GCC技术参考大全

第 1 章 GCC简介

GCC （ GNU Compiler Collection ， GNU 编译程序集合）是最重要的开放源码软件。事实上，其他所有开放源码软件都在某种层次上依赖于它。甚至其他语言，例如 Perl 和 Python ，都是由 C 语言开发的，由 GNU 编译程序编译的。

GCC 编译程序的历史很有趣，远远不止是一个时间和事件的列表。这个软件对于整个自由软件运动而言具有根本性的意义。事实上，如果没有它或类似的软件，就不可能有自由软件运动。 GCC 为 Linux 的出现提供了可能性。

本章概要介绍了 GCC 编译程序集合，以及它的相关工具。这些编译中使用的工具可以跟踪源代码、编辑文件、控制编译过程、提供调试信息。

本章介绍的内容包括一个列表以及对处理过程的一些描述。该列表描述了组成编译程序集合的文件和程序。之后介绍了将源文件变成可连接和可执行程序的步骤。

1.1 GNU

GCC 是 GNU 项目的一个产品。该项目始于 1984 年，目标是以自由软件的形式开发一个完整的类 UNIX 的操作系统。像所有这种规模的软件一样， GNU 项目也经历了一些波折，但目标最终还是实现了。实际上现在一个功能完备的类 UNIX 操作系统—— Linux ，已经在世界上广为流传了，并被不计其数的公司、政府和个人成功应用。而该系统及其所有工具和应用都是基于 GCC 的。

可用于 Linux 以及其他系统的自由软件的范围很广泛，并且还在日益增长。作为整体 GNU 项目的一部分而开发的免费 UNIX 被列在 http://www.gnu.org/directory 中的自由软件目录（ Free Software Directory ）中。

成千上万的程序员都在为各种 GNU 项目（及其他自由软件项目）作贡献，而实际上所有这些都在某种程度上依赖于 GCC 。

1.2 测量编译程序

我们可以在编译的速度、生成代码的速度，以及生成代码的尺寸上对编译程序进行比较。但是很难进行更深入的比较，因为虽然可以得出一些数字，却很难对这些数字赋予某种实际意义。例如，源文件的数目（ make 程序的描述文件、配置文件、头文件、可执行代码，等等）显示共有超过 15 000 个的各种文件。源文件编译成的目标文件、库和可执行程序的数目成千增长。代码的行数（这 15 000 多个文件的行数）超过 3 700 000 。从任何标准来看，这都是一个大程序。

代码的质量参差不齐——因为有如此之多的程序员参与开发过程，而且代码注释中也内嵌了大量的内部文档，所以文档的质量和数量也有变化。所幸的是，有大量的程序员正在努力地提高代码和注释的质量。而且，也不是必须阅读内嵌的注释才能使用编译程序。但如果要对编译程序做些工作，你会发现还是要花些时间阅读代码中内嵌的注释。

测量编译程序的质量的惟一方法是询问它的用户。全世界的用户数目很难估计（自由软件就有这样的特性），但一定是巨大的。它被用于某些版本的 UNIX ，这些 UNIX 自带系统供应商提供的本地编译程序。事实上，我知道一个很大的 UNIX 供应商就在自己内部的项目中使用 GCC ，即便该供应商也有自己的很优秀的编译程序。

GCC 编译程序从未停止过改进。如同第 2 章所描述的，通过下载某个特定版本的源代码便可安装已经发布的某个版本的 GCC ，也可以直接下载最新的（或测试中的）版本。测试中的版本时刻都在改进。有些更正是修改已有的 bug ，还有一些是为了加入新的语言和功能，还有一些是为了去掉某些不再应用的功能。如果你曾经使用过 GCC ，隔一段时间再用最新版，一定会发现一些变化。

1.3 命令行选项

每个命令行选项都以一个或一对连字号开始。例如，下面的命令行会编译 ANSI 标准 C 程序 muxit.c ，再产生一个非连接的目标文件 muxit.o ：

```
gcc -ansi -c muxit.c -o muxit.o
```

这些单字母选项后面跟着的名字可以和字母之间留有空格。例如，选项 -omuxit.o 和 -o muxit.o 是一样的。

下面的命令用 -v 代表详细说明，而 --help 会打印可用的选项，而且会打印一个详细的包括所有命令行选项的列表，包括那些适用于特定语言的选项。

gcc -v --help

有可能构造一些实际不做任何事情的命令行。例如，下面的命令将目标文件交给编译程序，然后指定 -c 选项防止激活连接程序：

gcc -c brookm.o

所有的命令行选项大致可分为三类：

· 指定语言 GCC 编译程序有能力编译多种语言，有些选项只可用于其中的一两种。例如， -C89 选项只应用于 C 语言，指定适用于 1989 年的标准。

· 指定平台 GCC 编译程序可以为多种平台生成目标代码，而有些选项只能应用于为某个指定平台生成代码。例如，如果输出平台是 Intel 386 ，那么 -fp-ret-in-387 选项可用来指出要将函数调用返回的浮点数保存在硬件的浮点寄存器中。

· 普适 很多选项对所有语言和平台都适用。例如， -O 选项指示编译程序要优化输出代码。

使用编译程序不知道的选项总会产生出错消息。使用目标平台不适用的选项也会产生出错消息。

gcc 程序能够自己处理所有已知的选项，而将未知其他选项传递下去来编译指定语言。如果传递下去的选项对指定语言的处理器是未知的，就会报错。

选项可以指示 gcc 只执行特定的操作（例如连接或预处理）或什么都不做，这就是说有些标志没有什么特殊目的。除非用 -W 选项产生特殊警告，否则这些标志可被识别但不采取任何动作，只简单忽略而已。

1.4 平台

GCC 编译程序集合可以在很多平台上运行。平台是指特定计算机芯片及其运行的操作系统的组合。

尽管 GCC 已经被移植到数以千计的硬件 / 软件的组合上，但只有一些基本平台可以用来测试发布的正确性。这些列在表 1-1 中的基本目标平台是最流行的，而且它们对 GCC 支持的其他平台具有代表性。

表 1-1 GCC 应用的主要平台

硬件	操作系统
Alpha	Red Hat Linux 7.1
HPPA	HPUX 11.0
Intel x86	Debian Linux 2.2 、 Red Hat Linux 6.2 和 FreeBSD 4.5
MIPS	IRIX 6.5

（续表）

硬件	操作系统
PowerPC	AIX 4.3.3
Sparc	Solaris 2.7

要注意保证 GCC 可以在表 1-1 中列出的主要平台上正确运行，而且也要很好地处理其次列在表 1-2 中的平台。

表 1-2 GCC 应用的次要平台

硬件	操作系统
PowerPC	Linux
Sparc	Linux
ARM	Linux
Intel x86	Cygwin

在这样少的主要和次要平台上测试的原因是人力问题。即使你的平台没有被列在上面，编译程序还是可能在系统上能够良好运行的。而且，完整的测试集合和编译程序的源代码一起提供，所以很容易验证编译程序是否工作正常。另一种方法就是作为志愿者来测试你的平台，这样编译程序就可以在每次发布之前得到测试。

1.5 编译程序的功能

编译程序是一个翻译器。它读入一种语言格式的指令（通常是文本形式的编程语言），并将它们翻译成可在计算机上运行的指令集合（通常是二进制硬件指令的集合）。

大体上讲，编译程序可以分为两部分：前端和后端。前端读出程序的源代码，将找到的内容以树的形式转换到内存驻留表（ memory-resident table ）中。一旦构造了该树，编译程序的后端就会读出树中保存的信息，并将它们转换成目标机器上的汇编语言。

下面是关于将源文件翻译成可执行程序的大致步骤：

· 词法分析是编译程序前端的最开始部分。它从输入中读出字符，确定哪些是在一起的，形成符号、数字和标点符号。

· 语法分析处理会读入来自词法浏览器的符号流，以及后面跟着的一个规则集合，确定它们之间的关系。语法分析器的输出结果是树结构，会被传递给编译程序的后端。

· 语法分析树结构会被翻译成伪汇编语言（ psuedo-assembly language ），叫做寄存器传送语言（ Register Transfer Language ， RTL ）。

· 编译程序的后端由分析 RTL 代码开始，然后执行一些优化操作。代码中冗余和未被使用的部分会被去掉。树中有些部分会被移动到其他位置以防止语句被不必要地多次执行。总的说来，有十个以上的优化操作，而且有些优化操作会多次浏览代码。

· RTL 被翻译成目标机器上的汇编语言。

· 激活汇编器去将汇编语言翻译成目标文件。该文件不是可执行格式 —— 它包括可执行的目标代码，但并不是最终运行的形式。另外，它更可能包括未解析的到其他模块例程和数据的引用。

· 连接程序将来自汇编器的目标文件（其中有些可能保存在包含目标文件的库中）组合成可执行程序。

注意，前端和后端是完全分离开的。任何语言都可用语法分析器产生树结构，而由 GCC 进行编译。类似地，任何机器只要能将程序的树结构翻译成汇编语言，就能够编译由前端处理的所有语言。

实际操作过程绝对不像描述的那样简单，但这确实能够实现。

1.6 语言

GCC 可以编译多种语言，但所有这些语言之间有个基本关系。语法分析器由于每种语言语法的惟一性而完全不同，但随着编译过程的前进，这些语言的代码就越来越相似。如前所述， GNU 编译集合可以接受任何形式编程语言的输入，产生的输出也可以在很多不同平台上运行。

1.6.1 C 是基本语言

GCC 的基本语言是 C 语言。整个编译系统由 C 编译程序开始，然后渐渐加了其他的语言进来。幸运的是 C 语言是系统级的语言，能够直接处理计算机程序的基本元素，因此在它上面再创建其他语言的编译程序就相对容易得多。

如果你用其他语言而不是 C 语言编程，而你也对 GCC 很熟悉，你会发现很多东西都是以 C 语言的形式出现的。可将 C 语言想成一种位于 GCC 编译程序的汇编语言之下的语言。大多数编译程序本身都是由 C 语言实现的。

1.6.2 C++ ： 第一个附加语言

C++ 语言是 C 语言的直接扩展（只有很小的改动），因此要向 GCC 加入其他语言，它是最佳首选。 C++ 能够完成的所有事情 C 语言都可以做到，所以没有必要修改编译程序的后端——只需要在前端载入一个新的语法语义分析器。一旦产生中间语言，编译程序的其他部分就和 C 语言完全一样了。

1.6.3 Objective-C

Objective-C 并不像 C 语言或 C++ 语言那样流行，但这是另一种源自（并基于） C 语言的语言。它被看作是“对象化的 C 语言”，事实上也是如此。很大程度上，可以编写 C 程序，而被当作 Objective-C 编译并运行。与基本 C 语法完全不同的特殊语法是用来定义对象的，所以它和纯粹的 C 代码没有什么混淆与冲突。

1.6.4 Fortran

Fortran 可以做到而 C 不能做到的事情就是：科学计算。标准 Fortran 函数库（这是 Fortran 的精髓，因为它就是语言的一部分）是一种扩展，已经趋于完美，而且也历经很多年了。 Fortran 如今被用于科学计算是因为它的基本能力就是快速而准确地实现复杂计算。 Fortran 甚至还将复杂的数字作为它的基本数据类型，而基本数值数据类型可具有很高的精确度。

这种语言的结构比其他的现代语言要麻烦一些，但它包含的一些基本函数和功能的实现是结构化编程所必需的。最新的 Fortran 标准在这一点上有所扩展，无疑是一种非常现代的语言。

1.6.5 Java

Java 是包含进 GCC 的最年轻的语言。 Java 语言和 C++ 一样是基于 C 语言的，但它使用了一些不同的方法来实现类的语法。 C++ 更加灵活， Java 则是通过限制对象的构造函数和析构函数去除了 C++ 的不确定性，它继承的是一些严格无歧义的形式。

Java 和 GCC 包含的其他语言有很大的区别，这是由它对象代码的形式决定的。 Java 会编译成一种对象代码的特殊格式，作为字节码（ bytecodes ）被解释器（叫做 Java 虚拟机）执行。所有 Java 程序都按照这种方式运行，直到 GCC 编译程序增加选项，通过挂接一个 Java 前端到已存在的 GCC 后端来产生本地可执行代码。另外，还添加了一个前端来读出 Java 字节码，并作为源代码用来产生本地可执行的二进制代码。

1.6.6 Ada

最新增加到 GCC 家族的是 Ada 。它是作为一个功能完善的编译程序而加入的，最早是由 Ada Core Technologies 公司独立开发作为 GNAT Ada 95 编译程序，在 2001 年 10 月捐赠给 GCC 。

Ada 编译程序的前端和其他语言的不同，它是由 Ada 编写的。一般来说，安装了 Ada 编译程序就可以了，但在一些系统中会需要特殊的引导过程。而所有其他语言都是由 C 和 C++ 编写的，因此几乎都可以普遍移植。

作为一种语言， Ada 是专门为多个程序员编写大型程序而设计的。在编译 Ada 程序的时候，它交叉引用程序其他部分的源代码来验证正确性。这种语言的语法要求每个函数和过程都要被声明为包的一部分，而包的配置是和声明相匹配的。 C 和 C++ 语言用原型来声明外部引用函数，而 Java 使用文件命名规则定位包的成员，但这两种技术都不像 Ada 那样严格。

1.6.7 不再支持 Chill

GCC 在 3.0 版之后不再支持 Chill 语言。就在发布版本 3.1 之前， Chill 语言的源代码也从 GCC 中移走了。但 GCC 非常复杂，而 Chill 语言完整地作为其一部分已经存在一段时间了，所以还会从 GCC 在线文档中和源代码的各种不同位置中看到对 Chill 语言的引用。本书是在这种转换过程中编写的，所以还会涉及到 Chill 编译程序选项和文件类型。

1.7 部分列表

GCC 是由许多组件组成的。表 1-3 列出了 GCC 的各个部分，但它们也并不总是出现的。有些部分是和语言相关的，所以如果没有安装某种特定语言，系统中就不会出现相关的文件。

表 1-3 GCC 安装的各个部分

部分	描述
c++	gcc 的一个版本，默认语言设置为 C++ ，而且在连接的时候自动包含标准 C++ 库。这和 g++ 一样
cc1	实际的 C 编译程序
cc1plus	实际的 C++ 编译程序
collect2	在不使用 GNU 连接程序的系统上，有必要运行 collect2 来产生特定的全局初始化代码（例如 C++ 的构造函数和析构函数）
configure	GCC 源代码树根目录中的一个脚本。用于设置配置值和创建 GCC 编译程序必需的 make 程序的描述文件
crt0.o	这个初始化和结束代码是为每个系统定制的，而且也被编译进该文件，该文件然后会被连接到每个可执行文件中来执行必要的启动和终止程序

cygwin1.dll	Windows 的共享库提供的 API ，模拟 UNIX 系统调用
f77	该驱动程序可用于编译 Fortran
f771	实际的 Fortran 编译程序
g++	gcc 的一个版本，默认语言设置为 C++ ，而且在连接的时候自动包含标准 C++ 库。这和 c++ 一样
gcc	该驱动程序等同于执行编译程序和连接程序以产生需要的输出

（续表）

部分	描述
gcj	该驱动程序用于编译 Java
gnat1	实际的 Ada 编译程序
gnatbind	一种工具，用于执行 Ada 语言绑定
gnatlink	一种工具，用于执行 Ada 语言连接
jc1	实际的 Java 编译程序
libgcc	该库包含的例程被作为编译程序的一部分，是因为它们可被连接到实际的可执行程序中。它们是特殊的例程，连接到可执行程序，来执行基本的任务，例如浮点运算。这些库中的例程通常都是平台相关的
libgcj	运行时库包含所有的核心 Java 类
libobjc	对所有 Objective-C 程序都必须的运行时库
libstdc++	运行时库，包括定义为标准语言一部分的所有的 C++ 类和函数

表 1-4 列出的软件和 GCC 协同工作，目的是实现编译过程。有些是很基本的（例如 as 和 ld ），而其他一些则是非常有用但不是严格需要的。尽管这些工具中的很多都是各种 UNIX 系统的本地工具，但还是能够通过 GNU 包 binutils 得到大多数工具。安装 binutils 的过程将在第 2 章中介绍。

表 1-4 GCC 使用的软件工具

工具	描述
addr2line	给出一个可执行文件的内部地址， addr2line 使用文件中的调试信息将地址翻译成源代码文件

	名和行号。该程序是 binutils 包的一部分
ar	这是一个程序，可通过从文档中增加、删除和析取文件来维护库文件。通常使用该工具是为了创建和管理连接程序使用的目标库文档。该程序是 binutils 包的一部分
as	GNU 汇编器。实际上它是一族汇编器，因为它可以被编译或能够在各种不同平台上工作。该程序是 binutils 包的一部分
autoconf	产生的 shell 脚本自动配置源代码包去编译某个特定版本的 UNIX
c++filt	程序接受被 C++ 编译程序转换过的名字（不是被重载的），而且将该名字翻译成初始形式。该程序是 binutils 包的一部分
f2c	是 Fortran 到 C 的翻译程序。不是 GCC 的一部分
gcov	gprof 使用的配置工具，用来确定程序运行的时候哪一部分耗时最大
gdb	GNU 调试器，可用于检查程序运行时的值和行为
GNATS	GNU 的调试跟踪系统（ GNU Bug Tracking System ）。一个跟踪 GCC 和其他 GNU 软件问题的在线系统

（续表）

工具	描述
gprof	该程序会监督编译程序的执行过程，并报告程序中各个函数的运行时间，可以根据所提供的配置文件来优化程序。该程序是 binutils 包的一部分
ld	GNU 连接程序。该程序将目标文件的集合组合成可执行程序。该程序是 binutils 包的一部分
libtool	一个基本库，支持 make 程序的描述文件使用的简化共享库用法的脚本
make	一个工具程序，它会读 makefile 脚本来确定程序中的哪个部分需要编译和连接，然后发布必要的命令。它读出的脚本（叫做 makefile 或 Makefile ）定义了文件关系和依赖关系
nlmconv	将可重定位的目标文件转换成 NetWare 可加载模块（ NetWare Loadable Module ， NLM ）。该程序是 binutils 的一部分
nm	列出目标文件中定义的符号。该程序是 binutils 包的一部分
objcopy	将目标文件从一种二进制格式复制和翻译到另外一种。该程序是 binutils 包的一部分
objdump	显示一个或多个目标文件中保存的多种不同信息。该程序是 binutils 包的一部分
ranlib	创建和添加到 ar 文档的索引。该索引被 ld 使用来定位库中的模块。该程序是 binutils 包的

	一部分
ratfor	Ratfor 预处理程序可由 GCC 激活，但不是标准 GCC 发布版的一部分
readelf	从 ELF 格式的目标文件显示信息。该程序是 binutils 包的一部分
size	列出目标文件中每个部分的名字和尺寸。该程序是 binutils 包的一部分
strings	浏览所有类型的文件，析取出用于显示的字符串。该程序是 binutils 包的一部分
strip	从目标文件或文档库中去掉符号表，以及其他调试所需的信息。该程序是 binutils 包的一部分
vcg	Ratfor 浏览器从文本文件中读取信息，并以图表形式显示它们。而 vcg 工具并不是 GCC 发布中的一部分，但 -dv 选项可被用来产生 vcg 可以理解的优化数据的格式
windres	Window 资源文件编译程序。该程序是 binutils 包的一部分

1.8 联系方式

GNU 的主页是 http://www.gnu.org ，而 GCC 项目的主页是 http://gcc.gnu.org 。

GCC 编译程序的范围很广——从简单的批处理工具程序到几百万行的规模的系统。总的来说，当软件项目变得更大或者在某些方面变得特殊，在不能处理某些奇怪问题的时候，就会出现各种情况。有些是 bug ，有些是特殊习惯，但有的不可避免地需要澄清——或者至少能够在正确的方向上引起注意。所幸的是有帮助信息可供使用，而且可以找到关于 GCC 的一切信息。

信息的主要来源是邮件组。开放的邮件组（所有的成员都可以收发邮件）的好处是可以立即展开讨论。如果它有所帮助，我建议注册到 gcc-help 的邮件列表中。开放邮件组上的对话会继续到情况被澄清或问题得到解决。表 1-5 包括所有 GCC 开放邮件组的简要描述。只读邮件组列在表 1-6 中。

表 1-5 GCC 的开放邮件组

邮件组名	描述
gcc	这是开发 GCC 的基本讨论区。如果只是要注册一个邮件组，这就是合适的选择。它能够让你了解到最新的新闻和开发情况。该邮件组信件很多
gcc-bugs	讨论 bug 和报告 bug 的邮件组。这里的邮件也很多
gcc-help	该邮件组适用于那些寻找问题答案的人。该邮件组信件很多

gcc-patches	源代码补丁和关于补丁的讨论会被提交到这个邮件组。该邮件组信件很多
gcc-testresults	测试结果和对测试以及测试结果的讨论都会发到这里
java	关于 GCC 的 Java 前端的开发和维护的讨论列表，以及 Java 的运行时库的讨论列表
java-patches	关于 Java 前端和 Java 运行时库的源代码补丁被发布到该讨论组及 gcc-patches 邮件组
libstdc++	该讨论组用来讨论标准 C++ 库的开发和维护

表 1-6 GCC 的只读邮件组

邮件组名	描述
gccadmin	该邮件组收到的消息来自 gcc.gnu.org 的 gccadmin 账号运行的长时间任务
gcc-announce	该邮件列表信息较少，主要用来公布最新的版本发布，以及其他关于 GCC 的重要事件
gcc-cvs	每个登入到 CVS 仓库的人都会发消息到该邮件组
gcc-cvs-wwwdocs	每个登入到 HTML 文档的 CVS 仓库的人都会发消息到该邮件组
gcc-prs	每次当报告的问题进入 GNATS 数据库的时候都会向该邮件组发消息
gcc-regression	发送到这个邮件组的消息包含的是 GCC 回归测试的运行结果
java-announce	这个邮件组信息很少，它是用来发布关于 Java 前端或者 Java 运行时基本函数的消息的
java-cvs	每次登入到 Java 编译程序和 CVS 仓库运行时部分的时候，都会向该邮件组（以及 gcc-cvs 邮件组）发送消息
java-prs	每次报告的与 Java 有关的问题进入 GNATS 数据库的时候，都会向该邮件组（以及 gcc-prs 邮件组）发送消息
libstdc++-cvs	每次登入到 CVS 仓库的 libstc++ 部分时，都会向该邮件组发送消息

有的邮件组都可在网站 http://www.gnu.org/software/gcc/lists.html 中访问到。在该页面中可以订阅和退订这些邮件组。每个邮件组都有自己的网站，可用来查找并浏览该邮件组收到的消息。该邮件组的名字以 gcc.gnu.org/ml/ 开头是网站的名字。例如，要定位 gcc-announce 的文档网站，就要访问 http://gcc.gnu.org/ml/gcc-announce 。

附录A GNU 通用公共许可证

GCC 编译程序是在 GNU 通用公共许可证（也被叫做 GNU GPL ，或就叫做 GPL ）之内的。由 GPL 保证的这种许可叫做 copyleft （版权所无） 。简单的说，这就意味着任何人都有权利复制并使用该软件，但如果它被集成进产品，该产品就必须也是 GPL 许可证的。就是说，不能使用 GPL 软件将它转换成专有的软件。然而，并不限制任何人用 GCC 作为工具创建自己需要形式的软件。变成生成程序中的二进制位以及片断不要求该程序得到 GPL 许可。

GPL 的另一种方案是弱通用公共许可证（ Lesser General Public License ， LGPL ）。该许可证以前叫做库 GPL ，但这个名字由于可能产生误导就改变了——它满足一些库，但不会是所有库。 LGPL 允许专有程序使用的库例程以及共享的和非静态连接的库。例子就是标准 C 库的 GNU 版本。

下面是 GPL 的文本。它用非常清楚的语言描述了许可的细节。文档的末尾是对处理过程的描述，遵照此过程可以将自己的软件放在 GPL 下。

- # GNU GPL

  Version2, June 1991

  59 Temple Place, Suite 330, Boston , MA 02111-1307 USA

导言

大多数软件许可剥夺你共享和修改软件的自由。相比之下， GNU 通用公共许可力图保证你共享和修改自由软件的自由——保证软件对所有用户是自由的。通用公共许可适用于大多数自由软件协会的软件，以及由使用这些软件而承担义务的作者所开发的软件。（自由软件协会的其他软件受 GNU 库通用公共许可的保护）。你也可以将它应用到你的程序中。

当我们谈到自由软件（ free software ）时，我们指的是自由而不是价格。 GNU 通用公共许可保证你有发布自由软件的自由（如果你愿意，可以对此项服务收取一定的费用）；保证你能收到源程序或者在你需要时能得到它；保证你能修改软件或将它的一部分用于新的自由软件；而且还保证你知道你能做这些事情。

为了保护你的权利，我们作出规定：禁止任何人不承认你的权利，或者要求你放弃这些权利。如果你修改了自由软件或者发布了软件的副本，这些规定就转化为你的责任。

例如，如果你发布了这样一个程序的副本，不管是收费的还是免费的，你必须将你具有的一切权利给予你的接受者；你必须保证他们能收到或得到源程序；并且将这些条款给他们看，使他们知道他们有这样的权利。

我们采取两项措施来保护你的权利。（ 1 ）给软件以版权保护。（ 2 ）给你提供许可证。它给你复制、发布和 / 或修改这些软件的法律许可。

同样，为了保护每个作者和我们自己，我们需要清楚地让每个人明白，自由软件没有担保（ no warranty ）。如果由于其他某个人修改了软件，并继续加以传播。我们需要它的接受者明白：他们所得到的并不是原来的自由软件。由其他人引入的任何问题，不应损害原作者的声誉。

最后，所有自由软件都不断受到软件专利的威胁。我们希望避免这样的风险，自由软件的再发布者以个人名义获得专利许可证，也就是将软件变为私有。为防止这一点，我们必须明确：任何专利必须以允许每个人自由使用为前提，否则就不准许有专利。

下面是有关复制、发布和修改的确切的条款和条件。

GNU 通用公共许可证

有关复制、发布和修改的条款和条件

0. 凡著作权人在其软件或其他著作中声明，该软件或著作得在通用公共许可证条款下才能发布，本许可对其均适用。以下所称的“程序”，是指任何一种适用通用公共许可的程序和著作；“基于本程序的著作”，则指程序或任何基于著作权法所产生的衍生著作，换言之，是指包含本程序全部或部分的著作，不论是否完整或经过修改的程序，以及（或）翻译成其他语言的程序（以下“修改”一词包括但不限于翻译行为）。被许可的人则称为“您”。

本许可不适用于复制、发布及修改以外的行为；这些行为不在本许可范围内。执行本程序的行为并不受限制，而本程序的输出只有在其内容构成基于本程序所生的著作（而非只是因为执行本程序所造成）时，是受到本许可约束的。至于程序输出的内容是否构成本程序的衍生著作，则取决于本程序的具体用途。

1. 您可以对所收受的本程序源码，无论以何种媒介，复制与发布其完整的复制品，然而您必须符合以下条件：以显著及适当的方式在每份复制品上发布适当的著作权标示及无担保声明；维持所有有关本许可以及无担保声明的原貌；并将本许可的副本连同本程序一起交付其他任一位程序收受者。
您可对授让复制品的实际行为请求一定的费用，也可自由决定是否提供担保作为收费的代价。

2. 您可以修改程序的一个或多个复制品或者程序的任何部分，以此形成基于本程序所生成的著作，并依前所述第一条规定，复制与发布修改过的程序或著作，但必须满足以下条件：

a ）您必须在所修改的文件上附加显著的标示，说明您修改过该文件，以及修改日期。

b ）必须就您所发布或发行的著作，无论是包含程序的全部还是部分的著作，或者是自程序或其他任何部分所衍生的著作，整体授权所有第三人可根据本许可规定使用，且不得因此项授权行为收取任何费用。

c ）若经过修改的程序在执行时通常以互动方式读取命令时，您必须在最常用的方式下，于开始进入互动方式时，列出或展示以下宣告：适当的著作权标示以及无担保声明（或声明由您提供担保）、使用者可以根据这些条件再发布此程序，以及告知使用者如何浏览本许可的副本。（例外：若程序本身是以交互方式执行的，然而通常却不回列出该宣告时，那您基于本程序所生成的著作便无需列出该宣告。）

这些要求对修改过的著作是整体适用的。如果著作中可识别的一部分并非衍生自本程序，并且可合理地认为是一个独立、个别的著作，则当您将其作为个别著作加以发布时，本许可及其条款将不适用于该部分。
当然当您将上述部分作为基于程序所生的著作的一部分而发布时，整个著作的发布就必须符合本许可条款的规定，而不管这些部分的作者是谁。
因此，本条规定的意图不在于主张或剥夺您对完全由您所完成的著作的权利；应该说，本条规定意在行使对于基于程序的衍生著作或集合著作的发布行为的控制权。

此外，非基于本程序所生的其他著作与本程序（或基于本程序产生的著作）在同一存储或发布媒介上的单纯聚集行为，并不会使该著作因此受到本许可条件的约束。

3. 可根据前面第一、二条规定，复制和发布程序（或第二条所述基于程序产生的著作）的目标代码或可执行形式，但必须符合以下条件：

a ）附上完整的相对机器可读的源码，而这些源码必须依前面第一、二条规定在经常作为软件交互的媒介上发布；或

b ）附上至少三年有效的书面报价文件，提供任何第三人在支付不超过实际发布源码所需成本的费用下，取得相同源码的完整的机器可读的复制品，并依照前面第一、二条规定在经常用以作为软件交互的媒介上发布该复制品；或

c ）附上所收受的有关发布相同源码的报价信息。（本项选择仅在非营利发布、且只在您依照前面 b 项方式自该书面报价文件收受程序目标代码或可执行形式时，才能适用。）

著作的源码是指对著作进行修改时适用的形式。对于可执行的著作而言，完整的源码是指著作中包含所有模式的全部源码，加上相关接口定义文件，还有用以控制该著作编译及安装的描述。然而，特别的例外情况是，所发布的源码并不需包括任何通常会随着所执行操作系统的主要组成部分（编译程序、核心等）而发布的软件（无论以源码或二进制格式），除非该部分本身就附加在可执行程序中。
如果可执行代码或目标代码的发布方式，是以指定的地点存取，供人复制，则提供可自相同地点复制源码的使用机会，看作是对源码的发布，然而第三人并不因此而负有将目标代码连同源码一起复制的义务。

4. 除本许可所明示的方式外，不得对程序加以复制、修改、再授权或发布。任何企图以其他任何方式进行复制、修改、再授权或发布程序的行为均为无效行为，并将自动终止您基于本许可所享有的权利。但依照本许可规定，从您手中收受复制品或权利的人，只要遵守本许可规定，他们所获得的许可并不会因此终止。

5. 因为您并未在本许可上签名，所以无需接受本许可。但除此之外，别无其他方式可以修改或发布程序或其衍生作品的授权许可。如不接受本许可，则这些行为在法律上都是被禁止的。因此对程序（或任何基于本程序所生成的著作）的修改和发布行为，表示已经接受了本许可，以及接受了所有关于复制、发布及修改程序（或基于程序生成著作）的条款和条件。

6. 每当再次发布程序（或任何基于程序所生成的著作）时，收受者就自动获得原授权人所有的关于复制、发布或修改程序的权利。不得就本授权所赋予接收者行使的权利附加任何进一步的限制。对于第三人是否履行本许可，无须负责。

7. 如果法院判决、专利侵权主张或其他任何理由（不限于专利争议）的结果，使得加诸于您的条件（无论是由法院命令、协议或其他方式造成）与本许可规定有所冲突，他们并不免除您必须遵守本许可的规定。如果无法同时符合本许可条件所生成的义务及其他相关义务而进行发布，那么后果就是不得发布该程序。例如，如果专利授权不允许直接或间接通过您而取得复制品的人，以免付权利金的方式再发布该程序时，惟一能够同时满足该义务及本许可的方式就是彻底避免发布本程序。

如果本条任何一部分在特殊情况下被认定无效或无法执行时，本条其余部分仍应适用，且本条全部内容在其他情况下仍然应该适用。

本条的目的并不是诱使您侵害专利或其他财产权的权利主张，或就此类主张的有效性加以争执；本条的惟一目的是保障公共授权惯例所执行的自由软件发布系统的完整性。许多人信赖该系统一贯使用的应用程序，而对经由此系统发布的大量软件有相当多的贡献；作者 / 贡献者有权决定他或她是否希望经由其他系统发布软件，而被授权人则无该种选择权。

本条的用意在于将本许可其他不确定部分彻底解释清楚。

8. 如果因为专利或版权保护的接口问题，而使得本程序的发布与 / 或使用局限于某些国家时，则将本程序置于本许可规范之下的原著作权人得增加明确的发布地区限制条款，将一些国家排除在外，而使发布的许可只限制在未排除的国家之内。在该情况下，将限制条款如同以书面方式订定于本许可内容中，而成为本许可的条款。

9. 自由软件协会可以随时发布通用公共许可的修正版和 / 或新版本。新版本在精神上将近似于目前的版本，但在细节上有所不同以满足新的问题或状况。

每个版本都有单独的版本编号。如果程序指定授权版本编号，表示其适用于该版本或是“任何新版本”时，得选择遵循该版本或任何由自由软件协会日后所发布的更新版本的条款或条件。如果程序没有指出授权版本编号，便要选择任一自由软件协会所发布的版本。

10. 如果想要将部分程序纳入其他自由程序，而其发布的条件有所不同时，请写信取得作者的许可。如果是自由软件协会享有著作权的软件，请写信到自由软件协会；我们有时会以特殊方式予以处理。我们的决定取决于两项目标：确保自由软件的所有衍生著作均维护在自由状态，并广泛地促进软件的共享或再利用。

无担保声明

11. 由于本程序是无偿授权的，因此在法律许可范围内，本许可对程序不负担责任。非经书面声明，著作权人和 / 或其他提供程序的人，无论显式或隐式，均是依照“现状”提供程序而并无任何形式的担保责任，其包含并不限于，就适售性以及特定目的的使用性为默认性担保。有关程序品质与效能的全部风险都由自己承担。如果程序被证明有瑕疵，您应该承担所有服务、修复或改正的费用。

12. 非经法律要求或书面同意，任何著作权人或任何可能依前述方式修改和 / 或发布程序的人，对于您因为使用或不能使用程序所造成的一般性、特殊性、意外性或间接性损失，不负任何责任（包括但不限于数

据损失、数据执行不精确、或应由您或第三人承担的损失，或程序无法与其他程序运作等），即便前述的著作权人或其他团体已被告知这种损失的可能性。

条款结束

如何将这些条款用于你的新程序

如果你开发了新程序，而且希望它得到公众最大限度的利用。最好的办法就是将它变为自由软件。使得每个人都能在遵守条款的基础上对它进行修改和重新发布。

为了做到这一点，给程序附上下列声明。最安全的方式是将它放在每个源程序的开头，以便最有效地传递拒绝担保的信息。每个文件至少应有“版权所有”行以及在什么地方能看到声明全文的说明。

<one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author>

这一程序是自由软件，你可以遵照自由软件协会出版的 GNU 通用公共许可证条款来修改和重新发布这一程序。或者用许可证的第二版，或者（根据你的选择）用任何更新的版本。

发布这一程序的目的是希望它有用，但没有任何担保。甚至没有适合于特定目的的隐含的担保。更详细的情况请参阅 GNU 通用公共许可证。

你应该已经和程序一起收到一份 GNU 通用公共许可证的副本。如果还没有，写信给自由软件协会（ The Free Software Foundation, Inc. ），地址： 59 Temple Place ， Suite 330, Boston, MA 02111-1307 USA 。

还应加上如何和你取得联系的相关信息。如果程序以交互方式进行工作，当它开始进入交互方式时，使它输出类似下面的简短声明：

Gnomovision version 69, Copyright (C) year name of author

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.

This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

假设的命令‘ show c ' 及‘ show w ' 应显示通用公共许可证的相应条款。当然，你使用的命令名称可以不同于‘ show c ' 及‘ show w ' 。根据程序的具体情况，也可以用菜单或鼠标选项来显示这些条款。

如果需要，你应该取得你的上司（如果你是程序员）或你的学校签署放弃程序版权的声明。下面只是一个例子，你应该改变相应的名称：

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

signature of Ty Coon, 1 April 1989

Ty Coon, President of Vice

这一许可证不允许你将程序并入专用程序。如果你的程序是一个子程序库，你可能会认为用库的方式和专用应用程序连接更有用。如果你想这样做，使用 GNU 库通用公共许可证代替本许可证。

附录B 环境变量

有大量的环境变量可供设置以影响 GCC 编译程序的方式。利用这些变量的控制也可使用合适的命令行选项。

一些环境变量设置在目录名列表中。这些名字和 PATH 环境变量使用的格式相同。特殊字符 PATH_SEPARATOR （安装编译程序的时候定义）用在目录名之间。在 UNIX 系统中，分隔符是冒号，而 Windows 系统中为分号。

C_INCLUDE_PATH

编译 C 程序时使用该环境变量。该环境变量指定一个或多个目录名列表，查找头文件，就好像在命令行中指定 -isystem 选项一样。会首先查找 -isystem 指定的所有目录。

也见 CPATH 、 CPLUS_INCLUDE_PATH 和 OBJC_INCLUDE_PATH 。

COMPILER_PATH

该环境变量指定一个或多个目录名列表，如果没有指定 GCC_EXEC_PREFIX 定位子程序，编译程序会在此查找它的子程序。

也见 LIBRARY_PATH 、 GCC_EXEC_PREFIX 和 -B 命令行选项。

CPATH

编译 C 、 C++ 和 Objective-C 程序时使用该环境变量。该环境变量指定一个或多个目录名列表，查找头文件，就好像在命令行中指定 -I 选项一样。会首先查找 -I 指定的所有目录。

也见 C_INCLUDE_PATH 、 CPLUS_INCLUDE_PATH 和 OBJC_INCLUDE_PATH 。

CPLUS_INCLUDE_PATH

编译 C++ 程序时使用该环境变量。该环境变量指定一个或多个目录名列表，查找头文件，就好像在命令行中指定 -isystem 选项一样。会首先查找 -isystem 指定的所有目录。

也见 CPATH 、 C_INCLUDE_PATH 和 OBJC_INCLUDE_PATH 。

DEPENDENCIES_OUTPUT

为文件名设置该环境变量会让预处理程序将基于依赖关系的 makefile 规则写入文件。不会包括系统头文件名字。

如果环境变量设置为单名，被看作是文件名字，而依赖关系规则的名字来自源文件名字。如果定义中有两个名字，则第二个名字是用作依赖关系规则的目标名。

设置该环境变量的结果和使用命令行选项 -MM 、 -MF 和 -MT 的组合是一样的。也见 SUNPRO_DEPENDENCIES 。

GCC_EXEC_PREFIX

如果定义了该环境变量，它会作为编译程序执行的所有子程序名字的前缀。例如，如果将变量设置为 testver 而不是查找 as ，汇编器首先会在名字 testveras 下查找。如果在此没有找到，编译程序会继续根据它的普通名进行查找。可在前缀名中使用斜线指出路径名。

GCC_EXEC_PREFIX 的默认设置为 *prefix* /lib/gcc-lib/ ，这里的 *prefix* 是安装编译程序时 configure 脚本指定的名字。该前缀也用于定位标准连接程序文件，包含进来作为可执行程序的一部分。

如果使用 -B 命令行选项，会重写该设置。也见 COMPILER_PATH 。

LANG

该环境变量用于指出编译程序使用的字符集，可创建宽字符文字、串文字和注释。

定义 LANG 为 C-JIS ，指出预处理程序将多字节字符按照 JIS （日语工业标准）字符进行解释。 C-SJIS 可用来指出 Shift -JIS 字符而 C-EUCJP 指出日文 EUC 。

如果没有定义 LANG ，或定义为不可识别，函数 mblen() 被用来确定字符宽度，而 mbtowc() 用来将多字节序列转换为宽字符。

LC_ALL

如果设置，该环境变量的值重写 LC_MESSAGES 和 LC_CTYPE 的所有设置。

LC_CTYPE

该环境变量指出引用串中定义的多字节字符的字符分类。主要用于确定字符串的字符边界，字符编码需要用引号或转义符，可被错误地解释为字符串的结尾或特殊字符串。对 Australian English ，可将它设置为 en_AU ； 对 Mexican Spanish ，可将它设置为 es_MX 。如果没有设置该变量，默认为 LANG 变量的值，或如果没有设置 LANG ，那就使用 C 英语行为。也见 LC_ALL 。

LC_MESSAGES

该环境变量指出编译程序使用何种语言发出诊断消息。对 Australian English ，可设置为 en_AU ； 对 Mexican Spanish ，可设置为 es_MX 。如果变量没有设置，使用 LANG 变量的默认值，或如果没有设置 LANG ，那就使用 C 英语行为。也见 LC_ALL 。

LD_LIBRARY_PATH

该环境变量不会影响编译程序，但程序运行的时候会有影响。变量指定一个目录列表，程序会查找该列表定位共享库。只有当未在编译程序的目录中找到共享库的时候，执行程序必须设置该变量。

LD_RUN_PATH

该环境变量不会影响编译程序，但程序运行的时候会有影响。该变量在运行时指出文件的名字，运行的程序可由此得到它的符号名字和地址。地址不会重新载入，因而可能符号引用其他文件中的绝对地址。这和 ld 工具使用 -R 选项完全一样。

LIBRARY_PATH

该环境变量可设置为一个或多个目录名字列表，连接程序会搜寻该目录，以查找特殊连接程序文件，和由 -l （字母 *l* ）命令行选项指定名字的库。

由 -L 命令行选项指定的目录在环境变量的前面，首先被查找。也见 COMPILER_PATH 。

OBJC_INCLUDE_PATH

在编译 Objective-C 程序的时候使用该环境变量。一个或多个目录名的列表由环境变量指定，用来查找头文件，就好像在命令行中指定 -isystem 选项一样。所有由 -isystem 选项指定的目录会首先被查找。

也见 CPATH 、 CPLUS_INCLUDE_PATH 和 C_INCLUDE_PATH 。

SUNPRO_OUTPUT

为文件名设置该环境变量会令预处理程序将基于依赖关系的 makefile 规则写入文件。会包含系统头文件名。

如果环境变量被设置为单个名字，它将会被当作文件名，依赖关系规则中的名字将由源文件的名字中获得。如果定义中有两个名字，第二个名字就是依赖关系规则中的目标名。

设置该环境变量的结果与在命令行中使用参数 -M 、 -MF 和 -MT 的效果一样。参见 DEPENDENCIES_OUTPUT 。

TMPDIR

这个变量包含了供编译程序存放临时工作文件的目录的路径名。这些文件通常在编译过程结束时被删除。这种文件的一个例子就是由预处理程序输出并输入给编译程序的文件。

附录C 命令行对照表

本附录是命令行选项的对照表。命令行选项按照关键字和类别顺序列出。关键字就是从选项名中析取出来的字，类别则由选项的功能导出。你可以按照选项所作用的语言名称查找，也可以按照这些选项是否作用于编译程序内部操作等进行查找。例如，如果你想要知道哪个命令行选项会影响预处理程序或连接程序，可以在这里查找。

对照表

Ada -gnat , -I , --include-directory , --no-standard-includes , -nostdinc

alias -fargument-alias , -fargument-noalias , -fargument-noalias-global , -fstrict-aliasing

align -falign-functions , -falign-jumps , -falign-labels , -falign-loops , -Wcast-align

argument -fargument-alias , -fargument-noalias , -fargument-noalias-global , -fugly-args , -Wformat-extra-args

asm --assemble , -fasm , -fdata-sections , -ffunction-sections , -finhibit-size-directive ,

--for-assembler , -fverbose-asm , -Wa

assert -A , -A- , --assert

atexit -fuse-cxa-atexit

bitfields -fsigned-bitfields , -funsigned-bitfields

boehm -fuse-boehm-gc

bounds -fbounds-check , -ffortran-bounds-check

C - -ansi , -ansi , -aux-info , -c , -C , -fallow-single-precision , -fasm , -fbuiltin , -fcommon ,

-fcond-mismatch , -fdollars-in-identifiers , -fdump-translation-unit , -ffreestanding , -fhosted ,

-finline , -fshort-wchar , -fsigned-bitfields , -fsigned-char , -funsigned-bitfields , -funsigned-char ,

-fwritable-strings , -pedantic , -pedantic-errors , -std , -traditional-cpp , -Waggregate-return ,

-Wbad-function-cast , -Wcast-align , -Wcast-qual , -Wchar-subscripts , -Wcomment , -Wconversion ,

-Wdeprecated-declarations , -Werror-implicit-function-declaration , -Wformat ,

-Wformat-extra-args , -Wformat-nonliteral , -Wformat-security , -Wformat-y2k , -Wimplicit ,

-Wimplicit-function-declaration , -Wimplicit-int , -Wimport , -Winline , -Wlarger-than-size ,

-Wlong-long , -Wmain , -Wmissing-braces , -Wmissing-declarations , -Wmissing-format-attribute ,

-Wmissing-noreturn , -Wmissing-prototypes , -Wmultichar , -Wnested-externs , -Wpacked ,

-Wpadded , -Wparentheses , -Wpointer-arith , -Wredundant-decls , -Wreturn-type ,

-Wsequence-points , -Wshadow , -Wsign-compare , -Wstrict-prototypes , -Wswitch ,

-Wsystem-headers , -Wtraditional , -Wtrigraphs , -Wundef , -Wuninitialized , -Wwrite-strings

C++ --ansi, -ansi , -faccess-control , -falt-external-templates , -fasm , -fcheck-new ,

-fconserve-space , -fconst-strings , -fdefault-inline , -fdollars-in-identifiers , -fdump-class-hierarchy ,

-fdump-translation-unit , -fdump-tree-switch , -felide-constructors , -fenforce-eh-specs ,

-fexternal-templates , -ffor-scope , -fgnu-keywords , -fimplement-inlines ,

-fimplicit-inline-templates , -fimplicit-templates , -finline , -fmemoize-lookups , -fms-extensions ,

-fnonansi-builtins , -foperator-names , -foptional-diags , -fpermissive , -frepo , -frtti , -fshort-wchar , -fstats , -ftemplate-depth-number , -fuse-cxa-atexit , -fvtable-gc , -fweak , -fwritable-strings , -nostdinc++ , -pedantic , -pedantic-errors , -Waggregate-return , -Wcast-align , -Wcast-qual , -Wchar-subscripts , -Wcomment , -Wconversion , -Wctor-dtor-privacy , -Wdeprecated , -Wdeprecated-declarations , -Weffc++ , -Wextern-inline , -Wformat , -Wformat-extra-args , -Wformat-nonliteral , -Wformat-security , -Wformat-y2k , -Wimport , -Winline , -Wlarger-than-size , -Wlong-long , -Wmain , -Wmissing-braces , -Wmissing-format-attribute , -Wmissing-noreturn , -Wmultichar , -Wnon-template-friend , -Wnon-virtual-dtor , -Wold-style-cast , -Woverloaded-virtual , -Wpacked , -Wpadded , -Wparentheses , -Wpmf-conversions , -Wpointer-arith , -Wredundant-decls , -Wreorder , -Wreturn-type , -Wshadow , -Wsign-compare , -Wsign-promo , -Wswitch , -Wsynth , -Wsystem-headers , -Wundef , -Wuninitialized , -Wwrite-strings

call -fcall-saved-register , -fcall-used-register , -fcaller-saves , -fnon-call-exceptions , -foptimize-sibling-calls

case -fcase-initcap , -fcase-lower , -fcase-preserve , -fcase-strict-lower , -fcase-strict-upper , -fcase-upper , -fignore-case , -fintrin-case-spec , -fmatch-case-spec , -fsource-case-spec , -fsymbol-case-spec

cast -Wbad-function-cast , -Wcast-align , -Wcast-qual , -Wold-style-cast

char -fsigned-char , -funsigned-char , -Wchar-subscripts

check -fbounds-check , -fcheck-new , -fcheck-references , -fdelete-null-pointer-checks , -fforce-classes-archive-check , -ffortran-bounds-check , -fruntime-checking , -fstack-check , -fstore-check

Chill -fchill-grant-only , -fgrant-only , -fignore-case , -flocal-loop-counter , -fold-string , -fruntime-checking , -I , --include-directory , -lang-chill

class --bootclasspath , -fconstant-string-class , -fdump-class-hierarchy ,

-fforce-classes-archive-check , -foptimize-static-class-initialization , -foutput-class-dir , --main ,

--output-class-directory

comment -C , -Wcomment

compile -c , --compile , -E , -fassume-compiled , -fcommon , -fcompile-resource ,

-fgnu-linker , -fgrant-only , -fhosted , -fmerge-all-constants , -fmerge-constants , -fsyntax ,

-ftest-coverage , -ftime-report , --help , -pass-exit-codes , -pipe , --preprocess , -Q , -s , -S ,

-save-temps , -syntax-only , -time , -v

complex -femulate-complex , -fugly-complex

constant -fconst-strings , -fconstant-string-class , -fkeep-static-consts ,

-fmerge-all-constants , -fmerge-constants , -fsingle-precision-constant

conversion -Wconversion , -Wpmf-conversions

cse -fcse-follow-jumps , -fcse-skip-blocks , -ffunction-cse , -frerun-cse-after-loop

debug -a , -d , --debug , -fdump-class-hierarchy , -fdump-translation-unit ,

-fdump-tree-switch , -fdump-unnumbered , -finstrument-functions , -fmem-report , -foptional-diags ,

-fpermissive , -fsilent , -fstats , -ftemplate-depth-number , -ftrapping-math , -ftrapv , -fverbose-asm ,

-g , -gcoff , -gdwarf , -gdwarf-2 , -ggdb , -gstabs , -gvms , -gxcoff , -H , -v

declaration -gen-decls , -Wdeprecated-declarations , -Werror-implicit-function-declaration ,

-Wimplicit-function-declaration , -Wmissing-declarations , -Wredundant-decls

dependencies --dependencies , -M , -MD , -MF , -MG , -MM , -MMD , -MP , -MQ , -MT ,

-pass-exit-codes , --print-missing-file-dependencies, --user-dependencies, -Wout-of-date,

--write-dependencies, --write-user-dependencies

deprecated -Wdeprecated , -Wdeprecated-declarations

directory -B , --bootclasspath directory, -foutput-class-dir, -I, -I-, -idirafter, -include,

--include-barrier, --include-directory, --include-directory-after, --include-prefix,

--include-with-prefix, --include-with-prefix-after, -iprefix, -isystem, -iwithprefix,

-iwithprefixbefore, -L, --library-directory, --output-class-directory, --prefix,

-print-multi-directory , -print-prog-name , -print-search-dirs

dollar -fdollar-ok , -fdollars-in-identifiers

dump -d , --dump , -dumpbase , -dumpmachine , -dumpspecs , -dumpversion ,

-fdump-class-hierarchy , -fdump-translation-unit , -fdump-tree-switch , -fdump-unnumbered

error -fmessage-length , -pedantic-errors , -Werror , -Werror-implicit-function-declaration

exception -fasynchronous-unwind-tables , -fcheck-new , -fenforce-eh-specs , -fexceptions ,

-fnon-call-exceptions , -fnonansi-builtins

extern -falt-external-templates , -fexternal-templates , -Wextern-inline , -Wnested-externs

file -aux-info , -B , --bootclasspath, -include , --include-barrier, --include-directory-after,

--include-prefix, --include-with-prefix, --include-with-prefix-after, --language, -llibrary, -MF,

--output, -print-file-name, -print-libgcc-file-name, --print-missing-file-dependencies,

-print-prog-name , -remap , -save-temps , -x

float -ffloat-store , -fpretend-float , -Wfloat-equal

form -ffixed-form , -ffree-form

format -Wformat , -Wformat-extra-args , -Wformat-nonliteral , -Wformat-security ,

-Wformat-y2k , -Wmissing-format-attribute

Fortran -fautomatic , -fbackslash , -fbadu77-intrinsics-spec , -fbounds-check , -fcase-initcap ,

-fcase-lower , -fcase-preserve , -fcase-strict-lower , -fcase-strict-upper , -fcase-upper , -fdollar-ok ,

-femulate-complex , -ff2c , -ff2c-intrinsics-spec , -ff66 , -ff77 , -ff90 , -ff90-intrinsics-spec ,

-ffixed-form , -ffixed-line-length-len , -ffortran-bounds-check , -ffree-form , -fglobals ,

-fgnu-intrinsics-spec , -finit-local-zero , -finline , -fintrin-case-spec , -fmatch-case-spec ,
-fmil-intrinsics-spec , -fonetrip , -fpedantic , -fsecond-underscore , -fsilent , -fsource-case-spec ,
-fsymbol-case-spec , -fsyntax , -ftypeless-boz , -fugly-args , -fugly-assign , -fugly-assumed ,
-fugly-comma , -fugly-complex , -fugly-init , -fugly-logint , -funderscoring , -funix-intrinsics ,
-fversion , -fvxt , -fvxt-intrinsics , -fzeros , -maligned-data , -pedantic , -pedantic-errors , -Wglobals ,
-Wimplicit , -Wsurprising , -Wuninitialized

function -falign-functions , -ffunction-cse , -ffunction-sections , -finline-functions ,
-finstrument-functions , -fkeep-inline-functions , -Wbad-function-cast ,
-Werror-implicit-function-declaration , -Wimplicit-function-declaration , -Wunused-function

garbage -fuse-boehm-gc , -fvtable-gc

gcse -fgcse , -fgcse-lm , -fgcse-sm

global -fargument-noalias-global , -fglobals , -fvolatile-global , -Wglobals

gnu -fgnu-intrinsics-spec , -fgnu-keywords , -fgnu-linker , -fgnu-runtime

implicit -fimplicit-inline-templates , -fimplicit-templates ,
-Werror-implicit-function-declaration , -Wimplicit , -Wimplicit-function-declaration ,
-Wimplicit-int

include -include , --include-barrier, --include-directory, --include-directory-after,
--include-prefix, --include-with-prefix, --include-with-prefix-after, --include-with-prefix-before,
--no-standard-includes, --trace-includes

inline -fasm , -fdefault-inline , -fimplement-inlines , -fimplicit-inline-templates , -finline ,
-finline-functions , -finline-limit , -fkeep-inline-functions , -Wextern-inline , -Winline

intrinsics -fbadu77-intrinsics-spec , -ff2c-intrinsics-spec , -ff90-intrinsics-spec ,
-fgnu-intrinsics-spec , -fmil-intrinsics-spec , -funix-intrinsics , -fvxt-intrinsics iprefix

Java --bootclasspath, -C, -D, --define-macro,--encoding, -fassume-compiled ,

-fbounds-check , -fcheck-references , -fcompile-resource , -fencoding ,

-fforce-classes-archive-check , -fhash-synchronization , -fjni , -foptimize-static-class-initialization ,

-foutput-class-dir , -fstore-check , -fuse-boehm-gc , -fuse-divide-subroutine , -I , --include-directory,

--main, --output-class-directory, -Wextraneous-semicolon , -Wlarger-than-size , -Wout-of-date ,

-Wredundant-modifiers , -Wshadow

label -falign-labels , -Wunused-label

length -ffixed-line-length-len , -fmessage-length

lib -B , -L , -l , --library-directory, --no-standard-libraries, -print-libgcc-file-name ,

-print-multi-lib , -shared , -shared-libgcc , -static , -static-libgcc , -symbolic

link -c , --compile, -fcommon, -fgnu-linker, -fhosted, -fmerge-all-constants,

-fmerge-constants, --for-linker, --force-link, -fvtable-gc, -L, --library-directory, -llibrary,

-no-standard-libraries, -nodefaultlibs , -nostartfiles , -nostdlib , -s , -shared , -shared-libgcc , -static ,

-static-libgcc , -symbolic , -u , -Wl , -Xlinker

machine -b , --target, --target-help

macro --ansi, -D, --define-macro, -ffast-math, -ffixed-register, -imacros, -U, -undef,

--undefine-macro

math -fallow-single-precision , -femulate-complex , -ffast-math , -ffloat-store , -fmath-errno ,

-fpretend-float , -fschedule-insns , -fschedule-insns2 , -fshort-double , -fsingle-precision-constant ,

-ftrapping-math , -ftrapv , -ftypeless-boz , -fugly-complex , -funsafe-math-optimizations ,

-funsigned-bitfields , -funsigned-char , -fuse-divide-subroutine , -Wdiv-by-zero , -Wfloat-equal ,

-Wsign-compare , -Wsign-promo , -Wsurprising

missing --print-missing-file-dependencies, -Wmissing-braces, -Wmissing-declarations,

-Wmissing-format-attribute , -Wmissing-noreturn , -Wmissing-prototypes

Objective-C --ansi,-ansi , -fasm , -fbuiltin , -fconstant-string-class , -fgnu-runtime , -finline ,
-gen-decls , -Waggregate-return , -Wcast-align , -Wcast-qual , -Wchar-subscripts , -Wcomment ,
-Wconversion , -Wdeprecated-declarations , -Wformat , -Wformat-extra-args , -Wformat-nonliteral ,
-Wformat-security , -Wformat-y2k , -Wimport , -Winline , -Wlarger-than-size , -Wlong-long ,
-Wmissing-braces , -Wmissing-format-attribute , -Wmissing-noreturn , -Wmultichar , -Wpacked ,
-Wpadded , -Wparentheses , -Wpointer-arith , -Wprotocol , -Wredundant-decls , -Wselector ,
-Wshadow , -Wsign-compare , -Wswitch , -Wsystem-headers , -Wundef , -Wuninitialized

optimization -fasynchronous-unwind-tables , -fbranch-probabilities , -fcall-saved-register ,
-fcall-used-register , -fcaller-saves , -fcommon , -fconserve-space , -fcprop-registers ,
-fcse-follow-jumps , -fcse-skip-blocks , -fdata-sections , -fdefer-pop , -fdelayed-branch ,
-fdelete-null-pointer-checks , -fdiagnostics-show-location , -felide-constructors ,
-fexpensive-optimizations , -ffloat-store , -ffunction-cse , -ffunction-sections , -fgcse , -fgcse-lm ,
-fglobals , -fguess-branch-probability , -finit-local-zero , -fkeep-static-consts , -fmemoize-lookups ,
-fmove-all-movables , -fomit-frame-pointer , -foptimize-register-move , -foptimize-sibling-calls ,
-foptimize-static-class-initialization , -fpack-struct , -fpeephole , -fpeephole2 , -fppc-struct-return ,
-fprefetch-loop-arrays , -freduce-all-givs , -freg-struct-return , -fregmove , -frename-registers ,
-frerun-cse-after-loop , -frerun-loop-opt , -fruntime-checking , -fschedule-insns , -fschedule-insns2 ,
-fshort-double , -fshort-enums , -fssa , -fssa-ccp , -fssa-dce , -fstack-check , -fstore-check ,
-fstrength-reduce , -fstrict-aliasing , -fthread-jumps , -funroll-all-loops , -funroll-loops ,
-funwind-tables , -fvtable-gc , -fzeros , -O , --optimize optimize, --optimize, --param,
-Wdisabled-optimization

preprocessor -A, -A-, --assert, -C, -D, --define-macro, --dependencies directory, -E,
-fident, -fpreprocessed, -H, -I, -I-, -idirafter, -imacros, -include, --include-barrier,
--include-directory, --include-directory-after, --include-prefix, --include-with-prefix,
--include-with-prefix-after, --include-with-prefix-before, -iprefix, -isystem, -iwithprefix,
-iwithprefixbefore, -M,-MD, -MF, -MG, -MM, -MMD, -MP, -MQ, -MT, --no-line-commands,
--no-standard-includes, -nostdinc, -nostdinc++, -P, --preprocess,
--print-missing-file-dependencies, -remap, --trace-includes, -trigraphs, -U, -undef,

--undefine-macro, --user-dependencies, -Wp, --write-dependencies, --write-user-dependencies,

-Wsystem-headers, -Wundef, -Wunknown-pragmas

profile -a , -fdata-sections , -fprofile-arcs , -ftest-coverage , -p , -pg , --profile, --profile-blocks

prototypes -Wmissing-prototypes , -Wstrict-prototypes

register -fcall-saved-register , -fcall-used-register , -fcprop-registers , -ffixed-register ,

-fforce-addr , -fforce-mem , -foptimize-register-move , -freg-struct-return , -fregmove ,

-frename-registers , -fstack-limit-register , -remap

return -fppc-struct-return , -freg-struct-return , -Waggregate-return , -Wreturn-type

sign -fsigned-bitfields , -fsigned-char , -Wsign-compare , -Wsign-promo

ssa -fssa , -fssa-ccp , -fssa-dce

stack -fstack-check , -fstack-limit-register , -fstack-limit-symbol

standard --ansi , -ansi , -ff2c , -ff2c-intrinsics , -ff66 , -ff77 , -ff90 , -ff90-intrinsics ,

-ffixed-form , -ffixed-line-length-len , -ffor-scope , -ffree-form , -fgnu-keywords , -fmil-intrinsics ,

-fms-extensions , -fnext-runtime , -fnonansi-builtins , -foperator-names , -fpedantic , -fpermissive ,

-fsigned-bitfields , -fsigned-char , -ftrapping-math , -ftrapv , -fugly-args , -fugly-assign ,

-fugly-assumed , -fugly-comma , -fugly-complex , -fugly-init , -fugly-logint , -fvxt , -fvxt-intrinsics ,

-fwritable-strings , --no-standard-includes, --no-standard-libraries, -pedantic , -std , -traditional-cpp ,

-Wtraditional

static -fkeep-static-consts , -foptimize-static-class-initialization , -fvolatile-static , -static ,

-static-libgcc

strings -fconst-strings , -fconstant-string-class , -fold-string , -fwritable-strings ,

-Wwrite-strings

syntax -fsyntax , -syntax-only

template -falt-external-templates , -fexternal-templates , -fimplicit-inline-templates ,

-fimplicit-templates , -ftemplate-depth-number , -Wnon-template-friend

underscore -fleading-underscore , -fsecond-underscore , -funderscoring

version -dumpversion , -fversion , --use-version , -v , -V

warn --all-warnings, --extra-warnings, -fmessage-length,--no-warnings, -w, -W,

-Waggregate-return, -Wall, --warn-, -Wbad-function-cast,-Wcast-align,-Wcast-qua 1,

-Wchar-subscripts, -Wcomment, -Wconversion, -Wctor-dtor-privacy, -Wdeprecated,

-Wdeprecated-declarations, -Wdisabled-optimization, -Wdiv-by-zero, -Weffc++, -Werror,

-Werror-implicit-function-declaration, -Wextern-inline, -Wextraneous-semicolon, -Wfloat-equal,

-Wformat, -Wformat-extra-args, -Wformat-nonliteral, -Wformat-security, -Wformat-y2 k,

-Wglobals, -Wimplicit, -Wimplicit-function-declaration, -Wimplicit-int, -Wimport, -Winline,

-Wlarger-than-size, -Wlong-long, -Wmain, -Wmissing-braces, -Wmissing-declarations,

-Wmissing-format-attribute, -Wmissing-noreturn, -Wmissing-prototypes, -Wmultichar,

-Wnested-externs, -Wnon-template-friend, -Wnon-virtual-dtor, -Wold-style-cast, -Wout-of-date,

-Woverloaded-virtual, -Wpacked, -Wpadded, -Wparentheses, -Wpmf-conversions,

-Wpointer-arith, -Wprotocol, -Wredundant-decls, -Wredundant-modifiers, -Wreorder,

-Wreturn-type, -Wselector, -Wsequence-points, -Wshadow, -Wsign-compare, -Wsign-promo,

-Wstrict-prototypes, -Wsurprising, -Wswitch, -Wsynth, -Wsystem-headers, -Wtraditional,

-Wtrigraphs, -Wundef, -Wuninitialized, -Wunknown-pragmas, -Wunreachable-code, -Wunused,

-Wunused-function, -Wunused-label, -Wunused-parameter, -Wunused-value, -Wunused-variable,

-Wwrite-strings

附录D 命令行选项

本附录按照字母顺序列出所有的命令行选项。这些选项中的某些可以作用于每种语言，而另一些则只能作用于一种或一些语言，它们只是多种语言中的一小部分。也有一些作用于预处理程序、汇编器或是链接器

的选项。对每个选项，都在相应选项的右边标出了它所作用的一种或多种语言。如果某个选项作用于所有语言，就没有这样的标记了。

事实上用 gcc 命令可以编译任何一种语言，但可能就无法使用某种限制语言的选项了。每种语言都有自己的前端驱动处理，因此如果某选项只针对一种语言，为了识别这些选项，就需要使用相应的驱动作为 gcc 的前端。

D.1 选项前缀

所有选项都以连字符开始。它们中的一些还会用两个连字符。某些以 -f 和 -W 开始的选项还有一些特殊的含义。

前缀 --

在命令行中，标识选项的传统方法是在字母前加一个连字符。新的方式则是以两个连字符开始。本索引中列出的许多选项既有旧的形式，也有新的形式，但意义完全相同。例如，设定在生成的代码中携带调试信息的传统选项是：

-g

同一选项还可以用如下较长的形式，如：

--debug

前缀 -f

字母 f 代表 flag （标记）。大多数选项只有两种状态，开和关。例如，下面的选项设定是否打开窥孔（ peephole ）优化：

-fpeephole

由于这一标记要么是开，要么是关，因而每个标记选项都有一个相反的选项，它使用相同的名字，但以 no 作为前缀。例如：

-fno-peephole

几乎所有的选项都会触发一个真或假的标记设置，因此选项中的一个值会是默认设置。但也有例外。比如，如下两个选项中的任意一个都能指出 for 循环中变量的作用域：

-ffor-scope

-fno-for-scope

两个作用域设置中的任何一个都不是默认的。默认方式会符合标准，但这两个设置都是标准的变体。

任何 -f 选项都可以具有一个双连字符的选项形式。例如，如下两个选项功能相同：

-frtti

--rtti

前缀 -W

前缀 -W 通常用来指定是否需要编译程序产生某些警告消息。与 -f 标记的设定方式相似，警告也可以打开或是在名字前加 no 来关闭。例如，如下的设定可以令编译程序在对函数调用使用了太多的参数时产生警告：

-Wformat-extra-args

如果需要抑制警告消息，可以使用如下选项：

-Wno-format-extra-args

D.1.1 命令行的顺序

选项的顺序很重要。如果命令行中有两个选项，并且它们是冲突的，通常第二个选项会改变第一个选项所产生的设定，从而覆盖了第一个选项。命令行一般由左向右进行分析，每个选项设定一个值或是一个标记（或是一组值或标记），因此在命令行中设置好的内容可能在其后被修改。

对顺序的要求其实很有实用价值。例如， -O3 优化选项将打开 -finline-functions 选项。但如果想使用 -O3 选项，又要关闭函数内嵌功能，可以按下面的顺序输入选项：

-O3 -fno-inline-functions

D.1.2 文件类型

编译程序通过匹配表 D-1 中的后缀检查文件名，从而判断文件的内容。任何具有未知后缀的文件都视作目标机上链接器的输入，并在链接阶段传递给链接器。选项 -x 可以指示编译程序忽略后缀并将文件作为某一类型对待。

表 D-1 GCC 可识别的文件名后缀

后缀	文件内容
.a	包含一个或多个供链接器使用的 .o 文件的静态库（也称为归档文件）
.c	需要进行预处理的 C 源代码
.adb	Ada 体文件，就是包含库单元体的源文件
.ads	Ada 规范文件，就是包含库单元声明或库单元重命名声明的源文件
.C .c++ .cc .cp .cpp .cxx	需要进行预处理的 C++ 源代码
.class	一个包含通过编译 Java 程序产生的字节码的文件
.f .for .FOR	不需要预处理的 Fortran 源代码
.F .fpp .FPP	需要预处理的 Fortran 源代码
.h	C ， C++ 或 Objective-C 的头文件
.i	不需要预处理的 C 源代码
.ii	不需要预处理的 C++ 源代码
.java	Java 源代码
.m	需要预处理的 Objective C 源代码
.mi	不需要预处理的 Objective C 源代码
.mo	包含国际化支持翻译的二进制文件
.o	可供链接器使用的目标文件

.po	包含国际化支持翻译的文本文件
.r	需要由 RATFOR 预处理程序预处理的 Fortran 源代码
.S	需要预处理的汇编语言源代码
.s	不需要预处理的汇编语言源代码
.so	包含一个或多个供链接器使用的 .o 文件的动态库（也称共享库）
<other>	具有不可识别后缀或无后缀的文件，这些文件均视为链接器的输入并且不做任何修改

D.2 选项字母顺序列表

-###

显示编译程序当前版本号，并显示编译和链接中每个阶段可用的所有命令，但并不执行任何命令。当被单独使用时，该选项显示编译程序的当前版本号。当和选项 --help 一起使用时，将会列出完整的命令行选项。

参见 -v 。

-a

在可执行代码每个基本逻辑块的顶端，都会产生额外代码用来进行记录。运行一次基本逻辑块，就会记录一次。所记录的信息包括基本逻辑块的起始地址和逻辑块中的函数名称。如果同时给出选项 -g ，则每个基本逻辑块所记录的信息将包括文件名和每个逻辑块的起始行。这些信息会写到文件 bb.out 中（除非在机器描述中给出了其他的名字）。

参见 -ax 、 -fprofile-arcs 和 -ftest-coverage 。该选项也可写作 --profile-blocks 。

-A question （ answer ）

按如下形式为某个断言指定问题和答案：

#if #question(answer)

该选项也可以写作 --assert 。参见 -A- 。

Pre

-A-

关闭通常用于描述目标机的标准断言。参见 -A 。

--all-warnings

参见 -Wall 。

C C++ ObjC

--ansi

与 -ansi 相同。

C C++ ObjC

-ansi

该选项指示编译程序编译符合标准的程序，但并不限制与标准不冲突的其他方式。也就是说，不冲突的 GNU 扩展仍然有效。对于 C 而言，该选项能够正确地编译符合 ISO C89 的程序。对于 C++ 而言，与 ISO C++ 冲突的 GNU 扩展将被禁止。

该选项同时打开选项 -fno_asm ， -fno_nonansi_builtin ， -trigraphs 和 -fno_dollars_in _identifiers 。对于 C++ ，它还打开了选项 -fno_gnu_keywords 和 -fno-nonasni-builtins 。

该选项定义了宏 __STRICT_ANSI__ ，它可以防止包含某些头文件，从而不会引起被编译程序中的名字和头文件中声明的函数或宏之间产生冲突。

该选项关闭了 GNU C 扩展关键字 asm ， typeof 和 inline ，但它们的替代形式 __asm__ ， __typeof__ ， __inline__ 仍然有效。

如果要限制代码严格符合标准，可以和 -ansi 一起使用 -pedantic 。参见 -std 。

该选项也可以写作 --ansi 。

--assemble

与 -S 相同。

Pre

--assert question （ answer ）

与 -A 相同。

C

-aux-info filename

输出所有已声明和在一个独立编译单元（一个 C 源文件和其包含的所有头文件）中定义的函数声明。

-b machine

该选项指示需要编译程序的目标机。如果没有指定该选项，默认是为编译程序所运行的目标机编译代码。目标机通过指定包含编译程序的目录来确定，通常为 /user/local/lib/gcc-lib/ *machine/version* 。

参见 -B 和 -V 。该选项也可以写作 --target 。

-B prefix

路径 prefix 给出库文件的位置，包括编译程序的文件、执行程序和数据文件。如果需要运行子程序，如 cpp ， as 或 ld ，就会用该前缀来定位这些程序。可以根据自己的意愿在前缀的末尾给出目录分隔符或不给。

所有的过程均使用同样的标准搜索过程。下面的步骤按顺序执行，直至要寻找的目标已被定位：

1. 如果用选项 -B 给出前缀，就用它来构建路径名。

2. 使用前缀 /usr/lib/gcc/ 构建路径。

3. 使用前缀 /usr/local/lib/gcc-lib/ 构建路径。

4. 按照变量中的出现顺序，使用变量 PATH 中定义的每个目录路径。

选项 -B 也可以用来为链接器定位库文件，这是因为编译程序将该选项翻译为选项 -L 并传递给链接器。

选项 -B 还可以用来定位头文件，这是因为所给出的前缀具有追加的目录名，它被翻译为选项 -isystem 并传递给预处理程序。

环境变量 GCC_EXEC_PREFIX 可以被设置为前缀目录的名字，并具有和选项 -B 相同的效果。

引导编译程序的特殊例子：如果前缀具有的形式从 dirpath/stage0 到 dirpath/stage9 ，它将被替换为 dirpath/include 。

该选项也可以写作 --prefix 。

Java

— bootclasspath= pathname

pathname 指定标准 Java 包和类（如 java.lang.String ）的位置。路径名可以是目录、 jar 文件或 zip 文件。

参见 --classpath 和 -I 。

-c

不调用链接器。这一选项会将源文件编译或汇编为目标文件，但不会运行链接器来创建可执行体。产生的目标文件将存放在与源文件同名但以 .o 为后缀的文件中。表 D-1 中所有不可识别的输入文件或是 -x 所指示的类型都将被忽略。

这一选项也可以写作 --compile 。

Java

-C

引起编译程序产生字节码类文件，而不是默认的可执行目标代码。参见 -foutput-class-dir 。

Pre

-C

与选项 -E 一同使用，该选项可以去除所有的注释。

该选项也可以写作 --comments 。

--compile

与 -c 相同。

-d letters

这里可以给出一个或多个字母，它们用来指定什么时候进行调试输出和输出哪些内容。这一选项是用来调试编译程序的，可以在编译的不同阶段检查详细信息。每个输出文件都有一个后缀标识，该后缀标识是阶

段号和某些标识字母。例如，在阶段 21 后输出的文件内容为全局寄存器分配，当编译文件 doline.c 时，该输出文件可能命名为 doline.21.greg 。

参见选项 -dumpbase ， -fdump-unnumbered ， -fdump-translation-unit ， -fdump-class-hierarchy 和 -fdump-tree-switch 。该选项还可以写作 --dump 。

表 D-2 列出了 -d 选项中可以使用的字母，它们可以按照任意组合和顺序插入。这一特征是完全为了调试编译程序所实现的，所以你会发现并不是在每个发布中都有这些字母。注意，当同选项 -E 一同使用时，字母 D ， I ， M 和 N 对预处理程序具有特殊的含义。

表 D-2 与 -d 选项联合使用的字母

字母	作用
A	用各种调试信息注释汇编语言输出
a	设置标记，输出除文件 name.pass.vcg 以外的所有命名文件，文件 name.pass.vcg 只能通过选项字母 v 设定
b	在计算分支概率后输出到文件 name.14.bp 中
B	在进行块重排序后输出到文件 name.29.bbro 中
c	在组合指令后输出到文件 name.16.combine 中
C	在首次 if 变换后输出到文件 name.17.ce 中
d	在延迟分支调度后输出到文件 name.31.dbr 中
D	与选项 -E 一同使用，除了预处理后的正常输出外，还输出所有的宏定义
e	在静态赋值优化后输出到文件 name.04.ssa 和 name.07.ussa 中
E	在第二次 if 变换后输出到文件 name.26.ce2 中
f	在数据流分析后输出到文件 name.13.cfg 中，并在活性分析后输出到文件 name.15.life 中
F	在修剪 ADDRESSOF 代码后输出到文件 name.09.addressof 中
g	在全局寄存器分配后输出到文件 name.21.greg 中
G	在 GCSE 后输出到文件 name.10.gcse 中

h	在异常处理结束后输出到文件 name.02.eh 中
i	在同属调用优化后输出到文件 name.01.sibling 中
I	当同选项 -E 一同使用时，预处理程序输出 #include 指示字，还输出其他预处理程序的输出
j	在首次跳转优化后输出到文件 name.03.jump 中
k	寄存器到栈（ register-to-stack ）的转换后，输出到文件 name.28.stack 中；而在寄存器到栈的变换结束后，输出到文件 name.32.stack 中
l	在局部寄存器分配后输出到文件 name.20.lreg 中
L	在循环优化后输出到文件 name.11.loop 中
M	在机器依赖重构阶段后输出到文件 name.30.mach 中。当同选项 -E 一同使用时，预处理程序将在所有的预处理后输出有效的宏定义列表
m	在编译结束时，向标准错误输出内存使用信息
n	在寄存器重编号后输出到文件 name.25.rnreg 中
N	在寄存器移动阶段后输出到文件 name.18.regmove 中。当同选项 -E 一同使用时，除了在预处理后的正常输出外，还包括以 #define name 简单形式定义的所有宏的列表
o	在后寄存器重载优化之后 ，输出到文件 name.22.postreload 中
p	用每条指令的长度以及标明使用了哪种模式进行优化的注释标注汇编语言

（续表）

字母	作用
P	使用可以产生每条指令的 RTL 代码标注汇编语言。选项 -dp 也会打开该功能，而且会有更多的标注
r	在生成 RTL 后输出到文件 name.00.rtl 中。参见字母 x
R	在第二次调度阶段后输出到文件 name.27.sched2 中
s	在 CSE 以及有时跟在首次 CSE 阶段后的跳转优化之后输出到文件 name.08.cse 中
S	在首次调度阶段后输出到文件 name.19.sched 中
t	在第二次 CSE 阶段以及有时跟在第二次 CSE 阶段后的跳转优化之后输出到文件 name.12.cse2

	中
u	在 SSA 优化后输出到文件 name.06.null 中
v	将每一个其他文件（除文件 name.00.rtl 之外）控制流图表示输出到文件 name.pass.vcg 中。该文件可以由 vcg 读取和显示
w	在第二次流程阶段后输出到文件 name.23.flow2 中
W	在 SSA 条件代码传播后输出到文件 name.05.ssaccp 中
X	在 SSA 不可达代码删除阶段后输出到文件 name.06.ssadce 中
x	为函数生成 RTL 但不编译它。该字母通常与字母 r 一起使用
y	解析器将调试信息输出到标准错误中
z	在窥孔优化阶段后输出到文件 name.24.peephole2 中

Java

-D property[= string]

该选项 可以在使用选项 --main 的命令行中使用 ， 同时它将定义一个可以通过调用 java.lang.System.getProperty() 方法获取的属性。如果 string 没有指定 ， 所定义的字串就是一个空字符串。

该选项也可以写作 --define-macro 。

Pre

-D macro[= string]

如果指定了 string ， 宏名字就被定义了 ， 就好像它就是程序的一部分。例如 ， -Dbrunt=logger 将产生如下的定义 ：

#define brunt=logger

如果没有指定 string ， 宏就定义为字符串 “1” 。例如 ， -Dminke 产生如下的定义 ：

#define minke 1

所有 -D 选项均在 -U 选项前处理，所有 -U 选项均在 -include 或 -imacros 选项前处理。

该选项也可以写作 --define-macro 。

--debug [level]

与选项 -g 相同。

Pre Java

--define-macro macro[=string]

与选项 -D 相同。

Pre

--dependencies

与选项 -M 相同。

--dump letters

与选项 -d 相同。

-dumpbase base

base 是基本文件名，用于命名选项 -d 所产生的输出文件。

该选项也可以写作 --dumpbase 。

-dumpmachine

打印该编译程序的目标机名字，同时不采取任何动作。

-dumpspecs

打印构建编译程序的规范信息，同时不采取任何动作。所打印的信息是一个很长的列表，包括用来编译、汇编和链接 GCC 编译程序自身用到的所有选项（和默认选项）。

-dumpversion

打印编译程序的版本号，同时不采取任何动作。

Pre

-E

当源代码经过预处理后暂停并输出结果。除非利用选项 -o 指定输出文件，否则所产生的输出将写到标准输出中。不需要预处理的输入文件将被忽略，这一过程可以通过表 D-1 中的文件名后缀判断，或用选项 -x 来指示表中不包含的文件类型。

该选项定义了环境变量 __GNUC__ ， __GNUC_MINOR__ 和 __GNUC_PATCHLEVEL__ 。

选项 -dD ， -dI ， -dM 和 -dN 仅当和选项 -E 一同使用时才具有特殊含义。参见选项 -C 和 -P 。这一选项还可以写作 --preprocess 。

Java

--encoding= name

与选项 -fencoding 相同。

--extra-warnings

与选项 -W 相同。

C++

-faccess-control

这是默认选项。如果指定了 -fno-access-control ，编译程序将不会检查访问权限。这一选项的惟一用途是绕过编译程序中的一个访问权限错误。

-falign-functions [=number]

将函数的起始地址在大于或等于 number 的 2 的幂次方的边界上对齐，但仅当必须略过不多于 number 个字节时才使用。例如，如果 number 是 20 ，其最终对齐的边界将是 32 字节边界，同时不会略过多于 20 个字节。

将 number 设置为 2 的幂次方将引起所有的函数均边界对齐。如果没有指定 number ，将使用机器的默认值。对于某些机器而言，这个数值是 2 的幂次方，因此会使所有的函数边界对齐。将 number 指定为 1 与使用选项 -fno-align-functions 等价，不会造成边界对齐。

-falign-jumps [=number]

将分支目标在那些大于或等于 number 的 2 的幂次方的边界上对齐，但仅当无需略过多于 number 个字节时有效。例如，如果 number 是 20 ，则有效的对齐边界是在 32 个字节的位置上，但同时又不能忽略多于 20 个字节。和与之相似的选项 -falign-labels 不同，该选项不需要在分支目标前插入空指令。

如果没有指定 number ，将使用机器的默认值，一般为 1 。指定数值 1 等价于使用选项 -fno-align-jumps 且不会造成对齐。

-falign-labels [=number]

将所有分支目标对齐于那些大于或等于 number 的 2 的幂次方的边界，但仅当无需略过多于 number 个字节时有效。例如，如果 number 是 20 ，则有效的对齐边界是在 32 个字节的位置上，但同时又不能忽略多于 20 个字节。该选项将使代码变慢且变大，其原因是在分支目标前插入的空指令造成的。如果需要功能相仿但性能较好的版本，参见选项 -falign-jumps 。

如果使用了 -falign-loops 或是 -falign-jumps ，但所指定的 number 大于该选项中指定的值，将使用这个更大的数值。如果没有指定 number ，将使用机器的默认值，一般为 1 。指定数值 1 等价于使用选项 -fno-align-labels 且不会造成对齐。

-falign-loops [=number]

将所有循环顶端对齐于那些大于或等于 number 的 2 的幂次方的边界，但仅当无需略过多于 number 个字节时有效。例如，如果 number 是 20 ，则有效的对齐边界是在 32 个字节的位置上，但同时又不能忽略多于 20 个字节。该选项将使代码变大，其原因是在分支目标前插入的空指令造成的，但根据机器的不同，循环可能因为每次迭代末尾的对齐跳转而运行得更快。

如果没有指定 number ，将使用机器的默认值，一般为 1 。指定数值 1 等价于使用选项 -fno-align-loops 且不会造成对齐。

C

-fallow-single-precision

这是默认选项。其含义是不用双精度浮点运算替代单精度浮点进行数学操作。如果使用了 -traditional ，所有浮点操作均将按双精度执行。但如果使用了这一选项，就可以使用单精度运算了。

C++

-falt-external-templates

该选项已经废止了。依赖于原始模板实例化的位置，模板实例可以或无需发生。实例化将符合 #pragma interface 和 #pragma implementation 。参见 -fexternal-templates 。

-fargument-alias

指定传递给函数的参数可以是其他参数的别名。也就是说，两个或多个参数可能代表同一个内存位置。还有一种可能就是参数还可以是全局变量的别名。该选项一般只用于编译程序内部。

对 C ， C++ 和 ObjC 而言，这是默认选项。

参见 -fargument-noalias 和 -fargument-noalias-global 。

-fargument-noalias

指定传递给函数的所有参数均不能是其他参数的别名。也就是说，两个或多个参数将不会对应同一个内存位置。但有可能某个参数是一个全局变量的别名。该选项一般只用于编译程序内部。

参见 -fargument-alias 和 -fargument-noalias-global 。

-fargument-noalias-global

指定传递给函数的所有参数均不能是其他参数的别名。也就是说，两个或多个参数将不会对应同一个内存位置。而且该参数也不可能是全局变量的别名。该选项一般只用于编译程序内部。

对 Fortran 而言，该选项是默认的。

参见 -fargument-alias 和 -fargument-noalias 。

C C++ ObjC

-fasm

这是默认选项，它将打开关键字 asm ， inline 和 typeof 。

对 C 而言，指定 -fno-asm 将关闭关键字 asm ， inline 和 typeof 。该选项对关键字 __asm__ ， __inline__ 和 __typeof__ 没有影响。

对 C++ 而言，指定 -fno-asm 将关闭关键字 typeof ，但由于关键字 asm 和 inline 是语言本身的一部分，因此对它们没有影响。

其他影响这些关键字的标志还有 -ansi ， -gnu-keywords 和 -std 。

Java

-fassume-compiled=classname

根据某些类是否已经编译为本地代码，编译程序可以产生不同的代码。可以重复使用选项 -fassume-compiled 和 -fno-assume-compiled 来构造一组类的列表，假定它们是编译过的或是未编译的。

-fasynchronous-unwind-tables

如果目标机支持，产生 DWARF2 格式的回退表。所生成的表格格式可由异步事件使用，如调试器和垃圾收集器。

参见选项 -fexceptions ， -fnon-call-exceptions 和 -funwind-tables 。

Fortran

-fautomatic

这是默认选项。指定选项 -fno-automatic 与为每个局部变量和数组指定一条 SAVE 指令的效果相同。该选项对公用块没有影响。可以通过指定选项 -fno-automatic 关闭此功能。

参见选项 -finit-local-zero 。

Fortran

-fbackslash

这是默认选项。指定 -fno-backslash 后将阻止反斜线字符在 Hollerith 串中用做转义字符（像 C 字串中的用法）来定义特殊字符。

Fortran

-fbadu77-intrinsics- spec

spec 指定了具有不恰当形式的 UNIX 特质的状态。 spec 可以具有如下几种可能：

· enable 特质是可识别的且是打开的。这是默认状态 。

· hide 特质是可识别的且当在一条 INTRINSIC 语句中首次出现时打开 。

· disable 特质是可识别的，但对它们的引用必须通过 INTRINSIC 语句完成 。

· delete 特质不可识别 。

Java Fortran

-fbounds-check

对 Java 而言，这是默认选项。指定选项 -fno-bounds-check 将关闭所有对数组访问的边界检查。该选项将提高数组索引的性能，但当超出数组边界时，可能会造成不可接受的行为。

对 Fortran 而言，该选项 产生的代码在运行时检查数组下标，并检查对 CHARACTER 子串的访问是否在所声明的最小和最大值内。

参见选项 -ffortran-bounds-check 。

-fbranch-probabilities

在使用选项 -fprofile-arcs 编译程序，并执行它来创建包含每个代码块执行次数的文件之后，程序可以利用这一选项再次编译，文件中所产生的信息将被用来优化那些经常发生的分支代码。如果没有这些信息，GCC 将猜测哪一分支经常发生并进行优化。这类优化信息存放在一个以源文件为名字以 .da 为后缀的文件中。

参见选项 -fguess-branch-probability 。

C ObjC

-fbuiltin

这是默认选项，用于通过名字来识别内建函数。选项 -fno-builtin 指出，除非利用前缀 __builtin_ 进行引用，否则不识别所有内建函数。例如，为了获得内建版本，应该调用 __builtin_strcpy() 而不是名为 strcpy() 的函数。

为了避免使用选项 -fno-builtin 而抑制所有内建函数，可将函数的名称追加到选项名之后，有选择地指定不使用哪些内建函数。例如，想使用除 bzero() 和 sqrt() 之外的所有内建函数，可以使用如下形式：

-fno-builtin-bzero -fno-builtin-sqrt

对 C++ 而言， -fno-builtin 始终有效，因此惟一可以直接调用 C 内建函数的方法就是使用 __builtin_ 前缀， GNU C++ 标准库就使用了很多内建函数。

参见选项 -fbuiltin- function ， -ffreestanding 和 -fnonansi-builtins 。

-fcall-saved- register

将指定的 register 作为一个可分配来包含数据的寄存器，而且即使经过函数调用仍然可以获取数据。利用这一选项编译的函数必须完成对该寄存器内容的保存和恢复。

对于具有固定作用的寄存器（如栈指针和堆指针），绝对不能使用该选项。

寄存器的名字和机器平台相关，均在机器描述的宏 REGISTER_NAMES 中给出。

参见选项 -fcall-used-register 和 -ffixed-register 。

-fcall-used- register

将指定的 register 看作可用于分配的寄存器，但其值将在函数调用中被破坏。该寄存器可用做临时存储，但经过函数调用后必须重载。

对于具有固定作用的寄存器（如栈指针和堆指针），绝对不能使用该选项。

寄存器的名字和机器平台相关，均在机器描述的宏 REGISTER_NAMES 中给出。

参见选项 -ffixed-register 和 -fcall-saved-register 。

-fcaller-saves

在调用函数之前需要加入额外的指令来保存寄存器，并且在返回函数时需要恢复它们。之后，这些寄存器就可以在函数调用和函数体内使用了。只有包含有用数据的寄存器需要保存，而且只有保存和恢复这些寄存器相比在需要时重新加载它们更好时才会这么做。默认情况下，该选项在某些机器上是打开的，而且在指定了选项 -O2 ， -O3 和 -Os 时均是打开的，但可以通过选项 -fno-caller-saves 将其关闭。

Fortran

-fcase-initcap

要求大部分源代码用起始大写字母书写。设置选项 -fintrin-case-initcap ， -fmatch-case-initcap ， -fsource-case-preserve 和 -fsymbol-case-initcap 。

Fortran

-fcase-lower

将源代码映射为小写的。设置选项 -fintrin-case-any ， -fmatch-case-any ， -fsource-case-lower 和 -fsymbol-case-any 。

Fortran

-fcase-preserve

保留用户定义符号的大小写，允许任意大小写的关键字和特质匹配。设置选项 -fintrin-case-any ，-fmatch-case-any ， -fsource-case-preserve 和 -fsymbol-case-any 。

Fortran

-fcase-strict-lower

要求大部分源代码用小写字母书写。设置选项 -fintrin-case-lower ， -fmatch-case-lower ，-fsource-case-preserve 和 -fsymbol-case-lower 。

Fortran

-fcase-strict-upper

要求大部分源代码用大写字母书写。设置选项 -fintrin-case-upper ， -fmatch-case-upper ，-fsource-case-preserve 和 -fsymbol-case-upper 。

Fortran

-fcase-upper

将源代码映射为大写的。设置选项 -fintrin-case-any ， -fmatch-case-any ， -fsource-case-upper 和 -fsymbol-case-any 。

C++

-fcheck-new

在 C++ 的分配内存操作中，插入代码检查 new 运算符返回的指针不为 NULL 指针。这一操作通常是不必要的，因为 C++ 库中的 new 在内存不足时会产生一个异常。如果重载后的 new 可能返回而不产生异常，该选项就可以用来检查返回指针了。

Java

-fcheck-references

当通过引用访问对象时，插入内嵌代码检查空指针引用。因为大多数处理机都会检测到这种空指针引用，这一操作通常是不必要的。

C

-fcommon

这是默认选项。指定 -fno-common 将使编译程序在数据区为每个全局变量显式地分配空间。默认情况是在一个公用块中分配空间，并由链接器解析，因此即使将同一全局变量声明多次也会使链接器将其解析到同一地址。

可以使用选项 -fno-common 来检验需要编译和链接到其他不使用 GCC 程序的系统。

对 Fortran 而言，决不可以使用 -fno-common 选项。

参见第 4 章的 4.3.5 节“属性”。

Java

-fcompile-resource= resourcename

resourcename 是包含属性定义及其他资源的文件名，这些资源将编译到目标代码中，并在运行时由核心协议处理函数作为 core:/resourcename 访问。

C

-fcond-mismatch

在条件表达式中允许类型不匹配。

C++

-fconserve-space

将在编译时不初始化的全局变量放在公用段中（如 C 中的做法）。由于直到程序加载时才分配空间，这将降低可执行文件的尺寸。对大部分平台来说，由于已有这样的支持，能够将变量保存在 BSS 段中而无需将变量作为公用的，因此这一选项已不再有价值了。

警告：如果使用这一选项造成程序在终止时崩溃，可能是因为某个对象的多次定义被合并且分配到同一地址上，从而进行两次释放造成的。

C++

-fconst-strings

这是默认选项。如果指定了选项 -fno-const-strings ，字串声明将被定义为 char * 而不是 const char * 。为了能够对字串进行写操作，同时还需要使用选项 -fwriteable-strings 。

ObjC

-fconstant-string-class= classname

对于每个形式为 @"..." 的字串，使用指定的 classname 作为实例化的类名。默认的 classname 是 NXConstantString 。

-fcprop-registers

在所有寄存器分配完成后，该选项分析复制到寄存器中的数据模式，试图发现那些不需要在寄存器中复制数值的地方，但为了能够再次使用，会向前传播前一次复制。该选项由选项 -O 设置，但可以使用选项 -fno-cprop-registers 关闭。

-fcse-follow-jumps

当跳转目标除了所采用的跳转以外均不可达时，在跳转路径后将产生公用子表达式消除扫描。也就是说，所有在跳转前存在的数值将在跳转目的那一点仍然存在并可以使用。选项 -O2 ， -O3 和 -Os 将设置这一选项，但可以利用选项 -fno-cse-follow-jumps 关闭。参见选项 -fcse-skip-blocks 和 --param 。

-fcse-skip-blocks

如果一条 if 语句体非常简单，以至于不包含任何会影响先前计算的数值，公用子表达式分析流将会跳过这一 if 语句并作用于随后的语句。选项 -O2 ， -O3 和 -Os 将会打开这一选项，但可以通过选项 -fno-cse-skip-blocks 关闭这一选项。参见 -fcse-skip-blocks 和 --param 。

-fdata-sections

在汇编语言输出中，每个数据项均被放置在其自己的小节中。小节名是通过数据项名派生的。仅在具有可以使用小节方式来优化空间分配的链接器的机器上，这一选项才有优势。如果对可执行代码进行同样的优化，参见 -ffunction-sections 。

在汇编器代码不支持分节的机器上，设置该选项将会产生一条警告消息，并且被忽略。即使在支持分节的机器上，这一选项也没有优势，除非链接器利用这一组织进行优化。事实上，由于会造成目标代码更大和加载更慢，该选项还会具有副作用。

如果设置选项 -p 进行性能分析，该选项将失效。而且，由于代码被重新组织，可能会在使用选项 -g 和进行调试时出现问题。

-fdefault-inline

这是默认选项。在类声明时，如果成员函数具有已定义的函数体，不管是否使用了关键字 inline 进行声明，它都将被定义为是内嵌的。为了防止自动内嵌，可以使用选项 -fno-default-inline 。参见 --param 。

-fdefer-pop

调用函数时压入栈的参数将不会在函数返回时立即弹出，而且允许在多次函数调用中累积，并在最后一次清除。默认情况下该选项被打开，因此要强制在每次函数调用后清除栈，需要使用选项 -fno-defer-pop 。

-fdelete-null-pointer-checks

如果数据流分析表明某指针不会为空，检查指针是否为空的代码将被删除。在某些环境下，可能要处理检查空指针的结果，因此该选项不能在依赖这一结果的程序中使用。选项 -O2 ， -O3 和 -Os 将打开这一选项，但可以通过选项 -fno-delete-null-pointer-checks 关闭这一选项。

-fdelayed-branch

该选项仅在具有延迟分支通道的机器上有效。这一选项不得不在处理加载和执行指令时，同时决定是否采用某一分支。在作出了决策后，根据指令的位置和决策的结果，指令的结果可能会被丢弃。如果目标机支持，每一级优化均会打开该选项，但可以通过选项 -fno-delayed-branch 关闭该选项。参见 --param 。

-fdiagnostics-show-location= where

诊断消息（错误和警告）有可能很长且是分行的，因此显示时会有很多行。 where 的默认设置是 once ，它会使引发诊断消息的源文件位置仅被包含一次。将 where 设置为 every-line 时将使诊断消息的每一行均包含源代码位置。

该选项可能在某些情况下没有实现，而且只有将选项 -fmessage-length 设置为大于 0 时才会有效。

Fortran

-fdollar-ok

允许在符号名中使用美元符号 $ 。

C C++

-fdollars-in-identifiers

在标识符中接受字符 $ 作为合法字符。使用选项 -fno-dollars-in-identifiers 可以显式地禁止这一功能。默认设置取决于平台和语言。传统 C 允许使用，但现在的标准则禁止使用，因此如果你需要使用的话，最好显式指定该选项。

C++

-fdump-class-hierarchy [-format]

每个类都会将层次信息和虚函数表输出到一个同名但以 .class 为后缀的文件中。可选的 format 可为如下形式之一：

· address 打印每个节点的地址，并利用这些地址交叉引用其他输出中的树节点，如利用选项 -d 产生的内容。

· slim 通过禁止输出函数体或是域成员减少输出的尺寸。

· all 打开所有选项增加输出的尺寸。

C C++

-fdump-translation-unit [-format]

将编译程序的内部树结构输出到与源代码同名但以 .tu 后缀结尾的文件中。可选的 format 可为如下形式之一：

· address 打印每个节点的地址，并且利用这些地址交叉索引其他输出的树节点，如选项 -d 产生的内容。

· slim 通过禁止输出函数体或是域成员减少输出的尺寸。

· all 打开所有选项增加输出的尺寸。

-fdump-tree - switch[-format]

将中间语言树的不同阶段输出到文件中。文件名由源文件名加上 switch 部分中指定的部分作为后缀组成。switch 必须是如下形式之一：

· original 在对树进行任何优化之前，将树结构输出到文件 name.original 中。

· optimized 在所有对树的优化后，将树结构输出到文件 name.optimized 中。

· inlined 在完成函数内嵌后，将树结构输出到文件 name.inlined 中。

可选的 format 可为如下形式之一：

· address 打印每个节点的地址并且利用这些地址交叉索引其他输出的树节点，如选项 -d 产生的内容。

· slim 通过禁止输出函数体或是域成员减少输出的尺寸。

· all 打开所有选项增加输出的尺寸。

-fdump-unnumbered

当使用选项 -d 调试编译程序时，该选项将抑制输出文件中的指令号和行号，这样可以方便使用 diff 比较输出。

-felide-constructors

这是默认选项。当产生的代码所调用的函数按值返回一个对象时，如果能直接在返回的位置上产生对象，而不是复制构造函数中已构建的对象，这将使代码大大简化。但如果构造函数具有副作用，这也会引发问题，可以利用 -fno-elide-constructors 关闭这一选项。

Fortran

-femulate-complex

通过仿真而不是可以提供直接支持的 gcc 后端实现复数算术。

该选项是为了绕过 gcc 复数算术中的错误而实现的，但这个错误应该已经修正了。

Java

-fencoding= name

指定用于读取源文件的某一特定字符集的编码名。默认情况下是计算机当前的本地设置，或者如果没有指定本地设置的话，默认会使用 UTF-8 。

C++

-fenforce-eh-specs

这是默认选项。根据 C++ 标准， GCC 所产生的代码将强制检查异常违例，但可以利用选项 -fno-enforce-eh-specs 抑制这些代码的产生。不检查违例的代码大小会有所减少。

-fexceptions

打开异常处理。这一选项会生成必要的代码来处理异常的抛出和捕获。如果没有指定这一选项，对于 Ada ，Java 和 C++ 等会触发异常的语言来说，默认都会指定该选项。

所生成的代码不会造成性能损失，但会造成尺寸上的损失。因此，如果想要编译不使用异常的 C++ 代码，可能需要指定选项 -fno-exceptions 。

参见选项 -fnon-call-exceptions ， -funwind-tables 和 -fasynchornous-unwind-tables 。

-fexpensive-optimizations

该选项打开一些耗费编译时间但却非常有效的优化操作。例如，在全局公用子表达式删除后再次执行公用子表达式删除。当该选项被打开时，其他一些优化操作会执行得更彻底。选项 -O2 ， -O3 和 -Os 自动打开这一选项，但可以通过 -fno-expensive-optimizations 强行关闭这一选项。

C++

-fexternal-templates

该选项已废止。根据模板定义的位置，模板实例可能会产生或不产生。实例化过程将遵从 #pragma interface 和 #pragma implementaion 。参见选项 -falt-external-templates 。

Fortran

-ff2c

这是默认选项，它指定生成与 f2c 兼容的代码。

指定选项 -fno-f2c 将抑制产生与 f2c 兼容的代码并使用 GNU 调用规范。但该选项对与库 libf2c 接口的代码无效，只是不允许来自 libf2c 的特质作为参数传递。

如果要链接为一个程序，所有代码都必须一致地使用选项 -fno-f2c 。

Fortran

-ff2c-intrinsics- spec

spec 指定具有不恰当形式的 f2c 相关特质的状态。 spec 可以是如下形式之一：

· enable f2c 相关特质可以识别并打开。这是默认状态。

· hide 只有当首次在 INTRINSIC 语句中涉及名字时， f2c 相关特质才可被识别并打开。

· disable f2c 相关特质可以识别，但对它们的引用必须通过 INTRINSIC 语句完成。

· delete f2c 相关特质不被识别。

Fortran

-ff66

被编译的源码具有 Fortran 66 方言。

参见选项 -ff77 和 -ff90 。

Fortran

-ff77

被编译的源码具有 Fortran 77 方言。

该选项同时也指定这种方言是 f2c 所需要的， f2c 是种可以将 Fortran 代码转换为 C 代码的工具。

参见选项 -ff66 和 -ff90 。

Fortran

-ff90

编译程序将识别某些 Fortran 90 方言中的内容。

很多 Fortran 90 中的内容可以通过使用选项 -fvxt 和 -ff90-instrinsics-enable 打开。

参见选项 -ff66 和 -ff77 。

-ff90-intrinsics- spec

spec 指定具有不恰当形式的 Fortran 90 特质的状态。 spec 可为如下形式之一：

· enable Fortran 90 特质可以识别并打开。这是默认情况。

· hide 仅在首次涉及 Fortran 90 特质名字是在 INTRINSIC 语句中时，才可以识别并打开。

· disable Fortran 90 特质可以识别，但对它们的引用必须通过 INTRINSIC 语句完成。

· delete Fortran 90 特质不可以识别。

参见选项 -ff90 和 -fvxt 。

-ffast-math

在违反某些 ISO 和 IEEE 规则时，一些数学运算可以加速完成。例如，如果设置了这一选项，就可以认为所有给 sqrt() 的参数均为非负，因此所有的浮点值就都合法了。

设置这一选项将使预处理程序定义宏 __FAST_MATH__ 并同时设置选项 -fno-math-errno ，-funsafe-math-optimizations 和 -fno-trapping-math 。设置选项 -fno-fast-math 的同时也将设置选项 -fmath-errno 。

-ffixed- register

视指定的 register 为一个不能由编译程序分配的固定寄存器。但它仍可能用作堆指针、栈指针或其他某些固定用途。

寄存器的名称与平台相关，并定义在机器描述的宏 REGISTER_NAMES 中。

参见选项 -fcall-used-register 和 -fcall-saved-register 。

-ffixed-form

这是默认选项。它指定 Fortran 源代码必须是传统的固定形式，而不能使用 Fortran 90 中的自由格式。

该选项与选项 -fno-free-form 相同。

Fortran

-ffixed-line-length- len

在固定形式的源代码输入中，数字 len 指定了其后字符全部忽略的列号。

设置选项 -ffixed-line-length-0 和 -ffixed-line-length-none 将可以允许任意长度的源代码输入。

len 的数值一般为 72 （在串接的 80 列卡片中留 8 个空白）。

-ffloat-store

不分配寄存器保存浮点值。在某些机器上，该选项可能引起寄存器将浮点值扩展到超出语言所定义的精度范围，因此将精度更高的浮点值存储到内存中。

默认设置是 -fno-float-store ，即可以使用寄存器。

当程序必须严格地将精度限制为 IEEE 标准定义的精度时，该选项非常有用。

C++

-ffor-scope

默认情况是遵守标准。这项设置确定了在 for 语句初始化部分声明的变量的作用域。

指定选项 -ffor-scope 将变量的作用域限制为循环体。

指定选项 -fno-for-scope 可将变量的作用域进行限制，由声明该变量的位置开始，到包含 for 语句的循环体的结束位置。下面是使用该选项的合法例子：

```
#include stdio.h

int main(int argc,char *argv[])

{

for(int i=0; i10; i++) {

printf("Loop one %d\n",i);

}

printf("Out of loop %d\n",i);

return(0);

}
```

-fforce-addr

必须将地址复制到寄存器中才能对它们进行算术运算。由于所需要的地址经常在前面就已经加载到寄存器中，之后就不需要加载了，因此这将改进所生成的代码。默认选项是 -fno-force-addr 。参见选项 -fforce-mem 。

Java

-fforce-classes-archive-check

该选项强制检查类文件 java.lang.Object 中的一个属性，保证它是由 GNU 编译程序编译的。该属性具有零尺寸，名称为 gnu.gcj.gcj-compiled 。除了编译程序的输出是字节码外，该属性均被检查。

-fforce-mem

必须首先将数值复制到寄存器中才能对它们进行算术运算。所需的数值经常在前面已经被加载到寄存器中（而无需再次加载），因此这将大大改进代码。选项 -O2 ， -O3 和 -Os 将打开该选项；默认情况是选项 -fno-force-mem 。参见选项 -fforce-addr 。

-ffortran-bounds-check

该选项会进行运行时的检查，验证数组子脚本和 CHARACTER 子串访问在所声明的最小和最大范围内。

与选项 -fbounds-check 相同。

Fortran

-ffree-form

指定 Fortran 源代码采用与 Fortran 90 相似的自由格式。

与选项 -fno-fixed-form 相同。

C

-ffreestanding

所编译的程序将在独立环境中运行，该环境可能没有标准库，而且可能也不从 main() 开始运行。该选项设置 -fno-builtin 并与选项 -fno-hosted 相同。

-ffunction-cse

这是默认选项。函数调用通过将函数地址保留在寄存器中实现。使用选项 -fno-function-cse 将引起每条调用指令隐式地包含函数地址。默认情况下会生成非常有效的代码。参见选项 --param 。

-ffunction-sections

在汇编语言输出中，每个函数均被放置在其自己的小节中。小节名是由函数名派生的。只有在某些链接器根据分节方式优化空间分配的机器上，这种方法才有优势。要对数据进行同样的优化，参见选项 -fdata-sections 。

在不支持分节的机器上使用该选项将引起警告消息，而且该选项将被忽略。甚至在支持分节的机器上，如果没有链接器进行优化组织，这也是没有效果的。事实上，它也会使目标代码变大而加载速度变慢，带来不利影响。

如果设置了选项 -p 进行性能分析，该选项将失效。而且，由于代码会重新组织，在使用 -g 选项进行调试时，可能会发生问题。

参见第 4 章 4.3.5 节“属性”。

-fgcse

执行全局公用子表达式消除优化操作。如果程序中具有参与计算的 goto 语句，该选项可能具有不利影响。选项 -O2 ， -O3 和 -Os 将打开该选项；可以使用 -fno-gcse 关闭该选项。参见选项 --param 。

-fgcse-lm

通过探测循环中的加载和存储操作，执行全局公用子表达式消除优化操作，其中加载操作可以放置在循环外，可以只执行一次。这是默认选项，但在没有指定选项 -Os 时没有作用。可以利用 -no-fgcse-lm 关闭该选项。

-fgcse-sm

通过探测循环中的加载和存储操作，执行全局公用子表达式消除优化操作，其中存储操作可以放置在循环外，可以只执行一次。这是默认选项，但在没有指定选项 -Os 时没有作用。可以利用 -no-fgcse-sm 关闭该选项。

Fortran

-fglobals

这是默认选项。指定选项 -fno-gloabls 将关闭这种具有相同过程名，但具有不同参数类型的全局冲突诊断。该选项也会关闭函数内嵌以防止生成不正确代码而引发的崩溃。

Fortran

-fgnu-intrinsics- spec

spec 指定了具有不恰当形式的 GNU 特质的状态。 spec 可以是如下形式之一：

· enable GNU 特质可以被识别并设置为有效。这是默认选项。

· hide 仅在首次涉及 GNU 特质名字是在 INTRINSIC 语句中时， GNU 特质才能被识别并有效。

· disable GNU 特质可以被识别，但只有通过 INTRINSIC 语句才能引用它们。

· delete GNU 特质不可识别。

C++

-fgnu-keywords

这是默认选项。选项 -fno-gnu-keywords 禁止使用关键字 typeof 。参见选项 --ansi 。

-fgnu-linker

这是默认选项。如果指定了选项 -fno-gnu-linker ，则说明将不使用 GNU 链接器，某些执行全局初始化的代码（如构造函数和析构函数）就无法保证正确执行了。

在需要该选项的系统中， gcc 被配置为自动调用 collect2 。

ObjC

-fgnu-runtime

对大多数系统而言，这是默认选项。该选项指导编译程序产生使用标准 GNU Object C 运行时的代码。

-fguess-branch-probability

这是默认选项。 GCC 将猜测哪个分支更可能被选取，并以此优化代码。

某些情况下使用随机模型进行猜测，因此编译同样的源代码有可能产生不同的代码。如果需要关闭该选项并强制生成相同的代码，可以指定选项 -fno-guess-branch-probability 。对于其他防止随机化的选项，每次均生成相同的代码，参见选项 -fbranch-probabilities 和 -fprofile_arcs 。

Java

-fhash-synchronization

该选项将引起 synchronize ， wait 和 notify 的位置被存放于一个哈希表中（而非每个对象中）。

C

-fhosted

所编译的程序需要运行在宿主环境中，其中需要有完整的标准库而且 main() 具有 int 返回类型。该选项设置 -fbuiltin 。该选项与选项 -fno-freestanding 相同。

Pre

-fident

这是默认选项。如果指定了选项 -fno-indent ，预处理程序将忽略 #ident 指示字。

C++

-fimplement-inlines

这是默认选项。以嵌入方式生成的函数也具有定义它们时的函数体。选项 -fno-implement-inlines 将抑制生成 #pragma implementation 控制的嵌入函数体的代码。如果没有生成函数体，每一次调用都需要生成内嵌代码。

C++

-fimplicit-templates

这是默认选项。对尚不存在的模板或模板函数的引用将引起模板的实例化。除了那些编译为内嵌函数的模板以外，选项 -fno-implicit-templates 将抑制隐式的模板实例化。参见选项 -fimplicit-inline-templates 。

C++

-fimplicit-inline-templates

这是默认选项。对尚不存在的模板或模板函数的引用将引起模板的实例化。选项 -fno-implicit-templates 将抑制隐式的模板实例化，包括那些编译为内嵌函数的模板。参见选项 -fimplicit-templates 。

-finhibit-size-directive

不输出 .size 汇编指令，也不输出其他可能在分割函数为两部分时引起问题的指令。

该选项是一种用来编译 crtstuff.c （ GCC 的一部分）的特殊情况，预计不会有其他目的。

Fortran

-finit-local-zero

对编译单元声明的所有局部变量都初始化为 0 。对于公用块和作为参数的变量无效。

建议与选项 -fno-automatic 一起使用，以防止初始化自动变量而引起的运行时代价。

-finit-priority

该选项只供编译程序内部使用，它用来指定运行时的初始化次序。

C C++ ObjC Fortran

-finline

这是默认选项，它允许用关键字 inline 指定函数在其被调用的地方展开。使用选项 -fno-inline 将使编译程序忽略关键字 inline 。注意，除非利用选项 -O 设置了某一级别的优化操作，否则是不会展开内嵌函数的。参见选项 --param 和第 4 章的 4.3.5 节“属性”。

-finline-functions

允许编译程序选择某些简单的函数在其被调用处展开。如果函数的声明方式使其所用的调用均是已知的（例如， C 源文件中的静态函数不能从文件外部调用），由于它从不能被调用，其函数体将被忽略。除非使用选项 -fno-inline-functions ，否则当设置选项 -O3 时该选项会自动打开。参见选项 -fkeep-inline-functions ， --param 和第 4 章的 4.3.5 节“属性”。

-finline-limit= size

对伪指令超过一定数目的函数，编译程序将不进行展开。 size 的默认值是 600 。参见选项 --param 。

-finstrument-functions

在每个函数的入口和出口处插入代码调用某个函数。函数调用的原型如下：

```
void __cyg_profile_func_enter(void *this_fn,void *call_site);
void __cyg_profile_func_exit(void *this_fn,void *call_site);
```

参数 this_fn 是被调函数的地址，可以通过符号表信息进行标识。参数 call_site 标识调用者。（某些平台尚没有 call_site 信息。）

如果函数是展开的内嵌函数，函数调用就插在内嵌代码前后。为了达到标识的目的，即便所有调用均使用内嵌代码，也必须存在一个函数的非内嵌版本。

为防止某个函数被插入代码，它可被声明为具有属性 no_instrument_function 。这对于中断处理函数以及性能分析例程不能调用的函数是非常有用的。

参见第 4 章 4.3.5 节“属性”。

-fintrin-case- spec

spec 确定了特质名的大小写。可以为如下形式之一：

· initcap 名称的首字大写，其他小写。

· upper 名称全部大写。

· lower 名称全部小写。这是默认选项。

· any 名称可以是大小写的任意组合。

参见选项 -fmatch-case- ， -fsource-case- ， -fsymbol-case- 和 -fcase- 。

Java

-fjni

该选项将本地方法编译为 JNI ，而非默认的 CNI 。同时生成调用底层 JNI 方法的存根（ stub ）。

-fkeep-inline-functions

即使对函数的所有引用均被扩展为内嵌函数，也就是不存在对函数的调用，编译程序也将生成函数体。默认选项是 -fno-keep-inline-functions ，它指定编译程序不生成没有被调用的函数体。参见选项 -finline-functions 和 --param 。

-fkeep-static-consts

除非设置了某些级别的优化选项，否则这是默认选项。对编译体而言，常量的数值即使没被引用，也将分配存储空间。为了防止为无用常量分配空间，可以使用选项 -fno-keep-static-consts 。

-fleading-underscore

该选项强制写入目标文件的每个符号均以下划线开始。选项 -fno-leading-underscore 将抑制增加下划线字符。

该选项用于链接已有汇编代码。

Fortran

-fmatch-case- spec

spec 指定 Fortran 关键字的大小写。它可为如下形式之一：

· initcap 名称的首字母大写，其余小写。

· upper 名称全部大写。

· lower 名称全部小写。这是默认选项。

· any 名称可以是大小写的任意组合。

参见选项 -fintrin-case- ， -fsource-case- ， -fsymbol-case- 和 -fcase- 。

-fmath-errno

这是默认选项。诸如 sqrt() 之类的数学函数错误代码将被存放在名为 errno 的全局变量

中。设置选项 -fno-math-errno 将使错误代码不被存放在 errno 中，从而干扰标准的 IEEE 异常处理。参见选项 -ffast-math 。

-fmem-report

编译程序结束时，它会打印一张详细的列表，其中列出为每个数据类型所分配的空间和其他永久性存储分配信息。

C++

-fmemoize-lookups

最近的内部符号表查找结果将被缓存，以便加速后续查找。

-fmerge-all-constants

该选项设置 -fmerge-constants ，而且能够合并重复的字符串和数组。标准 C 和 C++ 要求每个变量都有不同的存储位置，因此该选项可能会生成不符合标准的目标代码。

-fmerge-constants

该选项试图将跨编译单元的所有常量值（除了字串）合并在一个副本中。当打开某级优化选项时，这是默认选项。默认选项是 -fno-merge-constants ，它只允许合并同一编译单元中的常量。

-fmessage-length= size

错误消息将被格式化，从而不会超过 size 个字符。如果 size 为 0 ，就不进行格式化，并且每个错误消息仅出现在同一行中。对 C++ 而言，默认尺寸是 72 ，而其他语言均为 0 。该选项在某些情况下可能没有实现。

Fortran

-fmil-intrinsics- spec

spec 指定了具有不恰当形式的 MIL-STD-1753 特质的状态。 spec 可以是如下形式之一：

· enable MIL-STD-1753 特质可以识别并打开。这是默认选项。

· hide MIL-STD-1753 特质可以识别，但仅在首次涉及特质名字是在 INTRINSIC 语句中时才被打开。

· disable MIL-STD-1753 特质可以识别，但只有通过 INTRINSIC 语句才能引用它们。

· delete MIL-STD-1753 特质不可识别。

-fmove-all-movables

所有的不变表达式将被移到循环体外。这一过程的好与坏取决于源码中循环的结构。除了 Fortran 外，默认选项是 -fno-move-all-movables 。参见选项 -freduce-all-givs 。

C++

-fms-extensions

当使用 MFC 定义的构造函数时，关闭警告消息，如数据声明中的隐式 int 定义，使用非标准语法获取成员函数的指针等等。默认选项是 -fno-ms-extensions ，它将产生警告消息。

-fnext-runtime

产生与 NeXT 运行时兼容的输出。对于基于 NeXT 的系统，如 Darwin 和 Mac OS X ，该选项是默认的。

-fno-*

任何以 -fno- 开头的选项都有两种形式，并按照字母顺序排在名字后面——这里没有加入 no- 。例如，可以在选项 -ffor-scope 下找到选项 -fno-for-scope 的描述。大多数以 -f 开头的选项均有一个相伴的 -fno 形式，但也有例外。

-fnon-call-exceptions

产生的代码可供陷阱指令（如非法浮点运算或非法内存寻址）抛出异常。由于需要平台相关的运行时支持，该选项并不普遍有效。

该选项只受限于硬件陷阱信号，但不包括通用信号，如 SIGALRM 或 SIGTERM 。

参见选项 -fexceptions ， -funwind-tables 和 -fasynchronous-unwind-tables 。

C++

-fnonansi-builtins

这是默认选项。指定选项 -fno-nonansi-builtins 将禁止内建函数的自动生成，这里的内建函数并非是 ASNI/ISO C 所需要的。参见选项 -ansi 和 -fbuiltin 。

-fomit-frame-pointer

对于不需要栈指针的函数，就不在寄存器中保存栈指针，因此忽略存储和检索地址的代码，并将其他寄存器用于普通用途。所有 -O 优化选项的级别都自动设置这一选项，但这只当调试器可以不依靠栈指针运行时才可行。如果调试器的运行不能没有栈指针，就必须显式地设置它。某些平台没有栈指针，因此该选项也没有效果。默认选项是 -fno-omit-frame -pointer 。

Fortran

-fonetrip

DO 循环至少被执行一次（循环测试是在循环底而非循环顶处执行的）。

在 Fortran 77 之前，某些编译程序在循环体的底部进行循环检测，而另外一些则在循环体的顶部进行检测。从 Fortran 77 开始，所有测试均在循环体顶部进行，也就是说，如果一开始的检测为假，那么循环体将不被执行。

C++

-foperator-names

这是默认选项。指定选项 -fno-operator-names 将阻止编译程序将 and ， bitand ， bitor ， compl ， not ， or 和 xor 识别为 && ， & ， | ， ~ ， ! ， || 和 ^ 等运算符的替代关键字。

-foptimize-register-move

通过改变移动内存中数据位置的操作中的寄存器赋值，可以优化寄存器的分配。这对某些可直接在内存间搬运数据的机器来说是非常有效的。选项 -O2 ， -O3 和 -Os 可以打开该选项，但可用 -fno-optimize-register-move 关闭这一选项。

-foptimize-sibling-calls

优化尾递归调用和同属调用。选项 -O2 ， -O3 和 -Os 将自动打开该选项。默认选项是 -fno-optimize-sibling-calls 。

下面是一个尾递归调用的例子：

```
int rewhim(int x,int y) {
. . .
return(rewhim(x+1,y));
}
```

优化操作后，程序不会产生一个新的调用，仅插入一条命令直接跳转到函数的顶部。类似地，下面的同属调用也可以进行优化：

```
int whim(int x,int y) {
. . .
return(wham(x+1,y));
}
```

在同属调用中，必须调用函数 wham() ，但函数 whim() 的堆栈框架可以直接删除，这样函数 wham() 就直接将返回值返回给函数 whim() 的调用者了。

Java

-foptimize-static-class-initialization

这是默认选项。当优化选项被设置为 -O2 ， -O3 或 -Os 并且输出的是目标代码而非字节码时，静态类将在首次使用时进行初始化。可以使用选项 -fno-optimize-static-class -initialization 关闭该优化。

C++

-foptional-diags

这是默认选项。指定选项 -fno-optional-diags 将抑制诊断消息，根据 C++ 标准，编译程序并不需要输出这些消息。

Asm

--for-assembler optionlist

与选项 -Wa 相同。

Linker

--for-linker option

与选项 -Xlinker 相同。

Linker

--force-link name

与选项 -u 相同。

Java

-foutput-class-dir= directory

同选项 -C 一起使用时，由编译程序输出的类文件将存放在 directory 中（或相应的 directory 子目录中），而非当前目录。

该选项也可以写作 --output-class-directory 。

-fpack-struct

将结构成员打包在一起，这样彼此间就不能插入对齐空间了。

该选项将引起访问可执行代码的结构成员的效率降低，而且可能引起代码与系统库的不兼容。

对 Fortran 而言，绝对不可以使用选项 -fpack-struct 。

参见第 4 章的 4.3.5 节“属性”。

Fortran

-fpedantic

参见 Fortran 的选项 -pedantic 。

-fpeephole

这是默认选项，但可通过选项 -fno-peephole 关闭。输出汇编语言时，打开窥孔优化。它将匹配指令的集合，并用优化的版本替代它们。该选项在优化选项没有打开时无效。该选项是与平台相关的，可能对某些平台无效。

-fpeephole2

在寄存器分配后但未进行调度时，打开 RTL 窥孔优化。该优化是将一组指令集合进行机器相关的翻译，翻译成另外一组指令。该选项是平台相关的，因此可能对某些平台无效。如果没有打开优化，该选项也无效。选项 -O2 ， -O3 和 -Os 将设置该选项，但可以通过选项 -fno-peephole2 关闭该选项。

C++

-fpermissive

指明代码与标准不相符的诊断消息会作为警告（而不是错误）输出。如果没有指定选项 -fpermissive 或 -pedantic ，默认打开选项 -fpedantic-errors 。

-fpic

生成可用于共享库的位置独立代码（ PIC ）。所有的内部寻址均通过全局偏移表 (GOT) 完成。要确定一个地址，需要将代码自身的内存位置作为表中一项插入。该选项需要操作系统支持，因此并不是在所有系统上均有效。

该选项产生可以在共享库中存放并从中加载的目标模块。

如果链接器对选项 -fpic 产生了错误消息，说明位置独立代码不能工作，可以使用选项 -fPIC 。

某些系统的偏移表有尺寸限制。 Motorola 的 m88k 上的尺寸限制是 16k ， m68k 和 RS/6000 上是 32k ， Sparc 上是 8k 。位置独立代码需要一定的支持，因此它可能仅在某些机器上有效。

参见选项 -fPIC 和 -shared 。

-fPIC

该选项与选项 -fpic 相同，但它可以克服在 m68k ， m88k 和 Sparc 上可能遇到的偏移表尺寸限制。

参见选项 -fpic 和 -shared 。

-fppc-struct-return

生成的代码可将结构存储在内存中返回，即使结构可能小到可以在寄存器中保留。将返回的结构存储在内存还是寄存器的具体规范取决于平台。

尽管产生的代码效率可能不高，但当需要连接其他编译程序的代码文件时就需要该选项了。

在 Fortran 中，该选项仅在程序要编译 libg2c 版本时才会使用。

参见选项 -freg-struct-return 。

-fprefetch-loop-arrays

如果平台支持，生成预取数组的指令，可改进循环的性能。

Pre

-fpreprocessed

即使命令行含有后缀，指出文件需要预处理，也不进行预处理。表 D-1 给出了这些后缀。

即使指定了该选项，也可以使用选项 -C 去掉预处理程序的注释。

-fpretend-float

当为另一个系统交叉编译目标文件时，该选项将浮点运算格式化成就好像在本地机上运行一样。其结果很可能是一种无法在目标机上运行的格式，但指令序列和为目标机而产生的指令序列一致。

-fprofile-arcs

在使用这一选项编译程序，并运行它创建包含每个代码块执行计数的文件后，程序可以再次使用选项 -fbranch-probabilities 编译，文件中的信息可以用来优化那些经常选取的分支。如果没有这些信息， GCC 将猜想哪条路径可能被选取而进行优化。这些信息被存放在一个与源文件同名但以 .da 为后缀的文件中。

该选项的另一个用处是同选项 -ftest-coverage 一起使用来支持 gcoy 。该选项组合将为程序中的每个函数创建一个流程图，然后由流程图确定一个生成树。然后，代码将被插到生成树中没有的函数里，生成树将为每次执行产生一个计数。对于只有一个入口和出口的代码块来说，代码将直接加在代码块中。具有多个入口和出口的代码块将会产生新的跟踪每个入口和出口的代码块。

用这些选项编译的程序和 gcov 一起运行，将会比用选项 -a 和 -ax 编译时慢一些，但用选项 -a 产生的计数不能提供足够的信息来预测所有分支可能性。

参见选项 -a ， -ax ， -fbranch-probabilities 和 -fguess-branch-probability 。

-freduce-all-givs

强制所有通用归纳变量（循环计数器）都要进行强度削弱。该选项是否会产生好的代码取决于源代码中循环的结构。除 Fortran 外，默认选项是 -fno-reduce-all-gives 。

参见选项 -fmove-all-movables 。

-freg-struct-return

生成用寄存器返回短结构的代码。如果不够小，无法容纳在一个寄存器中，将使用内存返回。用内存或寄存器返回结构的具体规范和平台相关。

在 Fortran 中，该选项仅在程序要编译 libg2c 版本时才会使用。

参见选项 -fpcc-struct-return 。

-fregmove

与选项 -foptimize-register-move 相同。

-frename-registers

这是一种优化技术，它会试图去掉被调度代码中的假依赖关系，这是通过使用寄存器

分配和调度完成后的寄存器实现。这一优化操作对具有大量寄存器的机器非常有效。通过该选项生成的代码非常难于调试。选项 -O3 将设置该选项，但可以通过选项 -fno-rename-registers 关闭该选项。

C++

-frepo

打开自动模板实例化。设置该选项将同时设置选项 -fno-implicit-templates ，它将抑制非内嵌模板的自动实例化。

-frerun-cse-after-loop

该选项将在循环优化后再次进行公用子表达式优化。这是因为在循环优化后可能会产生新的子表达式。选项 -O2 ， -O3 和 -Os 设置该选项，但可以通过 -fno-rerun-cse-after-loop 关闭该选项。参见选项 --param 。

-frerun-loop-opt

运行循环优化两次。第二次优化并不展开循环，但它会再次利用第一次优化中删除的指令对循环进行分析。O2 ， -O3 和 -Os 设置该选项，但可以通过 -fno-rerun-loop-opt 关闭该选项。

C++

-frtti

这是默认选项。运行时标识代码是为包含虚方法的每个类生成的。如果没有使用 dynamic_cast 和 typeid ，可以使用选项 -fno-rtti 抑制生成代码，从而节省空间。该选项对异常处理无效，将按需生成 rtti 代码。

-fschedule-insns

相对其他操作而言，在浮点运算速度较慢或内存访问操作较慢的机器上，以及支持同一时间运行多条指令的机器上，可以改变指令的序列来消除处理器的阻塞。在较慢的指令正在执行的同时，执行其他指令。选项 -O2 ， -O3 和 -Os 将设置该选项，但可以使用 -fno-schedule-insns 关闭该选项。

-fschedule-insns2

该选项与 -fschedule-insns 一样，只是它是在为每个函数分配全局寄存器和局部寄存器之后执行的。对于寄存器数目较少而且装载寄存器的指令较慢的机器，该选项非常有效。选项 -O2 ， -O3 和 -Os 将设置该选项，但可以使用 -fno-schedule-insns2 关闭该选项。

-fshared-data

该选项要求数据共享而非私有。该选项只对某些操作系统有意义，在这些操作系统中，可以在运行同一程序的独立进程间访问共享数据，而且每个进程都有自己的私有数据。

-fshort-double

double 数据类型使用与 float 数据类型相同的尺寸。

对 Fortran 而言，该选项可能会引起问题。

-fshort-enums

将某个 enum 的尺寸减少到保存所需数值范围的最小整数类型。

C C++

-fshort-wchar

将数据类型 wchar_t 转换为 unsigned short int ，而非当前目标机的默认类型。

C

-fsigned-char

使用该选项后，数据类型 char 默认是有符号的（从 0 到 255 的数值）。如果没有指定 char 数据类型是否有符号，默认情况将取决于平台。使用选项 -fno-signed-char 与选项 -funsigned-char 相同。

C

-fsigned-bitfields

这是默认选项。位字段按有符号 int 数据类型处理，但选项 -fno-signed-bitfields 将按无符号 int 数据类型处理位字段。使用选项 -traditional 将强制按无符号处理所有的位字段。使用选项 -fno-signed-bitfields 与选项 -funsigned-bitfields 相同。

Fortran

-fsilent

这是默认选项。使用选项 -fno-silent 将在 stderr 上输出所有正在编译的程序单元的名字。

-fsingle-precision-constant

按常量声明的浮点数值将按单精度存储，而非双精度。

Fortran

-fsource-case- spec

spec 确定源代码文本是否被翻译为全部大写、全部小写或是不变。Holerith 常量不受该选项影响。spec 可为如下形式之一：

· upper 源代码全部翻译为大写。

· lower 源代码全部翻译为小写。这是默认选项。

· preserve 源代码文本不变。

参见选项 -fintrin-case- ， -fmatch-case- ， -fsymbol-case- 和 -fcase- 。

-fssa

试验功能。整个函数体将转换为一个 SSA （ Static Single Assignment ，静态单赋值）流图，之后在其上进行优化操作，继而再将代码由 SSA 转换为原格式。

-fssa-ccp

试验功能。打开 SSA 条件代码传播，其中实际上被当做常量来使用的变量将被转换为常量，并且会消除从未选取的分支。该选项需要选项 -fssa 和任意级 -O 优化。

-fssa-dce

试验功能。打开 SSA 不可达代码消除功能，将不可能被执行的代码删除。该选项需要选项 -fssa 和任意级 -O 优化。

-fstack-check

生成的代码会为防止程序栈溢出进行必要的检测。所生成的代码实际上并不进行检测，它只保证操作系统可以检测到栈扩展。

可能有必要在多线程环境中运行一个程序，因为在单线程程序中，栈溢出是自动检测的。

-fstack-limit-register= register

指定包含限制栈尺寸地址的寄存器名称。该选项只能用来减少栈；不能用它来扩展超出操作系统指定的栈尺寸。

参见选项 -fstack-limit-symbol 。

-fstack-limit-symbol= symbol

指定包含限制栈尺寸地址的变量名。该选项只能用来减少栈；不能用它来扩展超出操作系统指定的栈尺寸。

地址的值取决于平台。例如，如果栈从地址 0x8000000 开始并且通过降低地址完成增长栈， 128k 的地址限额可以通过如下选项设置：

-fstack-limit-symbol=__stack_limit -W1,__stack_limit=0x7FFE0000

还可以在程序内声明地址限额变量，但依然需要在命令行中使用选项 -fstack-limit-symbol 。

参见选项 -fstack-limit-register 。

C++

-fstats

显示前端处理的统计信息。这些信息主要用于编译程序内部，对其输出没有影响。

Java

-fstore-check

这是默认选项。指定选项 -fno-store-check 将删除运行时检查，可保证在数组中保存正确的对象类型。

-fstrength-reduce

执行循环强度消除并消除在循环内部使用的变量。这是用简单而快速的操作（如加法和减法）替代耗时操作（如乘法和除法）的过程。选项 -funroll-loops 和 -funroll-all-loops 均打开该选项。选项 -O2 ， -O3 和 -Os 也打开该选项，但可以通过 -fno-strength-reduce 关闭该选项。

作为一个简单的例子，下面的循环使用一个临时变量作为索引：

```
for(int i=0; i<10; i++) {
index = i * 2;
frammis(valarr[index]);
}
```

内部变量 index 可被消除，乘法也可以被替代为简单的移位操作，如下所示：

```
for(int i=0; i<10; i++) {
frammis(valarr[i << 1]);
}
```

向左移循环计数器一位就是将其加倍，之后就可以直接将结构作为数组的索引，而无需使用一个临时变量保存。

-fstrict-aliasing

根据正在编译的语言，将使用最严格的别名规则。例如，在 C 中一个 int 不能成为一个 double 或指针的别名，但可以成为一个 unsigned int 的别名。即使使用了最严格的别名规则，对联合的成员的访问也没有问题，只要是通过联合而不是通过联合成员的地址进行访问。下面的一段代码将引起问题：

```
int *iptr;
union {
int ivalue;
double dvalue;
} migs;
. . .
migs.ivalue = 45;
iptr = &migs.ivalue;
frammis(*iptr);
migs.dvalue = 88.6;
frammis(*iptr);
```

在这个例子中，严格的别名规则可能无法识别 iptr 所指引的数值在两次函数调用间的变化。但直接通过联合成员引用就不会引起问题。

Fortran

-fsymbol-case- spec

spec 决定用户自定义符号的大小写。可以是如下形式之一：

· initcap 名字的首字母大写，其余小写。

· upper 名字均为大写。

· lower 名字均为小写。

· any 名字可以是大小写的任意组合。这是默认选项。

参见选项 -fmatch-case- ， -fsource-case- ， -fintrin-case- 和 -fcase- 。

Fortran

-fsyntax

对源代码进行语法检查，之后不进行任何处理。

C++

-ftemplate-depth- number

将最大模板实例化深度设置为 number 值，以探测递归或循环模板定义。符合标准的程序不能使用深度超过 17 的模板。默认深度是 500 。

-ftest-coverage

编译程序将在所生成的文件中包含 gcov 使用的信息。输出文件与源文件具有相同的名字，但具有不同的后缀以示内容上的区别。

后缀为 .bb 的文件包含了由基本执行代码块到源文件行号的映射。这些信息可供 gcov 将执行计数与源代码行号建立联系。

后缀为 .bbg 的文件包含了程序流图中的所有箭头。这些信息供 gcov 使用来重建流图并从选项 -fprofile-acrs 指定的以 .da 为后缀的文件数据中计算执行计数。

参见选项 -a ， -ax ， -ftest-coverage 和 -fprofile-arcs 。

-fthread-jumps

如果某跳转条件表达式的值将代码转移到某处，而该值在此处恰巧又是一个跳转，那么源跳转将被直接转移到最终目的地。所有的优化级均设置该选项，但可以使用选项 -fno-thread-jumps 将其关闭。

-ftime-report

编译完成时，编译程序会打印编译耗时的统计信息。每一次扫描所涉及的用户时间、系统时间以及实际时间都将被打印出来，并在最后给出一个概要信息。

-ftrapping-math

这是默认选项。设置选项 -fno-trapping-math 将使代码假设浮点操作不会引起异常，这些异常可能被捕获并产生信号。选项 -fno-trapping-math 可能会生成违反标准浮点操作规则的代码。

-ftrapv

生成捕获有符号加、减和乘法等溢出条件的代码。通常这些溢出是被忽略的，但该选项可以在软件测试中使用，并在有整数溢出时产生 core 文件。默认选项是 -fno-trapv 。

Fortran

-ftypeless-boz

指出带前缀基数的非十进制常量，如 Z'ABCD' ，是无类型的而不是默认的 INTEGER(KIND=1) 。

Fortran

-fugly-args

这是默认选项。指定选项 -fno-ugly-args 将不允许无类型 Hollerith 常量作为参数在函数调用中传递。例如，下面两个函数调用默认是合法的：

```
CALL FRED(4HABCX)
CALL SAM('123'O)
```

Fortran

-fugly-assign

使用同样的存储包含被赋值的标记和数值数据。例如，下面两条语句将使用同一存储位置：

```
I = 3
ASSIGN 10 TO I
```

如果程序想按数据访问被赋值的数值，就需要使用该选项，因为默认情况下编译程序会为不同类型的信息单独创建存储位置。

Fortran

-fugly-assumed

尺寸为 1 的数组将按声明尺寸为 * 的数组使用。例如，DIMENSION X(1) 被看做是以 DIMENSION X(*) 形式声明的。

Fortran

-fugly-comma

结尾的逗号隐含地将空参数传递给子例程。例如，使用该选项时， CALL BLOG() 会传递一个空参数，而 CALL RIM(,) 将传递两个空参数。

如果不使用该选项，结尾的逗号将被忽略，从而不会传递空参数，即使是列表中有其他参数也不会传递。

Fortran

-fugly-complex

允许在特质 REAL(expr) 和 AIMAG(expr) 中使用任何复数表达式。默认情况下会限制复数表达式为 COMPLEX(KIND=1) 。

使用选项 -ff99 时，这些特质将返回参数中未被转换的实部和虚部。

Fortran

-fugly-init

这是默认选项。指定选项 -fno-ugly-init 将不允许在 PARAMETER 和 DATA 语句中使用 Hollerith 格式的数据。同时不允许使用字符常量来初始化数值数据类型，反之亦然。

Fortran

-fugly-logint

大多数情况下，系统会自动在数据类型 INTEGER 和 LOGICAL 间进行转换，基本上这两种类型可以互换使用。

Fortran

-funderscoring

这是默认选项。在有一个下划线的名字后追加两个下划线。在没有下划线的外部名字后追加一个下划线。为防止与外部名字产生冲突，对具有下划线的内部名字同样追加两个下划线。

指定选项 -fno-underscoring 将禁止通过追加下划线对名字进行转换。指定选项 -fno-second-underscoring 只禁止追加第二个下划线。

除非试图控制输出以产生与其他编译程序兼容的代码，否则不建议使用这些控制下划线用法的选项。另外，控制下划线可能会产生与系统库的冲突。

Java

-fuse-boehm-gc

打开 Boehm 垃圾收集器位图标记代码的使用。

Fortran

-fsecond-underscore

这是默认选项。参见 -funderscoring 。

Fortran

-funix-intrinsics- spec

spec 指定了 UNIX 特质的状态。它可以是下列形式之一：

· enable 可以识别并打开 UNIX 特质。这是默认选项。

· hide 只有在特质中的名字首次出现在 INTRINSIC 语句中时，才识别和打开 UNIX 特质。

· disable 可识别 UNIX 特质，但只有通过 INTRINSIC 语句方可引用它们。

· delete 不可识别 UNIX 特质。

-funroll-all-loops

该选项设置 -funroll-loops 并去除循环展开的尺寸限制，即使当循环的次数无法确定时仍然展开循环。该选项通常会产生比不使用该选项更慢和更大的代码。

当要展开一个循环次数无法确定的循环时，循环被展开为一系列代码块，每个代码块的结尾都有循环测试。这将生成一个包含自身多个副本的更大循环，因此不会像其他情况下那样经常迭代。

-funroll-loops

如果在编译时可以确定迭代的次数非常小，而且循环中的指令也非常少，可以进行循环展开，通过去除循环和复制指令，从而保证正确的执行次数。如果 insns 的数值乘以迭代的次数小于一个常量（当前设置为 100 ），就认为循环足够小。该选项同时设置选项 -fstrength-reduce 和 -frerun-cse-after-loop 。

-funsafe-math-optimizations

去除浮点运算的检查并假设所有的数值均是合法的。这将允许违反 IEEE 和 ANSI 标准的数学操作。该选项还可能造成链接器包含以非标准方式优化硬件 FPU （浮点运算单元）操作的代码。

C

-funsigned-bitfields

使用该选项将按照 unsigned int 数据类型处理位字段。默认情况下，将按 signed int 数据类型处理位字段。使用选项 -traditional 也会将所有的位字段按照无符号类型处理。指定选项 -fno-unsigned-bitfields 与选项 -fsigned-bitfields 相同。

C

-funsigned-char

使用该选项后，默认情况下 char 数据类型为无符号类型（值域为 - 127 到 +128 ）。如果

没有使用该选项，默认情况下 char 数据类型是否有符号取决于所使用的平台。指定选项 -fno-unsigned-char 与选项 -fsigned-char 相同。

-funwind-tables

除了生成必要的静态数据外，该选项与选项 -fexceptions 相似，对所生成的代码没有其他影响。该选项只供内部使用，不应该在命令行中使用。

参见选项 -fexceptions ， -fnon-call-exceptions 和 -fasynchoronous-unwind-tables 。

C++

-fuse-cxa-atexit

造成全局析构函数按照构造函数完成的相反顺序执行。仅当某构造函数内部调用了另外一个构造函数时，顺序才会改变。该选项仅在函数 cxa_exit() 是 C 运行时库的一部分时工作。如果不使用该选项，可以使用函数 atexit() 。

Java

-fuse-divide-subroutine

调用库例程完成整数除法，从而可以在整数除数为 0 时抛出异常。

-fverbose-asm

在所生成的汇编语言中插入比正常情况下更多的注释，从而使其更易读。

该选项主要用来使汇编代码更易读，并用来调试编译程序本身。默认情况下，相比汇编列表，选项 -fno-verbose-asm 更有用处。

Fortran

-fversion

运行内部测试来验证 GNU Fortran 是否正确安装，并显示版本号。选项 -v 和 --verbose 均设置了该选项。

-fvolatile

将所有通过指针引用的内存视为易变的（ volatile ）。

参见选项 -fvolatile-global 和 -fvolatile-static 。

-fvolatile-global

将所有外部和全局内存引用视为包含易变数据。该开关不会将静态数据项（仅从编译单元访问的数据）视为易变的。

参见选项 -fvolatile 和 -fvolatile-static 。

-fvolatile-static

将所有引用静态数据项（仅从编译单元访问的数据）的内存引用视为易变的。

参见选项 -fvolatile-global 和 -fvolatile 。

C++

-fvtable-gc

导致重定位信息的创建，从而使连接程序可以删除没有使用的虚函数 vtable 表项。该选项需要同时使用 GNU 汇编器和链接器。

该信息也可以用来删除未使用的函数。参见选项 -ffunction-sections 和 -W1 。

Fortran

-fvxt

某些源代码构造函数在 GNU Fortran 和 VXT Fortran 中具有不同的含义。使用该选项将造成所有构造函数按 VXT Fortran 解释。

参见选项 -ff90 ， -ffvxt-intrinsics 和 -ff90-intrinsics- 。

Fortran

-fvxt-intrinsics- spec

spec 指定了 VXT 特质的状态。它可以是如下形式之一：

· enable 可以识别并打开 VXT 特质。这是默认选项。

· hide 仅当特质名字首次出现在 INTRINSIC 语句中时才识别并打开 VXT 特质。

· disable 可以识别 VXT 特质，但必须通过 INTRINSIC 语句方可进行引用。

· delete 不可识别 VXT 特质。

参见选项 -fvxt 。

C++

-fweak

这是默认选项。指定选项 -fno-weak 将禁止使用弱符号支持，即使链接器支持也不能使用。由于选项 -fno-weak 会产生较低级的代码，除了测试编译程序外没有其他用处，不应该使用选项 -fno-weak 。

C C++

-fwritable-strings

编译程序允许将数据写入字符串常量。选项 -traditional 将设置该选项。

为了能够真正写入 C++ 的字符串常量，同时需要设置选项 -fno-const-strings 。

Fortran

-fzeros

像其他值一样处理初始值 0 。如果没有这一选项，可以有多个 DATA 语句设置变量的初始值为 0 ，而编译程序无法诊断这一情况。

-g [level]

输出的调试信息格式可被 gdb 使用。其格式和内容依赖于编译程序所产生的目标文件格式（ stabs ， COFF ， XCOFF 或 DWARF ）。

level 设置是可选的。 level 数值指定了所包含的调试信息量。默认级别是 2 。级别 1 产生的全局信息可供反向跟踪，但不包括局部变量和行号。级别 2 包括所有级别 1 的内容，同时还有局部变量和行号信息。级别 3 包括级别 2 的信息以及宏定义等额外信息。

在使用 stabs 格式的系统中，该选项将按 gdb 可用的格式生成调试信息。

使用该选项的同时可用选项 -O 生成优化代码。当优化打开时，调试过程可能不像正常情况下那样容易跟踪，这是因为优化在生成代码时做了许多改动，源代码和所生成的目标文件间不存在一对一的对应关系。某些目标文件可以重定位，而某些源码可能根本不生成任何代码。

参见选项 -ggdb ， -gstabs ， -gcoff ， -gxcoff 和 -gdwarf 。该选项可以写作 --debug 。

-gcoff [level]

如果支持的话，按照 COFF 格式生成调试信息。在 SVR4 以前的 System V 的 SDB 中，该格式使用比较广泛。设置 level 是可选的。参见选项 -g 中关于 level 设置 1 ， 2 和 3 的说明。

-gdwarf [level]

如果支持的话，按照 DWARF 格式生成调试信息。设置 level 是可选的。只有将 level 设置为 + 才会包含 gdb 扩展，它可能令其他调试器无法使用。参见选项 -g 中关于 level 设置 1 ， 2 和 3 的说明。

这是大部分 SVR4 系统中的 SDB 使用的格式。

-gdwarf-2 [level]

如果支持的话，按照 DWARF 版本 2 的格式生成调试信息。设置 level 是可选的。参见选项 -g 中关于 level 设置 1 ， 2 和 3 的说明。

这是 IRIX 6 的 DBX 中的格式。

ObjC

-gen-decls

将源文件中类的接口声明写入名为 w.decl 的文件中。

-ggdb [level]

生成适用于 gdb 格式的详细调试信息，包含任意可能的 gdb 扩展。设置 level 是可选的。参见选项 -g 中关于 level 设置 1 ， 2 和 3 的说明。

Ada

-gnat option

指定为 Ada 前端 GNAT 提供的选项。所有选项都是单字母，跟在选项字 gnat 之后。例如，如果要指定选项 e 和 l ，可以在命令行中这样指定它们：

$ gcc -gnate -gnatl

还可以将两个选项组合成一个。下面的形式与前面的相同：

$ gcc -gnatel

某些选项需要提供数值。例如，选项 k 用来指定由编译程序输出的错误消息的最大数目，下面的命令将该值设置为 15 ：

$ gcc -gnatk15

需要数值的选项可以与其他选项组合在一起，但需要放在列表的最后。下面的例子给出选项 e ， l 和 k 的组合格式：

$ gcc -gnatelk15

下面的选项指定了数字但不带前缀字母，强制使用 Ada 83 或 Ada 95 定义的限制：

$ gcc -gnat83
$ gcc -gnat95

默认是选项 -gnat95 。表 D-3 包含了选项 -gnat 前缀可用的选项字母列表。

表 D-3 用于 -gnat 选项的字母和值

字母	描述
a	打开断言。该选项同时打开 Pragma Assert 和 Pragma Debug 。如果不使用该选项，这两个 pragma 将被忽略
b	即使同时设置了 verbose 选项，在标准错误输出产生简要消息
c	只检查语法和语义。生成 .ali 文件，但不生成可执行代码
e	当错误发生时显示错误消息，而不是在结束时保留和打印所有的错误
E	执行完全的动态检查
f	报告每一种可能的错误。每一行均可能检测出多个错误。报告所有未定义的引用，而不仅仅是第一个引用
g	打开 Ada 风格的检查（列对齐、缩进、大小写格式等）
ichar	char 的值指定了标识符的字符集。标准的 ASCII 字符（值域由 1 到 127 ）总是相同的，但 8 位数值中的其他部分具有不同的含义。其所标识的字符集如下： 1 拉丁 -1

（续表）

字母	描述
	2 拉丁 -2 3 拉丁 - 3 4 拉丁 - 4 p IBM PC f 全大写 n 无大写 w 宽字符

jchar	char 的值指定了用来编码宽字符的方法。可用的方法如下： n 没有编码 h Hex 编码 u 上半区编码 s Shift JIS 编码 e EUC 编码
knumber	number 的值指定了一个标识符中字母个数的上限（ 1~999 ）
l	输出源代码列表，包括错误消息
mnumber	number 的值是可以包括的最大错误数目
n	激活跨子程序单元边界的内嵌机制，这些子程序均指定了 inline pragma 。如果使用了选项 -fno-inline ， 该选项将被关闭
N	与使用选项 -gnatn 并为每行源代码加入 inline pragma 相同。可以使用 -fno-inline 关闭该选项
o	打开通常关闭的运行时检查，如整数溢出检查和描述前访问。该选项会造成程序更大且运行更慢。（该选项对浮点操作没有影响）
p	禁止运行时检查。这与 pragma Suppress(all_checks) 命令相同。通过去除运行时安全检查，程序可以更小而且运行更有效
q	在发现语法错误后不停止编译程序。之后将根据解析器所收集的信息继续代码生成
r	要求列格式匹配参考手册的要求
s	仅检查语法
t	生成用于不可达代码消除的 .adt 树文件
u	列出本次编译所涉及的所有单元
v	详细模式。在标准错误输出中给出完全的错误报告以及引起错误的源代码行号
wmode	该模式决定如何处理警告。如果 mode 是 s ，将禁止警告消息。如果 mode 是 e ，警告消息将被处理为错误消息。如果 mode 是 l ，将产生描述次序警告

ztype	指定生成存根。如果 type 是 r ，生成接收端存根。如果 type 是 s ，将生成发送端存根

-gstabs [level]

如果支持的话，按照 stabs 格式生成调试信息。设置 level 是可选的。只有将 level 设置为 + 才会包含 gdb 扩展。参见选项 -g 中关于 level 设置 1 ， 2 和 3 的说明。

该选项可以用在大多数 BSD 系统的 DBX 中，但无法在 MIPS ， Alpha 或 SVR4 上的 DBX 或 SDB 上使用。而且在 SVR4 系统上需要使用 GNU 汇编器。

-gvms [level]

如果支持的话，按照 VMS 调试格式生成调试信息。设置 level 是可选的。参见选项 -g 中对 level 设置 1 ， 2 和 3 的说明。

这是 VMS 系统中 DEBUG 所使用的格式。

-gxcoff [level]

如果支持的话，按照 XCOFF 格式生成调试信息。设置 level 是可选的。只有将 level 设置为 + 才会包含 gdb 扩展，它可能令其他调试器无法使用，而且与非 GNU 汇编器一起使用时还会出现问题。参见选项 -g 中关于 level 设置 1 ， 2 和 3 的说明。

这是 RS/6000 系统的 DBX 所使用的格式。

Pre

-H

打印使用的所有头文件的嵌套列表，以及那些没有多次包含保护的头文件。

该选项也可以写作 --trace-includes 。

--help

显示 gcc 理解的命令选项列表。如果和选项 -v 一起使用，列表还将包含 gcc 调用的各个进程所接受的选项。如果与选项 -W 一同使用，同时会列出未给出文档的命令行选项。参见选项 --target-help 。

--include-directory

与选项 -I 相同。

-I name

对预处理程序而言， name 是包含文件搜索目标主列表的目录。该选项可以重复使用，从而加入若干个目录。

该选项所指定的目录将被首先搜索，因此可用该选项重载通常所包含的系统头文件。

可以用 -I. 中的点引用当前目录，它向编译程序指定当前工作目录。

对 Ada 而言， name 是搜索源文件的目录。

对 Java 而言， name 是在搜索其他路径（如选项 --classpath 中命名的或在环境变量 CLASSPATH 中命名的）之前搜索的路径单元。该单元可以是目录、 jar 文件或 zip 文件。建议使用 -I 而非 --classpath 。

参见选项 -I- ， -isystem ， -B ， -nostdinc 和 -withprefixbefore 以及环境变量 CPATH 。如果要设置二级头文件搜索目录列表，参见选项 -idirafter 。该选项也可以写作 --include-directory 。

-I-

-I- 特殊形式可被指定一次，用来声明命令行中其前所有 -I 指示字对 #include "…" 生效，但不对 #include <…> 生效。任何跟在选项 -I- 后的 -I 选项都将对 #include <…> 生效。

选项 -I- 也将忽略对源文件所在目录的搜索。

该选项也可以写作 --include-barrier 。

Pre

-idirafter directory

将 directory 加到二级头文件搜索目录列表中。要查找一个头文件， GCC 将在一级列表（通过选项 -I 加入的目录）中的目录中搜索。如果在一级列表中没有找到头文件，就继续搜索二级列表。

该选项也可以写作 --include-directory-after 。

Pre

-imacros filename

预处理程序将在读取程序源文件前读取和处理所指定的文件。除了宏之外，存储在 filename 中的信息将被丢弃，只有宏会用于源文件。

命令行中的选项 -D 或 -U 将在选项 -imacros 之前处理。选项 -include 和 -imacros 将按其在命令行中出现的次序处理。

该选项也可以写作 --imacros 。

Pre

-include filename

预处理程序将在读取程序源文件前读取和处理所指定的文件，就如同其在源文件的第一行被包含了一样。

命令行中的选项 -D 或 -U 将在选项 -include 之前处理。选项 -include 和 -imacros 将按其在命令行中出现的次序处理。

该选项也可以写作 --include 。

Pre

--include-barrier

与选项 -I- 相同。

Pre

--include-directory-after directory

与选项 -idirafter 相同。

Pre

--include-prefix prefix

与选项 -iprefix 相同。

Pre

--include-with-prefix directory

与选项 -iwithprefix 相同。

Pre

--include-with-prefix-after directory

与选项 -iwithprefix 相同。

Pre

--include-with-prefix-before directory

与选项 -withprefixbefore 相同。

Pre

-iprefix prefix

指定用来构成选项 -iwithprefix 和 -withprefixbefore 中给出的目录路径的前缀。

该选项也可以写作 --include-prefix 。

Pre

-isystem directory

在二级包含路径的起始处增加名为 directory 的目录，并将目录标记为系统目录，并按标准系统目录的使用方式处理。

参见选项 -I 和 -B 。

Pre

-iwithprefix directory

在用于搜索头文件的二级目录列表中增加名为 directory 的目录。通过选项 -iprefix 中指定的前缀与名为 directory 的目录组合构成最终的路径名。如果在该选项之前命令行中没有指定前缀，默认使用包含安装编译程序的目录。

要搜索某个头文件， GCC 在（通过选项 -I 所增加的）一级列表中搜索目录。如果在一级列表中没有找到，就搜索二级列表。

该选项也可以写作 --include-with-prefix 或 --include-with-prefix-after 。

Pre

-iwithprefixbefore directory

在主包含路径的目录中增加一个名字。通过选项 -iprefix 所指定的前缀和名为 directory 的目录的组合构成名字。如果在该选项之前命令行中没有指定前缀，默认使用包含安装编译程序的目录。

该选项也可以写作 --include-with-prefix-before 。

Linker

--library-directory directory

与选项 -L 相同。

Linker

-L directory

将指定的目录加入选项 -l 搜索的库目录中。

参见选项 -B 和环境变量 LIBRARY_PATH 。该选项也可以写作 --library-directory 。

Linker

-l library

设置链接器解析引用的库名。实际的库名是通过在指定的名字中增加前缀 lib 和后缀 .a 构成的。例如，设置 -lconsole 后，实际的库名是 libconsole.a 。

对所指定的库文件的搜索将在标准库文件集以及选项 -L 所给出的目录中进行。

选项 -lobjc 是链接 Objective-C 程序的特殊情况。

要解析引用，按照其在命令行中出现的顺序搜索库，这意味着它们出现的顺序是至关重要的。例如，在下面的命令行中，任何 glower.o 对 libjpeg.a 中的目标文件的引用都可以被解析，但 flower.o 中的引用就无法解析。

gcc glower.o -ljpeg flower.o -o showall

当某个库中的成员需要另外一个库中的成员才能解析时，该顺序也非常重要。有可能在两个库间形成循环引用，这需要它们在命令行中出现两次：

gcc sprig.o -ldflat -lturbo -ldflat -o sprig

可以在命令行中按 libjpeg.a 和 -ljpeg 方式引用同一个库，但只有选项 -l 才会指示链接器在标准目录以及选项 -L 所指定的目录中搜索。

为了符合 POSIX 标准，在选项标志和库名之间可以留一个空格。

参见选项 -L 。

--language language

与选项 -x 相同。

-M

预处理程序输出一组适合 makefile 书写的规则。该规则包含目标文件名、冒号、源文件和其所包含的头文件。这些头文件将另起一行并用全路径名表示。如果选项 -include 和 -imacros 均在命令行中，这些文件也将列在该规则中。

选项 -M 隐式地使用选项 -E 。

产生的规则包含目标文件名及其依赖文件列表；但不包含编译源文件的规则。

如果没有使用选项 -MT 或 -MQ 指定名字，在所产生的规则中的目标文件名与输入的源文件名相同，但替换了后缀。

源文件可以是表 D-1 中需要处理的任何一种文件。例如，如果源文件是 Java ，系统 Jar 文件将被列出。

其他涉及生成 makefile 规则的预处理程序选项有 -MD ， -MMD ， -MF ， -MG ， -MM ， -MP 和 -MQ-MT 。

该选项也可以写作 --dependencies 。

Java

--main= classname

指定包含 main() 方法的类名，该方法是执行程序的起点。

每个类均可以有自己的 main() 方法，因此当将它们编译和链接在一起时有必要指定哪个作为程序入口。

Fortran

-maligned-data

该选项只作用于 Intel x86 上的 Fortran 。

频繁使用 REAL(KIND=2)(DOUBLE PRECISION) 的 Fortran 程序在双精度浮点数对齐于 64 位边界时，其性能会大大提高。

Pre

-MD

除了不隐式地打开选项 -E 外，该选项与选项 -M 相同。编译会正常执行， makefile 规则输出到具有与源文件同样名字但以 .d 为后缀的文件中。可以同该选项一起使用选项 -MF ， -o 或 -E 来指定规则输出文件的名字。

该选项也可以写作 --write-dependencies 。

Pre

-MMD

除了不列出系统头文件外，该选项与选项 -MD 相同。

该选项也可以写作 --write-user-dependencies 。

Pre

-MF filename

当与选项 -M ， -MM ， -MD 或 -MMD 一同使用时，该选项指定输出文件名。

另一种指定输出文件名的方法就是设置环境变量 DEPENDENCIES_OUTPUT 。

Pre

-MG

该选项可以同选项 -M 或 -MM 一同使用，指出缺少的头文件假设就为生成文件，并与源文件在同一目录中。假设头文件存在且不包含其他文件，并生成依赖关系列表。

该选项也可以写作 --print-missing-file-dependencies 。

Pre

-MM

除了不列出系统头文件外，该选项与选项 -M 相同。

该选项也可以写作 --user-dependencies 。

Pre

-MP

与选项 -M 或 -MM 一同使用，为每个包含文件生成哑目标。其惟一目的就是防止 make 在删除头文件而又没有更新 makefile 文件时产生错误消息。

Pre

-MQ filename

除了要用适合 makefile 的方式引用目标名外，该选项与选项 -MT 相同。例如，命令 gcc -M -MT '$(OBJMRK)mrk.o' brink.c 将产生如下内容：

$$(OBJMRK)mrk.o: brink.c

Pre

-MT filename

与选项 -M 或 -MM 一同使用，指定所产生的 makefile 规则的目标文件名。默认情况下，目标文件与输入的源文件名相同，但使用后缀 .o 。选项 -MT 可以用来指定一个不同的名字、一个完整的路径或是基于环境变量的名字。例如，命令 gcc -M -MT '$(OBJMRK)mrk.o' brink.c 将产生如下内容：

$(OBJMRK)mrk.o: brink.c

参见选项 -MQ 。

Pre

--no-line-commands

与选项 -P 相同。

Pre Ada

--no-standard-includes

与选项 -nostdinc 相同。

Linker

--no-standard-libraries

与选项 -nostdlib 相同。

--no-warnings

不产生警告消息。与选项 -w 相同。

Linker

-nodefaultlibs

标准系统库例程不应该作为被链接的可执行程序的一部分包含进来。用到的库仅仅是在命令行中所列出的那些库。

编译程序可能会产生对系统函数 memcpy() ， memcmp() ， memset() （对标准 C 和 System V 而言），bcopy() 和 bzero() （对 BSD 而言）的调用。这些引用通常都可在 libc.a 中进行解析，因此需要在命令行中亲自提供这些例程。

库 libgcc.a 包含了一系列目标平台相关的特殊例程，可以将它们视为编译程序的一部分，因此在没有标准库时应该指定 -lgcc 。

参见选项 -nostartfiles 和 -nostdlib 。

Linker

-nostartfiles

标准启动目标文件不应该作为链接的可执行程序的一部分被包含。参见选项 -nostdlib 和 -nodefaultlibs 。

Pre Ada

-nostdinc

禁止编译程序在标准系统目录中搜索头文件。搜索的仅仅是当前目录和选项 -I 中指定的目录。

对 Ada 来说，该选项指定了源文件不使用系统库。

该选项也可以写作 --no-standard-includes 。

Pre C++

-nostdinc++

禁止编译程序在标准 C++ 目录中搜索头文件，但可以在其他标准目录中搜索。该选项专供编译 C++ 库使用。

Linker

-nostdlib

只有命令行指定的项才传递给链接器。标准启动文件和库都不会传递给链接器。该选项隐式打开选项 -nostartfiles 和 -nodefaultlibs 。

该选项也可以写作 --no-standard-libraries 。

-O level

指定用于编译程序所生成的代码的优化级别。注意，在代码尺寸和执行速度间永远都需要权衡。默认选项是 -O0 ，即不进行优化。

如果没有指定优化级别，编译程序产生与输入源结构匹配的代码。优化不仅需要更多的处理，还需要更多的内存。没有优化的编译具有两个好处，可以缩短编译时间（优化则需要很长的时间），而且可以在调试器中轻松跟踪所产生的代码。对软件开发过程而言，所有这些都是理想的条件。也可以使用调试器调试已优化的代码，但由于输出的代码可能被重组，因此很难进行跟踪。

该选项也可以写作 --optimize 。

优化级别参见表 D-4 。

表 D-4 优化的 6 种级别

级别	描述
-O	编译程序尝试减少代码尺寸和执行时间 ， 但不会进行造成调试困难的修改。该级别会打开选项 -fno_optimize_size ， -fdefer_pop ， -fthread_jumps ， -jguess_branch_prob ， -cprop-registers 和

	-fdelayed_branch 。当调试器在某平台上无需栈指针就可以正常工作时才会设置选项 -fomit-frame-pointer
-O0	这是默认选项。关闭所有优化。关闭所有尺寸优化并设置选项 -fno-merge-constants
-O1	与选项 -O 相同
-O2	该优化级打开不涉及代码尺寸和运行速度权衡的所有优化。除了选项 -O 所打开的选项外，该优化级还打开选项 -foptimize-sibling-calls ， -fcse-follow-jumps ， -fcse-skip-blocks ， -fgcse ， -fexpensive-optimizations ， -fstrength-reduce ， -frerun-cse-after-loop ， -frerun-loop-opt ， -fcaller-saves ， -fforce-mem ， -fpeephole2 ， -fschedule-insns ， -fschedule-insn-after-reload ， -fregmove ， -fstruct-aliasing ， -fdelete-null-pointer-checks 和 -freorder-blocks 。该优化级不进行循环展开、函数内嵌或是寄存器重命名
-O3	除了 -O2 所打开的选项外，该优化级还打开选项 -finline-functions 和 -frename-registers
-Os	优化代码尺寸。打开 -O2 中打开的所有选项。将选项 -falign-loops ， -falign-jumps ， -falign-labels 和 -falign-functions 设置为 1 ，它将禁止任何为对齐而插入的空间

-o filename

将输出写入指定的文件中。该选项对各种输出均有效，可以是预处理过的源代码、汇编语言、目标文件或是已链接的可执行程序。由于只能指定一个输出文件，因此如果可能产生多个文件，就不要使用 -o 选项。

如果编译程序生成一个要链接的可执行程序，而且没有用 -o 选项指定其名字，默认名是 a.out 。

该选项也可以写作 --output 。

--optimize level

与选项 -O 相同。

--output filename

与选项 -o 相同。

Java

--output-class-directory= directory

与选项 -foutput-class-dir 相同。

-p

包含额外的代码，这些代码将输出适合性能分析程序 prof 分析的信息。该选项必须在编译源文件和链接目标文件的同时使用。

参见选项 -pg 。该选项也可以写作 --profile 。

Pre

-P

当同选项 -E 一起使用时，预处理程序不产生 #line 指示字。

该选项也可以写作 --no-line-commands 。

--param name = value

GCC 内部存在一些优化代码程度的限制，因此调整这些值就是调整整个优化。表 D-5 给出了参数的名字和可用的数值。

该选项也可以写作 --param 。

表 D-5 --params 选项可接受的名字

名字	值
max-delay-slot-insn-search	填充一个延迟段时所选择的最大数目的指令。较大的数值可以改进所生成的代码，但会降低编译速度。默认值是 100
max-delay-slot-live-search	当查找具有合法寄存器信息的代码块时的最大代码块搜索数目。较大的数值可以改进生成的代码，但会降低编译速度。默认值是 333

max-gcse-memory	可用来分配执行 GCSE （全局命令子表达式消除）的最大的内存量。如果没有足够的内存执行该操作，优化将无法进行。默认值是 50MB （ 52428800 ）
max-gcse-passes	GCSE （全局命令子表达式消除）的最大迭代次数。默认值是 1
max-inline-insns	在可以被扩展内嵌的方法中的最大指令数目。默认值是 600
max-pending-list-length	在跟踪机制重置槽位列表并重新开始前，可以由槽位调度器在一个等待依赖列表中排序的最大的分支元素数目。一个大函数可能产生成千上万的依赖关系。默认值是 32

-pass-exit-codes

忽略编译过程中所有阶段标志错误的退出代码，并且编译程序将继续运行。 GCC 的返回值是任何阶段中最大的错误代码值。通常，编译程序在返回值为非零值时停止并退出。

C C++ Fortran

-pedantic

严格地遵照 ISO C 和 C++ 标准产生所需的警告消息。如果没有该选项，GNU 扩展将被打开，但符合 ISO 标准的程序可以成功地编译（尽管某些可能需要使用选项 -ansi ）。

对 C 而言，所遵守的是选项 -std 所指定的标准。如果 -std 指定了 gnu89 ，选项 -pedantic 使用 C89 的规则。选项 -pedantic 只输出 ISO 标准所需要的诊断信息，因此很可能不完全符合标准的程序能够成功地编译，而不产生警告消息。 GCC 尚无计划提供强制遵从标准的选项。

对 C 而言，选项 -pedantic 对跟在 __extension__ 后的表达式没有影响。

对 C++ 而言 ， 如果既没有指定选项 -fpermissive 也没有指定选项 -pedantic ， 默认使用选项 -fpedantic-errors 。

对 Fortran 而言，仅在 Fortran 77 中使用扩展时给出警告消息。对在字符串常量中像 C 那样的构建字（如 \n ）会给出警告。对某些 GNU 语言扩展和某些传统 Fortran 功能会给出警告。但是，要注意的是，该选项并不强制要求程序严格地遵从标准。

该选项也可以写作 --pedantic 。

C C++ Fortran

-pedantic-errors

除了诊断消息被当作错误（而不是警告）输出外，该选项与 -pedantic 相同。

对 C++ 而言，如果既没有使用选项 -fpermissive 也没有使用选项 -pedantic ，默认使用选项 -fpedantic-errors 。

该选项也可以写作 --pedantic-errors 。

-pg

包含额外的代码，这些代码将输出性能分析程序 gprof 所需的信息。该选项必须在编译源文件和链接目标文件的同时使用。参见选项 -p 。

-pipe

使用管道（而不是临时文件）作为编译程序一个阶段到另一个阶段交换输出的方式。如果本地汇编器不能从管道中读取输入，该选项将失效。

该选项也可以写作 --pipe 。

--prefix prefix

与选项 -B 相同。

Pre

--preprocess

与选项 -E 相同。

-print-file-name= library

输出指定的库的全路径名。除此之外编译程序不执行其他动作。参见选项 -print-ligcc-file-name 和 -print-prog-name 。

该选项也可以写作 --print-file-name 。

-print-libgcc-file-name

输出库 libgcc.a 的全路径名。该选项与 -print-file-name=libgcc.a 相同。

该选项也可以写作 --print-libgcc-file-name 。

Pre

--print-missing-file-dependencies

与选项 -MG 相同。

-print-multi-directory

输出从命令行中的 multilib 选择项中选出的库的对应目录名。目录名由环境变量 GCC_EXEC_PREFIX 确定。

该选项也可以写作 --print-multi-directory 。

-print-multi-lib

输出多库（ multilib ）选择的目录名，同时输出用来作出该选择的命令行选项。

该选项的输出使用分号 (;) 分隔，并使用 at 符号 (@) 替代连字符（ - ）指定命令行选项名称。这样做可以简化文本的 shell 处理。

该选项也可以写作 --print-multi-lib 。

-print-prog-name= program

输出指定程序（如 cc1 或 cpp0 ）的全路径名。除了列出路径名列表，编译程序不进行其他动作。参见选项 -print-file-name 。

该选项也可以写作 --print-prog-name 。

-print-search-dirs

输出所有目录的完全路径名的列表，GCC 将在该列表中搜索作为子进程运行的程序和连接过程所用的库。编译程序不进行其他动作。

如果没找到程序，可以将其放在所搜索的一个目录中，或在环境变量 GCC_EXEC_PREFIX 中加入程序所在的目录。

该选项也可以写作 --print-search-dirs 。

--profile

与选项 -p 相同。

--profile-blocks

与选项 -a 相同。

-Q

编译程序输出它正在编译的每个函数的名字，并在每个阶段后输出诊断统计信息，包括编译和链接程序所需的时间。

Pre

-remap

该选项指示预处理程序在每个含有包含文件的目录中检查名为 header.gcc 的文件是否存在。如果存在，则使用该文件来确定被查看文件的真实文件名。该文件的每一行包含被查看文件和其真实的文件名。例如，如下 header.gcc 文件中的代码行可以用短文件名代替长文件名：

NotSupportedException.h notsup.h

RollbackException.h rollbak.h

TransactionRequiredException.h transreq.h

Linker

-s

删除可执行程序中的符号名和所有重定位信息。其结果与运行命令 strip 所达到的效果相同。

-S

不调用汇编器或链接器。该选项允许将源文件编译为汇编语言文件，但不运行汇编器将其生成为可执行程序。所产生的输出存放在与源文件同名但以 .s 为后缀的文件中。如表 D-1 中所示，任何没有可识别后缀的输入文件或没有使用选项 -x 给出其类型的输入文件将被忽略。

该选项也可以写作 --assemble 。

-save-temps

不进行编译程序的正常步骤，即删除编译程序所生成的临时文件。临时文件将保留在当前目录中，并具有与其内容相对应的后缀。文件的内容可以通过表 D-1 给出的文件后缀判断。

该选项也可以写作 --save-temps 。

Linker

-shared

链接器将生成共享目标代码，该共享库可在运行时动态链接到程序形成完整可执行体。而且，如果使用 gcc 命令创建共享库作为其输出，该选项可以防止链接器将缺失 main() 方法视为错误。

为了可以正确工作，应该一致地使用选项 -fpic ， -fPIC 以及目标平台选项编译构成同一库的所有共享目标模块。特别是，该选项可能需要生成特殊代码来让构造函数正常工作。由于不正确的选项设定而产生的错误可能很不明显，而且也没有警告消息。

参见选项 -shared-libgcc ， -static-libgcc 和 -static 。该选项也可以写作 --shared 。

Linker

-shared-libgcc

该选项指定使用共享版本的 libgcc 。在不支持共享库的系统上和没有该库的共享版本的系统上，该选项没有作用。

如果是通过 g++ ， gcj 或 g77 调用链接器的，该选项将被自动设定，这是由于需要对异常进行处理。当应用程序需要从一个共享库代码抛出另一个共享库捕获的异常时，需要使用共享版本的 libgcc 。在这些环境下异常抛出器和捕获器都需要使用共享版本的 libgcc 。

参见选项 -shared ， -static-libgcc 和 -static 。

-specs= filename

gcc 驱动程序读取配置文件以确定哪些选项应该传递给哪些子进程（参见第 19 章）。该选项可以通过指定配置文件来覆盖默认配置，它将在默认配置文件读取后进行处理以修改调用子进程的某些规则集。

该选项也可以写作 --specs 。

Linker

-static

链接器将忽略动态可链接库，同时通过将静态目标文件直接包含到结果目标文件完成对所有引用的解析。对于没有动态链接功能的系统，该选项无效。

参见选项 -shared 。该选项也可以写作 --static 。

Linker

-static-libgcc

该选项指定使用静态版本的 libgcc 。该选项可能会引起 C++ 和 Java 中的异常处理问题。

参见选项 -shared ， -shared-libgcc 和 -static 。

C

-std= name

该选项指定 C 语言标准。表 D-6 列出了可以识别的名字。

表 D-6 指定语言标准的选项

名字	描述
iso9899:1990	ISO C 89 标准。该选项同时设置了选项 -fno-traditional ， -fno-writable-strings ，

	-fno-asm ， -fno-nonansi-builtin 和 -fno-noniso-default-format-attributes
iso9899:199409	修正的 ISO C 89 标准。该选项同时设置了选项 -fno-traditional ，-fno-writeable-strings ， -fno-asm ， -fno-nonansi-builtin 和 -fno-noniso-default-format-attributes
iso9899:1999	ISO C 99 标准。该选项还同时设置了选项 -fno-traditional ， -fno-writeable-strings ，-fno-asm ， -fno-nonansi-builtin 和 -fno-noniso-default-format-attributes
c89	与 iso9899:1990 相同
c99	与 iso9899:1999 相同
gnu89	带有 GNU 扩展和某些 ISO C 99 功能的 ISO C 89 标准。该选项还同时设置了选项 -fno-traditional ， -fno-writeable-strings ， -fasm ， -fnonansi-builtin 和 -fnoniso-default-format- attributes

该选项禁止 GNU C 扩展关键字 asm ， typeof 和 inline ，但它们的替代关键字 __asm__ ， __typeof__ 和 __inline__ 仍然有效。

参见选项 -ansi 。该选项可以写作 --std 。

Linker

-symbolic

当构建共享目标代码时绑定对全局符号的引用。这是使用 -shared 或 -static 链接可执行文件的替代方式。只有很少一部分系统支持这一选项，如某些 SVR4 系统和 DG/UX 。

该选项也可以写作 --symbolic 。

-syntax-only

检查输入源代码的语法，报告任何警告或错误，并停止。

--target machine

与选项 -b 相同。

--target-help

显示目标机相关的命令行选项列表。参见选项 --help 。

Pre

--trace-includes

与选项 -H 相同。

C

-traditional

该选项已废止。它试图支持最初的 K&R C 语言中的某些功能。如果传统的程序包含了 ISO C 头文件，它将无法编译成功。该选项将设置选项 -traditional-cpp 和 -fwritable-strings 。

参见选项 -fallow-single-precision 。该选项也可以写作 --traditional 。

C

-traditional-cpp

C 预处理程序将支持最初的预处理程序功能。

该选项也可以写作 --traditional-cpp 。

Pre

-trigraphs

支持 ISO C 三字母词。选项 -ansi 和 -std 隐含该选项。

当设置这一选项时，三字母词（均以 ?? 开始）共有 9 个，每个三字母序列都按下表被翻译为单独的字符：

??= # ??([?? < {

??/ \ ??)] ?? > }

??' ^ ??! | ??- ~

该选项也可以写作 --trigraphs 。

-time

输出编译一个程序的每个子进程所消耗的时间。每一行所列出的时间有用户时间（子进程代码所消耗的时间）和系统时间（系统调用所消耗的时间）。下面的例子给出了将 C++ 程序编译为可执行目标文件的例子：

gcc -time fortest.cpp -o fortest.o

cc1plus 0.14 0.05

as 0.00 0.01

collect2 0.10 0.03

该选项也可以写作 --time 。

Linker

-u name

指定的名字将被插入链接器的符号表且要求必须被解析。之后，链接器将装载包含该符号定义的目标模块以解析该符号。

该选项也可以写作 --force-link 。

Pre

-U macro

如果先前已定义了宏 macro ，宏 macro 将被删除。

所有 -D 选项将在选项 -U 前处理，所有 -U 选项将在选项 -include 或 -imacros 前处理。

该选项也可以写作 --undefined-macro 。

Pre

-undef

预处理程序将不会预定义任何非标准宏。该选项将禁止体系结构定义，如 __unix__ ， __OpenBSD__ ，__mips__ ， __linux__ ， __vax__ ，等等。

Pre

--undefine-macro macro

与选项 -U 相同。

--use-version version version

与选项 -V 相同。

Pre

--user-dependencies

与选项 -MM 相同。

-v

显示编译程序的当前版本号，并显示用于运行编译和链接进程的每一阶段的命令。当单独使用时，该选项将显示编译程序的版本号。当同选项 --help 一起使用时，将显示完整的命令行选项列表。

参见选项 -### 。该选项也可以写作 --verbose 。对 Fortran 而言该选项还将设置选项 -fversion 。

-V version

指定 gcc 将要运行的版本。只有在安装多于一个版本的编译程序时该选项才有意义。默认情况是运行最新安装的版本。

其工作原理是修改用于选择编译程序及其组件的前缀，这些组件通常都安装在 /usr/local/lib/gcc-lib/machine/version 目录下。

参见选项 -b 和 -B 。该选项也可以写作 --use-version 。

--verbose

与选项 -v 相同。

-w

不产生警告消息。与选项 --no-warnings 相同。

-W

该选项将打开一系列的警告消息，这些警告消息会报告将可能引起问题的代码，但也可能是程序员所需要的代码。该选项将打开下面的警告：

· 比较 当无符号值进行小于 0 的比较时给出警告消息。例如，因为无符号数值不可能为负值，下面的测试永远为假：

unsigned int x;

. . .

if(x 0) . . .

· 比较 如将有符号值和无符号值进行比较则给出警告消息。为了进行比较，无符号值要转换为有符号值，可能会引起不正确的结果。该警告可以通过选项 -Wno-sign-compare 关闭。

· 比较 代数与 C 语法中比较的表示有所不同。下面的语句将会产生警告消息：

if(a < b < c) . . .

在代数表示中，该表达式只有在 b 处于由 a 到 c 的开区间才为真。在 C 中，该表达式与下面的例子相同，这与代数表示完全不同：

int result;

result = a < b;

if(result < b) . . .

· 常量返回值 当函数被声明为 const 时，将对函数返回值产生警告。由于函数的返回值是右值 (rvalue) ，因此这里的 const 没有意义。

· 聚集初始值 当为聚集数据类型指定初始值，但并不是聚集的所有成员均被指定了初始值时，会产生警告消息。在下面的例子中，数组和结构均会产生错误消息：

struct {

int a;

int b;

int c;

} trmp = { 1, 2 };

int arr[10] = { 1, 2, 3, 4, 5 };

· 没有副作用 对于没有作用的语句会产生警告消息。例如，下面的加法操作的结果并没有用处：

```
int a = 1;
int b = 2;
a + b;
```

· 溢出 在 Fortran 中，当浮点常量声明溢出时会产生警告消息。

· 返回值 当函数被写成可以返回（或不返回）值时产生警告消息。下面的例子中，如果 x 为负将不会有返回值：

```
ambigret(int x)
{
if(x >= 0)
return(x);
}
```

· static 语法 如果关键字 static 没有用在一行声明的起始将产生警告消息。该次序已不再是标准 C 的需求了 。

· 无用参数 如果选项 -Wall 或 -Wunused 与选项 -W 一同使用，函数体中任何没有作用的参数均会引发警告消息。

该选项可以写作 --extra-warnings 。

Asm

-Wa, optionlist

optionlist 是一个或多个由逗号分隔的可以传递给汇编器的选项列表。这些选项由逗号分隔且每一个均可作为命令行选项传递给汇编器。

参见选项 -Wp 和 -Wl 。该选项也可以写作 --for-assembler 。

C C++ ObjC

-Waggregate-return

如果函数返回一个结构、联合或是数组时，产生警告消息。

-Wall

对 C 和 ObjC 而言，设置该选项与设置选项 -Wreturn-type ， -Wunused ， -Wimplicit ， -Wswitch ，-Wformat ， -Wparentheses ， -Wmissing-braces ， -Wsign-compare 和 -Wmultichar 相同。选项 -Wunknown-pragmas 仅对那些没有在系统头文件中探测到的代码进行设定。选项 -Wuninitialized 仅在指定选项 -O 时设定。

对 C++ 而言，附加的选项设置有 -Wctor-dtor-privacy ， -Wnon-virtual-dtor ， -Wreorder 和 -Wnon-template-friend 。

对 Fortran 而言，有效的选项设置仅有 -Wunused 和 -Wuninitialized 。

对 Java 而言，这与设置选项 -Wredundant-modifiers ， -Wextraneous-semicolon 和 -Wunused 相同 。

该选项也可以写作 --all-warnings 。

--warn-

与选项 -W 相同。

C

-Wbad-function-cast

无论何时函数被转换为非匹配类型时均产生警告消息。例如，下面的例子将会在函数调用时产生警告：

```
int glim()
{
```

```
return(88);
}
. . .
char *cp;
cp = (char *)glim();
```

C C++ ObjC

-Wcast-align

当指针转换为另一不同类型而引起对齐问题时产生警告消息。例如，在某些机器上仅能在 2 或 4 字节边界上访问 int 类型，因此将一个 char 指针转换为 int 指针时会产生非法地址值。

C C++ ObjC

-Wcast-qual

无论何时去掉函数调用的限定词时产生警告消息。例如，下面的例子去除了 const 限定词：

```
const char *conchp;
char *chp;
. . .
chp = (char *)conchp;
```

C C++ ObjC

-Wchar-subscripts

如果 char 数据类型被当作数组下标使用时产生警告消息。 char 通常默认是有符号的，因此这很可能是某些错误的根源。

C C++ ObjC

-Wcomment

当在 /* … */ 注释中发现 /* 时产生警告消息。而且，在 // 注释结尾发现反斜线时（这将指出下一行也是注释）也会产生警告消息。

C C++ ObjC

-Wconversion

当原型引起类型转换（而不是原型不存在时引起的类型转换）时产生警告消息。这包括在实数和整数类型间进行转换，在有符号和无符号数值间进行转换，以及改变数值的宽度。仅对隐式的转换（强制转换）给出警告，而非指定的类型转换。例如，下面例子的第一个赋值语句将引起警告，但第二句则不会：

```
unsigned int recp;
recp = -1;
recp = (unsigned int)-1;
```

C++

-Wctor-dtor-privacy

当某个类显然是不可用时产生警告，这是由于它只有私有构造函数和析构函数，没有友员函数和公用或静态成员。选项 -Wall 设置了该选项。

C++

-Wdeprecated

这是默认选项。当使用已废止的 C++ 功能时产生警告消息，除非使用了选项 -Wno-deprecated 。

C C++ ObjC

-Wdeprecated-declarations

这是默认选项。当使用已废止的属性时产生警告消息，除非使用了选项 -Wno-deprecated=declarations 。

-Wdisabled-optimization

当所请求的优化被关闭时产生警告消息。出现这一情况的原因不在于代码，而是因为编译程序自身的限制。GCC 在优化太复杂或是完成优化耗时太多时会拒绝进行优化。

-Wdiv-by-zero

这是默认选项。当编译程序检测到整数除数为 0 时产生警告消息，除非使用了选项 -Wno-div-by-zero 。当浮点除数为 0 时不产生警告消息。

C++

-Weffc++

当使用了违反由 Scott Myers 所著 *Effective C ++* 一书中给出的编码风格的代码时产生警告消息。由于标准库头文件不遵守这些原则，因此这些代码也会引起警告消息。

-Werror

将所有的警告转换为错误消息。

C

-Werror-implicit-function-declaration

当在声明函数前使用函数时产生错误。参见选项 -Wimplicit-function-declaration 。

C++

-Wextern-inline

当函数被同时声明为 extern 和 inline 时产生警告消息。

Java

-Wextraneous-semicolon

当分号没有可以终结的语句时产生警告消息。空语句已经被废止。

-Wfloat-equal

当对两个浮点数进行相等比较时会产生警告消息，因为从程序逻辑的角度而言这通常是错误的。

很显然，计算实数的浮点运算很少会产生相同的数值。这就是说比较两个实数是否完全相等可能会失效，即使这两个数非常接近以至于可以从程序的逻辑上将它们认为是相同的。如果你的程序将一个数落在另一个数的 0.00001 范围内视为相等的话，下面给出了一种比较浮点数的方法：

```
double delta = 0.00001;
. . .
if((val1 > val2-delta) && (val1 < val2+delta) {
/* val1 and val2 are considered equal */
}
```

C C++ ObjC

-Wformat

检查对诸如 printf() 和 scanf() 等函数的调用，并当命令中的参数类型与格式字串中所指定的类型不匹配时产生警告消息。例如，下面的语句试图将一个 double 值格式化为一个 int 类型时产生警告消息：

```
double dvalue = 44.44;
. . .
printf("The value %d is bad.\n",dvalue);
```

格式将按照 GNU libc 版本 2.2 的功能进行测试，该版本包含了 C89 ， C99 ， POSIX 和某些 BSD GNU 扩展中的功能。如果同时指定了选项 -pedantic ，还会对非标准部分的格式给出警告消息。

该选项所测试的带有格式字串的函数有 printf() ， fprintf() ， sprintf() ， scanf() ， fscanf() ， strftime() ， vprintf() ， vfprintf() 和 vsprintf() 。对 C99 而言则包含 snprintf() ， vsnprintf() ， vscanf() ， vfscanf() 和 vsscanf() 。对 X/Open 而言则包含 strfmon() ， printf_unlocked() 和 fprintf_unlocked() 。

参见选项 -Wformat-extra-args ， -Wformat-nonliteral 和 -Wformat-security 。参见第 4 章的 4.3.5 节“属性”。选项 -Wall 设置了该选项。

C C++ ObjC

-Wformat=2

与设置了选项 -Wformat ， -Wformat-nonliteral 和 -Wformat-security 相同。

C C++ ObjC

-Wformat-extra-args

这是默认选项。如果同时指定了选项 -Wformat ，使用选项 -Wno-format-extra-args 将禁止对诸如 printf() 和 scanf() 等函数使用额外（未使用）参数进行调用的警告消息。

C C++ ObjC

-Wformat-nonliteral

如果使用了选项 -Wformat ，而且一些函数调用（例如 printf() 和 scanf() ）所使用的格式字符串不是可用于检查的常量时，就会发出警告。

C C++ ObjC

-Wformat-security

当使用选项 -Wformat 而且对诸如 printf() 和 scanf() 的函数调用可能存在安全问题时，发出警告消息。使用变量（而非常量）作为格式化字符串进行函数调用时，由于可能使用 %n ，因此不信任这类调用。

C C++ ObjC

-Wformat-y2k

这是默认选项。使用选项 -Wno-format-y2k 将禁止对 strftime() 产生的两位数年份产生警告消息。

Fortran

-Wglobals

这是默认选项。使用选项 -Wno-globals 将禁止对子例程、函数、数据块或是具有相同特质名的通用块的全局名产生警告消息。同时还会禁止对不一致的全局函数和子例程（如不同的参数数目或不同的参数类型）给出警告消息。

Fortran

-Wimplicit

无论何时隐式地声明了变量、数组或是函数，均给出警告消息。其效果与声明了 IMPLICIT NONE 相似。

C

-Wimplicit-int

当声明并没有指令某个类型时产生警告消息。选项 -Wimplicit 和 -Wall 会设置该选项。

C

-Wimplicit-function-declaration

无论何时在声明前使用函数将产生警告消息。参见选项 -Werror-implicit-function -declarations 。选项 -Wimplicit 和 -Wall 会设置该选项。

C

-Wimplicit

与选项 -Wimplicit-int 和 -Wimplicit-function-definition 相同。选项 -Wall 会设置该选项。

C C++ ObjC

-Wimport

这是默认选项。使用选项 -Wno-import 将会禁止由预处理程序对使用 #import 产生的警告消息。

C C++ ObjC

-Winline

当函数被声明为 inline 但不能被扩展为内嵌代码时产生警告消息。

Linker

-Wl, optionlist

optionlist 是由一个或多个逗号分隔的传递给链接器的选项列表。这些选项由逗号分隔且每一个选项均会作为命令行选项提供给链接器。

参见选项 -Xlinker ， -Wa 和 -Wp 。

C C++ ObjC Java

-Wlarger-than- size

无论何时目标代码大于所声明的 size 字节时产生警告消息，如果函数返回值比 size 字节大，也会产生警告消息。

C C++ ObjC

-Wlong-long

这是默认选项，但仅当同时使用了选项 -pedantic 时才会有效。当使用 long long 数据类型时该选项产生警告消息。可以使用 -Wno-long-long 关闭这一默认选项。

C C++

-Wmain

当 main() 定义存在可疑之处时产生警告消息。 main() 应该是一个外部可链接且返回一个 int 的函数。它应该具有 0 个、 2 个或 3 个正确类型的参数。

C C++ ObjC

-Wmissing-braces

当数组的初始值没有完全被括号包含时产生警告消息。在下面的例子中，数组 a 和 b 均初始化正确，但数组 b 的括号更准确地表达了数值的位置：

```
int a[2][2] = { 0, 1, 2, 3 };
int b[2][2] = { { 1, 2 }, { 3, 4} };
```

选项 -Wall 会设置该选项。

C

-Wmissing-declarations

当定义了全局函数但没有在之前进行声明，或者没有指定参数类型，此时产生警告消息。参见选项 -Wstrict-prototypes 和 -Wmissing-prototypes 。

C C++ ObjC

-Wmissing-format-attribute

针对 format 属性的候选函数产生警告消息。需要注意的是，对可能的 noreturn 候选函数产生警告消息。如果没有使用选项 -Wformat 或 -Wall ，该选项无效。

C C++ ObjC

-Wmissing-noreturn

针对可能用于 noreturn 属性的函数产生警告消息。需要注意的是，对可能的 noreturn 候选函数产生警告消息。特别要注意那些实际有返回值的函数，由于可能会在程序中引起很微妙的错误，决不能将它们声明为 noreturn 属性。

C

-Wmissing-prototypes

当定义了全局函数但没有在之前给出带有参数类型的原型声明时，产生警告消息。参见选项 -Wstrict-prototypes 和 -Wmissing-declarations 。

C C++ ObjC

-Wmultichar

这是默认选项。当字符常量被声明为包含多于一个字符时，如‘ ab '或‘ Plop '，产生警告消息。由这类声明产生的代码会依赖于所使用的平台，不能在可移植的代码中使用，但可使用选项 -Wno-multichar 关闭警告消息。

C

-Wnested-externs

对函数体内的 extern 声明产生警告消息 。

C++

-Wnon-template-friend

这是默认选项。当非模板化友元函数被声明为模板的成员时产生警告消息。

在 C++ 标准定义之前， GNU C++ 将友元的名字实现为一个非限定的 id 。默认情况下已不再是这样的行为了，但为了已有的代码，可以使用选项 -Wno-non-template-friend 关闭该警告消息。选项 -Wall 会设置该选项。

C++

-Wnon-virtual-dtor

当非虚析构函数应该定义为虚函数时产生警告消息。如果某个对象被作为其超类引用并被释放时，非虚析构函数将无法被执行。选项 -Wall 会设置该选项。

C++

-Wold-style-cast

当使用传统风格（ C 语言风格）进行类型转换（而非 C++ 标准中定义的新形式之一，该新形式带有类型转换操作符 static_cast ， const_cast 或 reinterpret_cast ）时，产生警告消息。例如：

```
class A { ... };
class B: public A { ... };
. . .
A* a = new A();
B* b = a; //implicit conversion
A* a2 = static_cast <A*> (b); //reversing an implicit conversion
```

Java

-Wout-of-date

这是默认选项。使用选项 -Wno-out-of-date 将禁止编译程序在源文件比其对应的类文件新时产生警告消息。

C++

-Woverloaded-virtual

当函数声明隐藏了基类中声明的虚函数时产生警告消息。在下面的例子中，类 A 中的虚函数 fn() 被隐藏了：

```
class A {
virtual void fn();
};
class B: public A {
void fn(int);
};
```

Pre

-Wp, optionlist

optionlist 是由一个或多个逗号分隔的传递给预处理程序的选项列表。这些选项由逗号分隔，且每一个选项均被作为命令行选项传递给预处理程序。

参见选项 -Wa 和 -Wl 。

C C++ ObjC

-Wpacked

当使用了 packed 属性但却没有作用时产生警告消息。例如，下面的 struct 在没有 packed 属性的时候也会使用 4 个字节：

```
struct fourbyte {
short x;
char a;
char b;
}__attribute__((packed));
```

参见第 4 章的 4.3.5 节“属性”。参见选项 -fpack-struct 和 -Wpadded 。

C C++ ObjC

-Wpadded

当编译程序在结构的成员间插入填充字，以保证某一项在结构内部对齐或使整个结构对齐时，产生警告消息。在某些情况下，可以通过重新组织结构来对齐成员，而且可通过去除填充字来达到减少结构尺寸的目的。例如，下面的结构将在 short 数据类型前插入一个字节来保证它在 2 字节边界上对齐：

```
struct pad {
char a;
short b;
char c;
};
```

C C++ ObjC

-Wparentheses

对那些语法上虽然正确，但由于操作符的优先级或代码的结构而引起程序员混淆的构建子产生警告消息。

下面的表达式将产生一条警告消息，这是因为很难记住逻辑操作符是从左向右还是从右向左计算的：

```
if(a && b || c) . . .
```

下面的代码会引发一条警告消息，这是因为没有括号的话， if 和 else 语句间的关系很容易被误解：

```
if(a)
if(b)
m = p;
else
a = 0;
```

很显然，从缩进来说，程序很可能将 else 语句和第一个 if 语句联系在一起，但事实正相反。

选项 -Wall 会设置该选项。

C++

-Wpmf-conversions

这是默认选项。使用选项 -Wno-pmf-conversion 将引起一条警告消息，指出存在一个从指针到成员函数地址的转换。

C C++ ObjC

-Wpointer-arith

针对那些依赖于函数类型大小或 void 大小的代码产生警告消息。由于指针运算的缘故，这些项在 GCC 中的默认大小是 1 。

ObjC

-Wprotocol

这是默认选项。当协议所需要的方法未在采纳该协议的类中实现时，指定 -Wno-protocol 将禁止产生警告消息。

C C++ ObjC

-Wredundant-decls

对在同一作用域中声明多次的内容产生警告消息。即使当声明是完全一样的也会产生警告消息。

Java

-Wredundant-modifiers

当在声明中使用了不必要的修饰词时会产生警告消息。例如，当某 interface 的方法被声明为 public 时会产生一个警告消息。

C++

-Wreorder

当编译程序重组成员初始化函数以匹配它们所声明的顺序时产生警告消息。例如，下面的初始化函数必须进行重组：

```
class Reo {
int i;
int j;
Reo(): j(5), i(10) { }
};
```

选项 -Wall 会设置该选项。

C C++

-Wreturn-type

当所有声明的函数没有声明返回类型，并默认使用 int 时产生警告消息。而且，如果在没有声明为 void 的函数中，返回语句没有返回值时会产生警告消息。选项 -Wall 会设置该选项。

Pre

--write-dependencies

与选项 -MD 相同。

Pre

--write-user-dependencies

与选项 -MMD 相同。

ObjC

-Wselector

如果选择子为不同类型定义了多个方法时产生警告消息。

C

-Wsequence-points

如果变量在表达式中被多次引用，且某次引用修改了它的值，则会产生警告消息。 C 语言的定义允许按照任意顺序计算顺序点间的表达式（只要保持运算符的优先级），因此在某处对变量进行修改将使在其他位置需要使用变量时，无法确定其值。

可以通过如下一些操作符指定代码中的顺序点：

; , && || ? :

下面是表达式的一些例子，由于违背了顺序点，因而具有不惟一的结果：

```
s = a[s++];
s = s--;
a[s++] = b[s];
a[s] = b[s += c];
```

C C++ ObjC Java

-Wshadow

当局部变量屏蔽了参数、全局变量或是其他局部变量时产生警告消息。如果内建函数被屏蔽，也产生警告消息。

C C++ ObjC

-Wsign-compare

当比较有符号值和无符号值，可能因在比较前将有符号值转换为无符号值而产生不正确结果时，产生警告消息。选项 -Wall 会设置该选项，但可以使用选项 -Wno-sign-compare 关闭该选项。

C++

-Wsign-promo

当从无符号数据类型（或枚举类型）向具有相同尺寸的有符号数据类型进行重载时产生警告消息。标准定义了这种类型的转换，但它可能会造成数据丢失。

C

-Wstrict-prototypes

当函数的定义或声明没有指明参数的数据类型和数目时，就会产生警告消息。参见选项 -Wmissing-prototypes 和 -Wmissing-declarations 。

Fortran

-Wsurprising

该选项将对可能有多种解释方法的构建子给出警告消息，这种情况可能会使程序员感到非常困惑。不同的编译程序对语言的构建子有不同的处理方式。这些警告消息包括：

· 在同一行中使用了两个运算符的表达式。这种情况的一个例子就是 x** y * z 。在没有括号的情况下，编译程序可能将该表达式解释为 x** (y * z) 或 ((x**y) * z 。选项 -fpedantic 也可能对这种情况发出警告消息。

· 带有模糊的一元负号的表达式。例如， -2 ** x 可以解释为 -(2 ** x) 或 (-2) ** x 。像 -x*y 这样的表达式，如果某个值接近 x 或 y 数据类型值域的最大范围时可能会产生令人吃惊的结果，表达式可能被解释为 -(x*y) 而非 (-x)*y 。

· 使用实数而非整数作为循环计数器的 DO 循环可能会产生奇怪的结果。通常这并不是问题，但结果可能会依编译程序的不同而不同。

C C++ ObjC

-Wswitch

当使用某个枚举类型作为 switch 语句的索引但又没有 default 和 case 语句处理所有可能值时，产生警告消息。选项 -Wall 设置该选项。

C++

-Wsynth

当操作符的合成与 cfront 不同时产生警告消息。在下面的例子中， GCC 合成操作符 A& operator = (const A&) ；而 cfront 则使用用户定义的默认 operator = 操作符。

```
class A {
operator int();
A& operator = (int);
};

main() {
A a1;
A a2;
a1 = a2;
}
```

C C++ ObjC

-Wsystem-headers

对系统头文件中的代码发出警告，该代码同时也在被编译程序中。通常禁止产生系统头文件产生的警告消息。

为了对系统头文件中未知 pragma 产生警告消息，需要使用选项 -Wunknown-pragmas ，这是因为使用选项 -Wall 仅会检查程序中的 pragma 而忽略系统头文件中的 pragma 。

-Wtraditional

为标准 C 构建子发出警告，这里的构建具有不同的含义或在传统 C 中不存在。其中一些构建子含义模糊或是很容易产生问题，因此应该尽可能避免。

· 转换 由于原型定义而引起定点和浮点数之间的转换时，产生警告消息。参见选项 -Wconversion 。例如，如下的函数调用可能会引起警告消息：

```
void takedouble(double dval);
. . .
int eighty = 80;
takedouble(eighty);
```

· 外部 如果在代码块内部将某函数声明为外部函数，且在该代码块的外部使用了该函数时，产生警告消息。

· 初始值 当对自动聚集数据类型（如数组和在函数体内声明的结构）进行声明时，产生警告消息。

· 初始值 当对联合赋予初始值时产生警告消息，除非它的初始值为 0 。

· 标号 当标号与变量具有相同名字的时候产生警告消息。

· 文字常量 当使用前缀 U 声明整数常量，并用前缀 F 和 L 声明实数常量时，产生警告消息。

· 文字常量 如果基数为 10 的 C 常量具有不同的宽度，或具有与传统 C 不同的有符号 / 无符号特性时，给出警告消息。仅对基数为 10 的常量给出警告消息，这是因为通常十六进制和八进制的常量均视为位模式。

· 文字常量 当使用标准 C 中的字串连接方法时产生警告消息。

· 预处理程序 当所命名的宏作为字符常量的一部分出现时产生警告消息。传统 C 中的字符串可以包含宏定义，但标准 C 中则不允许。

· 预处理程序 当预处理程序指示字第一列对传统 C 未知时，产生警告消息。可以通过缩进使用新型的预处理程序命令，如 #pragma 和 #elif ，从而使它们不出现在第一列中。（传统预处理程序要求所有的预处理程序命令均在第一列开始。）

· 预处理程序 当定义为函数形式的宏却没有参数时产生警告消息。

· 静态 如果非 static 函数在一个 static 函数后被声明时产生警告消息。（某些传统编译程序不接受这样的情况。）

· 开关语句 当 switch 语句的操作数是 long 数据类型时产生警告消息。

· 一元加号 当存在一元加号时产生警告消息。

C

-Wtrigraphs

当某个三字母词可能改变程序的含义时，产生警告消息。除了注释中的三字母词不被翻译以外，其他各处的三字母词均被翻译，因而可能某个三字母词会出现在字符串中。例如， Linux 内核的某个版本就包含这样一个字符串 "imm: parity error (???)\n" ，在标准 C 中它将被翻译为 "imm: parity error (?]\n" 。

C C++ ObjC

-Wundef

当没有定义的标识符出现在 #if 命令的表达式中时，产生警告消息。

-Wuninitialized

如果自动变量在初始化之前就被使用，产生警告消息。而且，如果 setjmp() 调用可能破坏自动变量值的时候也会产生警告消息。该选项只能同选项 -O 一同使用，因为这些情况的探测需要使用优化数据流的信息。

由于该选项要求数据流分析，因此不可能完全准确。例如，在下面的代码中，对 printf() 语句而言，不可能保证 value 已经初始化或未经初始化。

```
int value;
if(a b)
value = 5;
else if(a c)
value = 10;
printf("%d\n",value);
```

由于数据流分析的本质，该选项不能作用于结构、联合、数组、所有声明为 volatile 的变量、所有通过地址引用的变量或用来计算从不使用的数值的变量。

数据流分析可以理解 setjmp() 语句，但它不可能知道在哪里会调用 longjmp() ，因此即使没有任何问题也可能会产生警告消息。

Fortran 源代码需要某种数据流分析，如同下面的例子所示，可能无法确定 TVAL 是否总是被初始化了：

```
IF (IVAL .EQ. 1) TVAL = 5
IF (IVAL .EQ. 2) TVAL = 10
CALL SMON(TVAL)
```

该选项会报告一些假的警告消息，可以通过将函数声明为不返回值 noreturn 来禁止这种警告消息。

如果与选项 -O 一同使用，选项 -Wall 将设置该选项。

-Wunknown-pragmas

当遇到未知的 #pragma 指示字时产生警告消息。除非已由选项 -Wall 设置，不然该选项将对系统头文件中任何未知的 #pragma 指示字产生警告消息。

-Wunreachable-code

如果在当前的程序执行中检测到不可达代码，则产生警告消息。如果代码存在永远被选取的分支，或是调用了不会返回的函数时就存在不可达代码。

应该谨慎地去除由该警告所报告的代码。很可能从可以产生内嵌代码的函数或是展开的宏中产生警告消息，但也有其他实例不产生不可达代码。而且，不可达代码也可能是编译时的选项导致的内部可忽略代码。

-Wunused

该选项设置选项 -Wunused-function ， -Wunused-label ， -Wunused-parameter ， -Wunused-value 和 -Wunused-variable 。

对 Fortran 而言，当变量被声明但没有使用时产生警告消息。

选项 -Wall 会设置该选项。

-Wunused-function

当存在一个未使用的静态函数的定义或是只声明但没有定义的静态函数时，产生警告消息。选项 -Wunused 和 -Wall 将设置该选项。

-Wunused-label

当定义了未使用的标号而没有使用 unused 属性时，产生警告消息。选项 -Wunused 和 -Wall 也会设置该选项。

-Wunused-parameter

当声明了未使用的参数而没有使用 unused 属性时，产生警告消息。选项 -Wunused 和 -Wall 也会设置该选项。

-Wunused-value

当存在未使用的局部变量，或声明了非常量的静态变量而没有使用 unused 属性时，产生警告消息。选项 -Wunused 和 -Wall 也会设置该选项。

-Wunused-variable

当存在未使用的局部变量，或声明了非常量的静态变量而没有使用 unused 属性时，产生警告消息。选项 -Wunused 和 -Wall 也会设置该选项。

C C++

-Wwrite-strings

当 C 程序在声明为 const 的指针中保存了某字符串常量的地址时，产生警告消息。当 C++ 将一个字符串常量转换为 char * 类型时，产生警告消息。

仅当需要在代码中谨慎地使用 const 来声明数据类型和原型时，该选项才会有用，否则所产生的警告消息是非常烦人的。

-x language

指定命令行中所指文件的内容。如果不使用该选项，文件的内容一般通过文件名中的后缀确定。该选项作用于命令行中跟在其后的所有文件名。在下面的例子中，名为 morg.jmp 和 frampl 的文件均视为 C 源代码文件。

gcc -xc morg.jmp frampl

可以多次使用该选项来改变所需语言，而且还可以使用特殊名 none 来关闭该选项。例如，下面的命令行 指定名为 murk 和 stim.wad 的文件均为 C++ 源代码，名为 hummer.c 的文件为 Java 源代码，而 slamm.c 则默认为 C 源代码。

gcc -xc++ murk stim.wad -xjava hummer.c -xnone slamm.c

选项 -x 也可以写作 --language 。参见表 D-7 。

表 D-7 -x 选项的语言指示字

语言名	描述
ada	Ada 源代码
assembler	不需要预处理的汇编语言
assembler-with-cpp	需要预处理的汇编语言
c	需要预处理的 C 源代码
c++	需要预处理的 C++ 源代码
c++-cpp-output	不需要预处理的 C++ 源代码
c-header	C 头文件
cpp-output	不需要预处理的 C 源代码
f77	不需要预处理的 Fortran 源代码
f77-cpp-input	需要预处理的 Fortran 源代码
java	Java 源代码
objc-cpp-output	不需要预处理的 Objective C 源代码
objective-c	需要预处理的 Objective C 源代码
ratfor	需要由 RATFOR 预处理程序处理的 Fortran 源代码

Linker

-Xlinker option

向链接器传递一个选项。这主要用来指定与系统相关的链接器选项。

例如，如果你正在使用 System V 链接器而且需要对其使用选项 -all ，可以使用选项 -Xlinker-all 完成这一任务。在链接器命令行中，指定多个选项时应当多次重复使用 -Xlinker 。例如，要指定选项 -woff 5,17 ，可能需要使用选项 -Xlinker -woff -Xlinker 5,17 。如写成 -Xlinker "-woff 5,17" 则无法工作。

参见选项 -Wl 。该选项也可以被写作 --for-linker 。

附录E 术语表

absolute address （绝对地址）

绝对地址是一个惟一的可用来指定某个内存字节的数字数值。参见相对地址。

address （地址）

参见绝对地址和相对地址。

aggregate （聚集）

包含多于一个基本数据类型的数据类型。例如，数组是一种聚集， C 的结构（ struct ）也是。

aliasing （别名机制）

同一内存地址可直接或间接地通过两个或多个不同名字（可能还有不同的数据结构）进行访问，这被叫做别名。由于在寄存器中保留数值是比较常见的做法，因此这是对于优化采取的特殊考虑。

ANSI （ American National Standards Institute ，美国国家标准协会）

管理和协调美国自发标准的组织。

archive （文档）

参见库。

assembler （汇编器）

一个平台相关的程序，它读取汇编语言源文件（机器代码的助记表示）并将其翻译为作为连接程序输入的二进制目标文件。

backtrace （回退跟踪）

GNU 调试器可以打印函数名和地址的列表，它会被调用以达到程序的当前执行点。这些信息，包括函数地址和参数值，称为回退跟踪。

BFD （ Binary File Descriptor ， 二进制文件描述字 ）

包含处理各种二进制文件格式完成各种底层操作的例程的库。

BSD （ Berkeley Software/Standard Distribution ， Berkeley 软件 / 标准发行）

一种 UNIX 操作系统。它也是几种其他现代 UNIX 系统的基础。参见 SVR4 。

bss

由 UNIX 连接程序生成的可执行文件的未初始化数据段。它包含的数据只有地址但不包含任何空间。因此，直到程序被加载时才会分配空间。在可执行文件中， bss 变量只被赋以名字、尺寸和位置。参见 text 和 data 。

built in function （内嵌函数）

由编译程序所生成的函数体称为内嵌函数。一个内嵌函数可以是相应标准库函数的优化版本、编译程序的附加功能或是实现诸如变长参数列表等内部使用的函数。

bytecode （字节码）

编译 Java 程序所生成的目标代码的可移植形式。字节码由 Java 虚拟机解释以执行 Java 程序。

C89

1989 ANSI C 标准。

C99

1999 ANSI C 标准。

calling convention （调用规范）

参见调用序列。

calling sequence （调用序列）

用来调用函数的汇编语言的语句序列。该序列设置需要传递的参数、保存返回的地址以保证可以找到调用函数，执行调用并管理返回值（如果有的话）。也叫做调用规范。

CCP （ Conditional Code Propagation ， 条件代码传播）

一种优化技术，可以发现某值对所有可能的执行路径均为常量，并用这一事实来探测和删除不可能被执行的代码。

cfront

最初的 C++ 实现是一个称为 cfront 的 AT&T 程序，它将 C++ 源代码翻译为 C 源代码。

class （类）

1. 在面向对象编程中，一个类就是一个对象类型的定义。由它所生成的对象，由于它们具有相同的接口和行为集，称它们属于同一类。 2. 在 Java 中，包含已编译类的文件称为类文件，或简称为类。

clobber （破坏）

如果存储位置（通常为一个寄存器）已经被用作一个临时工作区，它就不再保存所预期的数值，这个存储位置就称为被破坏（ clobbered ）了。

CNI （ Cygnus Native Interface ， Cygnus 本地接口）

在 C++ 中，用于编写本地 Java 方法的工具。参见 JNI 。

code （代码）

这个术语是指计算机上运行的任何形式的指令列表。代码可以是所有形式，从人可读的编程源码到机器可读的操作码的位模式。

code propagation （代码传播）

参见 CPP 。

coercion （自动转换）

由一种基础数据类型到另一种的自动转换（没有强制类型转换或函数调用）。

COFF （ Common Object File Format ，通用目标文件格式）

一种可跨系统移植的并由各种不同的汇编器和连接程序所了解的目标文件的标准格式。参见 ECOFF 和 XCOFF 。

COMDAT （ Common Data ，通用数据）

可以在不只一个目标文件中被复制的一种数据或可执行项（或项目的集合）。当将目标

文件和库或可执行体组合起来时，连接程序会去掉所有通用数据，只留下其中一份数据。这也称为合拢或通用数据合拢。

common （通用）

可为公用块分配的全局变量属性。

common block （公用块）

GNU 连接程序创建一个公用块作为全局变量的分配空间。如果不同的目标文件多次声明了相同的全局变量，它们会被解析为公用块中单独的一个变量。参见 COMDAT 。

compilation unit （编译单元）

可被编译为单独目标文件的单独源代码单位。它通常就是单独的一个源文件，但也可以包含其他辅助编译的源代码（如 C 程序中 #include 文件）。也称为翻译单元。

compiler （编译程序）

读取计算机程序的源文件（或文本文件）的软件集合，可将指令翻译成计算机可执行的格式。编译程序也称为翻译器（ translator ）。

copyleft （版权所属）

同通用许可证，它声明程序是自由软件并且所有修改和扩展的软件版本均可以自由发行。参见 GPL 。

cross compile （交叉编译）

使用一种编译程序创建在完全不同平台上执行的可执行文件。

CPP （ C Preprocessor ， C 预处理程序）

预处理程序读取程序源码，并处理其中的指示字以产生源码的修改版本。

cruft

随着软件的发展，以及经历了修改 bug 和更新的若干周期，它的部分代码已不再使用但仍然保留在源码中。这种代码称为 cruft 。 cruft 的尺寸范围可由一两行无用代码到整个源文件模块。由于很难识别 cruft ，去除 cruft 往往很困难。

CSE （ Common Subexpression Elimination ， 通用子表达式消除）

一种优化技术，它可以识别重复的表达式并重用它的值，而不是再次执行相应的计算。参见 GCSE 。

ctor

构造函数 constructor 的通用缩写。参见 dtor 。

CVS （ Concurrent Version System ， 同步版本系统）

一种版本控制系统，它可以维护文本文件的版本历史信息。设计它是为使广泛分布于各地的不同开发者可同时访问。

data

由 UNIX 连接程序生成的可执行文件的一部分，其中包含的数据具有初始值。该段包含的项具有名字、大小，且会分配相应的空间包含它的数值。参见 bss 和 text 。

DBX

一种交互式调试器，它可以用来一行行地跟踪程序的执行。 DBX 是一种命令行调试器，但在许多变体中它具有 X GUI 界面和 emacs 界面。

DCE （ Dead Code Elimination ， 不可达代码去除 ）

一种删除不可能被执行的代码的优化技术。

dead code （不可达代码）

在优化期间，可能留下某些代码但从不会被执行。这些就是不可达代码。优化器应该去除这些代码。

demangle

解析出编码在已经拆分（ mangle ）了的函数名中的描述信息的过程。参见 mangle 。

deprecated （已废止的）

任何不再需要的编译程序的选项或功能（或是因为某些原因而认为不再恰当的）被称为是已废止的。它仍然存在但在以后的编译程序版本中可能去除。

dereference （去引用）

表达式可能会涉及保存在指针中的地址。用这种方式使用的指针就称为是去引用的。

directive （指示字）

1. 源代码中使用井号 (#) 开头的命令，它由预处理程序处理。 2. 在汇编语言中，一条指示字就是一条针对汇编器的指令，而不是产生代码的操作码。汇编器指示字也称为伪操作。

distention （膨胀）

该名字用于某些特定 Fortran 语言扩展，现在均认为非常“丑陋”而不该使用。通过 -fugly-* 标记， g77 可支持某些扩展。

dtor

析构函数 destructor 的通用缩写。参见 ctor 。

DWARF （ Debugging With Attribute Format ，带属性的调试格式）

一种用来在目标代码中插入调试信息的格式。

DWARF2 （ Debugging With Attribute Format 2 ，带属性的调试格式 2 ）

DWARF 的更新版本，该格式用来在目标代码中插入调试信息。

dynamic library （动态库）

参见库。

ECOFF （ Extended Common Object File Format ， 扩展通用目标文件格式）

目标文件的标准格式，它具备系统间的可移植性，且不同汇编器和连接程序均了解该格式使用。参见 COFF 和 XCOFF 。

EH

异常处理（ exception handling ）的缩写。

elaborate （详细阐述）

在 Ada 中，执行前的最后一步是通过插入必要的初始值和可执行指令来详细阐述代码，通常需要同一程序中其他编译单元的内容。参见可预阐述（ preelaborable ）。

elaboration （详细阐述）

在 Ada 语言中，详细阐述是所需的详细阐述软件包的过程。

ELF （ Executable and Linkable Format ， 可执行和可链接格式）

Linux 二进制目标文件格式，它包含动态加载库和可执行代码的信息。 ELF 由 COFF 格式演变而来，与 COFF 非常近似。

elide （省略）

一个被省略的函数就是使用调用者返回位置作为工作空间（而非建立自己的内部工作空间）返回数值的函数，之后它会将返回值复制到返回地址中。优化器可以省略函数调用。

entry point （入口点）

可执行程序中的地址，程序从这里开始执行并称为入口点。

fetch （读取）

当计算机由内存中的程序读取指令并加载到 CPU 执行时，这称为指令的读取。参见预取（ prefetch ）。

FPU （ Floating Point Unit ， 浮点处理单元）

同 CPU 一起工作处理浮点操作的硬件处理器。没有 FPU 的计算机将需要软件来仿真浮点操作。

folding （合拢）

参见 COMDAT 。

frame （栈）

参见堆栈框架。

function （函数）

函数是一块可执行代码，它被赋予一个名字并可以在其他位置调用。一个函数也可以被定义为带有参数，它指定一列由调用者传递的参数值。参见成员函数和方法。

garbage collection （无用信息收集）

一个运行的程序的过程，它可以自动恢复动态分配但不再使用的内存。实现该过程有许多机制，但所有这些均可以被认为是无用信息收集。

GCSE （ Global Common Subexpression Elimination ，全局公用子表达式去除）

一种优化技术，它可以识别重复的表达式并重用它的数值，而不是再次执行相应的计算。参见 CSE 。

GNAT （ GNU Ada Translator ， GNU Ada 翻译器）

原 Ada 前端的名字，现已成为 GCC 的一部分。

GNATS （ GNU Bug Tracking System ， GNU 错误跟踪系统）

用来跟踪 GCC 和其他 GNU 软件错误的在线系统。

GOT （ Global Offset Table ，全局偏移表）

目标文件中包含一组偏移的表格，可用来重新定位可执行代码。参见 PIC 。

GPL （ General Public License ，通用公共许可证）

一种许可证，在其许可下软件被编写为 copyleft 格式的自由软件。

header （头文件）

由预处理程序执行 include 指示字并被包含在源文本文件中的文件。在 C ， C++ 和 Objective C 中，传统上它是后缀为 .h 的文件。

Hollerith field （ Hollerith 字段）

在 Fortran 中，引用的字串被保存为字符数，后面跟着的是字符本身。可以用 Hollerith 字段创建字符串，它是一个长度后跟一个 H 字母以及字符串（例如， 10HPhillips66 ）。

host （主机）

参见平台。

i18n

单词 internationalization 的简写形式，它由字母 i 开头，后跟 18 个字符，最后是字母 n 。参见 l10n 。

if-conversion （ if- 转换）

一种优化过程，它会修改已生成的代码，以便分支之后的最可能被采用的路径比不可能采用的路径更有效。

immediate （立即数）

一个立即数值是一个常量，可以在汇编语言操作码中作为操作数使用。

include guard （包含文件防护）

习惯上使用预处理程序条件编译命令来定义一个环境变量，它可以在头文件的最先部分进行检测，以防止头文件被再次编译。如果变量尚未定义，头文件就会被编译了。

induction variable （归纳变量）

一种在循环体内递增的变量。循环计数器。

inline （内联）

一个函数的实体（或其他相似的语言成分）被包含在函数的调用点。就是说函数体将在代码中内嵌展开，而不是只作为对另一处函数体的一次调用。

insn

一种机器语言或 RTL 中间语言指令。该语言具有许多特殊用途的 insn ，但最重要的一些构成了一种元汇编语言，可被翻译为目标机器上的汇编语言指令。

instantiation （实例化）

由类定义创建对象实例。

intrinsic （特质）

在 Fortran 中，特质函数是一种内嵌函数，由于它可以不声明而直接使用，因此被视为语言的一部分。

invariant expression （不变表达式）

循环体内的表达式，每次循环时所计算的数值是不变量，在优化时可以转移到循环外。

ISO （ International Organization or Standardization ，国际标准化组织）

一个成立于 1946 年的国际标准组织， ISO 的成员来自 75 个国家的标准组织，包括 ANSI 。

jar

一种包含一个或多个 Java 类文件的档案文件。该文件还包含一个清单，是一个包含 jar 文件中类名列表的文本文件。

Java Virtual Machine （ Java 虚拟机）

参见 JVM 。

JNI （ Java Native Interface ， Java 本地接口）

一种标准编程接口，用于编写 Java 本地方法和将 JVM 嵌入到本地应用中。参见 CNI 。

JVM （ Java Virtual Machine ， Java 虚拟机）

一个程序，它可从标准 Java 字节码格式的文件中读取指令并执行。向一个平台上移植 JVM 和标准类库实际上就是向该平台移植了所有的 Java 程序。

l10n

单词 localization 的简写，其起始字母是 l ，其后是 10 个字母，结尾字母是 n 。参见 i18n 。

lexical analysis （词法分析）

也称为词法扫描，词法分析就是从程序源文件中读取输入字符流，并按照某种方式组合它们，形成名字、数字以及标点。由一些字符集合构成的单元称为一个 token 。

LGPL （ Lesser General Public License ，次要的通用公共许可证）

该许可证可用于某些（但不是全部） GNU 库。该许可证允许库函数用于某些私有程序，通常这是 GPL 所不允许的。

library （库）

库是一个包含一个或多个目标文件的文件，它可以链接到其他目标文件组成可执行文件。静态库就是包含由连接程序永久链接到可执行程序的模块的库。静态库也称为文案。共享库就是包含可以临时链接到可执行程序并在执行时进行真正链接的模块的库。共享库也称为动态库。动态链接的程序就是包含到这些函数的引用的可执行程序，当程序执行时，它会从动态库中加载这些函数。参见静态库和相对地址。

life analysis （生命周期分析）

确定哪个数值应该留在寄存器中可供后用的过程，同时也确定哪个寄存器不会被使用，因为其中包含的数值不再使用。

link editor （链接编辑器）

与连接程序相同。

linkage （链接）

当调用函数时，必须用标准方法来存储和读取传递给函数的参数以及函数所返回的数值。这种协议称为链接，而且是混合两种语言时要解决的主要问题。

linker （链接程序）

一种平台相关的程序，可将一组目标文件（某些必须由库中析取）组合在一起生成一个可执行程序。

lvalue （左值）

一种任意类型的表达式，可以解析为内存单元的某个地址。这个术语源于 left value ，就是指赋值语句中左边的部分的值。参见右值（ rvalue ）。

macro （宏）

在预处理程序中，使用 #define 指示字声明的名字及其值就是一个宏，之后会用它来替代源码中的文本。

Makefile （ make 程序的描述文件）

包含一组由 make 命令编译和连接程序用到的规则的文件，它也会根据文件的日期和时间戳执行一些其他任务。 makefile 通常被命名为 makefile 或 Makefile 。

mangle

C++ （和 Java ）编译程序修改成员函数（以及方法）的名字，来依据其参数个数和类型生成一个惟一名。这一称为 mangling 的过程允许重载简单成员函数（和方法）。参见 demangle 。

manifest （清单）

参见 jar 。

marshaling （列集）

将传递给远程函数的参数以及由远程函数返回的值进行串行化（转变为可以发送的字节流）的动作，这称为列集。将一个规整化的流转化为数据的过程称为反列集（ unmarshaling ）。参见串行化。

member function （成员函数）

在 C++ 中，在类中定义的函数是类的一个成员并称为成员函数。除非成员函数被声明为静态的，否则成员函数永远会在该类的某个对象的上下文中被调用并具有 this 指针，它可以在函数中引用该对象。 C++ 中的成员函数类似于 Java 中的方法。

method （方法）

Java 中的方法就是类中定义的函数，它是类定义的一部分。除非方法被声明为静态的，否则方法始终会在类的对象的上下文中被调用并且具有 this 指针，它可以在函数中引用该对象。 Java 中的方法就如同 C++ 中的成员函数。

MFC （ Microsoft Foundation Class ，微软基础类）

包装 Win32 用户接口 API 的类的层次。

mirror （镜像）

复制其他 Internet 站点的站点，可以使下载文件更方便。

mnemonic （助记符）

代表一个值的名字，目的是使其更容易记忆。这个术语通常用来指可由汇编器识别的 CPU 操作码，它可以转换为二进制代码。参见操作码。

multilib

如果单个目标机需要多于一种版本的库，这就是 multilib 。例如，某平台可能具有浮点运算单元或没有，因此同样的数学库就需要使用不同选项设置编译两次，并使其可以在相应的配置下链接相应的程序。

NaN （ Not a Number ，非数值）

专指非法浮点数的 IEEE 标准名词。在下溢、溢出或某些其他非法浮点操作时会产生这一数值。

NEXTSTEP

一种计算机操作环境，它提供 GUI 界面并可以用于 HP ， NeXT ， Sun 和其他计算机上。最初该环境是为 NeXT 计算机系统开发的。

NLM （ Netware Loadable Module ， Netware 可加载模块）

可执行程序，它已被格式化为可以在 NetWare 系统上执行。

NLS （ Native Language Support ，本地语言支持）

GCC 编译程序的一种能力，它可以以本地语言（而非美国英语）输出诊断消息。 NLS 是 i18n 和 l10n 的组合。

noop （空操作）

汇编语言指令，但它什么也不做。它通常用来作为填充字插入到可执行代码中，或用来作为分支目标处的指令。

object （对象）

1. 在面向对象编程中，对象就是数据项的集合，以及用于操作这些数据项的方法（或函数）。参见类。 2. 编译过程的输出是目标文件（或目标代码），因为运行编译程序或连接程序的目的就是生成这种格式的文件。

opcode （操作码）

计算机 CPU 的一条指令。操作码可以是一条指令，能够完成两个数的相加、向寄存器加载一个数、在内存中保存一个数值，或是硬件知道如何操作的其他指令。操作码是机器指令的一部分，不包括数据。参见助记符。

ordered comparison （有序比较）

两个浮点数之间的有序比较就是，当其中一个值为 NaN 时会产生异常的比较。参见无序比较。

package （包）

Java 中的包就是类的集合。而 Ada 中的包是过程的集合。

pass （阶段）

软件每次读取输入，进行分析、优化、代码生成、预处理或其他操作就称为一个阶段。一个阶段可为下一阶段修改输入，或者仅生成表格。

peephole optimization （窥孔优化）

一种优化技术，它仅检查相连的几条指令，以确定它们是否可由一组已改进的指令替换。

PIC （ Position Independent Code ，位置独立代码）

适合用于共享库的代码，因为它可被保存在内存中的任何一个位置并从那里开始执行。所有的内部地址都是相对于内部偏移，或是对全局表的引用。见 GOT 。

platform （平台）

某种特定计算机硬件和操作系统的组合。编译程序必须被配置为可在某特定的操作系统和某特定的硬件计算机上运行。平台也称为主机或目标机。

PMF （ Pointer to Member Function ，成员函数指针）

C++ 中一种特殊数据类型，它可以保存到某个对象的成员函数的地址。

POSIX

由 IEEE 标准和 Open Group 的 Single UNIX 规范融和的 UNIX 标准规范。

pragma

一种在源代码中插入的命令，它是和编译程序相关的信息，并可由不理解该信息的编译程序忽略。

preelaborable （可预阐述）

如果 Ada 编译单元被描述为不需要其他单元提供信息，就以一种独立方式进行描述并称其为可预阐述的。参见详细阐述（ elaborate ）。

prefetch （预取）

当计算机从程序读取指令并将其加载到 CPU 中准备执行时，这一过程称为指令的读取。许多 CPU 都会同时加载若干条指令，这称为预取。

preprocessor （预处理程序）

一种文本处理器，它可以读取程序源代码并用数值（或其他名字）替换名字，使用文本替换扩展宏，以及评估表达式以确定哪些代码可以被去除。

pseudo-op （伪操作）

参见指示字。

Ratfor （ Rational Fortran ）

一种可供公共使用的源代码预处理程序，它允许按类似 C 的语法书写的 Fortran 可被转换为标准 Fortran 。

relative address （相对地址）

相对地址是到某个已知地址的偏移。这种类型的寻址通常用于可重定位的模块（共享库），因为只有模块最高端的位置才是加载到内存中的可执行代码所需要的惟一信息。参见绝对地址。

relocatable （可重定位的）

参见相对地址。

RM

对 Ada 来说，这是参考手册（ reference manual ）的简写，它是定义 Ada 95 标准的文档。

RTL （ Register Transfer Language ，寄存器传送语言）

由某语言源代码所生成的内部代码，并可由此输出汇编语言。在 RTL 格式下，仍可能执行优化和其他操作。

RTTI （ Runtime Type Identification ，运行时类型标识）

在面向对象编程中，可以将一种类型的对象当做另一种使用。可以利用 RTTI 工具在运行时确定对象的真实类型。

runtime （运行时）

也称作运行时软件包，由于它们在应用程序运行时会被调用，运行时是函数的集合，它同编译程序一同发行并链接到被编译的程序。

rvalue （右值）

任何可以得到一个数值的表达式。该名词源于右值（ right value ），是指在赋值语句中右边的表达式所计算出的数值。参见左值。

scheduler （调度器）

如果机器能够一次执行多条指令，指令可以被调整（调度），这样一些较快的指令就可以与一条较慢的指令同时执行。

scope （作用域）

一个定义可以被识别的区域。在 C 中，函数定义的变量的作用域是整个函数。定义在由括号括起的代码块中的变量的作用域是那个代码块，在函数外定义的变量可以由很多函数引用。

semantics （语义）

一种编程语言的语句的含义。语句的实际含义依赖于它的上下文。例如，表达式 a+b 可以是整数加、浮点加，甚至是字符串连接。确定其含义的过程称为语义分析。

serialize （串行化）

在 Java 中，一个对象可以被转换为字符串并传递（或存储留作后用），之后它还可以被转化回执行对象。参见 marshaling 。

sequence point （顺序点）

程序执行中的一点，在这一点所有的计算全部完成而且变量均包含正确的结果。在这一点可以开始计算新表达式。某些优化技术会修改和改变两个顺序点之间的操作顺序。

shared library （共享库）

参见库。

sibling call （同属调用）

参见尾调用。

side effect （副作用）

一种会修改内存或文件中数据的操作（如函数调用或算术表达式）。这些变化称之为副作用操作。大多数副作用都是有意的并且是程序执行所必要的，但有些情况下副作用是有害的。

slot scheduler （槽调度器）

参见调度器。

Single UNIX Specification

参见 POSIX 。

Smalltalk

一种在 20 世纪 70 年代 Xerox PARC 上开发的面向对象语言。 Smalltalk 是 SIMULA 界面系统的编程语言，该系统引入了鼠标和窗口系统。

spec file （ spec 文件）

一种包含一套规则的文件，这些规则可以控制哪些参数需由 gcc 传递给每一个子进程，以及可接受参数的格式。

STABS （ Symbol TABle directiveS ，符号表命令）

插入到汇编语言源代码中的操作子和数据位置，以提供调试信息。随后汇编器和连接程序可在目标代码和可执行程序中包含这些表格，以达到调试的目的。

static library （静态库）

参见库。

SSA （ Static Single Assignment ， 静态单赋值）

通过代码块代表逻辑流的一种特殊形式。 SSA 可用于某种特定类型的优化，例如不可达代码的删除。 SSA 的一个优点就是能够跟踪寄存器保存的值。

stack frame （堆栈框架）

栈中保存局部变量以及当前函数寄存器值的区域。确切的格式依赖于处理器以及所使用的函数调用规范。

static link （静态链接）

静态链接的可执行程序就是包含了所有需要函数的程序。当程序被链接时，由运行时库中取出函数并复制到被链接的程序中。参见共享库。

static single assignment

参见 SSA 。

stringize （字串化）

预处理程序采取的动作，可将宏参数转化为一个引用的字串，而非简单地将其插入到源码中。

strip

strip 命令可以用来去除可执行文件中的所有调试信息。根据编译到可执行程序中的调试信息的多少，缩减的尺寸可能是非常惊人的。

stderr （ Standard Error ， 标准错误）

每个 UNIX 程序在开始执行时均打开该输出流，默认情况下定向到终端上。通常该流可用来输出错误消息。

stdin （ Standard Input ， 标准输入）

每个 UNIX 程序在开始运行时均打开该输入流，默认情况下会从键盘中读取所敲击的内容。

stdout （ Standard Output ， 标准输出）

每个 UNIX 程序在开始执行时均打开的输出流，默认情况下定向到终端上。通常该流可以用来进行输出。

strength reduction （强度弱化）

一种优化技术，它可以用代价较低的操作取代代价高的操作，例如用加法或移位操作取代乘法操作。

stub

一种本地函数，当它被本地程序调用时，会将调用信息打包，并将调用发送给真正的函数，即位于另一台计算机上的函数。

subexpression （子表达式）

表达式的一部分。例如，在表达式 a * (b + c) 中，因子 (b + c) 是一个子表达式。一个子表达式可以非常复杂，也可以简单到例如向寄存器加载地址。

SVR4 （ System Five Release Four ， 系统 5 发行 4 ）

由 AT&T 制造的 UNIX 版本。 SVR4 是许多现代 UNIX 版本的基础。参见 BSD 。

syntax （句法）

编程语言的物理结构。语法通过强制的规则确定了语言的形式，这些规则可以确定语言自身文本的合法顺序和结构。

tail call （尾调用）

如果函数最后一条语句递归地调用自身，这一逻辑可以变成一个循环而非调用，这样可以节省栈空间。同样的优化技术可以用来在两个或多个函数间进行递归的相互调用 ——这种优化也叫同属调用。

target （目标）

参见平台。

text （文本）

由 UNIX 连接程序产生的可执行文件中包含的可执行代码段。参见 bss 和 data 。

thunk （块）

用来产生地址的一块代码。由 ALGOL 60 引入，通过名字和产生代码来传递参数，它生成的代码可以在运行时解析地址，现在块也用来专指用一列新地址覆盖旧地址的动作。

time slice （时间片）

操作系统允许进程在其停止并由另一个进程使用时间片前执行的时间量。

token （权标）

参见词法分析。

translation unit （翻译单元）

编译单元的另一种名称。

translator （翻译器）

编译程序的另一个名字，这是因为编译程序的工作就是将源码翻译为可执行代码。见编译程序。

trap （陷阱）

一种发送给正在运行的程序的硬件信号，它用来指出硬件检测到的程序执行错误。这包括诸如浮点除法除数为 0 ，以及非法的内存地址等。

trigraph （三字母序列）

标准 C 的三字母序列，可以转换为单个字符。它被设计用来在不支持 ASCII 字符全集的计算机上使用 C 编程。有 9 个三字母序列：

??= # ??([?? {

??/ \ ??)] ?? }

??' ^ ??! | ??- ~

unalias （去别名）

去除所有对内存的调用，但一种可能的到内存位置的引用就是为该单元去别名。见别名。

unordered comparison （无序比较）

两个浮点数间的有序比较不会在其中一个值为 NaN 时产生异常。参见有序比较。

unroll （循环展开）

为了优化代码，足够小并且具有固定循环次数的循环可以通过复制代码相应的次数被展开，并且循环可以被删除。

UTF-8 （ Unicode Transformation Format ， Unicode 传输格式）

也写作 UTF8 ， Utf8 或 UTF 。用这种方式编码的 Unicode 与 8 位存储的 ASCII 字符相似，即 ASCII 文本文件可被当作就是 UTF-8 编码的。

vague linkage （模糊连接）

目标文件包含的信息可用于链接和运行程序，但除了数据以外的其他内容也需要解析地址引用。一个例子就是 C++ 中的虚函数。

variadic macro （变长宏）

变长宏就是具有可变参数个数的宏。 GCC 预处理程序具备对可变参数宏进行扩展的能力，这种宏会将参数列表的文本保存在名为 __VA_ARGS__ 的变量中。

vector （向量）

一组具有相同类型和尺寸的数据项的连续集合。

volatile （易变的）

内存中的某单元可能会在其他地方被修改（没有常规的途径），这被称为易变的。这对编译程序很重要，因为将数据保存在寄存器比每次在需要时从内存加载效率更高。

vtable （ virtual function table ， 虚函数表）

面向对象语言的对象维护内部表，称为 vtables ，它包含函数的地址。通过替换这些地址，子类可以重载并替代父类中选中的函数。

VXT

Fortran 的一种变体，与 VAX Fortran 非常相似，在某种程度上与 Fortran 90 也非常相似。

weak alias （弱别名）

与弱符号相同。

weak symbol （弱符号）

具有两个或多个相同名字的全局符号时，只要除一个之外的其他全部符号均声明为弱符号就不会产生冲突。连接程序将忽略弱符号的定义，并使用正常的全局符号定义解析所有的引用，但只有全局符号不可用时才会使用弱符号。一个弱符号可被用来命名可以由用户代码重载的函数和数据。弱符号也称为弱别名，或简单叫做弱。

width （宽度）

术语宽度通常指基本数据类型的相对尺寸。例如，在 C 中， char 可能需要 1 个字节， short 可能需要 2 个字节。在这种情况下，称 short 比 char 宽。

word （字）

本地机器整数的尺寸。在 16 位机器中，字是 16 位的，在 32 位机器中，字是 32 位的。

whitespace （空白）

空白字符就是通常情况下不可见的字符，并会在编译现代自由格式语言时作为输入被忽略。它们通常是空格、制表符、竖直制表符、进纸符、换行符和回车符。

XCOFF （ Extended Common Object File Format ，扩展的通用目标文件格式）

目标文件的一种标准格式，它具有跨平台移植性，且可以适用于不同的汇编器和连接程序。参见 COFF 和 ECOFF 。