.ttp' is a special version of the linker. His sole purpose is to create a special symbol file used by the GNU debugger. The following example show the usage. ('$' is the prompt of a CLI, '*' is the GDB prompt, '#' marks a comment)

```
$ gcc -c -g foo.c      # compile 'foo.c'
$ gcc -o foo.prg foo.o -lgnu # link with normal 'gcc-ld.ttp'
$ sym-ld -o foo.sym $(GNULIB)\crt0.o foo.o -lgnu
                       # (or -lgnu16 if you use -mshort)
                       # link with 'sym-ld.ttp' to get symbol file
$ gdb
* exec-file foo.prg    # executable ('gcc-ld.ttp' linked Atari
                         executable)
* symbol-file foo.sym  # symbols file ('sym-ld.ttp' '-o' linked)
* run
* <start doing gdb commands here>
 ...
* q
$                      # back
```

Note the line in the example, where 'sym-ld.ttp' is invoked. A library 'gnugdb.olb' is used to create the symbol file. This is just like the normal library 'gnu.olb' except, that is was compiled with the '-g' option. If you don't have this library, use the normal library ('-lgnu'). In this case you can't single step through library functions at the source level. Also note, that 'sym-ld.ttp' is invoked without the '-r' option. This option was only necessary for some very early versions of 'gdb'.

For a bit more detailed info about debugging with 'gdb' turn to chapter See Chapter 9 [Debugging], page 62.

8.2 The Archiver 'gcc-ar.ttp'

The archivers main purpose is to make things in programming life easier. The archiver combines several object modules into one large library. At a later time the linker will then retrieve the modules needed to resolve all references. Without the library you would have to supply all modules by hand on the command line or the linker would have to search through all the files to resolve the references (The library '`gnu.olb`' contains around 150 modules).

The general syntax for invoking '`gcc-ar.ttp`' is:

`gcc-ar` *option* [*position*] *library* [*module*]

The *option* specifies the action to be taken on the *library* or a *module* of that *library*. *option* also includes modifiers for the action. The optional *position* argument is a member of the *library*. It is used, to mark a specific position in the *library*; an '`add`' operation would than place a new module before or after that *position*. The next argument specifies the library. The recommended naming convention for the creation of a new libraries is '*library*`.olb`'. If you don't use this convention, the compiler driver '`gcc.ttp`' will have trouble to find them. *module* is usually an object code file generated by the compiler.

'`gcc-ar.ttp`' supports the following command line options. If you don't use a *position* the named module is appended or moved to the end of the library

'`a`' The '`add`', '`replace`' or '`move`' operation should place the *module* **after** *position*.

'`b`' The '`add`', '`replace`' or '`move`' operation should place the *module* **before** *position*.

'`c`' If the specified *library* does not exist, it is silently created. Without this option '`gcc-ar.ttp`' would give you a notice, that it created a new library.

'`d`' deletes *module* from the *library*.

'`i`' This is the same as option '`b`'.

'`l`' This option is ignored. (Why is there in the first place ??)

'`m`' Move a member around inside the library.

'`o`' preserves the modification time of a module, that is extracted from the library.

'`p`' This option pipes the specified *module* directly to '`<stdout>`'.

'`q`' A quick append is performed.

'`r`' causes *module* to be replaced. If the named module is not already present, it is appended. This is also the default action, when no *option* is given.

'`s`' creates special member in the library called '`__.SYMDEF`', which contains a directory of the external names defined by all the other members.

'`t`' lists the members, that are currently in the *library*. If the option '`v`' is also given, additional information about file permissions, user- and group-id's and last modification date of the members are displayed. Of course, file permissions and user- and group-id's don't make much sense on the Atari ST.

'u' If this option is given, an existing module in the library is only replaced, if the modification time of the new module is newer than the modification time of the one already in the library.

'v' gives you some additional information depending on the operation, that currently performed.

'x' Extract *module* from the *library*.

8.3 Miscellaneous Utilities

There are some additional utility programs, which are used to either modify an executable, or which give you some info about either object files, object libraries or executables.

Some programs, which are described in the following sections can only work correctly, if the symbol table is still attached to the executable. So, if you want to modify the memory usage of a program (with '`fixstk.ttp`') at a later time, you should keep the unstripped executables around or use the command '`xstrip.ttp`' and keep only the `_stksize` symbol.

8.3.1 '`gcc-nm.ttp`'

'`gcc-nm.ttp`' generates a listing about the symbols found in a GNU object file or library. To list the symbols, which are attached to a TOS executable use '`cnm.ttp`'.

The output of '`gcc-nm.ttp`' looks like the following sample:

```
00000870 b _Lbss
         U _alloca
000003b4 t _glob_dir_to_array
00000532 T _glob_filename
00000248 T _glob_vector
         U _malloc
0000086c D _noglob_dot_filenames
         U _opendir
         U _readdir
00000000 t gcc_compiled.
```

The first column displays the relative address of that symbol in the object file. If the symbol has the type `U` (undefined external) the space in left blank. The next column shows the type of the symbol. In general, symbols, which have an external scope (visible for other object module) are marked with an uppercase letter. Symbols, which are local to the object file are marked with lowercase letters. The following letters are possible:

'C' marks variables, which are defined in that source module, but not initialized. A declaration like

```
int variable;
```

would create a line marked with a '`C`'. The first column would show the size of that variable in bytes instead of the relative address in the object module.

'b' Variables, which are declared with

```
static int variable;
```

are displayed with a '`b`'.

'D' marks variables, which are initialized at declaration time. A declaration like

```
int variable = 1;
```

would show as a line with a 'D' in it.

'd' Variables, which are initialized at declaration time declared are displayed with a 'd'. A declaration like

```
static int variable = 1;
```

would create a line marked with a 'd'.

't,T' mark text (in other words: actual program code). Functions in your C source, which have the storage class `static`, would be displayed with a 't'. All other functions in that source module, which are visible to other modules, would show up with a 'T'.

'U' All functions, which are defined in other modules and referenced in this module, are displayed with a 'U'.

The last column shows the symbol name.

'gcc-nm.ttp' supports the following command line options.

'-a' In case a file is compiled with the '-g' or '-gg' option, special information for debugging purposes is included in the object code. This information is listed by supplying the '-a' option.

'-g' This option restricts the output to include only symbols, which have an external scope.

'-n' Without any options the output is sorted in ascii order. By supplying the '-n', the listing is sorted in numerical order by the addresses in first column.

'-o' If this option is given, every output line is preceded by a filename in the form '*file*:', naming the file in which the symbol appears. If the file to be listed, is an archive, the line begins in the form '*library*(*member*):'.

'-p' The symbols are listed in the order as they appear in the object code module.

'-r' The output is sorted in reverse ascii order.

'-s' Archives may contain a special member called '__.SYMDEF'. Don't ask me about it purpose. Anyway, using this option show the content of this member.

'-u' Only undefined symbols are listed.

8.3.2 'cnm.ttp'

'cnm.ttp' prints the symbols which are attached to an executable. This can be either a standard DRI or an extended symbol table. To list the symbols in an object file or library use 'gcc-nm.ttp'. The following options are supported by 'cnm.ttp':

'g' List only those symbols, which are flagged as global.

'u' List the symbols in the order as they appear in the symbol table (unsorted).

Without any options, all symbols, which appear in the symbols table, are listed and sorted according to their value.

8.3.3 'gcc-size.ttp'

'gcc-size.ttp' reports the sizes of the sections (text, data and bss) in an GNU object file or library. 'gcc-size.ttp' has no additional options.

The following example shows the output, when 'gcc-size.ttp' is executed with an object file and an object library as arguments:

```
c:\gnu\lib => gcc-size crt0.o iio.olb
text data bss dec hex
960 12 8 980 3d4 crt0.o
1580 0 0 1580 62c iio.olb(idoprnt.o)
1884 0 0 1884 75c iio.olb(iscanf.o)
116 0 0 116 74 iio.olb(fprintf.o)
164 0 0 164 a4 iio.olb(sprintf.o)
164 0 0 164 a4 iio.olb(fscanf.o)
144 0 0 144 90 iio.olb(sscanf.o)
```

The name in braces in the last column is the member of the library.

8.3.4 'printstk.ttp'

'printstk.ttp' works basically the same way as 'fixstk.ttp', but displays the current value at the location `_stksize` or `_initial_stack`. The usage is:

`printstk` *filename* ...

If 'printstk.ttp' is used on some of the executables of the GCC distribution, you should see a value of '-1', which means that all available memory is used by the program (at least for the programs 'gcc-cpp.ttp' and 'gcc-cc1.ttp').

8.3.5 'fixstk.ttp'

'fixstk.ttp' is used to modify the current stacksize of an executable. It does this by looking up the symbol `_stksize` in the symbol table portion of the file and than changes the values of the location where `_stksize` points to. The usage is:

`fixstk` *size file* ...

size is the stacksize in Bytes, KBytes or MBytes. To specify *size* in Kbytes or Mbytes, append a 'K' or a 'M' to the integer number.

For dumping applications like Scott Kolodzieski's port of GNU Emacs 18.57 'fixstk.ttp' looks up the symbol `_initial_stack` instead of `_stksize`.

```
fixstk 128K gcc-as.ttp
```

sets the stacksize of 'gcc-as.ttp' to 128 Kbytes.

8.3.6 'toglclr.ttp'

The general syntax to invoke 'toglclr.ttp' is:

`toglclr [`*options*`]` *file* ...

'toglclr.ttp' supports the following options:

'-fload'   Toggle the 'fast load' flag.

'`-frun`' Toggle the '`fast run`' flag.

'`-fram`' Toggle the '`fast ram malloc`' flag.

'`-fshare`' Toggle the '`shared text`' flag.

'`-private`'
Set memory protection to PRIVATE. Only the process itself and the AES have access to the process's memory. This is the normal state.

'`-global`' Set memory protection to GLOBAL. Every process has unrestricted access to the process's memory.

'`-super`' Set memory protection to SUPER. Every process in supervisor mode has unrestricted access to the process's memory.

'`-readable`'
Set memory protection to PRIVATE/READABLE. Every process has access to the process's memory, but only for read operations.

The last four option are not yet part of the official '`toglclr.ttp`'. I've send the changes to J.R.Bammi. They will probably be part of an future update.

If TOS launches an application, it clears all memory starting from the BSS section to the end of the TPA. With earlier TOS versions (pre TOS 1.4) this could take quite a considerable amount of time. The clearing algorithm was improved during the different TOS releases, but it is still used, although **most** of the existing programs don't need a cleared memory. Well, most is not all; therefore for compatibility sake the feature will stay in place.

With TOS 1.4 you can keep the GEMDOS loader from clearing all memory. The long word with offset 0x16 in the program header is used to determine whether the memory should be cleared or not. Setting the bit 0 of this longword to 1 prevents the loader from clearing all memory. '`toglclr.ttp`' serves exactly that purpose, namely toggling this long word.

TOS 2.x and 3.x gave another two bits in the above mentioned longword a meaning. The '`fast run`' bit 1 is used to determine, if a program should be started in ordinary ST-ram (bit 1 = 0) or in alternate ram. In case of the TT or the SST68030 from Dave Small, this is ram which is not slowed down by any video hardware.

The '`fast ram malloc`' bit 2 determines, if any subsequent malloc's, which a program might do, should be satisfied from slow ST-ram (bit 2 = 0) or from alterate ram. All these flags have been introduced to increase compatibility between the different TOS versions.

To toggle the '`shared text`' bit only makes sense, if you'r using MiNT 0.96 pl14 and higher or MultiTOS. An additional requirement is, that the program is compiled with the GCC '`-mbaserel`' option and linked with the corresponding libraries. In essence this means, if the '`shared text`' bit is set, multiple processes of the same program execute on the same text (program code) segment in memory.

Using the options, which modify the memory protection of the program shows only effect, if you're running MultiTOS on a machine with MMU (M68030).

8.3.7 'xstrip.ttp'

'xstrip.ttp' removes the symbol table from a TOS executable file. The default behaviour, which is to completely remove the symbol table (except the symbol `_stksize`), may be modified by specifying additional command line optione. The systax for the 'xstrip.ttp' command is:

xstrip [-a] [-g] [-k] [-l *names*] [-t] *file* ...

'-a' Specifying this option *completely* removes the symbol table.

'-g' This option causes 'xstrip.ttp' to keep all global symbols.

'-k' keeps the `_stksize` symbol, so that the stack size can be adjusted even for a nearly-stripped GCC produced executables.

'-l *names*'
keeps all symbols listed in a file *names* (one symbol per line).

'-t' to strip TurboC / PureC style executables.

Both '-k' and '-l' options convert the extended symbols into regular ones (DRI compatible).

8.3.8 'size68.ttp'

'size68.ttp' is used to print information, which is found in the the header of an executable program file. A sample output is shown in the following lines.

```
c:\ => size68 temacs
temacs:
        text size       255448
        data size       161044
        bss size        7684
        symbol size     38388
        File is relocatable.
        BSS and high mem cleared on startup.
        File is loaded into alternate ram
         and allocates memory from alternate ram.
        File runs with 'private' memory protection (under MultiTOS).
```

The value of '`text size`' is the actual size of the program code; the TEXT segment. The value of '`data size`' gives the size of initialized data; the DATA segment. For example, if you define a variable '`char array[10] = "foobar";`', the string '`foobar`' is moved to the data segment during the linking process. The value of '`bss size`' is the size of the BSS segment. If you define a global variable '`char array[10];`', this variable '`array`' would end up the BSS segment. The BSS segment is initialized to zero from the GEMDOS loader, when the program is loaded into memory. The memory usage during the programs runtime can't simply be calculated by adding the three values, since this doesn't take into account the memory, which might be dynamically allocated.

The value of '`symbol table`' is the size of the symbol table, which is appended to the three segments. The symbol table is only used, when the program is invoked under the control of a debugger. The symbol table doesn't use up any memory, when the program is launched from the desktop or a CLI. The next line says, that the program file is relocatable. As far as I know is every program file relocatable on the ST. The follwoing line indicates, that the BSS section and the all available memory (`high mem`) is cleared upon startup. On systems with lots of memory, this can take quite a bit of time. You can keep the GEMDOS loader from clearing all memory by toggling a bit in the header. See `toglclr.ttp` for more info.

9 Debugging Programs

In general, you have two choices for debugging; machine and source level debugging. Most of the time you will prefer the source level debugging.

This chapter is not ready yet. If you have some ideas, what should go into this section, please tell me.

Mention GDB 3.5 here.

Concept Index

(Index is nonexistent)

Index of all Command Line Options

Table of Contents