The Definitive Guide to GCC

KURT WALL AND WILLIAM VON HAGEN

Apress™

ISBN (pbk): 1-59059-109-7

Printed and bound in the United States of America 12345678910

Technical Reviewer: Gene Sally

Editorial Board: Steve Anglin, Dan Appleman, Gary Cornell, James Cox, Tony Davis, John Franklin, Chris Mills, Steven Rycroft, Dominic Shakeshaft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Sofia Marchant

Copy Editor: Ami Knox

Production Manager: Kari Brooks

Production Editor: Janet Vail

Proofreader: Elizabeth Berry

Compositor and Artist: Kinetic Publishing Services, LLC

Indexer: Valerie Perry

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

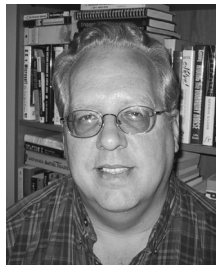
Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

About the Authors

Kurt Wall first touched a computer in 1980 when he learned FORTRAN on an IBM mainframe of forgotten vintage; happily, computer technology has improved considerably since then. A professional technical writer by trade and a historian by training, Kurt has a diverse working history. These days, Kurt works for TimeSys Corporation in Pittsburgh, Pennsylvania. His primary responsibility is managing TimeSys' Content Group, or as it is known in-house, the artists' colony. In addition to directing production of the technical and end-user documentation for TimeSys' embedded Linux operating system and development tools, he also writes much of the documentation for TimeSys' embedded Linux products. Kurt has written all or parts of 15 books on Linux system administration and programming topics. In his spare time . . . he has no spare time. Kurt, who dislikes writing about himself in the third person, receives entirely too much e-mail at kwall@kurtwerks.com.



Bill von Hagen holds degrees in computer science, English writing, and art history. Bill has worked with Unix systems since 1982, during which time he has been a system administrator, writer, systems programmer, development manager, drummer, operations manager, and (now) product manager. Bill has written a number of books including *Hacking the TiVo, Linux Filesystems, Installing Red Hat Linux 7,* and *SGML for Dummies,* contributed to *Red Hat 7 Unleashed,* and coauthored the *Mac OS X Power User's Guide* with Brian Proffitt. Bill has written articles and software reviews for publications including *Linux Magazine, Mac Tech, Linux Format* (UK), and *Mac Directory.* He has also written extensive online material for CMP Media, Linux Planet, and Corel. An avid computer collector specializing in workstations, he owns more than 200 computer systems. You can contact Bill at wvh@vonhagen.org.

Introducing GCC and *The Definitive Guide to GCC*

This book, *The Definitive Guide to GCC,* is about how to build, install, customize, use, and troubleshoot GCC 3.x, the GNU Compiler Collection version 3.x. GCC has long been available for most major hardware and operating system platforms and is often the preferred compiler for those platforms, at least among users. As a general purpose compiler, GCC produces high-quality, fast code. Due to its design, GCC is easy to port to different architectures, which contributes to its popularity. GCC, along with GNU Emacs, the Linux operating system, the Apache Web server, the Sendmail mail server, and the BIND DNS server, is one of the showpieces of the free software world and proof that sometimes you *can* get a free lunch.

Why a Book About GCC?

We wrote this book, and you should read it, for a variety of reasons: it covers version 3.x; it is the only book that covers general GCC usage; and it is better than GCC's own documentation. You will not find more complete coverage of its features, quirks, and usage anywhere else collected in a single volume. Moreover, save for GCC's own documentation, when we first started writing, no existing book was dedicated solely to using GCC—at most it got one or two chapters in programming books and only a few paragraphs, perhaps an entire section, in other, more general titles. GCC's existing documentation, although thorough and comprehensive, targets a programming-savvy reader. Although there is nothing wrong with this approach—indeed, it is the proper approach for advanced users—GCC's own documentation leaves the great majority of its users out in the cold. Fully half of *The Definitive Guide to GCC* is tutorial and practical in nature, explaining why you use one option or why you should not use another one. Showing you how to use GCC is this book's primary goal, but we also hope to explain how GCC works without descending into the minutiae of compiler theory.

The Definitive Guide to GCC is the first and only title that provides a general GCC reference and tutorial. Not surprisingly, our own curiosity provided the original motivation to write about GCC—we suspected that we were barely

scratching the surface of GCC's capabilities, and it did not take us long to discover that we were right. Most people, including many programmers, use GCC the way they learned or were taught to use it. That is, many GCC users treat the compiler as a black box, which means that they invoke it by using a small and familiar set of options and arguments they have memorized, shove source files in one end, and then receive a compiled, functioning program from the other end. Just as often, many end users, particularly nonprogrammers, use higher level tools, such as the `make` utility or RPM. Such tools make GCC easier to use because they hide GCC behind a simpler interface. Of course, there is nothing wrong with this utilitarian approach. RPM, `make`, and other build automation tools are valuable and essential tools, but they have the unfortunate side effect of hiding GCC's power and flexibility. So, another goal we had in mind when writing *The Definitive Guide to GCC* was to remove the obscurity, show the power, reveal what the high-level tools like `make` and RPM do behind the scenes, and, in the process, give you the opportunity to understand what the options you use by rote when invoking GCC directly actually mean and do.

Inveterate tweakers, incorrigible tinkers, and the just plain adventurous among you will also enjoy the chance to play with the latest and greatest version of GCC and the challenge of bending a complex piece of software to your will, especially if you have instructions that show you how to do so without harming your existing system.

What You Will Learn

The Definitive Guide to GCC follows the standard tutorial format, proceeding from simple to complex, each chapter building on the material in preceding chapters. You will start with basic GCC invocation for compiling C and C++ source code, move on to intermediate operations such as controlling the compilation process and defining the type of output GCC produces, and finish up with advanced topics such as code optimization, test coverage, and profiling. You will also learn how to download, compile, and install GCC from scratch, a poorly understood procedure that, until now, only the most capable and confident users have been willing to undertake. Programmers and advanced or merely inquisitive readers will appreciate exploring the extensions to the C and C++ programming languages that GCC supports. Developers will also discover how to use libtool and the GNU autoconfiguration tools to simplify configuring software to build with GCC. Finally, the book veers back to its focus for a more general audience by providing a complete summary of the GCC's command-line interface, a chapter on troubleshooting GCC usage and installation, and another chapter explaining how to use GCC's online documentation.

What You Need to Know

This is an end user's book, intended for anyone using GCC. Whether you are a casual end user who only occasionally compiles programs, an intermediate user using GCC frequently but lacking much understanding of how it works, or a programmer seeking to exercise GCC to the full extent of its capabilities, you will find information in this book that you can use immediately. Because Linux and Intel x86 CPUs are so popular, we have assumed that most of you are using one version or another of the Linux operating system running on Intel x86 and compatible system. Nevertheless, most of the material will be GCC specific, rather than Linux or Intel specific, because GCC is largely independent of operating systems and CPU features in terms of its usage.

What do you need to know to benefit from this book? Well, knowing how to type is a good start because GCC is a command-line compiler. Accordingly, you should be comfortable with working in a command-line environment, such as a terminal window or a Unix or Linux console. You need to be computer literate, too, and the more experience you have with Unix or Unix-like systems, such as Linux, the better. If you have downloaded and compiled programs from source code, you will be familiar with the terminology and processes constantly mentioned in the text. If, on the other hand, this is your first foray into working with source code, Chapters 2 and 3 will get you up and running quickly. You do not need to be a programmer, know how to write C or C++, or how to do your taxes in hexadecimal. If you have experience using a compiled programming language, though, this experience will help you. The only other prerequisite you need in order to benefit from this book is access to a system with GCC, but not even *that* requirement is absolute—jump ahead to Chapter 1 to find out how to download, build, and install GCC from scratch using your system's native compiler.

You should know how to use a text editor, such as vi, pico, or Emacs, if you intend to type the listings and examples yourself in order to experiment with them. Because the source and binary versions of the GCC are usually available in some sort of compressed format, you will also need to know how to work with compressed file formats, usually gzipped tarballs, although the text will explain how to do so.

What *The Definitive Guide to GCC* Does Not Cover

As an end user's book on GCC, a number of topics are outside this book's scope. In particular, it is not a primer on C or C++, although the examples used to demonstrate compiler features are written mostly in C (with a few examples using C++). Similarly, the text uses the `make` utility to simplify compilation, but using `make` is not covered beyond explaining the examples. Strictly speaking, as you will learn in the next chapter, GCC is a collection of front-end, language-specific interfaces to a common back-end compilation engine. The list of compilers includes C, C++,

Objective C, FORTRAN, Ada, and Java, among others, but the coverage in this book is generally limited to GCC's usage for C and C++, which reflects its use by the vast majority of its users. Compiler theory also gets short shrift in this book, because 98 percent of you could care less about compiler guts; the other 2 percent of you will have to satisfy yourselves with the limited material describing GCC's architecture and overall compilation workflow. That said, it is difficult to talk about using a compiler without skimming the surface of compiler theory and operation, so you will learn a few key terms and concepts as necessary.

The Definitive Guide to GCC, Chapter by Chapter

This introduction justifies our existence, or at least why you should buy the book. Seriously, the first half of this introduction details what you will learn, or at least what it proposes to explain, the assumptions made about your experience and knowledge of things digital, and the topics not covered for reasons of space or relevance. It also briefly describes the contents of each chapter. The second half of this introduction looks a little harder at what GCC is and does and includes the obligatory history of GCC. GCC being one of the GNU project's marquis products, the final pages of this introduction discuss GCC's development model in order to help you understand why it has the features it has, why it lacks other features, how to submit bug reports, and how you can participate in its development, if you are so inclined.

Conventional wisdom is that building and installing GCC from scratch should not be undertaken except by experts because of the possibility of rendering your system an inert lump of plastic, copper, and silicon. While compiling GCC from scratch clearly falls into the nontrivial task category, you will learn in Chapter 1, "Building GCC," that conventional wisdom has led you astray: building GCC from source is considerably simpler and more straightforward than commonly believed, due in large part to the excellent work of GCC's developers in automating the process. Chapter 1 shows you how to download, compile, test, and install GCC 3.x on a Unix or Linux system. Because the 3.x release represents such a milestone in GCC development, Chapter 1 also devotes considerable time and space to exploring GCC 3.x's new features and to highlighting changes and improvements relative to version 2.95. In a similar vein, Chapter 2, "Installing GCC on DOS and Windows Platforms" shows you how to get GCC 3.x up and running on a DOS or Windows system using DJGPP, Cygwin, and MinGW.

> **NOTE** *Although Chapter 1 spotlights the shiny new GCC wagon, discussion of new and changed features is peppered throughout* The Definitive Guide to GCC. *Everyone should read and heed the caveats, gotchas, qualifications, and disclaimers pertaining to the 3.x release to avoid weeping, wailing, and gnashing of teeth*

Chapter 3, "Basic GCC Usage," describes the fundamentals of using GCC. More experienced GCC users can probably safely skim it, but newcomers, novices, and even GCC veterans will find lots of useful information in Chapter 3's discussion of GCC options and arguments, controlling GCC's output, and GCC's usage with respect to the various dialects of C and C++. Chapter 3 also shows you how to control the C preprocessor, modify directory search paths for include files and libraries, and how to pass options to the linker and the assembler.

Chapter 4, "Advanced GCC Usage," delves into GCC customization, such as controlling its behavior using environment variables and spec strings. You will also learn how to use GCC as a cross-compiler and how to refine the code that GCC generates. GCC has extended the C and C++ standards with some delightfully useful keywords and additional functionality, so you will spend considerable time learning what these extensions are, how to use them, and, in the interest of fairness, why you might not want to use them.

Chapter 5, "Optimizing Code with GCC," devotes itself to using GCC's very capable code optimizer. After giving you a jogging tour of compiler optimization theory, Chapter 5 discusses how to use GCC's various code optimizations, which fall into two broad categories: processor-specific optimizations, and processor-independent optimizations. This material leads into Chapter 6, "Performing Code Analysis with GCC," which introduces you to GCC's code analysis capabilities. You will learn how to use `gcov`, a tool for analyzing test code coverage, and `gprof`, a code profiling tool. *Test code coverage* refers to determining how much of a program's code a test suite actually tests, undeniably a valuable piece of information if you wish to make sure a program has been adequately tested before releasing it. *Code profiling* refers to developing a picture of how often a program's code is executed and how much computing time a given section of code consumes, which allows you to focus your optimization efforts most effectively.

The next two chapters, "Using Autoconf and Automake" (Chapter 7) and "Using Libtool" (Chapter 8) endeavor to help you take better advantage of GCC's capabilities. Autoconf is a package designed to ease the process of configuring programs to compile on a given platform. Because the compiler in question is often GCC, you can use Autoconf to customize how source code is compiled for a given CPU or hardware platform. The idea is to write portable code and to let Autoconf figure out how to take best advantage of GCC's capabilities on a given architecture.

Automake similarly generates Makefiles more or less automatically (hence the name), which in turn instruct compilers what to build and how. Again, because the compiler in question is often GCC, it seems sensible to describe how to use Automake to exploit GCC.

Libtool is a wonderful utility that instructs the compiler (GCC, in this case) how to create libraries (either shared or static) for a given platform. Libtool handles the nitty-gritty of compiler and linker invocations necessary to create a proper library, which means that you do not have to remember the magic

incantations (for example, on some platforms, you have to use `-fpic` to create a shared library, while others use `-fPIC`, and still others use another construct altogether).

The last four chapters are devoted to GCC miscellanea and administrivia. Chapter 9, "Troubleshooting GCC," helps you solve common problems encountered using GCC, and suggests workarounds (spelled k-l-u-d-g-e-s) for the worst of GCC's known misfeatures and warts. For better or worse, the GNU project insists on using Texinfo for online documentation, which, in GCC's case at least, mars an otherwise excellent documentation set. To remedy this shortcoming and enable you to take advantage of GCC's documentation, Chapter 10, "Using GCC's Online Help," introduces you to the Info user interface and also suggests some alternative programs you can use if you simply cannot or will not use Info. Chapter 11, "GCC Command-Line Options," serves as a reference section, listing and briefly describing all of GCC's documented options. Chapter 12, "Additional GCC Resources," lists sources of GCC information, such as mailing lists, Web pages, and other additional reading that you might find useful as you work with GCC. Appendices A, "Building and Installing Glibc," and B, "Machine and Processor-Specific Options for GCC," provide complete reference information for all of GCC's command-line options and arguments and the architecture-specific options.

Ways to Read This Book

If we have done our job properly, you should be able to read *The Definitive Guide to GCC* cover to cover and follow its intended progression from introductory material on through the advanced topics. Alternatively, you can read, or ignore, lumps of chapters.

The chapters fall naturally into a few sections that, while providing useful for information for everyone, meet one group of readers' needs quite precisely. To wit: if all you want to do is get GCC up and running on your system, read the installation instructions in Chapters 1 and 2. Chapters 3 through 5 are best suited for new to intermediate GCC users because these chapters cover the range of normal GCC usage—you will be well ahead of most GCC users in terms of making GCC sing and dance to your tune after reading these three chapters. Advanced users will get the most benefit from Chapters 6 through 8 because these chapters focus on more advanced topics that explore GCC features and capabilities that would leave uninitiated readers dazed and confused. If you are having trouble installing or using GCC, have a look at Chapters 9 and 10. Chapter 11 can be used as a quick reference to all of GCC's known command-line options and arguments. Programmers, particularly those new to GCC and command-line compilation environments, will want to read Chapters 3 through 10. Whether you start with the appetizer and finish with dessert or simply graze, *The Definitive Guide to GCC*, and GCC, has something that you can use.

More About GCC and *The Definitive Guide to GCC*

This section takes a more thorough look at what GCC is and does and includes the obligatory history of GCC. Because GCC is one of the GNU Project's premier projects, GCC's development model bears a closer look, so we will also show you GCC's development model, which should help you understand why GCC has some features, lacks other features, and how you can participate in its development.

What exactly is GCC? The tautological answer is that GCC is an acronym for the GNU Compiler Collection, formerly known as the GNU Compiler Suite, and also as GNU CC and the GNU C Compiler. As remarked earlier, GCC is a collection of compiler front ends to a common back-end compilation engine. The list of compilers includes C, C++, Objective C, FORTRAN, and Java. GCC also has front ends for Pascal, Modula-3, and Ada 9x. The C compiler itself speaks several different dialects of C, including traditional and ANSI C. The C++ compiler is a true native C++ compiler. That is, it does not first convert C++ code into an intermediate C representation before compiling it, as did the early C++ compilers, such as the cfront "compiler" Bjarne Stroustrup first used to create C++. Rather, GCC's C++ compiler, g++, creates native executable code directly from the C++ source code.

GCC is an optimizing and cross-platform compiler. It supports general optimizations that can be applied regardless of the language in use or the target CPU and options specific to particular CPU families and even specific to a particular CPU model within a family of related processors. Moreover, the range of hardware platforms to which GCC has been ported is remarkably long. GCC supports platform and target submodels, so that it can generate executable code that will run on all members of a particular CPU family or only on a specific model of that family. A partial list of GCC's supported platforms, many of which you might never have heard of, much less used (we certainly have not), includes (at least) the architectures listed in Table 1.

Considering the variety of CPUs and architectures to which GCC has been ported, it should be no surprise that you can configure it as a cross-compiler and use GCC to compile code on one platform that is intended to run on an entirely different platform. In fact, you can have multiple GCC configurations for various platforms installed on the same system and, moreover, run multiple GCC versions (older and newer) for the same CPU family on the same system.

Table 1. Processor Architectures Supported by GCC

Architecture	Description
AMD29K	AMD Am29000 architectures
ARM	Advanced RISC Machines architectures
ARC	Argonaut ARC processors
AVR	ATMEL AVR microcontrollers
DEC Alpha	Compaq (neé Digital Equipment Corporation) Alpha processors
H8/300	Hitachi H8/300 CPUs
HPPA	Hewlett Packard PA-RISC architectures
Intel i386	Intel i386 (x86) family of CPUs
Intel i960	Intel i960 family of CPUs
M32R/D	Mitsubishi M32R/D architectures
M68k	The Motorola 68000 series of CPUs
M88K	Motorola 88K architectures
MCore	Motorola M*Core processors
MIPS	MIPS architectures
MN10200	Matsushita MN10200 architectures
MN10300	Matsushita MN10300 architectures
NS32K	National Semiconductor ns3200 CPUs
RS/6000 and PowerPC	IBM RS/6000 and PowerPC architectures
RT	IBM RT PC architectures
SPARC	Sun Microsystems family of SPARC CPUs
Hitachi SH3/4/5	Super Hitachi 3, 4, and 5 family of processors
TMS320C3x/C4x	Texas Instruments TMS320c3x and TMS320C4x DSPs

New GCC users might be confused by mention of EGCS, the Experimental (or Enhanced) GNU Compiler Suite, on the GCC Web site (http://gcc.gnu.org/) and elsewhere on the Web and in printed literature. EGCS was intended to be a more actively developed and more efficient compiler than GCC, but was otherwise effectively the same compiler because it closely tracked the GCC code base and EGCS enhancements were fed back into the GCC code base maintained by

the GNU Project. Nonetheless, the two code bases were separately maintained. In April 1999, GCC's maintainers, the GNU Project, and the EGCS steering committee formally merged. At the same time, GCC's name was changed to the GNU Compiler Collection and the separately maintained (but, as noted, closely synchronized) code trees were formally combined, ending a long fork and incorporating the many bug fixes and enhancements made in EGCS into GCC.

GCC's History

GCC, or rather, the idea for it, actually predates the GNU Project. In late 1983, just before he started the GNU Project, Richard M. Stallman, President of the Free Software Foundation and originator of the GNU Project, heard about a compiler named the Free University Compiler Kit (known as VUCK) that was designed to compile multiple languages, including C, and to support multiple target CPUs. Stallman realized that he need to be able to bootstrap the GNU system and that a compiler was the first strap he needed to putt. So, he wrote to VUCK's author asking if GNU could use it. Evidently, VUCK's developer was uncooperative, responding that the university was free but that the compiler was not. As a result, Stallman concluded that his first program for the GNU Project would be a multilanguage, cross-platform compiler. Undeterred and, in true hacker fashion, desiring to avoid writing the entire compiler himself, Stallman eventually obtained the source code for Pastel, a multiplatform compiler developed at Lawrence Livermore Labs. He added a C front end to Pastel and began porting it to the Motorola 68000 platform, only to encounter a significant technical obstacle: the compiler's design required many more megabytes of stack space than the 68000-based Unix system Stallman used at the time supported. This situation forced him to conclude that he would have to write a new compiler, starting from ground zero. That new compiler eventually became GCC. Although it contains none of the Pastel source code that originally inspired it, Stallman did adapt and use the C front end he wrote for Pastel. Development of this ur-GCC proceeded slowly through the 1980s, because, as Stallman writes in his description of the GNU Project (http://www.gnu.org/gnu/the-gnu-project.html), "first, [he] worked on GNU Emacs."

As suggested previously, GCC development forked during the 1990s into two, perhaps three, branches. While the primary GCC branch continued to be maintained by the GNU Project, a number of other developers, primarily associated with Cygnus Solutions, began releasing a version of GCC known as EGCS. EGCS tracked GCC closely, but was independently and more actively maintained than GCC. Eventually, the two compilers were merged into a single compiler and the EGCS steering committee became GCC's official maintainers. Meanwhile, the Pentium Compiler Group (PCG) project created its own version of GCC, PGCC,

a Pentium-specific version that was intended to provide the best possible support for features found in Intel's Pentium-class CPUs. During the period of time that EGCS was separately maintained, PGCC closely tracked the EGCS releases—the reunification of EGCS and GCC seems to have halted PGCC development because, at the time of this writing, the PCG project's last release was 2.95.2.1, dated December 27, 2000. For additional information, visit the PGCC project's Web site at http://www.goof.com/pcg/.

Table 2 highlights GCC's release history. As you read it, keep in mind that the EGCS project maintained two version numbers. The first was in the form 2.9m.nn and indicated the relationship between the GCC and EGCS trees. The second number identified EGCS releases and took the form 1.m.n. Version numbering reverted to a single number of the form 2.9m.nn after the April 1999 merger between GCC and EGCS.

Table 2. GCC Release History

Name	Version	Release Date
GCC 3.3.2	3.3.2	October 17, 2003
GCC 3.3.1	3.3.1	August 8, 2003
GCC 3.3	3.3	May 13, 2003
GCC 3.2.3	3.2.3	April 22, 2003
GCC 3.2.2	3.2.2	February 5, 2003
GCC 3.2.1	3.2.1	November 19, 2002
GCC 3.2	3.2	August 14, 2002
GCC 3.1.1	3.1.1	July 25, 2002
GCC 3.1	3.1	May 15, 2002
GCC 3.0.4	3.0.4	February 20, 2002
GCC 3.0.3	3.0.3	December 20, 2001
GCC 3.0.2	3.0.2	October 25, 2001
GCC 3.0.1	3.0.1	August 20, 2001
GCC 3.0	3.0	June 18, 2001
GCC 2.95.3	2.95.3	March 16, 2001
GCC 2.95.2	2.95.2	October 24, 1999

Table 2. GCC Release History (continued)

Name	Version	Release Date
GCC 2.95.1	2.95.1	August 19, 1999
GCC 2.95	2.95	July 31, 1999
EGCS 1.1.2	2.91.66	March 15, 1999
EGCS 1.1.1	2.91.60	December 1, 1998
EGCS 1.1.0	2.91.57	September 3, 1998
EGCS 1.0.3	2.90.29	May 15, 1998
EGCS 1.0.2	2.90.27	March 16, 1998
gcc 2.8.1	2.8.1	March 2, 1998
gcc 2.8.0	2.8.0	January 7, 1998
EGCS 1.0.1	2.90.23	January 6, 1998
EGCS 1.0	2.90.21	December 3, 1997

GCC passed a long-awaited milestone on June 18, 2001 with the release of version 3.0. Version 3.x's key features, which Chapter 1 discusses in detail, include the following:

- *Additional targets:* New CPU targets have been added and significant enhancements have been added for several existing platforms.
- *Better documentation:* Documentation has been updated, corrected, and, in some cases, rewritten.
- *New languages:* Additional languages have been added and support for existing languages has been improved and extended.
- *Optimization enhancements:* Various features of the code optimizer, especially for ISO C99 functions, have been improved.
- *Miscellaneous changes:* Additional enhancements include internal garbage collection and an extensive test suite for verifying GCC source builds.

Who Maintains GCC?

Formally, GCC is a GNU Project, which is directed by the Free Software Foundation (FSF). The FSF holds the copyright on the compilers and licenses the compilers under the terms of the GPL. Either individuals or the FSF hold the

copyrights on other components, such as the runtime libraries and test suites, and these other components are licensed under a variety of free software licenses. The FSF also handles the legal concerns of the GCC project. So much for the administrivia.

On the practical side, a cast of dozens maintains GCC. GCC's maintainers consist of a formally organized steering committee and a larger, more loosely organized group of hackers scattered all over the Internet. The GCC steering committee, as of August, 2001, is made up of 14 people representing various communities in GCC's user base that have a significant stake in GCC's continuing and long-term survival, including kernel hackers, FORTRAN users, and embedded systems developers. The steering committee's purpose is, to quote its mission statement, "to make major decisions in the best interests of the GCC project and to ensure that the project adheres to its fundamental principles found in the project's mission statement." These "fundamental principles" include the following:

- Supporting the goals of the GNU Project
- Adding new languages, optimizations, and targets to GCC
- More frequent releases
- Greater responsiveness to consumers, the large user base that relies on the GCC compiler
- An open development model that accepts input and contributions based on technical merit

The group of developers that work on GCC includes members of the steering committee and, according to the contributors list on the GCC project home page, over 100 other individuals across the world. Still, others not specifically identified as contributors have contributed to GCC development by sending in patches, answering questions on the various GCC mailing lists, submitting bug reports, writing documentation, and testing new releases.

Who Uses GCC?

GCC's user base is large and varied. Given the nature of GCC and the loosely knit structure of the free software community, though, no direct estimate of the total number of GCC users is possible. A direct estimate, based on standard metrics, such as sales figures, unit shipments, or license purchases, is virtually impossible to derive because such numbers simply do not exist. Even indirect estimates, based for example on the number of downloads from the GNU Web and FTP

sites, would be questionable because the GNU software repository is mirrored all over the world.

More to the point, we submit that quantifying the number of GCC users is considerably less important and says less about GCC users than examining the scope of GCC's usage and the number of processor architectures to which it has been ported. For example, GCC is the standard compiler shipped in every major and most minor Linux distributions. GCC is also the compiler of choice for the various BSD operating systems (FreeBSD, NetBSD, OpenBSD, and so on). Thanks to the work of D. J. Delorie, GCC works on most modern DOS versions, including MS-DOS from Microsoft, PC-DOS from IBM, and DR-DOS from Lineo. Indeed, Delorie's work resulted in ports of most of the GNU tools for DOS-like environments. Cygnus Solutions, now owned by Red Hat Software, Inc., created a GCC port for Microsoft Windows users. Both the DOS and Windows ports offer complete and free development environments for DOS and Windows users.

The academic computing community represents another large part of GCC's user base. Vendors of hardware and proprietary operating systems typically provide compiler suites for their products as a so-called value-added service, that is, for an additional, often substantial, charge. As free software, GCC represents a compelling, attractive alternative to computer science departments faced with tight budgets. GCC also appeals to the academic world because it is available in source code form, giving students a chance to study compiler theory, design, and implementation. GCC is also widely used by nonacademic customers of hardware and operating system vendors who want to reduce support costs by using a free, high-quality compiler. Indeed, if you consider the broad range of hardware to which GCC has been ported, it becomes quite clear that GCC's user base is composed of the broadest imaginable range of computer users.

Are There Alternatives?

What alternatives to GCC exist? As framed, this question is somewhat difficult to answer. Remember that GCC is the GNU Compiler Collection, a group of language-specific compiler front ends using a common back-end compilation engine, and that GCC is free software. So, if you rephrase the question to "What free compiler suites exist as alternatives to GCC?", the answer is, "Very few."

As mentioned earlier, the Pentium Compiler Group created PGCC, a version of GCC, that was intended to extend GCC's ability to optimize code for Intel's Pentium-class CPUs. Although PGCC development seems to have stalled since the EGCS/GCC schism ended, the PGCC Web site still exists (although it, too, has not been modified recently). See http://www.goof.com/pcg/ for more information.

If you remove the requirement that the alternative be free, you have many more options. Many hardware vendors and most operating system vendors will be happy to sell you compiler suites for their respective hardware platforms or operating systems, but the cost can be prohibitive. Some third-party vendors exist

that provide stand-alone compiler suites. One such vendor is The Portland Group (http://www.pgroup.com/), which markets a set of high-performance, parallelizing compiler suites supporting FORTRAN, C, and C++. Absoft Corporation also offers a well-regarded compiler suite supporting FORTRAN 77, FORTRAN 95, C, and C++. Visit their Web site at http://www.absoft.com/ for additional information. Similarly, Borland has a free C/C++ compiler available. Information on Borland's tools can be found on their Web site at http://www.borland.com/.

Intel and Microsoft also sell very good compilers. And they are not that expensive.

Conversely, if you dispense with the requirement that alternatives be collections or suites, you can select from a rich array of options. A simple search for the word "compilers" at Yahoo generated over 120 Web sites showcasing a variety of single-language compilers, including Ada, Basic, C and C++, COBOL, Forth, Java, Logo, Modula-2 and Modula-3, Pascal, Prolog, and Smalltalk. If you are looking for alternatives to GCC, a good place to start your search is the Compilers.net Web page at http://www.compilers.net/.

So much for the look at alternatives to GCC. This is a book about GCC, after all, so we hope you will forgive us for leaving you on your own when it comes to finding information about other compilers.