Basic Information about GCC

GCC Version 4.0.2

Abhijat Vichare (amvichare@iitb.ac.in)

Indian Institute of Technology, Bombay

(http://www.iitb.ac.in)

This is edition 1.0 of "Basic Information about GCC", last updated on January 7, 2008., and is based on GCC version 4.0.2.

Short Contents

Table of Contents

1 Introduction

The GNU Compiler Collection – GCC – is one of the most complex software systems available in full source form. It was initiated by the Free Software Foundation and today is steered by an independent steering committee. While initially, it started off as a efficient C compiler for 32 bit machines, it has evolved to a reasonable generic architecture that accommodates about 7 source languages and a large number of target machines officially. This architecture is highly re-forgettable in practice, and the emphasis has been to have a well tested compilation system for a large number of target machines. The GPL ensures that the evolution of the system will continue. The bazaar (see [The Cathedral and the Bazaar], page 21) model of development keeps the system reasonably updated in practice. The easy availability, retargetability, the GNU license and continuing development have combined to make it a standard reference implementation with respect to which specific implementations are often compared today.

The availability and maturity of the GCC has made it an attractive development system for a variety of interests: professional, academic & hobby. Despite the availability of the source code, the complexity of the system makes it accessible usually to professionals. Even for these people, a significant effort has to be invested in understanding the compiler internals. Although high development standards have been adhered to by the GCC developers, which include well commented code, adherence to standard coding and maintenance practices, and a few simple principles (like using simple algorithms and data structures), there is almost no description of the compiler at various useful levels of abstraction. The development of such descriptions has been less attractive than the development of the compiler itself. This has been so mainly due to the strong emphasis on creating a practical and an efficient compilation system. This emphasis on a practical working compilation system for a variety of source languages and target architectures has resulted in a complexity that is difficult to master.

This document is the first in a series of articles planned towards a gradual mastery of the GCC internals. For a start, to master the compiler internals, we need to understand it's code organization, the basic reasons that motivate this organization, the basic compiler building techniques and the basic debugging techniques. The use of the compiler is documented in the online manuals – in the conventional Unix `man` pages and the GNU style `info` pages and is therefore not much dealt with here. However, we do point out some useful switches that can help in understanding the compiler internals.

The description is organized as follows. We first discuss the basic goals and architecture of the GCC system. This is used to understand the GCC source organization. We then describe building a native GCC compiler. Although described in the installation notes of GCC, we examine the process in more detail with a aim to eventually build hacking abilities. This is illustrated by a description of a few simple debugging techniques. Files and documents of the GCC source base are indicated relative to a “home” in the file system where the sources have been extracted. This home is indicated by “`$GCCHOME`”.

2 Basic GCC Goals

GCC started it's life as a C compiler. It's goal then (and even today) was to be a *useful* compiler for general use (see [A Brief History of GCC], page 21). It was desirable that the compiler be *retargetable* to facilitate easy porting to new systems. A retargetable architecture postpones target machine specific decisions to build time instead of committing to specific machine properties earlier. This implies that the building process is required to incorporate the target machine properties into the compiler code base. The GCC today actively pursues the retargetability goal mainly because it results in a useful compiler. To be useful is the primary goal and retargetability is viewed as a way to reach the goal. It aims to support all the machines that are in operation at any given point in time. It is possible that support for some machines gets withdrawn as their decreasing usage is not worth the effort to support it. The main goals of GCC are summarized in the mission statement (see [the GCC mission statement], page 21) of the GNU Project. The design and development goals are:

- New languages
- New optimizations
- New targets
- Improved run time libraries
- Faster debug cycle
- Various other infrastructure developments

The GCC effort is mainly an engineering effort aimed at being of immediate use. As with any other software engineering effort, it's concerns are with:

- Adhering to standards prescribed from time to time,
- Bugs removal management,
- Performance benchmarking, and
- Testing.

2.1 Retargetability and GCC Structure

The GCC software architecture strongly reflects the retargetability requirements. As a consequence of retargetability, the build time, t_{build} acquires critical importance. The GCC system is designed and developed prior to this instant, and the compiler binary that would eventually be used to compile user programs is created after this instant. Prior to t_{build} the GCC system cannot make any assumptions about target properties[1]. Hence, the implementation must have at least two parts:

1. The main core of the system must be generic in the sense of being unassuming about target characteristics and be "parametrized" with respect to target properties, and
2. the "parameter values" on a per target basis must be specified separately for each target.

At t_{build} the specifications of the chosen target must be combined with the generic part to obtain the complete target specific compiler sources. These target specific sources are then built to obtain the compiler binary that is used to compile user programs.

[1] This is not strictly true. The targets are assumed to be at least 32 bit, for example!

The time period t_{build} separates two conceptually distinct phases. Prior to t_{build} the compilation phases have to be expressed generically and the target properties have to be specified. This is the development time, denoted by t_{cgf} . After t_{build} we have a complete target specific compiler executable that a user can use to compile program. This is the operation time denoted by t_{op} . At t_{build} target specific parts of the compiler code are generated from the target specifications available before t_{build} . This is shown in Figure 2.1 below. The top half of the figure denotes the implementation before t_{build} as developed by a GCC developer and hence is labeled as "GCC". The bottom half is the target specific compiler *generated* from the code in the top half during the build process at t_{build} and is therefore labeled as "gcc" and executable is used by a user to compile programs.

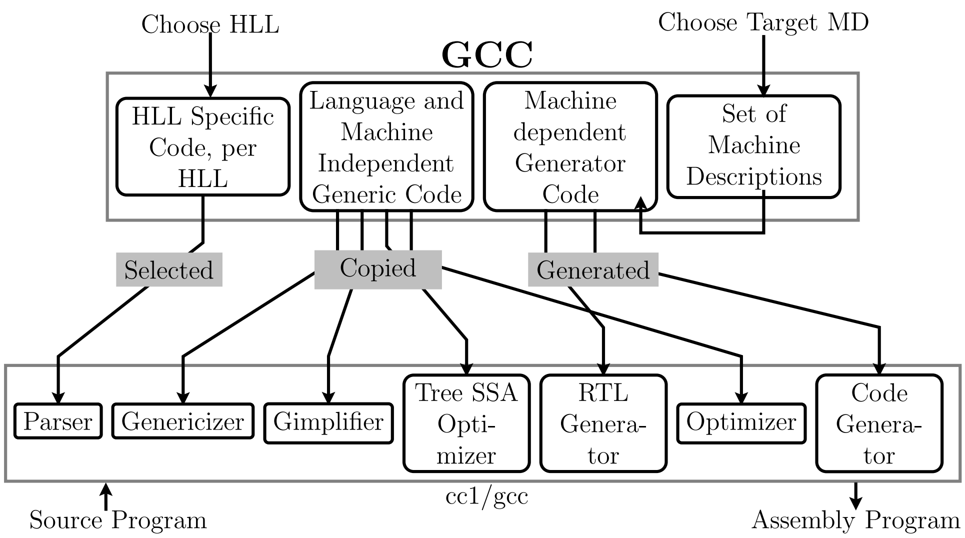

Figure 2.1: The GCC Compiler Generation Framework (CGF) and it's use to generate the target specific compiler (cc1/gcc) components. Some components of the compiler (cc1/gcc) are *selected* from the CGF, some are *copied* from the CGF and some are *generated* from the framework.

The figure also shows that GCC is "retargetable" with respect to front end languages too. Front end specific processing for various front end languages that GCC supports is also a part of the GCC system. The purpose of front end specific parts is to reduce the input source program to a common IR called the generic. In this series of documents we do not focus on front end specific processing of GCC. For our purposes the compiler starts from the program representation in generic IR.

The top half of the figure, labeled "GCC", will be called as the Compiler Generation Framework (CGF). The CGF *generates* the target specific compiler at t_{build} . An awareness of this distinction is useful to understand the GCC system. In this article, we describe the basic layout and logical structure of the CGF. Along the way we will introduce some terms that would be used in the other articles of this series. In particular, the source files are divided into implementation groups. The phase sequence wise grouping is described in [The Phase wise File Groups of GCC], page 21. Note that there are distinct groups of files that are used to *generate* the target compiler of the bottom half of the figure. These groups will be further refined in later documents (e.g. [The Phase wise File Groups of GCC], page 21). For completeness, we also briefly describe the build and use of a GCC compiler. Details of this can be consulted in the build instructions that accompany the downloaded compiler. We take this opportunity to bring out a few lesser known ways of using the compiler itself for our goal – the study of the internals. To focus more sharply on the internals, we ignore the other goals of the GCC development like development and implementation of new optimizations, improving run time library support, benchmarking etc. We will be concerned with development of new targets since that forms a part of study of the internals.

3 Source Organization

The pristine sources of GCC are downloadable from any official GCC distribution site on the Internet. The list of sites is available on the official GCC site (`http://gcc.gnu.org/`). A gzipped tape archive for GCC version 4.1.2 is named as `gcc-4.1.2.tar.gz`. These sources are extracted in a directory that we denote by `$GCCHOME`. Conventionally, the sources extract into a set of directories and files in a directory named `gcc-x.y.z`, where `x`, `y` and `z` are version digits. For example the GCC version 4.1.2 sources extract into a directory named `gcc-4.1.2`. Thus `gcc-x.y.z` is (usually) the last directory component of `$GCCHOME`. This description of the GCC source organization strives to build the intuition behind the structure that one obtains on unpacking the distribution. We emphasize that this is GCC specific, and some variations are possible in principle.

The HLL specific components, the target back end components and the actual compiler logic are separate. A driver is needed to collect the components for the chosen HLL and target pair, and "assemble" the final compiler sources which are subsequently compiled. This strategy allows creating various kinds of compilers like native, cross or Canadian cross.

The source and target independent parts of the compiler are within the `$GCCHOME/gcc` subdirectory of the main source trunk. It is in this directory that we find the code that

1. implements the complete *generic* compiler,
2. implements all the source and target independent manipulations, e.g. the optimization passes,
3. implements source specific routines housed in a *separate* sub directory, and
4. implements the back end specific routines again housed in a *separate* sub directory structure.

Corresponding to each HLL, except C[1], is a directory within `$GCCHOME/gcc` which all the code for processing that language exists. In particular this involves scanning the tokens of that language and creating the ASTs. If necessary, the basic AST tree node types need to be augmented with variations for this language. The main compiler calls these routines to handle input of that language. To isolate itself from the details of the source language, the main compiler uses a table of function pointers that are to be used to perform each required task. A language implementation needs to fill in such data structures of the main compiler code and build the language specific processing chain until the AST is obtained.

The back end specific code is organized as a list of directories corresponding to each supported back end system. This list of supported back ends is separately housed in `$GCCHOME/gcc/config` directory of the main trunk.

Parts of the compiler that are common and find frequent usage have also been separated into a separate library called the `libiberty` and placed in a distinct subdirectory of `$GCCHOME`. This facilitates a one-time build of these common routines. We emphasize that these routines are common to the main compiler, the front end code and the back end code (e.g. regular expressions handling); the routines common to only the main compiler still reside in the main compiler directory, i.e. `$GCCHOME/gcc`. GCC also implements a garbage collection based memory management system for *it's* use during a run. This code is placed in the subdirectory `$GCCHOME/boehm-gc`.

[1] GCC was originally aimed at being just a C compiler

We focus on files relevant to understanding the compiler. Hence files like `Changelogs`, `README`s, `COPYING`, `FAQ` and such have been omitted below.

3.1 Code other than the compiler proper

GCC uses internal garbage collection to manage it's own memory during a run. Supporting each front end may require additional libraries which are also bundled with the GCC sources, except the C library which is a separate package. A few other directories have code for different purposes like maintenance, description of the building and installation procedure etc. Here is a summary.

`$GCCHOME/boehm-gc`	Garbage collector
`$GCCHOME/config`	Collection of system specific flags
`$GCCHOME/contrib`	Useful contributed code
`$GCCHOME/fastjar`	Bundled Java archiver
`$GCCHOME/INSTALL`	Install instructions
`$GCCHOME/libf2c`	Fortran-to-C library
`$GCCHOME/libffi`	Bundled Foreign Function Interface
`$GCCHOME/libiberty`	Common GNU routines library
`$GCCHOME/libjava`	Java library
`$GCCHOME/libobjc`	Objective C library
`$GCCHOME/libstdc++-v3`	C++ Library
`$GCCHOME/maintainer-scripts`	Scripts used by maintainers
`$GCCHOME/zlib`	General purpose compression library

Apart from the directory organization, `$GCCHOME` also has code and data to build and install the sources. GCC uses `autoconf` generated `configure` script to obtain the detailed building requirements. This script is supported by a few other scripts. It emits the top level `Makefile` using a few data files in `$GCCHOME`. The `make` command that uses this `Makefile` also needs some supporting scripts which reside here. These scripts thus are used in various phases: configuration, building, and installation of the compiler.

`$GCCHOME/install-sh`	`$GCCHOME/libtool.m4`
`$GCCHOME/ltcf-c.sh`	`$GCCHOME/ltcf-cxx.sh`
`$GCCHOME/ltcf-gcj.sh`	`$GCCHOME/ltconfig`
`$GCCHOME/ltmain.sh`	`$GCCHOME/Makefile.def`
`$GCCHOME/Makefile.in`	`$GCCHOME/Makefile.tpl`
`$GCCHOME/missing`	`$GCCHOME/mkdep`
`$GCCHOME/mkinstalldirs`	`$GCCHOME/move-if-change`
`$GCCHOME/shmake`	`$GCCHOME/symlink-tree`
`$GCCHOME/ylwrap`	`$GCCHOME/config.guess`
`$GCCHOME/config.if`	`$GCCHOME/config.sub`
`$GCCHOME/configure`	`$GCCHOME/configure.in`
`$GCCHOME/config-ml.in`	

`configure` uses the `config.guess` script to guess the canonical name when the user has not supplied one. The canonical name of a system – build, host or target – is made up of a triple, or some times a quadruple of CPU type (sparc), Manufacturer (sun), operating system (unix), and sometimes the kernel (linux) as the third of the quadruple. The `config.sub` script is used to validate a given canonical name, i.e. it checks if the given name is supported or not. Adding a new backend *may* involve adding some code the `config.sub` to recognize the new target.

3.2 The source tree of the compiler proper

The main compiler sources reside in `$GCCHOME/gcc` directory. This directory contains five categories of code: the supported front ends, the supported back ends, auxiliary code for various purposes like internationalization support, hacks to fix vendor supplied files, the test suite etc., the include files, and the main compiler sources. Here are the various directories and files.

3.2.1 Front end support code

This code deals with processing the program as expressed by the user and corresponds to the "Language Specific Code" part of GCC box in Fig.(Figure 2.1).

`$GCCHOME/gcc/f`	Fortran front end
`$GCCHOME/gcc/ada`	Ada front end
`$GCCHOME/gcc/cp`	C++ front end
`$GCCHOME/gcc/java`	Java front end
`$GCCHOME/gcc/objc`	Objective C front end
`$GCCHOME/gcc/treelang`	Treelang front end

3.2.2 Back end support code

The back end support code resides in the `$GCCHOME/gcc/config` directory and corresponds to the "Machine Dependent Generator Code" part of the GCC box in Fig.(Figure 2.1). The specifications of supported target are found in individual subdirectories and are the input to the *generation* mechanism (files in section [gen:srcs], page 10) that generates the target specific information for the compiler in the bottom half of Fig.(Figure 2.1). This directory contains two main types of files. The common header files usually contain code for various target *systems* and reside in `$GCCHOME/gcc/config` (referred to as `$BACKEND` below) itself while the actual target machine description files are found in respective subdirectories.

Back end common files

```
$BACKEND/aoutos.h                       $BACKEND/chorus.h
$BACKEND/darwin-c.c                     $BACKEND/darwin-crt2.c
$BACKEND/darwin-protos.h                $BACKEND/darwin.c
$BACKEND/darwin.h                       $BACKEND/dbx.h
$BACKEND/dbxcoff.h                      $BACKEND/dbxelf.h
$BACKEND/divmod.c                       $BACKEND/elfos.h
$BACKEND/fp-bit.c                       $BACKEND/fp-bit.h
$BACKEND/freebsd-nthr.h                 $BACKEND/freebsd-spec.h
$BACKEND/freebsd.h                      $BACKEND/freebsd3.h
$BACKEND/freebsd4.h                     $BACKEND/freebsd5.h
$BACKEND/freebsd6.h                     $BACKEND/gnu.h
$BACKEND/gofast.h                       $BACKEND/interix.h
$BACKEND/interix3.h                     $BACKEND/libgcc-glibc.ver
$BACKEND/libgloss.h                     $BACKEND/linux-aout.h
$BACKEND/linux.h                        $BACKEND/lynx-ng.h
$BACKEND/lynx.h                         $BACKEND/netbsd-aout.h
$BACKEND/netbsd-elf.h                   $BACKEND/netbsd.h
$BACKEND/netware.h                      $BACKEND/openbsd-oldgas.h
$BACKEND/openbsd.h                      $BACKEND/psos.h
$BACKEND/ptx4.h                         $BACKEND/rtems.h
$BACKEND/sol2.h                         $BACKEND/svr3.h
$BACKEND/svr4.h                         $BACKEND/t-darwin
$BACKEND/t-freebsd                      $BACKEND/t-freebsd-thread
$BACKEND/t-gnu                          $BACKEND/t-interix
$BACKEND/t-libc-ok                      $BACKEND/t-libgcc-pic
$BACKEND/t-libunwind                    $BACKEND/t-linux
$BACKEND/t-linux-aout                   $BACKEND/t-linux-gnulibc1
$BACKEND/t-netbsd                       $BACKEND/t-openbsd
$BACKEND/t-rtems                        $BACKEND/t-slibgcc-sld
$BACKEND/t-svr4                         $BACKEND/tm-dwarf2.h
$BACKEND/udivmod.c                      $BACKEND/udivmodsi4.c
$BACKEND/usegas.h                       $BACKEND/x-interix
$BACKEND/t-openbsd-thread               $BACKEND/t-slibgcc-elf-ver
$BACKEND/t-slibgcc-nolc-override
```

Back end machine description

For each of the supported back end targets, GCC uses the following layout:

```
$BACKEND/<target-directory>
$BACKEND/<target-
directory>/<target>.h
$BACKEND/<target-
directory>/<target>.md
$BACKEND/<target-
directory>/<target>.c
$BACKEND/<target-directory>/<other
files>
```

3.3 Auxiliary files

The following directories contain auxiliary files as follows:

`$GCCHOME/gcc/doc`	Documentation in `texinfo` format
`$GCCHOME/gcc/fixinc`	Hacks to fix vendor's include files
`$GCCHOME/gcc/ginclude`	Additional includes for ISO C support
`$GCCHOME/gcc/intl`	GCC Internationalization support
`$GCCHOME/gcc/po`	Internationalization data strings
`$GCCHOME/gcc/testsuite`	GCC test suite

3.4 The Compiler includes

The common include files of the compiler reside in the `$GCCHOME/include` directory. This is referred to below as `$GCCINCLUDES`.

```
$GCCINCLUDES/ansidecl.h
$GCCINCLUDES/dyn-string.h
$GCCINCLUDES/floatformat.h
$GCCINCLUDES/getopt.h
$GCCINCLUDES/libiberty.h
$GCCINCLUDES/objalloc.h
$GCCINCLUDES/partition.h
$GCCINCLUDES/sort.h
$GCCINCLUDES/symcat.h
$GCCINCLUDES/xregex.h
$GCCINCLUDES/demangle.h
$GCCINCLUDES/fibheap.h
$GCCINCLUDES/fnmatch.h
$GCCINCLUDES/hashtab.h
$GCCINCLUDES/md5.h
$GCCINCLUDES/obstack.h
$GCCINCLUDES/safe-ctype.h
$GCCINCLUDES/splay-tree.h
$GCCINCLUDES/ternary.h
$GCCINCLUDES/xregex2.h
```

3.5 The Compiler sources

The bulk of the sources reside in the `$GCCHOME/gcc` directory. We will refer to this directory as `$MAINSRCS` below. We divide the sources into the following six types: scripts, templates to drive the scripts, definitions, C sources that are used to generate sources with target specific information at build time, C include files and C sources.

3.5.1 Scripts

```
$MAINSRCS/configure
$MAINSRCS/fixproto
$MAINSRCS/genmultilib
$MAINSRCS/mkinstalldirs
$MAINSRCS/move-if-change
$MAINSRCS/sort-protos
$MAINSRCS/mkmap-flat.awk
$MAINSRCS/mkmap-symver.awk
$MAINSRCS/configure.frag
$MAINSRCS/config.gcc
$MAINSRCS/config.guess
$MAINSRCS/aclocal.m4
$MAINSRCS/mkconfig.sh
$MAINSRCS/scan-types.sh
```

3.5.2 Script templates

```
$MAINSRCS/c-config-lang.in
$MAINSRCS/config.in
$MAINSRCS/configure.in
$MAINSRCS/c-parse.in
$MAINSRCS/cstamp-h.in
$MAINSRCS/gccbug.in
$MAINSRCS/gdbinit.in
$MAINSRCS/Makefile.in
$MAINSRCS/mkheaders.in
$MAINSRCS/mklibgcc.in
```

3.5.3 Definitions

Of particular interest for the study of the GCC compiler are the `tree.def`, `c-common.def`, `rtl.def` and `machmode.def` definition files. `tree.def` and `c-co-mmon.def` together define all the AST node types. `rtl.def` defines all the various RTL types that a given version GCC uses internally. Finally, the `machmode.def` file defines the RTL Abstract machine data types with their relative size in bytes.

```
$MAINSRCS/builtin-attrs.def
$MAINSRCS/builtins.def
$MAINSRCS/builtin-types.def
$MAINSRCS/c-common.def
$MAINSRCS/diagnostic.def
$MAINSRCS/machmode.def
$MAINSRCS/params.def
$MAINSRCS/predict.def
$MAINSRCS/rtl.def
$MAINSRCS/stab.def
$MAINSRCS/timevar.def
$MAINSRCS/tree.def
```

3.5.4 Generator C sources

HEADERS

```
$MAINSRCS/genattrtab.h
$MAINSRCS/gengtype.h
$MAINSRCS/gengtype-yacc.h
$MAINSRCS/gensupport.h
```

SOURCES

```
$MAINSRCS/genattr.c
$MAINSRCS/genattrtab.c
$MAINSRCS/genautomata.c
$MAINSRCS/gencheck.c
$MAINSRCS/gencodes.c
$MAINSRCS/genconditions.c
$MAINSRCS/genconfig.c
$MAINSRCS/genconstants.c
$MAINSRCS/genemit.c
$MAINSRCS/genextract.c
$MAINSRCS/genflags.c
$MAINSRCS/gengenrtl.c
$MAINSRCS/gengtype.c
$MAINSRCS/gengtype-lex.c
$MAINSRCS/gengtype-yacc.c
$MAINSRCS/genopinit.c
$MAINSRCS/genoutput.c
$MAINSRCS/genpeep.c
$MAINSRCS/genpreds.c
$MAINSRCS/gen-protos.c
$MAINSRCS/genrecog.c
$MAINSRCS/gensupport.c
```

3.5.5 C includes

$MAINSRCS/acconfig.h
$MAINSRCS/basic-block.h
$MAINSRCS/bitmap.h
$MAINSRCS/c-common.h
$MAINSRCS/cfglayout.h
$MAINSRCS/collect2.h
$MAINSRCS/conditions.h
$MAINSRCS/convert.h
$MAINSRCS/cppdefault.h
$MAINSRCS/cpphash.h
$MAINSRCS/cpplib.h
$MAINSRCS/c-pragma.h
$MAINSRCS/c-pretty-print.h
$MAINSRCS/cselib.h
$MAINSRCS/c-tree.h
$MAINSRCS/dbxout.h
$MAINSRCS/dbxstclass.h
$MAINSRCS/debug.h
$MAINSRCS/defaults.h
$MAINSRCS/df.h
$MAINSRCS/diagnostic.h
$MAINSRCS/dwarf2asm.h
$MAINSRCS/dwarf2.h
$MAINSRCS/dwarf2out.h
$MAINSRCS/dwarf.h
$MAINSRCS/errors.h
$MAINSRCS/et-forest.h
$MAINSRCS/except.h
$MAINSRCS/expr.h
$MAINSRCS/flags.h
$MAINSRCS/function.h
$MAINSRCS/gbl-ctors.h
$MAINSRCS/gcc.h
$MAINSRCS/gcov-io.h
$MAINSRCS/ggc.h
$MAINSRCS/glimits.h
$MAINSRCS/graph.h
$MAINSRCS/gstab.h
$MAINSRCS/gsyms.h
$MAINSRCS/gsyslimits.h
$MAINSRCS/gthr-aix.h
$MAINSRCS/gthr-dce.h
$MAINSRCS/gthr.h
$MAINSRCS/gthr-posix.h
$MAINSRCS/gthr-rtems.h
$MAINSRCS/gthr-single.h
$MAINSRCS/gthr-solaris.h
$MAINSRCS/gthr-vxworks.h
$MAINSRCS/gthr-win32.h
$MAINSRCS/hard-reg-set.h
$MAINSRCS/hashtable.h
$MAINSRCS/hooks.h
$MAINSRCS/hwint.h
$MAINSRCS/input.h
$MAINSRCS/insn-addr.h
$MAINSRCS/integrate.h
$MAINSRCS/intl.h
$MAINSRCS/langhooks-def.h
$MAINSRCS/langhooks.h
$MAINSRCS/libfuncs.h
$MAINSRCS/libgcc2.h
$MAINSRCS/limitx.h
$MAINSRCS/limity.h
$MAINSRCS/line-map.h
$MAINSRCS/location.h
$MAINSRCS/longlong.h
$MAINSRCS/loop.h
$MAINSRCS/machmode.h
$MAINSRCS/mbchar.h
$MAINSRCS/mkdeps.h
$MAINSRCS/optabs.h
$MAINSRCS/output.h
$MAINSRCS/params.h
$MAINSRCS/predict.h
$MAINSRCS/prefix.h
$MAINSRCS/pretty-print.h
$MAINSRCS/profile.h
$MAINSRCS/ra.h
$MAINSRCS/real.h
$MAINSRCS/recog.h
$MAINSRCS/regs.h
$MAINSRCS/reload.h
$MAINSRCS/resource.h
$MAINSRCS/rtl.h
$MAINSRCS/sbitmap.h
$MAINSRCS/scan.h
$MAINSRCS/sched-int.h
$MAINSRCS/sdbout.h
$MAINSRCS/ssa.h
$MAINSRCS/stack.h
$MAINSRCS/sys-protos.h
$MAINSRCS/system.h
$MAINSRCS/sys-types.h
$MAINSRCS/target-def.h
$MAINSRCS/target.h
$MAINSRCS/timevar.h
$MAINSRCS/toplev.h
$MAINSRCS/tree-dump.h
$MAINSRCS/tree.h
$MAINSRCS/tree-inline.h
$MAINSRCS/tsystem.h
$MAINSRCS/typeclass.h
$MAINSRCS/unwind-dw2-fde.h
$MAINSRCS/unwind.h
$MAINSRCS/unwind-pe.h
$MAINSRCS/varray.h
$MAINSRCS/version.h
$MAINSRCS/vmsdbg.h
$MAINSRCS/xcoffout.h
$MAINSRCS/unwind.inc

3.5.6 C sources

We further divide the sources depending on the concept being implemented by them as: front end processing, Interfacing with the rest of the compiler, main compilation phases, optimizations, tools chain interfacing, C preprocessing, measurements and diagnostics, error detection and reporting, debugging, the `gcc` driver files and other miscellaneous files. These divisions, however, are rough since a source file sometimes contains code that is useful in a different context too.

FRONT END PROCESSING

```
$MAINSRCS/attribs.c
$MAINSRCS/c-aux-info.c
$MAINSRCS/c-common.c
$MAINSRCS/c-convert.c
$MAINSRCS/c-decl.c
$MAINSRCS/c-dump.c
$MAINSRCS/c-errors.c
$MAINSRCS/c-format.c
$MAINSRCS/c-lang.c
$MAINSRCS/c-lex.c
$MAINSRCS/c-objc-common.c
$MAINSRCS/c-opts.c
$MAINSRCS/c-parse.c
$MAINSRCS/c-semantics.c
$MAINSRCS/c-typeck.c
$MAINSRCS/langhooks.c
```

INTERFACING WITH REST OF THE COMPILER

```
$MAINSRCS/bitmap.c
$MAINSRCS/builtins.c
$MAINSRCS/fix-header.c
$MAINSRCS/ggc-common.c
$MAINSRCS/ggc-none.c
$MAINSRCS/ggc-page.c
$MAINSRCS/ggc-simple.c
$MAINSRCS/sbitmap.c
$MAINSRCS/stringpool.c
```

MAIN COMPILATION PHASES

```
$MAINSRCS/caller-save.c
$MAINSRCS/calls.c
$MAINSRCS/conflict.c
$MAINSRCS/convert.c
$MAINSRCS/dummy-conditions.c
$MAINSRCS/emit-rtl.c
$MAINSRCS/et-forest.c
$MAINSRCS/explow.c
$MAINSRCS/expmed.c
$MAINSRCS/expr.c
$MAINSRCS/final.c
$MAINSRCS/floatlib.c
$MAINSRCS/fp-test.c
$MAINSRCS/function.c
$MAINSRCS/gcov.c
$MAINSRCS/global.c
$MAINSRCS/haifa-sched.c
$MAINSRCS/hashtable.c
$MAINSRCS/hooks.c
$MAINSRCS/ifcvt.c
$MAINSRCS/integrate.c
$MAINSRCS/line-map.c
$MAINSRCS/lists.c
$MAINSRCS/local-alloc.c
$MAINSRCS/main.c
$MAINSRCS/optabs.c
$MAINSRCS/params.c
$MAINSRCS/predict.c
$MAINSRCS/profile.c
$MAINSRCS/protoize.c
$MAINSRCS/ra-build.c
$MAINSRCS/ra.c
$MAINSRCS/ra-colorize.c
$MAINSRCS/ra-rewrite.c
$MAINSRCS/read-rtl.c
$MAINSRCS/real.c
$MAINSRCS/recog.c
$MAINSRCS/regclass.c
$MAINSRCS/regmove.c
$MAINSRCS/regrename.c
$MAINSRCS/reg-stack.c
$MAINSRCS/reload1.c
$MAINSRCS/reload.c
$MAINSRCS/reorg.c
$MAINSRCS/resource.c
$MAINSRCS/rtlanal.c
$MAINSRCS/rtl.c
$MAINSRCS/sched-deps.c
$MAINSRCS/sched-ebb.c
$MAINSRCS/sched-rgn.c
$MAINSRCS/sched-vis.c
$MAINSRCS/simplify-rtx.c
$MAINSRCS/ssa.c
$MAINSRCS/stmt.c
$MAINSRCS/stor-layout.c
$MAINSRCS/toplev.c
$MAINSRCS/tracer.c
$MAINSRCS/tree.c
$MAINSRCS/tree-inline.c
$MAINSRCS/varray.c
$MAINSRCS/version.c
$MAINSRCS/gengtype-lex.l
$MAINSRCS/c-parse.y
$MAINSRCS/gengtype-yacc.y
$MAINSRCS/libgcc-std.ver
```

Optimisations

$MAINSRCS/alias.c	$MAINSRCS/bb-reorder.c
$MAINSRCS/cfganal.c	$MAINSRCS/cfgbuild.c
$MAINSRCS/cfg.c	$MAINSRCS/cfgcleanup.c
$MAINSRCS/cfglayout.c	$MAINSRCS/cfgloop.c
$MAINSRCS/cfgrtl.c	$MAINSRCS/combine.c
$MAINSRCS/cse.c	$MAINSRCS/cselib.c
$MAINSRCS/df.c	$MAINSRCS/doloop.c
$MAINSRCS/dominance.c	$MAINSRCS/flow.c
$MAINSRCS/fold-const.c	$MAINSRCS/gcse.c
$MAINSRCS/jump.c	$MAINSRCS/lcm.c
$MAINSRCS/loop.c	$MAINSRCS/sibcall.c
$MAINSRCS/ssa-ccp.c	$MAINSRCS/ssa-dce.c
$MAINSRCS/unroll.c	

Tools chain interfacing

$MAINSRCS/collect2.c	$MAINSRCS/c-pretty-print.c
$MAINSRCS/crtstuff.c	$MAINSRCS/graph.c
$MAINSRCS/intl.c	$MAINSRCS/libgcc2.c
$MAINSRCS/mbchar.c	$MAINSRCS/prefix.c
$MAINSRCS/tlink.c	$MAINSRCS/varasm.c
$MAINSRCS/xcoffout.c	

C preprocessing

$MAINSRCS/cppdefault.c	$MAINSRCS/cpperror.c
$MAINSRCS/cppexp.c	$MAINSRCS/cppfiles.c
$MAINSRCS/cpphash.c	$MAINSRCS/cppinit.c
$MAINSRCS/cpplex.c	$MAINSRCS/cpplib.c
$MAINSRCS/cppmacro.c	$MAINSRCS/cppmain.c
$MAINSRCS/cppspec.c	$MAINSRCS/cpptrad.c
$MAINSRCS/c-pragma.c	$MAINSRCS/scan.c
$MAINSRCS/scan-decls.c	

Measurements and Diagnostics

$MAINSRCS/diagnostic.c	$MAINSRCS/gmon.c
$MAINSRCS/timevar.c	

Error detection and reporting

$MAINSRCS/doschk.c	$MAINSRCS/errors.c
$MAINSRCS/except.c	$MAINSRCS/rtl-error.c
$MAINSRCS/unwind-c.c	$MAINSRCS/unwind-dw2.c
$MAINSRCS/unwind-dw2-fde.c	$MAINSRCS/unwind-dw2-fde-darwin.c
$MAINSRCS/unwind-dw2-fde-glibc.c	$MAINSRCS/unwind-libunwind.c
$MAINSRCS/unwind-sjlj.c	

Debugging

$MAINSRCS/dbxout.c
$MAINSRCS/debug.c
$MAINSRCS/dwarf2asm.c
$MAINSRCS/dwarf2out.c
$MAINSRCS/dwarfout.c
$MAINSRCS/print-rtl1.c
$MAINSRCS/print-rtl.c
$MAINSRCS/print-tree.c
$MAINSRCS/ra-debug.c
$MAINSRCS/sdbout.c
$MAINSRCS/tree-dump.c
$MAINSRCS/vmsdbgout.c

THE gcc DRIVER FILES

$MAINSRCS/gcc.c
$MAINSRCS/gccspec.c

MISCELLANEOUS

$MAINSRCS/mips-tdump.c
$MAINSRCS/mips-tfile.c
$MAINSRCS/mkdeps.c

4 Building GCC

There are *four* directories[1] that are useful to describe the user level building of GCC. They are not required to be defined in practice.

1. The directory where we have downloaded the compressed sources. We denote this by `$DOWNLOADDIR`
2. The directory where the we extract the downloaded sources. We denote this by `$GCCHOME`
3. The directory where we build the compiler for the chosen source language and target machine. We denote this by
4. The directory where the built compiler is installed for use. We denote this by `$INSTALLDIR`

The GCC build instructions in `$GCCHOME/INSTALL/index.html` recommend the use of a distinct build directory and discourages building GCC in `$GCCHOME`. Any directory with suitable permissions that is different from `$GCCHOME` may be used.

The binaries, libraries, headers and documentation that is built is installed as a directory tree under `$INSTALLDIR`. This is any convenient directory with suitable permissions, and usually distinct from the others. The default is a system wide installation directory, e.g. `/usr/local`, but can be specified when GCC is configured for building.

There are four steps to building the compiler.

1. change to the `$BUILDDIR`,
2. configure the pristine GCC sources,
3. build the compiler binaries, libraries etc., and
4. install the compiler.

In the description below, unless otherwise stated, we assume a GNU/Linux system running on an i386 with the GNU Bourne Again SHell – `bash` – as the command shell. All commands are issued at the `bash` shell prompt, and shell commands or scripts are `bash` scripts.

4.1 Configuring GCC

The pristine GCC sources must be informed about some details like the system on which it will eventually run. A shell script called `configure` is used for this. Most pieces of required information have reasonable default values, and the usual way is to simply issue the `configure` command, which uses the defaults. However, specific non default values can be given to the `configure` command through some command line switches. Being a retargetable compiler that supports a number of high level languages (HLLs), the sources need to be informed about the particular source language and the target hardware on which the built compiler is to be used. By default, GCC is configured to build a compiler for the target on which it is being compiled – the so called compiler is desired, then the switch `--enable-languages` can be used. It also builds a compiler for each supported source language. The install directory defaults to `/usr/local`, but can also be specified using the

[1] Directories are also called as "folders".

`--prefix` switch. The `configure --help` command lists out various such options whose details are documented in `$GCCHOME/INSTALL/index.html`.

Here is a list of few configuration options.

- `--enable-languages`: The set of desired source languages separated by commas.
- `--target`: The target hardware for which the built compiler should generate code given as a GNU system triplet,
  e.g. `i386-linux-gnu`.
- `--prefix`: The absolute pathname of the directory below which the built compiler will be installed. This is `$INSTALLDIR`. This must be available (created fresh, if necessary) before issuing the `configure` command.

For example, on a typical Intel 386 based machines running the GNU/Linux systems, the following commands build a native compiler for C, C++, Java etc. that is installed in `/usr/local`.

1. Change to the build directory
   `cd $BUILDDIR`
2. Just use defaults.
   `$GCCHOME/configure`

To build only a C compiler for a i386 for running on a GNU/Linux operating system and `/home/amv/gcc-trial-install` as the installation folder[2], we configure as follows:

1. Change to the build directory
   `cd $BUILDDIR`
2. Specify that we need only the C compiler, to run on an i386 machine running GNU/Linux and `/home/amv/gcc-trial-install` as the installation folder (each option is shown on a separate line for clarity, but is one single command line)
   ```
   $GCCHOME/configure
   --enable-languages=c
   --target=i386-linux-gnu
   --prefix=/home/amv/gcc-trial-install
   ```

In any case, the `configure` program makes a number of checks for a successful build and generates a `Makefile` (as `$BUILDDIR/Makefile`) to start building the compiler if all the checks are successful. However, it occasionally can occur that this `Makefile` may result in a failure of the later build in which case it is a good idea to report the failure to GCC developers.

It is useful to redirect the output of `configure` to some file for later study as follows:[3]

```
$GCCHOME/configure > configure.log 2> configure.errors
```

4.2 Steps in Building GCC

Once the configuration successfully generates the required `Makefile`, to build the compiler one simply issues the `make` command. The steps are:

---

[2] We will describe this by saying that `$INSTALLDIR` is `/home/amv/gcc-trial-install`. In practice, we do **not** need to set a `$INSTALLDIR` variable and the complete pathname of `$INSTALLDIR` must be given.

[3] In fact, this is what we did for each stage of building to study some aspects of GCC!

1. `cd $BUILDDIR`
2. `make`

Building GCC involves building the compiler for each source language, the driver program `gcc`, the associated header files, any support libraries (but not the standard C library – 1 that is built separately outside of GCC), and the documentation. The driver program `gcc` is the *command* that users use to compile their source programs. The driver takes the user's source file to be compiled and invokes a sequence of programs – the compiler, the assembler and the linker – that generate it's binary.

The GCC build aborts in case an error is encountered.

It is useful to redirect the output of `make` to some file for later study as follows:

`$BUILDDIR/make > make.log 2> make.errors`

4.3 Installing GCC

An install follows a successful build. The various components of the compiler like the driver, the compiler proper, any libraries, the documentation etc. are installed under a well defined directory structure in the `$INSTALLDIR` directory. The following structure is typically used:

- `$INSTALLDIR/bin`: Directory where the various executables are installed.
- `$INSTALLDIR/include`: Directory where the various headers are installed.
- `$INSTALLDIR/lib`: Directory where the various libraries are installed.
- `$INSTALLDIR/man`: Directory where the various online manual pages are installed.
- `$INSTALLDIR/info`: Directory where the various online info[4] pages are installed.

The install the built sources, use the following command:

`$BUILDDIR/make install`

It is useful to redirect the output of install to some file for later study as follows:

`$BUILDDIR/make install > install.log 2> install.errors`

4.4 Using GCC

To use the newly built GCC compiler, it is useful to have the `$INSTALL-DIR/bin` directory in the path. On unix like systems, like GNU/Linux, a path is a standard shell variable called `PATH` whose value is a colon separated list of directories to be *sequentially* searched for locating the executable of the command given by the user. In case the `$INSTALLDIR/bin` is not in the path, the complete pathname of the executable must be given, as we use in the example commands that follow.

Assume that we have written a C program in a file named `prog.c` in the current directory. If the installation is successful, the following command can be used to compile `prog.c` and generate it's executable:

`$INSTALLDIR/bin/gcc prog.c`

If there are no errors, the executable named `a.out` is generated.

[4] Info pages are standard GNU online documentation system similar to unix man pages and accessed using the `info` command.

GCC has a number of useful options that can be used to control the details of the compilation. All the options can be found in the online documentation using the commands `man gcc`, or `info gcc`. Here are a few:

- `-Wall`: Turns on all warnings. This is extremely useful to trap many conventional errors we make while writing source code. We strongly recommend the use of this switch in normal program development. It is not necessary in final production compilation.
- `-o <file_name>`: The output generated by the compiler is stored in the file named `file_name`. Most outputs have default file names. For example, executables are named `a.out` by default. This switch is used if we wish to give the executable a specific name. For example, if we wish to name the executable of `prog.c` as `prog` then the command line is:
  `$INSTALLDIR/bin/gcc prog.c -o prog`
- `-S`: Generate the assembly output of the given program. The assembly code output of the program `prog.c` is stored by default in the file named `prog.s` (unless the `-o` switch is used).
- `-c`: Generate the object code, not the executable, of the given program. The object code output of the program `prog.c` is stored by default in the file named `prog.o` (unless the `-o` switch is used).
- `-dA`: Annotate the assembler output with some useful information. Useful information like the source variable to assembly register association can be obtained using this switch.
- `-fdump-tree-*`: The '*' here stands for a set of additional words like `all`, `raw` etc. giving us a family of switches like `-fdump-tree-all`, `-fdump- tree-raw` etc. This family of switches tell the compiler to dump the tree intermediate representations of the program being compiled. This is useful to study the internals of the compiler.
- `-fdump-rtl-*`: The '*' here stands for a set of additional words like `all`, `bbro` etc. giving us a family of switches like `-fdump-rtl-all`, `-fdump- rtl-bbro` etc. This family of switches tell the compiler to dump the RTL intermediate representations of the program being compiled. This is useful to study the internals of the compiler.

References

(**Note:** In the URLs below: `$GCCINTDOCSHOME` is `http://www.cfdvs.iitb.ac.in/~amv/gcc-int-docs`)

1. –
   GCC Development Mission Statement (1999-04-22)
   (`http://gcc.gnu.org/gccmission.html`)
   1999.
2. Abhijat Vichare.
   The Phasewise File Groups of GCC.
   (`$GCCINTDOCSHOME/html/gcc-source-blocks.html`)
   2007.
3. Eric Raymond.
   The Cathedral and the Bazaar
   (`http://www.catb.org/~esr/writings/cathedral-bazaar/`)
   2001.
4. Mathew Wilcox.
   A Brief History of GCC
   (`http://gcc.gnu.org/wiki/History`)
   2006.

List of Figures

Appendix A Copyright

A.1 GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document 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.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.