GCC

> **用于C语言的编译器**
>
> **gcc** 是GNU推出的基于C/C++的编译器，是开放源代码领域应用最广泛的编译器，具有功能强大，编译代码支持性能优化等特点。现在很多程序员都应用GCC，怎样才能更好的应用GCC。目前，GCC可以用来编译C/C++、FORTRAN、JAVA、OBJC、ADA等语言的程序，可根据需要选择安装支持的语言。

目录

一、编译过程

选项	功能
-E	预处理指定源文件（.i）
-S	编译指定源文件，产生汇编语言（.s）
-c	编译源文件，产生机器语言（.o）
-o	链接生成可执行文件

1、预处理

> 预处理指定源文件

```
gcc -E main.c - o main.i
```

2、编译

> 编译指定源文件，产生汇编语言

```
gcc -S main.i - o main.s
```

3、汇编

> 编译源文件，产生机器语言

```
gcc -c main.s - o main.o
```

4、链接

> 链接生成可执行文件

```
gcc main.o - o main
```

默认生成a.out 文件

5、快速生成

> 自动执行所有步骤

```
gcc main.c -o main
```

默认生成a.out 文件

二、编译参数

```
-o：指定生成的输出文件；
-E：仅执行编译预处理；
-g:编译带调试信息的可执行文件
-S：将C代码转换为汇编代码；
-O[n]:优化源代码
-wall：显示警告信息；
-c：仅执行编译操作，不进行连接操作。
-l | -L:库文件链接 | 库文件目录指定
-I：指定头文件
-C:保留注释信息
```

1、调试文件

> **-g**
>
> **编译可以GDB调试信息的可执行文件**

```
# -g 选项告诉 GCC 产生能被 GNU 调试器GDB使用的调试信息，以调试程序。 # 产生带调试信息的可执行文件test
gcc -g test.c
```

2、优化源代码

> ***-O[n]***
>
> **优化源代码**

所谓优化，例如省略掉代码中从未使用过的变量、直接将常量表达式用结果值代替等等，这些操作 会缩减目标文件所包含的代码量，提高最终生成的可执行文件的运行效率。

```
# -O 选项告诉 g++ 对源代码进行基本优化。这些优化在大多数情况下都会使程序执行的更快。
#-O2 选项告诉 g++ 产生尽可能小和尽可能快的代码。
#如-O2，-O3，-On（n 常为0-3）
# -O 同时减小代码的长度和执行时间，其效果等价于-O1
# -O0 表示不做优化
# -O1 为默认优化
# -O2 除了完成-O1的优化之外，还进行一些额外的调整工作，如指令调整等。
# -O3 则包括循环展开和其他一些与处理特性相关的优化工作。
# 选项将使编译的速度比使用 -O 时慢， 但通常产生的代码执行速度会更快。
# 使用 -O2优化源代码，并输出可执行文件
gcc -O2 test.c
```

3、库文件

> **-l | -L**
>
> **链接库文件 | 指定库文件路径**

`库的默认搜索路径`

> **/lib**
>
> **/usr/lib**
>
> **/usr/local/lib**

```
# -l参数(小写)就是用来指定程序要链接的库，-l参数紧接着就是库名
# 在/lib和/usr/lib和/usr/local/lib里的库直接用-l参数就能链接
# 链接glog库
gcc -lglog test.c
# 如果库文件没放在上面三个目录里，需要使用-L参数(大写)指定库文件所在目录
# -L参数跟着的是库文件所在的目录名
# 链接mytest库，libmytest.so在/home/bing/mytestlibfolder目录下
gcc -L/home/bing/mytestlibfolder -lmytest test.c
```

4、头文件

> **-I**
>
> **头文件的搜索目录**

`默认头文件路径`

> **/usr/include**

```
# /usr/include目录一般是不用指定的
#但是如果头文件不在/usr/icnclude 里我们就要用-I参数指定了，比如头文件放在/myinclude目录里，那编译命令行就要加上I/myinclude 参数了.
#-I参数可以用相对路径，比如头文件在当前目录
gcc -I/myinclude test.c
```

5、警告信息

1、打印警告

> **-Wall**
>
> **打印出gcc提供的警告信息**

```
# 打印出gcc提供的警告信息
gcc -Wall test.c
```

2、关闭打印警告

> **-w**
>
> **关闭所有警告信息**

```
# 关闭所有警告信息
gcc -w test.c
```

3、警告作为错误

> **-Werror**
>
> **将警告信息当作错误来处理**
>
> **一般与-Wall搭配使用`-Wall -Werror`只要有警告信息则编译停止**

6、设置编译标准

> **-std**
>
> **设置编译标准**

```
# 使用 c++11 标准编译
gcc -std=c99 test.c
```

7、定义宏

> **-D**
>
> **在使用gcc/g++编译的时候定义宏**
>
> **常用场景： -DDEBUG 定义DEBUG宏，可能文件中有DEBUG宏部分的相关信息，用个DDEBUG来选择开启或关闭 DEBUG**

```c
#include <stdio.h>
int main()
{
#ifdef DEBUG
    printf("DEBUG LOG\n");
#endif
    printf("in\n");
}
// 1. 在编译的时候，使用gcc -DDEBUG main.c
// 2. 第七行代码可以被执行
```

```
gcc [-c|-S|-E] [-std=standard]
           [-g] [-pg] [-Olevel]
           [-Wwarn...] [-Wpedantic]
           [-Idir...] [-Ldir...]
           [-Dmacro[=defn]...] [-Umacro]
           [-foption...] [-mmachine-option...]
           [-o outfile] [@file] infile...
```