学习和使用

TURBO C

语言（续编）

—TURBO C实用工具与库函数大全

主审 陈世福

编写 潘金贵 陆庆文 陈兆乾 樊莉萍

南京大学出版社

学习和使用 TURBO C 语言(续篇)

——TURBO C 实用工具与库函数大全

潘金贵　陆庆文
陈兆乾　樊莉萍　编写

陈世福　主审

南京大学出版社
1993·南京

(苏)新登字第 011 号

内 容 简 介

Tubro C 软件包提供了友好的集成环境和高效的编译系统。此外，它还提供了一组灵活方便的实用工具和为数 400 多个功能齐全的库函数，从而使其成为一个得力的软件开发工具。在 C 语言之林中独树一帜，引人注目，备受用户青睐。

本书共二十五章和四个附录，分为三个部分。第一部分系统地介绍了最新版本的 Turbo C 软件包提供的实用工具，包括：预处理程序、程序开发管理工具、连接程序、库管理工具、文件搜索工具、图形驱动程序和字体转换工具、目标模块交叉引用工具、目标文件重建工具、联机帮助工具等的功能和用法。第二部分，按类介绍了最新版本的 Turbo C 提供的 400 多个库函数的完整信息，包括：函数名、功能、用法、原型所在文件、使用说明和示例、返回值、可移植性等，并对每一个函数赋以编号，为便于查找函数的说明页，还提供了一个方便的索引。第三部分附录，给出了库函数使用的全程变量以及 main 函数的完整的信息，回答了用户在使用 Turbo C 时经常会碰到的那些公共问题，介绍了怎样利用 Turbo C 运行时刻库函数源程序的方法等。

本书及其上篇《学习和使用 Turbo C 语言》和《Turbo C 工具库》是一套适合广大程序员、大中专学生学习和使用 C 语言的极好教材和教学参考书，也可作为 Turbo C 软件包的使用手册和参考手册使用。

学习和使用 TURBO C 语言(续编)
——TURBO C 实用工具与库函数大全
潘金贵　陆庆文　陈兆乾　樊莉萍　编写
陈世福　主审
*
南京大学出版社出版
(南京大学校内，邮编 210008)
江苏省新华书店发行　　高邮市印刷厂印刷
*
开本：787×1092　1/16　印张：20　字数：492 千
1993 年 5 月第 1 版　　1993 年 5 月第 1 次印刷
印数 1— 6000，
ISBN　7—305—02128—8/TP・66
定价：13.80 元
责任编辑：顾其兵

前　言

C 语言是一种结构化、模块化、可编译的通用程序设计语言，被广泛地用于系统程序和应用程序的开发。例如，著名的 UNIX 操作系统就是用 C 语言书写的。C 语言有着良好的可移植性，用 C 语言书写的程序在不同的计算机系统之间很容易实现转换。因此，几乎所有的程序设计任务均可使用 C 语言来完成。

Turbo C 是美国 BORLAND 国际公司在 IBM PC 机上实现的一个高效、优化的 C 编译程序。编译速度快，例如 Turbo C 1.5 版的编译程序每分钟就可编译 10000 行源程序，Turbo C 2.0 的编译程序较 1.5 版快 20%－30%。它在编译时利用 RAM 存放中间数据结构，一趟扫描产生内部代码，只需要一次性对盘上的源文件读入和写出目标码（其他 C 语言编译程序则要 4－5 次读盘，每次执行一种功能）。编译产生的目标模块的格式与 PC-DOS 连接程序一致，可与汇编程序连接。支持极小、小、中、紧凑、大和特大六种存储模式。可以使用近、远指针的混合模式。

Turbo C 支持 IEEE 浮点标准，Turbo C 包含有处理 80x87 协处理芯片的浮点例程，80x87 协处理器能大大加快程序的运行速度，但不是必要的，当没有 80x87 协处理器时，可使用系统提供的仿真 80x87 协处理器的实用程序进行高速的浮点运算。

Turbo C 提供了一个完整的交互式集成开发环境，包括：

- 友善的用户接口。通过下拉菜单和多窗口，能够从集成开发环境中产生、调试或运行一个可执行文件。
- 高效能的全屏幕编辑程序。当编译过程中出现错误时，编辑程序自动地用彩色光标指出源程序中适当的出错位置。
- 强有力的“MAKE”功能。使得大型 C 程序开发的管理十分容易。
- 快速的内部 Turbo 连接程序和内含式上下文敏感的帮助功能等。

Turbo C 2.1 是 Turbo C 风格语言的最新一代。从 Turbo C 2.0 起，Turbo C 最为显著的特点是包含了可在源程序级进行符号调试的集成调试程序，可以完成单步执行（STEP OVER 和 TRACE INTO）、设置断点、监视和计算表达式等功能。该调试功能还支持 BORLAND 国际公司新增的独立的调试程序 Turbo DEBUGGER。提供了较以前版本更丰富的图形函数库，包括新增了许多函数和可安装的驱动程序及字体等。所以用现在的 Turbo C 来编制软件，可以使用户界面设计得相当完美。

除了上述的调试和图形功能以外，新版本的 Turbo C 还具有如下一些新的特点：

- 更快的内存分配和串函数
- 更快的浮点计算例行程序
- 更快的内部连接程序
- 可连接生成小模式的.COM 文件
- 新增加了 signal 和 raise 函数
- EMS 扩充存储器可用作编辑缓冲区
- 引入了允许用户在编译时向程序插入机器代码的__emit__机制
- 支持命令行上的通配符

- 支持 long double 常数和变量
- 能够自动地进行快缩进/回退及优化填充
- MAKE 实用程序可执行自动依赖关系检查
- 与以前版本向下兼容

Turbo C 提供了与 Turbo Prolog，Turbo Pascal 和汇编语言等多种语言的接口，能把用不同风格程序设计语言建立的目标模块连成单一的程序。Turbo C 实现了美国国家标准局(ANSI)建议的 C 语言标准，完全支持 Kernighan 和 Ritchie 的 C 定义。

一个好的 C 语言系统，不仅要提供一个好的编译器和开发环境，还要提供功能齐全，方便实用的工具和库函数，它们代表了 C 语言的能力。Turbo C 提供了 400 多个库函数和近十个可独立运行的实用程序(本书将予以详细介绍)，可得心应手地用于开发各类程序。

Turbo C 支持 CGA、EGA、VGA、HERCULES、3270PC、AT&T400 线和 IBM 8514 等高质量的图形库。此外还包括了一些混合模式程序设计任选的扩充，进一步挖掘了 PC 机的能力。

我们在长期使用 Turbo C 语言进行教学和编程的基础上，曾对 Turbo C 的多种版本进行了成功的汉化，为开发汉化应用软件提供了有力的工具。

此外，我们还根据最近国外流行的几种介绍 Turbo C 的书籍和软件的配套手册等编译和改写了一套(共三本)学习和使用 Turbo C 语言的教材和教学参考书。

本书是《学习和使用 Turbo C 语言》的续编。全书共二十五章和四个附录，分为三个部分：

第一部分系统地介绍了最新版本的 Turbo C 软件包提供的实用工具，包括预处理程序、程序开发管理工具、连接程序、库管理工具、文件搜索工具、图形驱动程序和字体转换工具、目标模块交叉引用工具、目标文件重建工具、联机帮助工具等的功能和用法。

学习和使用 Turbo C 语言，除了要掌握 C 程序的结构，各种语句的用法外，还要学会利用 C 语言提供的大量的库函数来完成自己要解决的任务。因此，我们在本书的第二部分按类介绍了最新版本的 Turbo C 提供的 400 多个库函数的完整信息，包括：函数名、功能、用法、原型所在文件、使用说明和示例、返回值、可移植性等，并对每一个函数赋以编号，为便于查找函数的说明页，还提供了一个方便的索引。

第三部分附录，给出了库函数使用的全程变量以及 main 函数的完整的信息，回答了用户在使用 Turbo C 时经常会碰到的那些公共问题，介绍了怎样利用 Turbo C 运行时刻库函数源程序的方法等。

本书及其上篇《学习和使用 Turbo C 语言》，还有《Turbo C 工具库》是一套适合广大程序员、大中专学生学习和使用 C 语言的极好教材和教学参考书，也可作为 Turbo C 软件包的使用手册和参考手册使用。

可配合本书使用的软件有 Turbo C 2.0、2.1(含西文原版和中西文版)系统盘，以及 Turbo C 工具库盘(Turbo C Tools 6.0)，以及 Turbo C 运行时刻库源程序等共二十多枚，需要的读者可与本书的作者联系获得有关信息。

参加本书原稿的翻译和改编的主要人员有潘金贵、陆庆文、陈兆乾和樊莉萍等同志，并由潘金贵同志主持，陈世福教授主审。

限于水平和时间仓促，难免有错误和不妥之处，恳请读者批评指正。

编　者　1992 年 3 月
于南京大学计算机科学系

目　录

第一部分　Turbo C 实用工具

第二部分 Turbo C 库函数

第三部分 附 录

第一部分　Turbo C 实用工具

Turbo C 语言系统不仅提供了两种当今最快的 C 语言编译程序，还提供了近十个可独立运行的十分有效的实用工具，这些实用工具既能用于 Turbo C 文件，也可用于其他模块。

这些工具是：CPP、MAKE、TLINK、TLIB、GREP、BGIOBJ、OBJXREF、CINSTXFR 等。

本部分介绍了这些实用程序的功能，并举例说明它们的用法。

第一章　Turbo C 预处理程序 CPP

CPP 程序用于扩充 Turbo C 编译程序的功能。它对于 C 语言程序的一般编译是根本不需要的。CPP 的目的在于产生一个对嵌入文件和宏定义进行了扩展的 C 源程序的列表文件。

通常，当编译程序在宏或嵌入文件内发现了错误时，如果能看到宏或嵌入文件扩展的结果，就可以更多地了解到有关此错误的信息。在许多多趟扫描的编译程序中，有专门一趟扫描负责这项工作，并且这次扫描的结果可被查看。

由于 Turbo C 使用集中式单趟扫描编译程序，CPP 充当了其他编译程序中第一趟扫描的功能。此外，CPP 还可用作宏预处理程序。

CPP 程序的使用方式和运行编译程序 TCC 一样。CPP 从同一个 TURBOC.CFG 文件中读入默认选项，且命令行选项也和 TCC 相同。

CPP 忽略与 CPP 无关的 TCC 选项。要想查看由 CPP 处理的自变量表，只要在 DOS 提示符下键入 CPP 即可。

CPP 命令行中的文件名和 TCC 中一样处理，且允许使用通配符。只是所有文件都作为 C 源文件处理，对于.OBJ、.LIB 和.ASM 文件没有特殊处理。

每一个由 CPP 处理的文件，其输出结果均写到当前目录（或由－n 选项命名的输出目录中）的文件中，其名为源文件名加扩展名.I。

该输出文件是一个正文文件，它包含了源文件的各行和所有嵌入文件。所有预处理指令以及所有不参加的编译条件正文行都被删去，正文行均前缀以文件名和源文件的行号或嵌入文件名及行号，正文行中所有宏指令都用它们的扩展正文代替。

由于每行前缀都附有文件名和行号，CPP 的输出结果不能被编译。

CPP 作为宏预处理程序

CPP 的-P 选项要求在每行前缀以源文件名和行号。若给定-P -，则 CPP 省略行号信息。这样，CPP 能被用作宏预处理程序，产生的.I 文件则可被 TC 或 TCC 编译。现举例说明。

下面的程序说明了 CPP 如何先使用-P 然后以-P -选项来预处理文件。

源文件:HELLOJOE. C

```
/ * 这是一个 CPP 输出的例子 * /
#define NAME "Joe Smith"
#define BEGIN {
#define END }
main ()
BEGIN
    printf( "%s\n",NAME);
END
```

将 CPP 作为预处理程序调用的命令行:

```
cpp hellojoe. c
```

输出:

```
hellojoe. c 2:
hellojoe. c 3:
hellojoe. c 4:
hellojoe. c 6: main()
hellojoe. c 7:{
hellojoe. c 8: printf( "%s\n","Joe Smith");
hellojoe. c 9: }
```

将 CPP 作为宏预处理程序调用的命令行:

```
cpp -P -hellojoe. c
```

输出:

```
main()
{
    printf( "%s\n","Joe Smith" );
}
```

第二章　大型程序管理工具 MAKE

Turbo C 具有功能强和灵活性好的特点，可以用来管理由多个嵌入文件、源文件和目标文件构成的大而复杂的程序，但要求记住哪些文件需用来产生其他文件。这是因为如果对某一文件作了改动，就要作必需的重编译和连接。当然，一种解决办法是每当作了改动就对所有模块重编译一次。但当程序越来越长，重编译就要化费越来越多的时间，那么，该怎么做呢？

答案很简单：使用 MAKE。MAKE 是一个聪明的管理程序，只要给定适当指令，它就能做所有必要的工作使得程序保持新貌。实际上，MAKE 能做的远远不止这个，它可以复制副本，从不同子目录中取出文件，甚至当程序使用的数据文件被修改了，它能自动运行程序。用熟了 MAKE 以后，我们还可发现各种新的、不同的有助于程序开发的手段。

MAKE 是一个可独立运行的程序，它与 Project－Make 不同，后者只是集成开发环境的一部分。

本节将介绍如何与 TCC 和 TLINK 一起使用 MAKE。

§2.1　一个简单的例子

下面以一个例子来说明 MAKE 的用途。假设有一显示银河系信息的程序 GETSTARS，它读入一个有关银河系的正文文件，进行一些处理，然后产生一个存有结果信息的二进制数据文件。

GETSTARS 使用了 STARDEFS. H 中的某些定义和 STARLIB. C(在 STARLIB. H 中说明)的某些子程序。GETSTARS 程序本身由三个文件组成：GSPARSE. C，GSCOMP. C，GETSTARS. C。前两个文件 GSPARSE 和 GSCOMP 具有相应的嵌入文件(GSPARSE. H 和 GSCOMP. H)，第三个文件 GETSTARS. C 包含了程序主体。在三个文件中，只有 GSCOMP. C 和 GETSTARS. C 使用了 STARLIB 程序。

下面是每个. C 文件必需的用户嵌入文件(除说明标准运行时刻库子程序的 Turbo C 嵌入文件)。

.C 文件	嵌入文件
STARLIB. C	无
GSPARSE. C	STARDEFS. H,
GSCOMP. C	STARDEFS. H,STARLIB. H,
GETSTARS. C	STARDEFS. H,STARLIB. H,GSPARSE. H,GSCOMP. H

为产生 GETSTARS. EXE(假设用中数据模式)，需键入下面的命令行：

```
tcc -c -mm -f starlib
tcc -c -mm -f gsparse
tcc -c -mm -f gscomp
tcc -c -mm -f getstars
```

```
tlink lib\com starlib gsparse gscomp getstars,getstars,getstars,lib\emu lib\mathm lib\cm
```

查看上面的信息，可看出一些文件的相互依赖关系：

(1) GSPARSE，GSCOMP 和 GETSTARS 均依赖于 STARDEFS. H。换句话说，如果对 STARDEFS. H 作了改动，那么这三个文件均要重新编译。

(2) 同样地，对 STARLIB. H 的任何改动也将要求重新编译 GSCOMP 和 GETSTARS。

(3) GSPARSE. H 的改动意味着 GETSTARS 必须重新编译，对 GSCOMP. H 也一样。

(4) 当然，对任何源代码文件(STARLIB. C，GSPARSE. C 等)的改动意味着该文件要重新编译。

(5) 最后，如果进行了重编译，就么就必须再进行连接。

这要保留相当多的信息。如果改变了 STARLIB. H，重新编译了 GETSTARS. C，但忘记重编译 GSCOMP. C，将发生什么情况呢？当然可以使用一个. BAT 文件进行四次重编译一次连接。但每改变一次，就必须再做一遍。让我们看看 MAKE 是如何简化这些工作的。

2.1.1 建立 MAKE 文件

由上述两组信息(依赖关系及相应的编译连接命令)即可组成一 MAKE 文件。例如，上述两组信息加在一起，稍微改变就成为：

```
getstars. exe:getstars. obj gscomp. obj gsparse. obj starlib. obj
    tlink lib\c0m starlib gsparse gscomp getstars, getstars,\
        getstars,lib\emu lib\mathm lib\cm
getstars. obj:getstars. c stardefs. h starlib. h gscomp. h gsparse. h
    tcc -c -mm -f getstars. c
gscomp. obj:gscomp. c stardefs. h starlib. h
    tcc -c -mm -f gscomp. c
gsparse. obj:gsparse. c stardefs. h
    tcc -c -mm -f gsparse. c
starlib. obj:starlib. c
    tcc -c -mm -f starlib. c
```

这仅仅是以前所说的重述，不过次序有点颠倒，MAKE 按如下方式解释这个文件：

(1) 文件 GETSTARS. EXE 依赖于 4 个文件：GETSTARS. OBJ，GSCOMP. OBJ，GSPARSE. OBJ 和 STARLIB. OBJ。四个. OBJ 文件中任何一个改变，GETSTARS. EXE 都必须重连接。这可以通过使用给出的 TLINK 命令实现。

(2) 文件 GETSTARS. OBJ 依赖于五个文件：GETSTARS. C，STARDEFS. H，STARLIB. H，GSCOMP. H 和 GSPARSE. H。如果其中之一改变，GETSTARS. OBJ 必须使用给出的 TCC 命令行进行重编译。

(3) GSCOMP. OBJ 文件依赖于三个文件：GSCOMP. C，STARDEFS. H 和 STARLIB. H，若其中之一改变，GSCOMP. OBJ 必须用给出的 TC 命令进行重编译。

(4) 文件 GSPARSE. OBJ 依赖于两个文件，GSPARSE. OBJ 和 STARDEFS. H，同样，若其中

之一改变，该文件需用给出的 TCC 命令行重新编译。

(5) 文件 STARLIB. OBJ 只依赖于一个文件：STARLIB. C，如果它改变了，则必须经 TCC 命令行重新编译。

将上述信息输入一文件(例如 MAKEFILE)，即可使用 MAKE. EXE。

2.1.2 使用 MAKE 文件

假设按上面描述生成了一个 MAKEFILE，同时假设各源代码和嵌入文件存在，即可键入命令：

```
make
```

MAKE 查寻 MAKEFILE(可以使用其他名，后面将要讨论)并且读描述 GETSTARS. EXE 依赖关系的第一行，检查 GETSTARS. EXE 是否有最新的存在。

这要求对 GETSTARS. EXE 依赖的每个文件(GETSTARS. OB，GSCOMP. OBJ，GSPARSE。OBJ 和 STARLIB. OBJ)作相同的检查。其中各个文件所依赖的其他文件也要被检查。为更新各. OBJ 文件，需多次调用 TCC 命令，最后执行 TLINK(若必要的话)产生 GETSTARS. EXE 的最新版本。

如果 GETSTARS. EXE 和所有. OBJ 文件均已存在，MAKE 则将每个. OBJ 文件的最后一次修改日期、时间和它依赖的文件的日期、时间作比较。如果有一依赖的文件比. OBJ 文件新，MAKE 知道从上次. OBJ 文件生成以来已经有了修改，则执行 TCC 命令。

如果 MAKE 更新了任一. OBJ 文件，则当它将 GETSTARS. EXE 的日期、时间与它们比较时就知道必须执行 TLINK 命令以产生 GETSTARS. EXE 的最新版本。

2.1.3 使用 MAKE 举例

下面举一例子以帮助理解以上描述。假设 GETSTARS. EXE 和所有的. OBJ 文件均存在，并且 GETSTARS. EXE 比所有. OBJ 文件新，同样地每个. OBJ 文件比它所依赖的各文件新。这样，若打入命令：

```
make
```

则不会有任何事发生，因为没有任何文件需要更新。

现在，假设修改 STARLIB. C 和 STARLIB. H，如改变某些常量的值，当打入命令

```
make
```

MAKE 看到 STARLIB. C 比 STARLIB. OBJ 新。因此它发出命令：

```
tcc -c -mm -f starlib.c
```

然后它看到 STARLIB. H 比 GSCOMP. OBJ 新，于是它发出命令：

```
tcc -c -mm -f gscomp.c
```

STARLIB. H 也比 GETSTARS. OBJ 新，于是下一条命令是：

```
tcc -c -mm -f getstars.c
```

最后，由于这三条命令，文件 STARLIB. OBJ，GSCOMP. OBJ 和 GETSTARS. OBJ 都比 GETSTARS. EXE 要新，因比 MAKE 发出的最后一条命令是：

```
tlink lib\c0m starlib gsparse gscomp getstars, getstars,getstars, lib\emu lib\mathm lib\cm
```

它将所有的.OBJ 文件连接在一起生成 GETSTARS.EXE 的一个新版本(注意,TLINK 命令行必须为一行)。

现已对 MAKE 有了一些基本的认识:它是干什么的,如何生成一个 make 文件,以及 MAKE 如何解释该文件。下面将更详细地介绍之。

§2.2 MAKE 文件的组成及生成

Make 文件包含了帮助 MAKE 使程序保持"最新"的定义和关系。可以生成多个 make 文件,并可随意给它们命名。MAKEFILE 只是当运行 MAKE 而又未指定 make 文件时,MAKE 查找的一个默认文件名。

可以使用任何 ASCII 码正文编辑程序(如 Turbo C 的内部交互式编辑程序)来生成 make 文件。所有的规则、定义和指令均需以换行符结束。若一行太长,可用反斜杠(\)作为行的最后一个字符来继续到下一行。

空白(空格和制表键)用于分隔相邻的标识符(例如依赖文件名),并且在规则中用于对齐命令。

生成 make 文件类似于编写程序,它带有定义、命令和指令。make 文件中允许出现下述成分:

(1) 注解

(2) 显式规则

(3) 隐式规则

(4) 宏定义

(5) 指令:嵌入文件、条件执行、错误检测、取消宏定义。

以下,详细描述各个组成部分。

2.2.1 注解

注解以字符#开始,该行#后的部分被 MAKE 忽略,注解可放在任何位置,并且不必在某一特定列开始。

不能用反斜杠(\)把注解继续到下一行,因而每行必须使用一个#。实际上,在含有注解的行中不能使用反斜杠作为继续字符,如果它在#前面,它不再是行的最后一个字符;如果它跟着#,则它是注解本身的一部分。

下面是 make 文件中几个注解的例子:

```
#makefile for GETSTARS.EXE
    #does complete project maintenance
      getstars.exe:getstars.obj gscomp.obj gsparse.obj starlib.obj
    #can't put a comment at the end of the next line
      tlink lib\c0m starlib gsparse gscomp getstars,\
        getstars, getstars, lib\emu lib\mathm lib\cm
```

```
#legal comment
#can't put a comment between the next two lines
  getstars.obj:getstars.c stardefs.h starlib.h\
    gscomp.h gsparse.h
  tcc -c -mm -f getstars.c #you can't put a comment here
```

2.2.2 显式规则

前面的例中已使用了显式规则。显式规则的形式为：

```
target [target…]:[source source…]
    [command];
    [command]
    …;
```

这里 target 是要更新的文件，source 是 target 所依赖的文件，command 是任意有效的 MS-DOS 命令(包括调用.BAT 文件和执行.COM 及.EXE 文件)。

显式规则定义一个或多个目标名，零个或多个源文件以及一个可选的命令表。在显式规则中列出的目标文件名、源文件名，可以含有驱动器号和指定目录，但不能含有通配字符。

这里的语法格式相当重要。target 必须始于行首(第 1 列)，而每个 command 前面至少有一个空白或制表符。如前所述，当源文件表或一给定命令太长，一行无法容纳时，可用反斜杠(\)作为继续符。注意，源文件和命令都是任选的。只含 target[target…]:的显式规则也是合法的。

显式规则的意图是：用列出的命令通过使用源文件生成或更新目标文件。当 MAKE 遇到一个显式规则，它首先在 makefile 中检查各源文件本身是否为其他地方的目标文件。如果是，首先处理那些规则。

一旦根据其他显式(或隐式)规则生成或更新了所有源文件，MAKE 就检查目标文件是否存在。如果不存在，则按给定次序执行每个命令。如果存在，则将各源文件的最后修改的日期、时间和目标文件的日期、时间比较。若某源文件比目标文件更晚修改，则执行命令表。

在一次给定的 MAKE 执行中，一个给定的文件名只能在一显式规则的左侧出现一次。

在显式规则中，每个命令行以空白开始。MAKE 认为在显式规则后的所有内容都是此规则命令表的一部分，直至从第一列开始的行(前面无空白)或文件的结束。空行将被忽略。

特殊情况

没有命令行的显式规则的处理与有命令行的显式规则的处理稍有不同：

(1) 如果某目标的显式规则中有命令行，则此目标依赖的文件仅为显式规则中列出的那些文件。

(2) 如果无命令行，目标不仅依赖于显式规则中给出的文件，还依赖于任何与本目标的隐式规则匹配的文件。

参见下节介绍的隐式规则。

例子

下面是显式规则的几个例子：

```
myprog.obj:myprog.c
        tcc -c myprog.c
prog2.obj:prog2.c include\stdio.h
        tcc -c -k prog2.c
prog.exe:myprog.c prog2.c include\stdio.h
        tcc -c myprog.c
        tcc -c -k prog2.c
        tlink lib\c0s myprog prog2,prog, , lib\cs
```

(1) 第一个显式规则说明 MYPROG.OBJ 依赖于 MYPROG.C，而 MYPROG.OBJ 是通过执行给出的 TCC 命令行生成的。

(2) 第二个显式规则说明 PROG2.OBJ 依赖于 PROG2.C 和 STDIO.H(在 INCLUDE 子目录中)，并且 PROG2.OBJ 是由给出的 TCC 命令行生成的。

(3) 最后一条显式规则说明 PROG.EXE 依赖于 MYPROG.C，PROG2.C 和 STDIO.H，并且若有一个改变，PROG.EXE 就可由给定的命令串重建。但是这可能产生不必要的工作，因为即使只有 MYPROG.C 改动了，PROG2.C 仍将重编译。这是由于只要规则的目标过时了，此规则的所有命令就将被执行。

(4) 如将规则：

```
prog.exe:myprog.obj prog2.obj
    tlink lib\c0s myprog prog2, prog, , lib\cs
```

作为 make 文件的第一条规则，后面跟(对 MYPROG.OBJ 和 PROG2.OBJ)给出的规则，那么只有那些需要重编译的文件才被重编译。

2.2.3 隐式规则

MAKE 同样允许定义隐式规则。隐式规则是显式规则的拓广形式。下面的例子说明了两种规则之间关系。考虑前面例子中的显式规则：

```
starlib.obj:starlib.c
        tcc -c -mm -f starlib.c
```

此规则遵循一般规则：.OBJ 文件依赖于具有相同名字的.C 文件，并且通过执行 TCC 命令行生成。实际上 make 文件可能有若干个(甚至几十个)具有这种格式的显式规则。

将显式规则重定义成隐式规则能避免所有具有相同形式的显式规则。隐式规则形式如下：

```
.c.obj:
        tcc -c -mm -f $<
```

这规则意为“所有以.OBJ 结尾的文件依赖于同名的以.C 结尾的文件，并且该.OBJ 文件通过使用命令行 tcc -c -mm -f $< 生成。其中 $<代表带有.C 扩展名的文件”(符号 $<是

一个特殊的宏命令，将在下一节讨论）。

隐式规则的语法为：

```
source _ extension. target _ extension:
    {command}
    {command}
    ...
```

与前面一样，其中诸命令是任选的且必须缩进书写。source _ extension（必须在第一列开始）是源文件的扩展名。即它适用于任何如下格式的文件：

```
fname. source _ extension
```

类似地，target _ extension 是目标文件的扩展名，它适用于文件：

```
fname. target _ extension
```

其中 fname 对两个文件来说是相同的。换句话说，此隐式规则代替了所有如下格式的显式规则：

```
fname. target _ extension:fname. source - extension
    {command}
    {command}
    ...
```

其中 fname 为任何名。

如果找不到给定目标文件的显式规则，或者其显式规则不带命令，则使用隐式规则。此时该目标文件名的扩展名用于确定使用哪个隐式规则。若发现一个文件具有与目标相同的名，且具有提到的源扩展名，则使用该隐式规则。例如，假设有一个 make 文件（名为 MAKEFILE）其内容为：

```
. c. obj:
    tcc - c - ms - f $<
```

如果有一个名为 RATIO. C 的 C 程序，需编译成 RATIO. OBJ。可以使用命令：

```
make ratio. obj
```

MAKE 将 RATIO. OBJ 看作目标。由于没有显式规则能生成 RATIO. OBJ，MAKE 使用隐式规则并产生命令：

```
tcc - c - ms - f ratio. c
```

当然，该命令确能通过编译来生成 RATIO. OBJ。

当一个显式规则不带命令时也使用隐式规则。假设在 make 文件开始有如下隐式规则：

```
. c. obj:
tcc - c - mm - f $<
```

原显式规则可重写成下面形式：

```
getstars. obj:stardefs. h    starlib. h    gscomp. h    gsparse. h
gscomp. obj:stardefs. h    starlib. h
gsparse. obj:stardefs. h
```

由于没有关于如何生成. OBJ 文件的显式信息，MAKE 使用以前定义的隐式规则。由于 STARLIB. OBJ 只依赖于 STARLIB. C，该规则不用列出，MAKE 将自动地使用之。