C語言程序

参考手册

用于8086和8088微处理机以及MS—DOS操作系统

广东省中梅电脑开发公司资料服务部译

一九八四年九月

译 者 的 话

C语言程序设计参考手册是适用于8086和8088微机以及MS—DOS操作系统，如IBM—PC微机就是其中之一。

C语言是世界上比较新颖的有效的计算机语言。为了尽快使国内广大工程技术人员和广大师生掌握这种先进的计算机语言，我们引进了C语言程序设计参考手册和操作磁盘，并把参考手册翻译出版，供大家学习、掌握C语言以及掌握C语言程序设计的编译、连接和库管理的方法。这本参考手册是技术性很强的参考书，它对决定C语言程序设计的文本有广泛参考价值，C编译操作磁盘是包含了整个C编程语言的程序包。所以，初学者在使用本手册之前，要学习C语言编程序的基本知识，然后，参考本手册，利用C编译操作磁盘，边上机边学习，这样，可能更为有效。

由于我们翻译水平所限，错误难免，請大家批评指正。

译 者

目　录

第 一 篇

C语言程序设计编译程序
参 考 手 册

（适用于8086和8088微处理机以及MS—DOS操作系統）

前　言

本手册给出了实施一种称为C语言的高级程序语言的编译程序即MicrosoftC编译程序的功能描述。手册并不打算讨论程序设计基本原理，也不打算讨论如何用C语言本身进行编程。本手册广泛地引证了由布赖思·W·克尼亨（Brian W. Kernghen）和丹尼斯·M·里奇（Dennis M. Ritchie）所著的权威著作《C程序语言》（Prentice—Hall, Inc. Englewood Cliffs, New Jersey, 1978），这部著作也包含在本软件包里。若没有这部著作，下面对C语言的叙述就不完整。这部著作也是这一语言的极好的入门读物。

本手册分为三章。第一章给出了这一特定实施方法的细节，开头部分是操作说明（如何运用编译程序，连接和执行程序等等）。第二章描述了编译程序所接受的语言，这种语言只在几个次要细节上与标准语言有所不同。在第三章里，按功能组给出了库函数，并给出了调用次序和例子。因为本手册打算作为一本参考书，所碰到的每一题目通常都详尽地给了所有技术细节，叙述中涉及到某些未读到章节是不可避免的，但这时将特别注明。若要获得这一编译程序的概貌，請阅读实施叙述（第一章）中的每一主要小节的开头部分，语言定义（第二章）一章的开头部分的语言提要以及库组一章（第三章）的开头部分的函数摘要。

第一章　MS—DOS的实施

Microsoft（R）　C编译程序是运行在Microsoft　MS（TM）—DOS操作系统。它接受用C程序语言（整个语言，而不是其中一个子集）写的程序并以Intel(TM)　8086目标模块格式产生适合Microsoft　LINK连接程序使用的浮动机器代码。程序库定义了一套广泛的输入输出子程序，这些子程序在MS—DOS操作系统下能执行大部分在克尼亨和里奇的著作①里所描述的与UNIX相容的标准函数。

8086指令系统很适合用于实现象C语言这样的高级语言，而Microsoft　C编译程序则产生充分利用了它的特点的机器代码。虽然8086体系能支援1兆字节的可寻址存储器，却缺乏直接有效地对此存储器寻址的能力。因此这种实施方案将C程序的长度限制程序段（函数）最大容量为64K字节，加上数据段（包括静态数据、自动或堆栈数据及动态可分配存储器）最大内存容量为64K字节。但即使有这一局限性，仍能编制出相当复杂和有较强能力的程序（包括编译程序本身）。

1．1　操作指令。

Microsoft　C编译程序MS—DOS操作系统里由下列文件所组成的程序包而提供：

MC1．EXE	C编译程序〔段1（phase　1）〕。
MC2．EXE	C编译程序〔段2（phase　2）〕。
FXU．EXE	函数抽出实用程序。
MD．EXE	目标模块分解程序。
C．OBJ	C程序输入／输出模块。
CC．OBJ	用于产生·COM文件的C．OBJ文本。
MC．LIB	运行时间库与I／O程序库。
MC．BAT	用于执行MC1和MC2的成批处理程序。
STDIO．H	标准I／O标题文件。
CONIO．H	控制台I／O标题文件。
CTYPE．H	字符类型标题文件。
MAIN．C	“main”的标准程序库文本。
TINYMAIN．C	“main”的缩略文本。
FTOC．C	华氏—摄氏采样程序。
CAT．C	文件连接采样程序。

C——指《C程序语言》一书　译者注

SIEVE．C　　Eratosthenes筛选法采样程序。

FXU．C　　实用功能抽出源程序。

CONIO．C　　基本控制台I／O功能。

C．ASM　　C．OBJ汇编程序源文件。

CC．ASM　　CC．OBJ汇编程序源文件。

IO．ASM　　采样汇编程序。

而且还包括了Microsoft　LINK连接程序和Microsoft　LIB库管理程序。

这些磁盘文件占据了磁盘存储器约250K字节。编译程序本身每一段各占据约50K字节的程序段，并且每一段还需要最少为4K字节的数据区。这样编译程序除了MS—DOS本身所需的以外还要大约64K字节的暂时存贮器，而且在编译大型源文件时需要额外的存贮器。

MC1和MC2组成了编译程序本体。它们各自执行编译过程的一个部分，并且必须用各别的命令来调用。当MC1完成了它的处理过程之后，并不自动装入MC2。在通常情况下如果源文件没有错误，MC2应紧接在MC1之后执行。成批处理文件MC．BAT用于以同样的文件名称（正常顺序）连续执行MC1和MC2。编译过程可图解如下：

file．C　→　MC1　→　file．Q

file．Q　→　MC2　→　file．OBJ

MC1读入一个源文件，这一文件必须具有．C扩展名，并且（在没有致命错误的情况下）产生具有．Q扩展名的同名中间文件。MC2读入由MC1产生的中间文件，并产生具有．BOJ扩展名的同名目标文件。当MC2完成它的处理过程时，．Q文件即被删除。通常，每一段都在与输入文件相同的驱动下产生它的输出文件。注意如果一个源文件定义不止一种函数，那么由它所产生的目标文件也一样。当一个程序被连接时，单个函数不能从目标文件分割开来，关于这点在1．3．2节有更多的叙述。

为了产生一个可执行的程序文件，必须提供OBJ文件作为连接程序的输入。在连接过程中，除了由用戶引入的任何OBJ文件之外，还必须包括两个专用文件。连接过程可图解如下：

C．OBJ　+　user．OBJ　+　．．．　+　MC．LIB　→　LINK　→　user．EXE

所需的专用文件是C．OBJ和MC．LIB。首先，C．OBJ文件必须被规定为LINK执行命令的第一个模块，这一模块定义了任何用Microsoft　C编译程序产生的程序的执行进入点和出口点。其次，MC．LIB文件必须被规定为库；这一文件定义了作为Microsoft　C软件包的一部分而包括在内的所有运行时间和I／O程序库功能。在连接时间内，用戶还必须规定要被包括在内的任何OBJ文件的名称以及将要被连接程序所产生的EXE文件的名称。

为说明程序的产生过程，这里举出了为编译、连接和执行华氏——摄氏采样程序所需的命令。在这一例子中假设所有的EXE文件（MC1，MC2和LINK）都位于同一个磁盘上。

命令用大写字母写出，虽则小写字母也同样好用。

第1步：打入下列命令执行编译程序的第一段

MC1 FTOC<RETURN>

注意没有提供.C扩展名（虽然如果提供了这一命令将能恰当地运行）

第2步：在MC1完成了它的处理过程后MS—DOS就发出提示符号，就可以打入下列命令执行编译程序的第二段

MC2 FTOC<RETURN>

在这里也没有规定扩展名，MC2提供.Q段扩展名。

第三步：在MC2完成了它的处理过程后，发出提示符时，打入下列命令执行连接程序

LINK C FTOC<RETURN>

注意C（指C.OBJ）被规定为LINK命令里的第一个目标模块。对于任何C程序的连接都需要这样。然后FTOC（指刚由MC2产生的FTOC.OBJ）被规定为附加的目标模块。对于其它连接程序的提示的响应如下：

RUU File〔C.EXE〕：FTOC<RETURN>

List File 〔NUL.MAP〕：<RETURN>

Libraries 〔.LIB〕：MC<RETURN>

这些响应使运行文件以FTOC.EXE为其名称，使程序跳过产生连接地址变换这一过程，而且使LINK程序对外部引用检索MC.LIB。

第四步：通过打入下列命令执行EXE文件：

FTOC<RETURN>

一个华氏温度值和它的摄氏等效值的表将写入用户控制台。

注意开头的两步骤可以由下面单个命令完成：

MC FTOC<RETURN>

这一命令运用MC.BAT成批处理文件连续执行MC1和MC2。也应注意FTOC.OBJ文件仍然存在，并也许应该被擦除。

对于编译、连接和执行程序的详尽说明在下面几节给出。对于两个编译程序段所进行的处理的详细讨论请参阅1.3节。

在描述各种命令行格式时，将使用“字段”这一术语来表述在命令行中非空格字符的一个次序。任选字段将用方括号括起起来（〔 〕）；在打入实际的命令时，不包括这些括号。研究每节末尾的例子就可以清楚实际命令应该是怎样的。

1.1.1 段1

编译程序的段1读入一个C源文件并产生一个称为四元组的逻辑记录中间文件。称进行的处理过程的更详尽讨论参阅1.3.1节。调用编译程序段1的命令格式是

MC1 〔=stack〕 〔>listfile〕 filename 〔options〕 <RETURN>

各种命令行区分符，它们在命令中必须依照的顺序示出。任选区分符用括号括起。前两个任选区分符是所有C程序的通用命令行任选区分符的一部分（参阅1．1．4节）。

=stack

第一个任选区分符是用于取消保留给堆栈的字节数目（C程序结构的完整叙述参阅1.4节）。缺席为2048（十进制）字节，这对于大多数的程序来说足夠的。如果出现，堆栈容量取消字段必须紧接段1名称（MC1）之后。它被规定为一个等号后随十进数目字（例如，=4096表示4096个十进制字节的）。

因为本编译程序运用归递方式处理C语句，所以具有许多嵌套的语句比线性顺序会使编译程序应用更多的堆栈空间。如果一个带有许多嵌套的语句（if里有if，if里又有if，如此等等）的源程序引起段1在一个编译程序中间莫明其妙地终止，或者为一个并不存在的错误而发生错误信息，或者表现出其它异常的行为时，增加堆栈容量可以解决这一问题。另一方面，我们有时会发现只是发生了一个人为的编译错误。报告这样的问题的过程請参阅附录B。在受到存贮容量限制的系统里，为了力图消除“存贮容量不足”的错误，可以減小堆栈容量。然而，这并不能保证编译必定成功，特别是如果堆栈容量減小到低于1024字节的话。

>listfile

第二个任选区分符用于将段1信息导向一个规定的文件。这些信息包括编译程序符号开始信息和任何可能产生的错误或报警信息。必须给出完整的文件名称，包括扩展名。如果一文件是已有的，它被截断并被重新使用。这一任选区分符对于检查错误信息长表时有用。

filename

这是唯一必须出现的命令行字段。它规定了要被编译的C源程序的名称。规定文件名时应该不用．C扩展名，段1自动提供．C扩展名。注意只有．C扩展名的文件才能被编译。如果规定了另外一些扩展名，编译程序会将它忽略，并试图找到“name．C”。（另一方面，#include文件必须用扩展名规定）除非规定了其它的驱动器，否则使用当前驱动器。在与源文件相同的驱动器下产生四元组文件，除非使用3—0任选区分符（见下文）。在文件名称里字母可用大写或小写给出。

options

编译时间任选区分符规定为一个负号后随一个字母。字母必须用小写打入。相应的大写字母任选区分符不起作用。每一个任选区分符必须用单独的负号和字母单独地规定（也就是说，它们不能象某些UNIX程序那样结合起来）。现行任选区分符有：

－a　使编译程序假设最坏情况的混淆现象，也就是放弃任何建立在有关指示字的有利假设基础上的优化过程。在通常的情况下，编译程序假设通过指示字引用的目标与在这一程序的同一区段直接引用的目标不同。这一任选分符取消了那种有利假设。－a任选区分符几乎是从来都不需要的，除非遇到要用指示字做十分复杂工作的情况。更多的资料可以在1．3．4节找到。

－b　对于所有区距计算强迫进行字节排列。段1通常把不是“char”的所有目标排列在一个字边界上。这保证了程序在一台8086上有效地取数据（在8086处理器的奇数字节边界上取一个字需要四个额外的时钟周期）。

设置这一任选区分符使我们能为8088处理器编出有效地利用空间的程序。它对于某些结构说明来说也是有用的，在这些结构说明中，字项必须相隔奇数字节区距，以便与特定的记录格式相适应（例如，MS—DOS所使用的FCB结构就包含有一个落在奇数字节边界的长整数）。

－C　使注解不用嵌套进行处理。MicrosoftC 编译程序通常假设注解可以用嵌套；这使得我们可以非常容易注释大段的代码。这一任选区分符使用户能迫使编译程序以标准的无嵌套的方式工作。

－d　使调试信息被包括在四元组文件里，特别是行分隔符记录被散布在通常的四元组文件里。如果使用了这一任选区分符，段1产生一个包括了输入行数与程序段区距之间关系的信息的目标文件，然后这一文件被目标模块分解程序（OMD）所处理产生一个原有源文件的表，示出所产生的机器语言指令（参阅1．1．6节）。

－id　从磁盘“p”读出所有#include文件，在这里“d”是大写或小写的单个字母，规定了一个磁盘驱动器（“a”用于A:，“b”用于B:等等）。驱动器字母必须紧接“－i”（中间不留空格）。通常，所有的#include文件都是从当前记入的磁盘中读出的。这一任选区分符使得在需要时可在其它一些磁盘读出#include文件。

－od　在“d”驱动器产生输出文件（四元组文件），在这里“d”是大写或小写字母，它规定了一个磁盘驱动器（“a”用于A:，“b”用于B:等等）。驱动器字母必须紧接“－o”（中间不留空格）。

－x　将外说明（在一种功能体外作出的）缺席存贮类从“外部定义”改变到“外部引用”。没有显式存贮类的外说明的通常意义是对目标定义一个存贮，并使它在其它文件里可见：外部定义。－X任选区分符使这样的说明就象在它们前加上了“extern”的关键字那样进行处理。也就是被说明的目标在其它一些文件里存在。设置这一任选区分符用于供用BDS C编译程序写成的程序。更详尽的资料請参阅附录C，

“Conversion of CP／M Programs”

例：

```
MC1 xyz －ob －x
```

用文件xyz．C作为输入执行编译程序的段1，在B:上产生文件XYZ．Q，并解释所有外说明，而不用“extern”说明存储类。

```
mc1 =4096 >tns.err tns
```

用文件TNS.C作为输入执行编译程序的段1，在现用驱动器上产生文件TNS.Q，将堆栈容量置定在4096＋进制字节上，并产生一个TNS.ERR文件包含所有由编译程序产生的

信息。

1．1．2 段2

编译程序读出由段1产生的一个四元组文件并以标准MS—DOS格式产生一个目标文件。有关所进行的处理过程的更详尽的讨论請参阅1．3．2节。调用编译程序段2的命令格式是：

MC2 filename 〔obtions〕〈RETURN〉

这一命令格式与段1的非常相似。也可以使用堆栈容量取消任选区分符和 listfile任选区分符，但是一般说来它们的用处不那么大，因此在这里就不详述了。注意编译程序的两个程序段都不对标准输入作任何的处理，所以<任选区分符在这两个程序段里都没有作用（对于一般的C程序执行任选区分符的情况，請参阅1．1．4节）。

filename 这一字段必须存在，它规定了中间文件的名称，代码是为它而产生的。这一中间文件一个由段1产生的、带有．Q扩展名的四元组文件。文件名应不带．Q扩展名。段2 自动提供扩展名。字母可以用大写也可以用小写除非规定了某些驱动口名称，否则使用缺席驱动口，而且除非使用了－O任选区分符，否则目标文件在与四元组文件的同一驱动口下产生（见下文）。

options 编译时间任选区分符规定为一个负号后接一个字母。字母必须用小写，大写字母无效。每一个任选区分符必须单独用单独的负号和字母规定（也就是说，它们不能对在某些UNIX程序那样结合起来。）现行的任选区分符包括：

－od 在〃d〃驱动口产生输出文件（目标文件）在这里〃d〃是单个大写或小写字母，它规定了一个磁盘驱动器（〃a〃规定A：〃b〃规定B：等等）。驱动字母必须紧接在〃－O〃之后（不留空格）。

例如：

mc2 u790 －oc

用文件U790．Q作为输入执行段2，在驱动器C：上产生文件U790．OBJ。

1．1．3 程序连接

当一个程序的所有部分源模块被编译后，必须把它们连接在一起来组成一个可执行的程序文件。有几个原因要求进行这一步骤。首先，由编译程序段2所产生的目标文件的状态并不适于执行，其次，大部分的程序都使用现在模块里没有定义的函数，在这样的程序能夠被执行之前，它们必须与其它模块“连接”。这些外部函数可以由用戶定义，这时它们必须被编译并作为OBJ目标文件可供利用，或者它们可以由与编译程序一起提供应用的库定义。（可移植函数功能在第3章中描述；其它只在MS—DOS里定义的在1．5节中描述）。第三，虽然通常C语言定义称为〃main〃的函数是一个C程序的执行点，但是常常有相当大量的随系统而异的处理过程必须要在〃main〃实际被调用之前进行。进行这一处理过程的模块在程序被连接时被纳入程序里。

虽然通常的连接概念包括了外部函数调用，C语言也允许访问在其它模块中的定义的数据单元的函数。这类引用是可能的，因为由目标代码所支援的外部连接机构一个外部符号与一个存贮单元相联系。这一符号是用于在一个C程序中说明目标的标识符。程序员必须小心地在定义目标和访问这一目标的两个模块中以相同的属性说明这一目标，因为连接程序的不能检验所作的引用的类型——它只是用外部符号连接各次存贮器访问。对于普通的外说明使用include文件常常可以避免这类错误。

一般来说，连接过程要求一个程序的所有组成部分都要直接地或间接地有规定，作为连接程序的输入。需要三种输入。

1． C.OBJ文件必须被规定为连接程序所包含的第一个模块。这一文件对于所有用cMirosoft C编译程序所编译的所有C程序定义了MS—DOS进入点。

2． 由用户所产生的函数必须被规定为要被包含的附加模块。这些模块包含了主模块以及任何在其它源模块中定义的附加函数。

3． 文件MC.LIB必须被规定为在连接过程中被检索的程序库。

这些输入由下列各项规定：

1、在“LINK”命令里使“C”成为第一个模块。

2、在规定“C”之后，在“LINK”命令里要纳入用户的目标文件的名称（不用.OBJ扩展名）。

3、在连接程序给出“Libraries”的提示符时打入“MC”。

注意在第2步骤时，“LINK”命令中所包含的其中一个文件必须是主模块。

如果连接程序不能找到在“LINK”命令中所提到的其中一个OBJ文件，它就会停止处理，不产生EXE文件。如果连接程序不能找到在所规定的OBJ文件中所引用的所有外部项目，也会产生错误情况。在这种情况下，我们会得到大意为存在“不满足外部引用”的信息，接着便是不满足的外部项目名称。不要试图以不满足的外部引用去执行一个程序，除非你有相当把握失却的函数不会被调用。

外部名称的讨论参阅1．2．2节。本说明中所使用的目标代码的技术描述参阅1．4节。如果你的系统里所提供的连接程序文本所具有的提示与这里所列举的有所不同，請查阅Microsoft LINK连接装入程序用户指南，这本手册包括在你的软件包里。一般来说对于其它的提示符的缺席响应是正确的。因为连接程序允许产生一个公用的符号地址变换，所以我们可以产生一个MAP文件，并看看在所得到的装入模块中所存在的组成部分。用CC.OBJ代替C.OBJ作为连接命令中的第1个模块也可以产生一个EXE文件，运用MS—DOS实用EXE2BIN，可以将这一文件转换成一个COM文件。这一目标文件（CC）定义了初始顺序与C.OBJ所定义的初始状态顺序是类似的，除了（1）没有定义STACK段，以及（2）不需要程序段安排。这里有三个应该注意的事项。首先，整个程序的容量，包括程序段和数据段在内，总的不应超过64K。第二，因为没有定义STACK类型，连接程序将发出报

警：

Marning: no STACK segment

并在得到检测出一个错误的信息情况下完成连接。这两个信息都可以被忽略，因为它们并不指示真正的错误情况，第三，那些包含具有与段有关的数值的汇编语言程序的程序不能转变为COM文件，即使在连接过程中使用的是CC.OBJ，因为COM文件不支援以段为基础的安排。

C.OBJ和CC.OBJ的源文件分别作为C.ASM和CC.ASM提供。

例如：

```
LINK  C  XYZ  QRS
      Run  File  〔C.EXE〕:XYZ<RETURN>
      List  File  〔NUL.MAP〕:<RETURN>
      Libraries  〔.LIB〕:MC<RETURN>
```

执行连接程序，产生XYZ.EXE作为一个可执行程序，并在程序中包含了XYZ.EXE和QRS.OBJ文件。

1．1．4　程序的执行

当执行一个C程序时，引入“main”功能就开始执行了。在它接受控制前对它进行了两个重要的服务。

1、执行程序的命令受到分解，并且来自命令行的信息作为参数提供给“main”。下面将详细讨论所进行的分解和所提供的参数性质。这一特性是设计用来使程序的命令行输入变得更容易。

2、缓冲正文文件“stdin”(标准输入)“stdout”(标准输出)和“stderr”(标准错误)被打开并可供程序使用。通常的情况下，所有三个单元都分配到用户控制台，但“stdin”和“stdout”可以由下列描述命令行任选区分符分配到别处。这一特性使我们在利用标准的“getchar”和“putchar”宏命令执行与正文文件输入和输出一起工作的程序时有更大的灵活性。

执行一个C程序的最简单的方法就是只把EXE文件的名称打入（不带.EXE扩展名）然后紧接着<RETURN>。因为命令行提供了一种方便的方式将输入提供给一个程序，所以一个程序执行请求常常包含其它信息。执行一个C程序命令行的一般格式是：

pgmname 〔=stack〕 〔<infile〕 〔>outfile〕 〔args〕 <RETURN>

在“pgmname”之后的每一项目都是任选的，所以用括号括着。各种附加的项目，如果有的话，可以用任何顺序规定，第一个字符使专用的任选区分符从普通的命令行自变量中识别出来。

pgmname 这一字段名是要被执行的程序。这是程序被连接时所产生的EXE文件名称。它必须指定不用.EXE扩展名。

=stack　第一个任选区字段用于规定在执行程序时为堆栈保留的十进制字节数。如

果这一字段不存在的话，所使用的缺席值是2048字节。堆栈容量用紧接在等号后的一个＋进数目字来规定。所有被“auto”说明的目标都从堆栈分配存贮单元，但是当说明它们的操作返回它的调用程序时，这些被分配用到的存贮器就恢复自由了。这种分配的动态性质通常使我们很难预知某一特定程序实际需要多少堆栈空间。命令行上的堆栈容量任选区分符使用户能调节保留给堆栈的存贮器总数而无须再编译这一程序。保留给堆栈的存贮器影响由3．1节中的描述的各种库函数所进行的动态分配所能利用的存贮器总数。有关C程序的结构的更多信息参阅1．4节。

＜infile　第二个任选字段命名了将被标准输入(“stdin”)赋值的一个文件或设备。这一任选字段只在被执行的程序实际使用标准输入时有用（也就是说，它利用“getchar”或“scanf”处理正文输入或调用“stdin”产生显式输入“getc”或“fscanf”）。文件或设备名称必须紧接前面的一个＜字符。如果是文件，则必须规定全称，如果有扩展名，必须包括在内，有效的设备名称的表参阅1．5．2节。文件必须存在，否则程序将出现异常终止的错误信息“can't open stdin file”

＞outfile　第三个任选字段命名了将被标准输出(“stdout”)赋值的文件或设备。这一任选段只在被执行程序实际使用标准输出时有用（也就是说，它利用“putchar”或“printf”产生正文输出或调用“stdout”产生显式输出“.utc”或“fprintf”）。文件或设备名称必须紧接前面一个＞字符。如果被规定的是一个文件，则必须给出包括扩展名(如果有的话）在内的全名。有效的设备名称的表请参阅1．5．2节。这一文件作为新文件被打开，如果先前已经存在内容，则它舍弃先前的内容并产生一个空文件。如果所规定的文件名称无效，或者没有足够的目录空间供产生新的文件，这一程序就以
“Can't create stdout file”的错误信息被异常终止。

如果使用了两个＞字符而不是一个，则这一文件被打开是用于增补文件，而任何的输出都被附加在文件末端。这一任选字符在积累存入信息时有用。如果不存在这一文件，则这一文件就被产生。

args　任何在不属于上述三个任选字段中的程序名称之外的附加字段都被抽出并作为两个自变量被传送到“main”函数：

```
main( argc, argv )
int argc;              /☆ number of arguments ☆/
char ☆argv[ ];         /★ array of ptrs to arg
                       s rings★/
```

每一变量串都以一个零字节终止。在大多数支持C语言的系统里，“argv[0]”是调用这一程序的名称。在MS—DOS操作系统里，程序名称不是马上可供应用的，虽然所有来自命令行的其它信息是如此。因此提供了“argv[o]”这一个哑元（根据“argv[o]”所有的程序都命名为“C”），但是随后的“argv”元素都被适当地定义。在“argv”里的变量