Microsoft

C 语言编译和链接

用户参考手册

（适用于8086和8088微处理器
和MS-DOS操作系统）

杜子德　叶亚明

科　海　培　训　中　心

一九八六年九月

Microsoft

C语言编译和链接

用户参考手册

（适用于8086和8088微处理器
和MS-DOS操作系统）

杜子德　叶亚明

科　海　培　训　中　心

一九八六年九月

编辑：科海培训中心教材部
发行：科海培训中心资料组
地址：北京2725信箱　科海培训中心
资料组
（北京海淀路332路黄庄站旁）
印刷：河北省蔚县印刷厂

译 者 序

程序设计语言C是一种十分流行的语言，国内市面上已有若干种介绍C语言的文章，但多为泛泛而谈，或在原文本的基础上进行改编。以某机器某C编译为背景而介绍的资料实不多见。目前国内微处理机以PC/XT为主，该手册就是为PC机上的C语言而配制的。为了满足用户在这一方面的要求，我们翻译了这本手册。

该手册由四部分组成：第一部分为C编译手册；第二部分为链接用户指南；第三部分为链接参考手册；第四部分为库管理参考手册。第一部分由杜子德翻译，其余三部分由叶亚明翻译。

作者详尽地描述了在装有MS-DOS操作系统的8086和8088微处理器上的编译和链接装配过程，编译所使用的硬件特点，所有实用函数和宏，几种内存模式的使用方式以及编译，链接出错信息和索引等。此外还给出了链接方面的技术性资料和系统库管理方式。特别是对系统函数和宏逐条地对其调用方式、参考说明、功能描述、返回代码、注意事项进行了描述，无论是对程序设计初学者还是对已具有丰富的程序设计经验的软件工作者来说，都是一本不可多得的参考手册和技术资料。

但正如该书作者所指出的，本书不是一本语言文本，也没有什么C语言程序设计方面的例子。关于C语言本身的定义，读者可参阅B.W.Kernighan和D.M.Ritchie所著的《程序设计语言C》（《The C Programming Language》，PRENTICE—HALL，WC出版，1978）。该文本的中译本请看《计算机工程与应用》1982年第1，3期(华北计算技术研究所)。读者手中如有该文本和这本手册，且有初步程序设计知识，则用C语言进行程序设计不成问题。

译者已用该手册在装有MS—DOS操作系统的PC/XT上进行了一年多的程序设计实践，证明它非常适用于Microsoft C Compiler（Microsoft C编译），且非常方便适用。

译文最后由杜子德进行了校对，对有些专业名词进行了统一，使其更为符合我国读者的习惯。即使如此，由于译者水平有限，错误在所难免，敬希读者提出宝贵的意见和建议。

译者谨识

一九八六年八月

于中科院计算所

Microsoft C 编译器（MCC）
（适用于8086和8088微处理器和MS-DOS操作系统）

参 考 手 册

前　言

这本手册提供了Microsoft C编译器实现的功能描述，此C编译器是高级程序设计语言C的编译器。这里并不打算讨论程序设计基础或如何用C去编程，更广泛的资料可参考由Brian W. Kernighan和Dennis M. Ritchie所写的定义文本《程序设计语言C》（Prentice—Hall，Inc. Englewood Cliffs，New Jersey，1978）*，该语言文本已包括在这个软件包中。下面关于C语言的说明，如果没有C语言文本则是不完整的。这个语言文本还提供了对程序设计语言的极好的指导性的入门教材。

该手册分为三章。第一章是对系统实现的详细说明，并以操作指令开始（如何运行编译器、链接以及执行程序等）。第二章描述了由该编译器能够接收的语言，该语言仅在很少细微地方与标准语言有所不同。第三章按照功能组提供了库函数，且有调用顺序及例子。由于这本手册是打算用来作参考的，故每个函数的标题一般按所遇见的那样给出了完整的技术名称。

对一些仍没有遇到的节参考是不可避免的，但对这些参考作出了特别的注记。为了得到该编译器的概观，应先阅读每一章的主要段落的第一部分。第一章中的实现说明，第二章中语言定义开头的语言概述，第三章中库函数组开始的函数概述。

---

*此书见《计算机工程与应用》1982年1，3期——译者注。

目　录

第一章　MS—DOS的实现

Microsoft（R）C编译在Mircosoft MS（TM）—DOS操作系统下运行。它接收以C语言（完整语言，不是子集）写的程序并按照Intel（TM）8086目标模块格式产生可浮动的机器代码，该代码是适于Microsoft Link链接器上使用的。库定义了大量的I/O子程序，它是在MS—DOS下对大多数在Kernighan和Ritchie所著的文本中描述的与UNIX兼容的标准函数的实现。

8086指令集特别适于实现象C这样的高级语言，并且Microsoft编译产生的机器代码完全利用了它的特点。虽然8086体系结构支持多达1兆字节编址的内存空间，但它的段式编址方法最有效的工作空间是64K字节。编译支持四种不同的内存编址环境，或"模式"，这样，程序员可对特殊的应用选择有效的组合和所要求的编址，这些模式将在1.4节中更为详细地讨论。开始，将以例子表示最简单且最有效的模式（model）：按照所谓的S模式程序员可获得最大为64K字节的程序段（函数），再加最大为64K字节的数据段（包括静态字节，自动或栈数据以及动态可分配内存）。尽管有这些限制，也能开发非常复杂和功能很强的程序（包括编译本身）。

1.1　操作指令

关于编译包所提供的文件一览表参见附录D。可执行文件MC1和MC2构成实际的编译。每一个完成编译的一部分且须以各自的命令调用之。当MC1完成其功能后，并不自动调用MC2。一般地，MC2应立刻在MC1后执行，如果没有错误的话。编译过程可用下面图表示：

file.C→　MC1→file.Q

file.Q→　MC2→file.OBJ

MC1读必须以·C为其扩展名的C文件，如果没有什么本质性的错误的话，则将其编成以·Q为其扩展名中间文件。MC2读MC1产生的中间文件并产生以·OBJ为其扩展名的相同名的目标文件。当MC2完成其功能时自动删除·Q文件。每一趟一般都在与输入文件相同的设备上输出目标。注意，如果原文件不只一个，则目标亦然。当程序联接时，局部函数不能从目标文件中解出。详细参看1.3.2。

OBJ文件必须作为联接程序的输入以便产生可执行文件。二个特殊的文件必须包含在联接过程中，除了由用户创建的OBJ目标文件外。联接过程可用图表示如下：

CS·OBJ + user·OBJ + … + MCS·LIB→LINK→user·EXE

注意

实际所使用的文件名取决于所选择的内存模式；详见1.4。在以下的讨论中，使用了S (Small) 模式描述联接过程

所要求的特殊文件首先是CS.OBJ和MCS.LIB，CS.OBJ文件必须在LINK执行命令中确定为第一个模块，该模块定义了对任何的用MicrosoftC Compiler (MCC) * 所产生的程序的执行入/出口点。其次，MCS.LIB必须说明成为一个库；该文件定义了所有运行时刻的文件和作为Microsoft+C程序包的一部分的I/O库文件。用户也必须确定在联接过程中应包括的任何OBJ文件，以及由联接程序产生的EXE文件的名字。

为了论述程序产生顺序，这里有一些命令，它们对编译，联接和执行华氏/摄氏温度转换程序是必须的。这个例子假定所有的可执行文件 (MC1，MC2，LINK)都在同一盘上。命令必须以大写字母键入，显然小写字母也有同样的功能。

步骤1：执行编译的第一趟

MC1 FTOC<〈RET〉**

注意.C扩展名是不必提供的（当然即使提供了也不算错）。

步骤2：在执行了MC1并出现了MS—DOS的提示符后，开始执行编译的第二趟

MC2 FTOC〈RET〉

同样，不提供.Q扩展名，MC2自动提供.Q。

步骤3：在执行了MC2并出现了提示符后，执行链接程序

LINK CS+FTOC，FTOC，NUL，MCS〈RET〉

注意CS（即CS.OBJ）是作为LINK命令的第一个目标模块；这是链接任何C程序所要求的。FTOC(即FTOC.OBJ,它刚由MC2产生)是作为又一个目标模块。命令还产生以FTOC.EXE命名的运行文件，使程序跳过产生链接图，以及使LINK查找MCS.LIB作为外部访问。

步骤4：执行EXE文件

FTOC〈RET〉

一列华氏温度值以及与其等值的摄氏温度值将显示到用户的控制台上。

关于编译、链接和执行程序的详细指令将在下面几节中说明。关于编译程序完成的过程将在1.3中详细讨论。

在提供各种命令行格式时，“field”（域）这一词将用来描述在命令行中的一串非空格字符。可选域括在方括号([ ])中；当实际打入命令时并不打入方括号。琢磨一下该节末尾的例子，看看实际的命令是什么样子。

1.1.1 第一趟编译

编译的第一趟读C源文件并产生称之为“四元组”(quadruples)逻辑记录的中间文件，该过程详细的讨论请看1.3.1。调用第一趟编译的格式是

MC1 [=stack][>listfile] filename[options]〈RET〉：

---

*简略起见，Microsoft C 编译为MCC——译者注。

**（RET）为回车之意——译者注。

所示的各种命令行的描述字必须按上述顺序出现。可选描述字括在[]中。前二个选择是对所有的C程序的一般命令行选择的部分（见1.1.4）。如果检测出错误的话，返回0出口代码，否则为1。这允许在批量文件中使用“if”表达式，例如：

```
MC1 %1
if  错误级 1 goto errs
```

=stack　第一个选择用于优先为栈保留字节数（见1.5中关于C程序的完整的结构描述）。省缺是2048（十进制）字节，这对大多数程序来说已经是足够了。如果提供的话，其大小一定是MC1后所提供的数，它是等号（=）后随一十进数（例如=4096，即为4096个字节）。

由于编译使用递归的方法处理C语句，嵌套很深的语句要比一般“平面”语句使用更多的栈空间，如果有过多嵌套语句的源程序(ifs within ifs within ifs等等）引起栈溢出而导致第一趟执行终止时，若增加栈容量比如4096，则会使程序成功地通过MC1编译。需要有确定必要栈大小的经验。在对内存有约束的系统中，栈大小可能被适当下调以避免出现“无足够空间”之类的错误；但是没有保证编译将会成功，特别是栈被减少到1024字节。

>listfile　第二个选择用于将第一趟的信息放入一指定的文件中。这些信息包括编译符号信息以及可能产生的错误和警告信息。必须提供文件的全名，包括扩展名在内。如果文件已经存在，那就被截去重新使用。这个选择对于浏览较多的错误信息是很有用的。

filename 这是在命令行中唯一的必须提供的域；它指出要被编译的C文件。但最后的.C不要提供，编译将自动提供之。注意编译只能编.C的文件，如果不是这类文件，编译将略过它去找"name.C"文件。(#include文件却必须提供全名）如果没样其他目录提供，则使用省缺的目录（directory）。quad文件以与源文件同样的形式在同一目录中创建，除非使用了‘-o’的选择。文件名中的字母字符既可以是大写也可以是小写。

options 编译时的选择项是用减号‘-’后随一字母刻划。字母必须是小写的；相应的大写字母将不起作用。每一个选择必须分别说明，用一个‘-’和字母（不能象有些UNIX那样可以组合）。目前的选择包括：

-a　使编译作出最坏情况的决定，而消除任何基于对指针优惠假设的优化。一般地，编译假设通过指针访问目标是与直接访问同一段中程序目标不同的。这个选择消除了这个假定。-a选择在执行下一语句前强行完成所有赋值语句（即对实际内存的存取）。一般地，赋值的代码生成使一个值取到寄存器中，但它可能不立刻保存；-a标记强行进行保存操作。这对下面几点是重要的：（1）联合，一个值存进去然后立刻进行检测或通过另一个成员传送到一个函数中；（2）实时处理，共享数据值用于“锁”字，赋值语句的立刻执行对序列动作是关键的；（3）内存图（memory—mapped）的I/O赋值，值必须在同一内存单元中重复保存。

-b　对所有的增量运算，强行对齐字节。第一步一般将所有的指针，结构和联合对齐在字界上。

-c 使注释按无嵌套处理。MCC假定注释可能是嵌套的，这使得很容易地将大段的代码注出来。这个选择允许用户强制编译成为标准的、无嵌套的操作方式。

-d 使得调试信息包括在quad文件中。特别地，行分割符记录用常规的quad点级。如果使用了该选择，第二趟编译创建OBJ文件，它包括有关输入到程序段增量的行数的信息，然后这个文件可以通过目标模块反汇编来处理，反汇编（OMD）产生一列最初的源文件，它表明了所产生的机器语言指令（见1.1.6）。

-iprefix 指出#include文件是准备通过加串"prefix"的文件名来查找的，除非在#include中的文件名已经由驱动或目录标识符加上前缀。如果"prefix"是一个字符，则加一冒号；所以-ia会成为"a:"。顶多能确定4个不同的-i串。注意，未加前缀的#include文件名是首先在当前的目录中查找，然后加由-i选择项确定的前缀，其顺序是与命令行提供的一致的，从左到右。

-mM 使编译产生特定内存模式的代码。模式以单字母说明，大写小写均可，或以模式名字，或以数字指示器0～3（S=0，P=1，D=2，L=3）。模式说明必须紧靠"m"，不能有空格。

-n 使编译可允许39个字符的标识符，包括#define符号。省缺的标识符长是8个字符。

-oprefix 指出输出（.Q）文件名是由输入文件（被编译有.C的文件）加"prefix"。如果prefix是单个字符，应加冒号":"；因此-ob应加"b:"。任何加在输入文件名上的驱动器和路经前缀在加前缀之前统统都要扔掉。

-s 改变方式代码，它是为在D和L模式中四字节的指针而产生的，参见1.4.5。

-x 改变省缺的外部说明存储类（在函数体外所作出的说明）从外部定义到外部参考。一般的外部说明的含义（显式存储类并不是为它提供的）是为目标定义存储并使它可以在其他文件中访问：外部定义。-x选择使这种说明这样处理：由于"extern"关键字仿佛使它们提前了，那就是说，被说明的目标在其他的文件中表出。这种选择是为用BDS C编译写的程序而提供的。详细参考附录C中"CP/M程序的转换"。

例子

MC1 XYZ -ob -x

将文件XYZ.C作为输入进行第一趟编译，并在B:上创建XYZ.Q文件，并将所有没有存储类的外部说明解释成"外部"说明。

MC1 =4096 >tns.err tns

将文件TNS.C作为输入文件进行第一趟编译，并在当前的目录上创建TNS.Q文件，置栈大小为4096字节，创建一个TNS.ERR文件以存放编译过程的信息。

MC1 XYZfile -ib:\headers\

将XYZ FILE.C作为输入进行第一趟编译，并在当前的目录上创建XYZFILE.Q文件。任何#include文件如在当前目录中找不到的话，将在B:\HEADERS中查

找。注意前缀尾部的反斜线'\'紧靠-i标志；编译并不自动提供。

1.1.2 第二趟编译

编译的第二趟读由第一趟生成的中间文件并按MS—DOS的格式生成目标文件。关于详细的处理过程参见1.3.2。调用第二趟编译的格式是

MC2 filename [options] <RET>

其命令格式是与第一趟编译十分相仿的。也可以使用栈大小优先和表文件选择，因为一般它们没有什么大的用处，故此处就不再赘述了。注意哪一趟编译也不对标准输入作任何处理，所以'<'选在二趟编译中都不起什么作用（参见1.1.4中关于一般C程序执行选择）。

filename 该域必须有。它指出为其产生的中间文件名。该中间文件名是由第一趟编译生成以.Q作为其扩展的文件，filename不应该有.Q扩展名。第二趟编译将自动提供。字母字符可以大写或小写。除非提供其他的路径名，否则按省缺路径处理。如果没有指出-O可选项的话，则目标文件在与输入同样的目录上生成文件。

options 编译时刻选择是用减号'-'后随一字母指出的。字母必须是小写；相应的大写字母将不起作用。每个选择必须分别确定；'-'和字母（即它们不能象UNIX那样可以组合在一起）。当前的选择包括：

-ggroup 指出名字"group"是用于.OBJ模式中的代码组。"group"不能超过15个字符（包括15）且必须紧贴-g（不能有空格）。

-oprefix 指出输出文件（.OBJ）名是由输入名加"prefix"构成。如果"prefix"是单个字符，则要加":"；因此-ob意即"b:"。任何前缀于中间文件的目录或驱动器号在加前缀以前统统扔掉。

-ssegment 指出名字"segment"是用于.OBJ模块的代码段。"segment"不能大于15个字符，且必须紧靠"-s"。

-v 使代码生成器略掉到每个函数入口处的代码，这些函数是检查栈溢出的。

-g和-s选择是为了强制省缺代码组和段名的。只有需要与每一个特定的应用接口的用户（用别的语言什么的）才需要使用这些选择。

例子

MC2 u790 -oc

将U790.Q作为输入进行第二趟编译，并在驱动器C：上生成U790.OBJ文件。

1.1.3 程序链接（linking）*

当一个程序的各个源模块编完之后，它们必须链接到一起构成一个可运行文件。这一步由于若干原因是必要的。其一，第二趟编译产生的目标是不可执行的。其二，大多数程序利用没有定义在当前模块中的函数；这些程序在可运行之前，它们必须与其他模块链接在一起。这些外部函数可由用户定义，这种情况下，外部函数必须编译成.OBJ文件，或者它们按照编译提供的库来定义。（可移植程序在第三章中叙述；仅在MS—DOS下定义的

* 链接（link）也有译作连接或联接的——译者注。

程序在1.6中叙述)其三,虽然C一般定义了一个C程序的执行点所谓"main",仍然有许多与系统相关的处理,这必须在调用"main"之前完成。完成该过程的模块当链接时被加在程序中。

虽然通常连接的概念包括外部函数的调用,C也允许函数存取在其他模块中定义的数据。这种访问是可能的,因为由目标代码支持的外部链接机构将外部符号与一内存地址联系了起来。该符号是用于说明C程序中目标的一个标识符。程序员必须很小心地说明目标,它在定义它的模块和访问它的模块中具有相同的特性,因为链接器不能验证所访问的类型——它简单地使用外部符号与内存访问连系起来。作为公共外部说明的包括文件的使用一般会防止这种错误的发生。

链接过程中,按照一般的感觉,要求指出一个程序的所有的各个部分,直接的或间接的,以便作为链接的输入。要求三种类型的输入:

1. 起始文件(即CS.OBJ,CP.OBJ,CD.OBJ和CL,OBJ中之一)必须被指出,作为链接程序所含的第一个模块。该文件定义了用MCC编译的所有C程序的MS—DOS入口点。

2. 由用户产生的函数必须刻划为欲包括的附加模块。这些模块必须包括主模块,以及任何定义在其他源模块中的附加函数。

3. 库文件(MCS.LIB,MCP.LIB,MCD.LIB或MCL.LIB)必须说明为在链接时查找的库。

这些输入由以下说明:

1. "CS"(或"CP"或"CL")成为在链接时的第一个模块。

2. 包括在链接命令中用户目标文件的名字(无.OBJ扩展)。

3. 打入"MCS"(或"MCP"或"MCD"或"MCL")以响应从链接器来的"Libraries"提示符。

注意,在第二步,在"LINK"命令包括的文件必须有一个是主模块。

如果链接程序没有找到在"LINK"命令中提到的OBJ文件,它将停止处理,也不产生EXE文件。如果链接程序不能找到在确定的目标文件中要访问的外部项将出现错误条件。这种情况下,你会看到下面的信息:未满足的外部访问,其后是一列未满足的外部名字。不要期望执行有未满足的外部访问的程序,除非你有把握那些没有的函数将不会调用。

参看1.2.2关于外部名字的讨论。参看1.5关于用在这个实现中的目标代码特点的技术描述。如果提供给你的系统的链接版本有与此处的提示符不同,看看Microsoft LINK链接装入用户指南,你的程序包中有这些内容。一般来说,对其他的提示符的不予响应是正确的。由于链接器允许生成公共符号图(map),你可创建MAP文件并看看在结果装入模块中表示的各项。

所有起始模块版本的源程序是C.ASM。

例子

```
LINK CS XYZ QRS
  Run File  [CS.EXE] : XYZ<RET>
  List File [NUL.MAP] : <RET>
  Libraries [.LIB] : MCS<RET>
```

执行链接程序，产生XYZ.EXE作为可执行程序，在程序中包括XYZ.OBJ和QRS.OBJ。

1.1.4 程序执行

当一个C程序执行时，开始先调用"main"。在它接受控制权之前，要为它完成二项重要的服务性工作：

1. 要分析执行程序的命令，以命令行中来的信息作为"main"的参数。完成的分析和提供的参数的特性下面将详细讨论。这个特点将使得很容易将命令行输入到程序。

2. 缓冲正文文件"stdin"（标准输入）"stdout"（标准输出），和"stderr"（标准错误）被打开，且程序可用它们。一般情况，这三个文件都指向用户的控制台，但是"stdin"和"stdout"可能通过命令行选择指向别的地方，这将下面叙述。这个特点使得程序的使用具有灵活性，该程序与使用标准的"getchar"和"putchar"宏的正文输入，输出文件一起工作。

执行C程序的最简单的方法是打入EXE文件的名字（不带.EXE），后随<RET>（回车）。由于命令行提供了一个方便的方法将输入送到程序，所以一个程序的执行要求常常包含其他信息。执行C程序的命令行的一般格式是：

pgmname [=stack] [<infile>] [>outfile] [args]<RET>

"pgmname"之后的那些是可选的，就象[ ]所表示的那样。各种附加项如果有的话，必须出现在程序名之后所有命令行参数之前。

pgmname　该域指出了要执行的程序的名字；这是在链接时生成的EXE文件的名字。打入时它不能有.EXE。

=stack　第一个可选域是用于确定一个十进制字节的当程序执行时使用的栈。如果该域省缺则隐含为2048字节，栈大小是在等号后随一十进制字节。任何说明为"auto"的目标都从栈中分配，但是用于这些分配的内存当有内存申请说明的函数返回到调用者时是随便的。一般地很难预测一特特的程序究竟用多大的栈空间。命令行中的栈选择允许用户调整用于栈的内存量而不必重新编译程序。用于栈的内存影响由各种库函数作动态申请时的有用内存量，库函数将在3.1中描述。关于C程序之结构可参见1.5更详细的内容。

<infile　第二个可选域是文件或标准输入（"stdin"）指向的设备。这个选择仅当执行的程序实际使用标准输入时才是有用的（即，它使用"getchar"或"scanf"或进行明显的"gefc"或"fscanf"函数调用处理正文输入）。文件或设备名必须紧跟在"<"之后；对于一个包括扩展名的文件，若有的话，必须要说明清楚。参见1.6.2中关于有效设备名表。文件必须存在，否则程序将中止执行并显示"Can't open stdin file"信息。

>outfile　第三个可选域是文件或标准输出（stdout）指向的设备。这个选择仅当执行的程序实际使用标准输出时才是有用的（即，它使用"putchar"或"printf"或明显使用"stdout"产生调用"putc"或"fprintf"）。文件名或设备名必须紧跟在">"字符之后；如果给出了一个文件，则必须给出全名。参见1.6.2中关于有效设备名表。文件被打开作为新文件，它将该文件原有的内容全部抛掉并产生一个

空文件。如果确定的文件名是无效的，或没有足够的空间创建新文件，程序将中止执行并显示"Can't create stdout file"信息。

如果使用了两个>而不是一个，文件打开作为附加。任何输出将加在文件其尾。这个选择用于累计记录。如果不存在就创建新文件 。

args　程序名之外的任何附加域，不是上述选择域中之一，将被取出并当作两个参数传到函数"main"之中：

```
main (argc, argv)
int argc;                    /*参数个数*/
char * argv[ ];              /*指向参数串的指针数组*/
```

每个参数串以空字节结束。在大多数支持C的系统里，"argv[0]"是调用程序的名字。MS—DOS中，程序名并不是容易可用的，虽然其他所有从命令行中来的参数是可用的。哑元"argv[0]是提供了（按照"argv[0]"所有的程序命名为"C"），而"argv"的相继元素也被适当定义了。在"argv"中的参数，其顺序与命令行中的顺序一致。注意可选项栈和文件并不包括在"argv"中。

虽然以上所有的特点是为了方便写在MS—DSO下的实用程序，但许多库I/O函数被强制成为程序的一部分，就是因为这一种处理方式（特别是打开带缓冲的输入输出文件）。对于欲使用带缓冲的I/O函数的程序而言，这倒不是什么问题，即使这些程序加了许多的代码字节到被链接的程序上。一定要关心程序大小的用户以及不使用这些功能的用户能够通过提供一个特殊的"-main"版本而避免包括多余的模块，"-main"为调用"main"的库函数，详细见1.6.4。

例子

```
CPROG =8000 <INPUT.R PQP 12
```

执行CPROG.EXE文件，栈容量为8000字节，标准输入为"INPUT.R"。"main"函数将得到一个值为3的"argc"，在"argv"数组中有串"C"，"PQP"和"12"。

```
errlog>>errors.log data
```

执行errlog.EXE文件，将"stdout"与ERRORS.LOG联接，增加信息（加在文件尾）。将给"main"函数提供有2值的"argc"，在"argv"数组中是串"C"和"data"。

1.1.5 函数提取实用程序

由于编译对所有定义在源文件中的函数产生单一的不可分割的目标模块，所以提供了函数提取实用程序以便一组小函数能放入一个源文件中且为它们单独产生目标模块。实用程序通过提取源正文作为单一指定的函数而工作，因此创建一个能够编译的源模块以产生一个仅定义在那个特定的函数中的目标模块。

对该实用程序有点疑问的程序员会发现下面的例子是有帮助的。假如一个用户有一个模块STRING.C，其内定义了若干个串处理函数，以及调用这些函数（如strcnt）的程序。如果STRING.C作为一个单一的源模块编译，那么结果目标模与其他若干函数一起定义"strcnt"。当一个程序被链接时，包括了"strcnt"的机器代码（当STRING.C编译时，作为产生的目标模块的一部分），但是所有其他函数的目标模块的代码也包含了进去，虽

然程序并不使用它们：只有通过将"strcnt"编译成定义它源模块中的唯一的函数，编译才能产生只定义那个函数的目标模块。FXU能用于产生这样的源文件。

调用函数提取程序的命令格式是：

FXU [<header—file][>output-file] file name function

各种命令行描述子按它们在命令中的顺序表示；可选的描述子表示在方括号中。前二个可选项是所有C程序的通用命令行选择的部分（见1.1.4）。

<header—file 第一个选择确定了当指出的函数找到时要考贝到输出文件中的文件。在任何函数的正文被写入以前将整个文件考入。如果只有函数本身要写到输出文件中，则应该用<NUL选择。如果该选择省略，正文将从用户的控制台读入且将它考到输出文件中直到打入control-Z为止。

>output—file 第二个选择指出了将包括被提取的函数正文的输出文件（如果有的话，冠之以头文件正文）。如果没有这个选择，正文就显示到用户控制台上。

filename 指出包括要使用的函数的文件名。

function 指出要被从指定文件提取出来的函数名。函数名必须与其定义的名字完全一致，除非在大小写字母上有别。

函数提取实用程序计算定义在函数体内的括号以便确定函数何时达到结尾。虽然它能够识别注释，在计算括号时不会出错，但它假设注释可以嵌套，该假设与编译的假设一般是一致的。但是编译可由命令行的选择要求将注释处理成非嵌套的。FXU没有这种选择。

提取的正文由函数名后括号之间的所有字符组成，甚至包括提取的函数的前后花括号"{}"。如果指出的函数是定义在源文件中的第一个文件，那么从文件开始到函数的后花括号"}"之间的所有字符都包括了进去。注意访问定义在源模块中的外部数据项的函数不能很容易地用函数提取实用程序所处理。但正如下面的例子所说明的，头文件选择能用于避免这种限制。

如果指出的函数在指定的源文件中没有遇到，则输出文件中将包括一条错误信息"指出的函数没有找到"。注意FXU仅在一个函数上操作，而不是一列函数上。但是定义多于一个提取函数的源模块能够被生成，通过重新执行FXU，然后使用CAT程序组合所提取的正文即可，CAT作为源文件中的一个例子。

所提供的FXU版本当向下扫描的时候使用内部缓冲去存储函数中的字符。缓冲的大小可以扩大，只要对源文件FXU.C作一稍稍修改即可，如果是必要的话。

例子

FXU <NUL STRING.C strcnt

从正文文件STRING.C中提取名叫strcnt的函数；并不包括前面任何正文；且将提取出的正文显示到用户的终端台上。

FXU <IOS.H>INPUT.C IOFUNC.C input

从正文文件IOFUNC.C中取出称为"input"的函数并将文件IOS.H中的正文加到输出中；将结果正文写到INPUT.C中，如果在IOFUNC.C中的每个函数访问外部地址"flag1"和"flag2，且需要从标准I/O头文件中来的信息，那么IOS.H应包括正文

```
#include<stdio.h>
```

```
extern int flag1, flag2;
```

同样的技术可用于那些需要更高代价的外部访问的函数。

1.1.6 目标模块反汇编

对于那些想在机器代码一级调试C模块的程序员来说，目标模块反汇编提供了一组为特殊的C源模块产生的机器语言指令。如果模块用－d选择来编译以便将行号/增量信息包括在目标文件中，那么反汇编能产生解释了的源代码行。这些指令(代码)能与程序的链接图联系起来用MS—DOS调试程序完成交互的调试。当然这个程序的实用只限于那些程序员，他们对8086体系结构和指令系统是熟悉的。

调用目标模块反汇编的命令格式是

OMD [>listfile][options] objfile[textfile]

命令行中的各项是按其出现的顺序表示的。可选的描述项用方括号[ ]括起来。

>listfile　第一项选择用于将由OMD产生的结果指向特定的文件或设备。如果该项省缺，则输出到用户的控制台上。

options　可指出四种强制的选择；每一种由一个'－'后随一字母组成，它指出强制的种类和确定强制值（override value）的十进制数字串。在任何单一选择中，不能有空格，但每一个选择必须独立地说明为一个域。有效选择是：

－Pnnn　代替为正处理的目标模块的程序段而提供的省缺大小。"nnn"指出十进制的存储字节数以对程序段进行分配。省缺值是1024字节。

－Dnnn　代替为正处理的目标模块的数据段而提供的省缺大小。"nnn"指出十进制的存储字节数以对数据段进行分配。省缺值是1024字节。

－Xnnn　代替能由OMD处理的外部项的省缺最大数；这个数分别提供给外部定义和外部访问。"nnn"指出了能被处理的外部项的十进制数。省缺值是200。

－Lnnn　代替行号和增量信息表的省缺尺寸。这些表仅当目标文件用－d选择处理时才使用；来自文件的行号/增量信息放在这些表中。省缺尺寸（它定义了能被处理的行号/增量对的最大值）是100。

objfile　确定目标文件名。它由编译产生，但是将由OMD处理。即使扩展.OBJ也必须包括在文件名中。

textfile　确定C源代码文件名字，它要按照反汇编指令进行列表。如果有该选择，则在MC1编译中必须用－d可选项编译源文件。文件中的扩展.C也必须包括进去。

OMD仅处理一个目标模块。在产生表项之前将整个模块读出并装入内存中去。各种强制（override）选择对处理很大的目标模块和减少OMD在系统中所需的内存量是十分有用的，因这些系统中的内存是吃紧的。

如果使用了"textfile"可选项，只将一个指定文件中的原正文列了出来；如果它访问任何_include文件，则这些文件不会被列出来。对"textfile"选择的一些限制是需要注意的。首先为"FOR"语句的第三部分产生的代码放入循环的底部；在循环结束后该代码出现在下面一个语句的前面。第二，编译要推迟保存寄存器直到最后可能的时刻，因此赋值语句的代码常常仅由将值取到寄存器的代码组成；寄存器其后才被保存。最后进入到函数的入口的代码常常出现在定义那个函数的源行前面。所以检查外层代码对于确定为一个