Introduction

A brief history of GCC

The original author of the GNU C Compiler (GCC) is Richard Stallman, the founder of the GNU Project. The GNU project was started in 1984 to create a complete Unix-like operating system as free software, in order to promote freedom and cooperation among computer users and programmers. Every Unix-like operating system needs a C compiler, and as there were no free compilers in existence at that time the GNU Project had to develop one from scratch. The work was funded by donations from individuals and companies to the Free Software Foundation, a non-profit organization set up to support the work of the GNU Project.

The first release of GCC was made in 1987. This was a significant breakthrough, being the first portable ANSI C optimizing compiler released as free software. Since that time GCC has become *the foundation of all free software*.

A major revision of the compiler came with the 2.0 series in 1992, which added the ability to compile C++. In 1997 an experimental branch of the compiler (EGCS) was created to improve optimization and C++ support. These features were integrated back into the main-line of GCC development and became widely available in the 3.0 release of GCC in 2001.

Today GCC has been extended to support many additional languages, including Fortran, ADA and Java. The acronym GCC is now used to refer to the "GNU Compiler Collection". Its development is guided by the *GCC Steering Committee*, a group composed of representatives from GCC user communities in industry, research and academia.

Major features of GCC

This section describes some of the most important features of GCC.

- First of all, GCC is a portable compiler -- it runs on most platforms available today and can produce output for many types of chips, from 8-bit microcontrollers through to processors used by supercomputers.

- GCC is not only a compiler -- it can also *cross-compile* any program, producing executable files for a different system from the one where the compiler is running. This is particularly useful for embedded systems which are not capable of running a compiler. GCC is written in C and can compile itself, so it can be ported to new systems in the future.

- GCC has multiple language *front-ends*, for parsing different languages. Programs in each language can be compiled, or cross-compiled, for any architecture. For example an ADA program can be compiled for a microcontroller, or a C program for a supercomputer.

- GCC has a modular design, allowing support for new languages and architectures to be added easily. Adding a new language front-end to GCC

enables the use of that language immediately on any architecture. Similarly, adding support for a new architecture makes it immediately available to all languages.

- Finally and most importantly GCC is free software. This means you have the freedom to use and to modify GCC, as with all GNU software. If you need support for a new type of CPU, a new language, or new language feature you can hire someone to enhance GCC for you. You can hire someone to fix a bug if it is important for your work.

- Furthermore you have the freedom to share any enhancements you make to GCC. As a result of this freedom you can also make use of enhancements to GCC developed by others. The many features offered by GCC today show how this freedom to cooperate works to benefit you and everyone else who uses GCC.

- In addition to C and C++ the GNU operating system also provides other high-level languages such as GNU Common Lisp (gcl), GNU Smalltalk (gst), the GNU Scheme extension language (guile) and the GNU Compiler for Java (gcj) -- these languages do not require the user to access memory directly, eliminating the possibility of memory access errors. They provide a safer alternative to C and C++ for many applications.