Microsoft C 程序库参考手册

中国科学院　软件研究所　开发公司

1987年5月

Microsoft C 程序库参考手册

目　录

第一部分　概　要

第一部份　概要

第一章　引言

1.1　关于C程序库

Microsoft C程序库 (The Microsoft C Run-Time Library)(简称"C库")是用于C语言程序中的200多个预先定义好的函数和宏。由于提供了以下功能，使程序设计变得容易：

1. 与操作系统功能的接口（例如打开和关闭文件）；
2. 既快速又高效地实现程序设计中常见任务的函数（例如字符串处理），这就节省了程序员用于写这些函数的时间和精力。

因为C语言本身不提供输入、输出、存储分配和进程控制等基本函数，所以使用C语言的程序员必须依靠程序库来提供这些函数。因此，在用C语言进行程序设计中，程序库就特别重要。

在设计 Microsoft C库的函数时，保证了它们在 MS-DOS 和 XENIX 或 UNIX 系统之间的最大程度的兼容性。在这本手册中，凡涉及到 XENIX 的内容，对 UNIX 和类似 UNIX 的系统同样适用。

MS-DOS 中的C库中的大多数函数同 XENIX 中的C库中同名的函数功能上兼容。如果你对可移植性感兴趣，请参阅附录B，"XENIX 和 MS-DOS 的通用程序库"。这个附录列出了 MS-DOS 中的C库中特有的函数，并且叙述了 MS-DOS C库中的函数与其在 XENIX 中的同名者在功能上的差异（如果有差异的话）。

为进一步提高兼容性，Microsoft C库中的数学函数都经扩充具有例外处理功能，处理方式与 UNIX 系统V中的数学函数相同。

为方便那些有兴趣利用 MS-DOS 特别功能的程序员，程序库包含了与 MS-DOS 接口的函数，这些函数使C程序可以援用 MS-DOS 中的系统调用和中断。C库中还包含了控制台输入输出函数，以便从用户控制台上有效地进行读和写。

为了利用 Microsoft C编译程序的类型检查功能，对伴随C库的嵌入文件进行了扩充。除去库中的函数和宏所需要的定义和说明之外，嵌入文件中还包括带有形式参数（形参）类型表的函数说明。类型表使系统能对库函数的调用进行类型检查。这个特点对于查出由于函数的形参与实参类型不一致造成的程序错误帮助极大，建议尽量加以利用。

但是你也不是非使用参数类型检查功能不可。在嵌入文件中带参数类型表的函数说明包含在预处理段的#if defined ( ) 块中，仅当你定义了标识符 LINT-ARGS 后才起作用。

为了给库中的函数提供参数类型表，在C程序库的标准嵌入文件表中增加了几个新的嵌

入文件。这些新的嵌入文件的命名尽量做到了与 ANSI（美国国家标准学会）的C语言标准以及 XENIX 和 UNIX 中的名称一致。

1.2 关于本手册

本手册描述了 Microsoft C 库的具体内容。我们假定你熟悉C语言和 MS-DOS。我们还假定你知道怎样在你的 MS-DOS 系统中编译和连接C程序，而且会用环境变量建立起编译和连接的环境。如果你对编译、连接或建立环境有问题，请参阅《Microsoft C 编译器用户指南》一书。如果你不熟悉C语言，请参阅《Microsoft C 语言参考手册》。

《Microsoft C 程序库参考手册》有两大部分。第一部分是“概要”，介绍C库。它所讨论的是适用于整个库的一般规则并概括了该库的各成分。

第二部分是“参考手册”，按英文字母顺序叙述库中的函数，以便快速查阅。一旦你熟悉了有关库的一般规则之后，你就会主要使用这一部分。

第一部分余下章节的内容如下：

第二章“使用C程序库”，给出了理解和使用C程序库的一般规则，指出了适用于某些库函数的特殊要求。建议你在使用C程序库之前先阅读这一章，当你对使用库的过程有疑问时也请再看第二章。

第三章“全局变量和标准类型”，描述了在库中定义、并为库函数所用的变量和类型。这一章还提供了一个说明（或定义）这些变量（或类型）的嵌入文件的相互参照表。你会发现这些变量和类型在你自己的程序中也有用。这些变量和类型在第二部分“参考手册”中也有相应的描述。

第四章“库程序的分类”，对库中的程序进行分类，并讨论每一类的特殊性。此章是作为第二部分的补充，使用户易于按照功能确定要用的库程序。如果你找到了所需的库程序的名称，你还需要阅读第二部分（“参考手册”）中的相应部分以获得详尽的描述。

第五章“嵌入文件”概括了程序库中用到的各嵌入文件的内容。

第二部分之后的附录提供了出错信息的详细说明和与 XENIX 兼容的库程序。附录A，“错误信息”，描述了使用库程序时可能出现的错误值和信息。附录B，“XENIX 和 MS-DOS 的通用程序库”，列出了 MS-DOS C 程序库中那些与 XENIX（和 UNIX）系统中的同名库程序功能上兼容的库程序，附录B还给出了这些库程序在 MS-DOS 和 XENIX 中的所有差异。通用全局变量和嵌入文件在附录B中也有讨论。附录C列出了中国科学院软件研究所开发公司为方便用户而附加的绘图库程序；附录D描述我们建议的增强C程序可读性的编码方法。

还有一点，在以下部份，“程序”、“函数”或“例程”都代表C函数。

1.3 记号的约定

以下是关于本手册中使用的记号的约定：

约定的记号	意义
省略点（…）	省略点用来表示程序范例中省去了一部分程序段。例如在下面的片断

中，两条语句间的省略点表示它们之间还有其他语句没有被写出来：

```
int x, y;
  ...
  y = abs(x);
```

省略点也表示在某一个项之后还可以出现更多的具有相同形式的项，例如，={expression[,expression]…}表示花括号中间可以有一个或多个由逗号分隔的表达式。

【黑体方括号】　黑体方括号在库函数的说明中表示可有可无的参数。

例如在：

int open (pathname, oflag[,pmode])；中，pmode两边的黑体方括号表示这个参数可有可无，而且如果有的话，须用逗号将其与前一参数隔开。

由于C语言中数组说明和下标表达式里也用到方括号，因此在谈及包含数组和下标表达式的文法和例子时，这些方括号用单线方括号表示。

举例说明：

```
char *args[4];
```

上例为一个有四个元素的数组的说明。'4'两边的方括号是C语言要求的。

“双引号”　双引号界出文中定义的术语。例如当“token”被定义时，它就出现在双引号中。

有一些C语言的成份如字符串也要用到双引号，这些双引号形如 " " 而不是“ ”。例如

```
"abc"
```

是一个C程序的字符串。

第二章　使用C程序库

2.1　引言

在使用C程序库时，只需要直接调用库中的例程，就象它们是在你的程序中定义的那样。C库例程是按编译以后的形式存放在库文件中的。

在连接时，你的程序必须与相应的一个或多个库文件连接在一起，以便连接被调用的库例程的代码，与C库进行连接的办法在《Microsoft C编译器用户指南》中有详细的介绍。

在大多数情况下，作为调用库函数的准备，你必须履行下面两个步骤或其中之一：

1. 在你的程序中嵌入一个指定的“.h”文件，许多库函数都要引用由某个嵌入文件提供的定义和说明。
2. 为那些返回值的类型不是int的库函数提供说明。因为如果没有说明函数返回值的类型，那么编译器就认为它返回值的类型为int。你可以通过嵌入包含有这些说明的C库文件或者在你的程序中给出显式的说明两种方法之一来提供必要的说明。

这些是使用C库函数的起码要求；你也许还得做一些其他步骤，如使编译器实现对函数调用进行参数类型检查。

这一章余下部分讨论使用库函数要做的准备和一些对某些库例程适用的特殊规则（如文件名和路径名的约定）。

2.2　鉴别函数和宏

在本手册中，“函数”和“例程”是作为同义使用的。事实上，C库中大多数例程都是C函数，即它们是由编译过的C语句组成的。但是有些例程是用“宏”实现的。一个宏是由C预处理命令#define定义的标识符，它代表一个值或一个表达式。和函数一样，宏定义中也可以带零个或多个形参。宏的定义和使用在《Microsoft C语言参考手册》中有详细的讨论。

C库里定义的宏的使用与函数相似：它们也带参数并返回值，而且调用方式也与函数相似。使用宏的主要优点是运行速度快；它们的定义在预处理阶段就被展开，省去了函数调用所需的额外时间。但正是由于宏要在编译以前被展开（即用其定义去替换），它们会使程序的代码的长度增大，在程序中宏出现次数较多时更是如此。无论函数被调用多少次，它们只被定义一次；与此不同，宏的每一次出现都要被展开。因此函数和宏提供了程序的运行速度和长度间的一种权衡。有时，C库为某例程提供了函数和宏两种实现方式，使你可以进行选择。

下表列举了函数和宏之间的一些重要区别：

1. 有些宏对具有副作用的参数的处理可能会不正确，这发生在宏定义中对那些有副作用的参数的计算多于一次的时候。请看表后的范例。

2. 宏标识符与函数标识符的性质不同。特别之处是，函数标识符的值代表地址，而宏标识符的值则不然。所以不能把宏标识符用在要求指针的情况下。例如假如你将一个宏标识符用作一个函数调用的参数，所传递的是该宏表示的值；假如将一个函数标识符用作函数调用的参数，传递的则是该函数标识符所表示的函数的地址。
3. 因为宏不是函数，所以它们不能被说明，同样也不能说明指向宏标识符的指针。因此无法对宏的参数进行类型检查。但编译器能够检查出提供给宏的参数个数不对的情况。
4. 用宏的方式实现的库例程是在库的嵌入文件中通过#define 定义的。要使用库里的宏，你必须包含相应的库文件，不然宏就没有定义。

用宏实现的例程在第二部分“参考手册”中将通过注释予以指出。为了确定带副作用的参数是否会引起问题，你可以在相应的嵌入文件中查看某个宏的定义。

范例

```
#include <ctype.h>
int a = 'm';
a = toupper (a + +);
```

上例用到C 库里的例程 toupper。它是用宏实现的；在 ctype.h 中它的定义为：

```
#define toupper(c) ((islower(c))?_toupper(c):(c))
```

定义中使用了条件操作符 (?:)。在条件表达式中，参数 c 被计算了两次：一次是为了确定它是否为小写字母，另一次是返回适当的值。这使得实参 a + + 被计算了两次，因此使 a 增值两次而不是一次。其结果是 islower 所处理的值与_toupper 处理的值不一样。

并非所有的宏都有这样的效果。你可以通过研究宏定义来确定某个宏是否会对副作用处理得当。

2.3 嵌入文件

很多库例程用到在别的嵌入文件中定义的宏、常量和类型。要使用这些例程，就必须在提交给编译器的源文件中嵌入那些指定的文件（用预处理命令#include）。

根据特定的库例程的需要，每个嵌入文件的内容都不同。但是嵌入文件通常包含以下项目的组合：

- 常量定义
  例如常量 BUFSIZ，用来确定带缓冲区的输入输出操作的缓冲区的大小，是在 stdio.h 中定义的。
- 类型定义
  有些库例程用数据结构作参数或者返回结构类型的值。嵌入文件中设置所需要的结构类型定义。大多数流式输入输出操作使用指向结构类型 FILE 的指针。FILE 是在 stdio.h 中定义的。
- 两组函数说明
  第一组函数说明给出了库函数的返回值类型和参数类型表；第二组说明只给出返回值的类型。对任何返回值类型不是 int 的函数都必须说明其返回值的类型（见第

2.4节“函数说明”)。提供了参数类型表则有可能对函数调用进行参数类型检查；请参看第2.5节“参数类型检查”对这个选项的讨论。

- 宏定义

  库中的有些例程是用宏实现的。这些宏的定义放在嵌入文件中。要使用某一个宏就必须嵌入相应的库文件。

在第二部分里介绍每一个库例程时都列举出了它需要的嵌入文件。

2.4 函数说明

每当你用到一个返回非 int 型值的库函数时，必须保证它调用之前被说明了。最简单的办法是嵌入那个含有该函数说明的库文件，使相应的说明出现在你的程序中。

在每个嵌入文件中有两组函数说明。第一组说明了函数的返回值类型和参数类型表。只有当你使用第 2.5 节中介绍的参数类型检查时，才需要包含这一组说明。因为实参和形参间的类型不匹配会导致严重的而且难以查出的错误。我们积极建议用户使用参数类型检查功能。

第二组只说明了函数的返回值类型。当不使用参数类型检查时就使用这一组函数说明。

你的程序中可以包含同一函数的几个说明，但这些说明必须相容。如果你有的旧程序中的函数说明没有包括参数类型说明，记住这条性质是很重要的。例如，假如你的程序中有如下说明

```
char *calloc ( ) ;
```

你就可以再包含下面的说明：

```
char *calloc (unsigned, unsigned) ;
```

尽管这两个说明不完全相同，但它们是相容的，故不会发生冲突。

如果你愿意的话，也可以自己提供函数说明，而不使用库里的嵌入文件。但是建议你参考嵌入文件中的说明以确保你的说明是正确的。

2.5 参数类型检查

Microsoft C 编译器提供了对函数调用进行参数类型检查的功能。只要在函数说明中描述了参数类型表，并且函数说明出现在该函数在程序中的使用或定义之前，就进行类型检查。参数类型表的形式以及类型检查的方法在《Microsoft C 语言参考手册》中有详细描述。

对用户自己写的函数，用户要为参数类型检查提供参数类型表。用户也可以在命令行中使用选项/Zg，使编译器为某特定的源文件中定义的所有函数产生一个函数说明表；然后将这个说明表嵌入用户程序中。请参阅《Microsoft C 编译器用户指南》中第三章“编译”中对使用选项/Zg 的详细讲解。

对于 C 库里的函数，你可以使用本节讲述的方法来进行参数类型检查。每一个库函数都在某一个或几个嵌入文件中有说明。每个函数有两个说明：其中一个有而另一个没有参数类型表。函数说明放在预处理段的#if defined ( ) 块中。如果你定义了一个叫 LINT-ARGS 的标识符，包含参数类型表的函数说明将会被处理并参加编译，从而可以进行参数类型检查。如果没有定义标识符 LINT-ARGS，不含参数类型表的函数说明将被采用，也就不会进行参

数类型检查。

在缺省时，LINT-ARGS 被认为没有定义，因此对库函数的参数不进行类型检查。定义 LINT-ARGS 的方法有两种：

1. 在编译时用命令项选项/D 来定义。
2. 在源文件中用#define 命令来定义。为使带函数说明的文件有效，#define 命令必须出现在#include 命令之前。

LINT-ARGS 的值没有意义，它可以被定义为任何值（包括空白）。

请注意，LINT-ARGS 的定义只对在嵌入文件中的库函数说明起作用，而不影响直接出现在源程序或用户自己的嵌入文件中的函数说明。通过使用#if或#if defined（ ）伪指令，你可以使自己的函数定义与 LINT-ARGS 有关。具体例子请参看库嵌入文件。

对那些参数个数不定的函数，类型检查的能力很有限，以下这些库函数受到此局限：

- 对 cprintf，cscanf，printf，和 scanf 的调用，只对第一个参数（即格式串）进行类型检查。
- 对 fprintf，fscanf，sprintf 和 sscanf 的调用中，只对第一个和第二个参数（即文件或缓冲区以及格式串）进行类型检查。
- 对 open 的调用中，只对头两个参数（即路径名和打开标志）进行类型检查。
- 对 sopen 的调用中，只对头三个参数（即路径名、打开标志和共享方式）进行类型检查。
- 对 exel，execle，execlp 和 execlpe 的调用中，只对头两个参数（路径名和第一个参数指针）进行类型检查。
- 对 spawnl，spawnle，spawnlp 和 spawnlpe 的调用中，对前三个参数（方式标志、路径名和第一个参数指针）进行类型检查。

2.6 出错处理

当调用函数时，最好提供对可能出现返回错误值的情况的检查和处理，否则程序可能会产生意料之外的结果。

对于库函数，可以从手册中知道它们预期的返回值是什么。在有的情况下，函数在出错时并不能返回一个确定的错误值，这通常是由于该函数的正常返回值的范围太大，不可能为错误规定一个唯一确定的值。

对某些函数，当出错时全局变量 errno 会被置为代表某一错误类型的值。注意：除非在对函数的描述中明确地提到变量 errno，你不能依靠它的设置来判断错误。

当使用会给 errno 置值的库函数时，你可以对照 errno.h 文件中定义的错误值，来检查 errno 的值。也可以使用函数 perror 或者 strerror。如果需要把系统的出错信息输出到标准错误文件（stderr）中，就使用 perror；如果需要把错误信息存放到一个串中以备以后使用，就使用 strerror。变量errno 的值以及相应的出错信息表见附录 A“出错信息”。

当使用 errno、perror和 strerror 时，请记住 errno 的值是被调用过的最后一个在 errno 置值的函数置的。为避免出现错误结果，在存取 errno 之前，应测试一下返回的值以证实的确发生了错误。一旦确定已发生错误，就应该立即使用 errno 或 perror，否则 errno 的值可

能被以后的函数调用修改。

数学函数在出错时在 errno 置值的方式将在第二部分“参考手册”中描述。数学函数通过调用一个名叫 matherr 的函数来处理错误。你可以写一个自己的错误处理函数并把它命名为 matherr，从而按你自己的方式去处理错误。当你自己提供函数 matherr 时，它就取代与之同名的库函数，正如手册第二部分中介绍 matherr 时指出的那样，你在写这个函数时必须遵守一定的规则。

你可以通过调用 ferror 来检查对流式文件的操作中的错误。函数 ferror 检查对某个给定的流式文件是否已设置了错误标志。当流式文件被关闭或反绕时，错误标志会自动清除。也可以调用函数 clearerr 来清除错误标志。

低级输入和输出操作（如 read（ ），write（ ））的错误会置 errno 的值。

函数 feof 测试一个流式文件是否已到文件尾。在低级输入输出操作中，测试文件结束状态用函数 eof，或者“读”（read）操作返回零作为被读出的字节数。

2.7 文件名和路径名

许多库函数用字符串代表文件名和路径名作为参数。这些函数对它们进行处理然后传递给操作系统，最终是由操作系统负责创建和维护文件和目录。因此重要的是不但要记住 C 语言对字符串的规定，还要记住操作系统对于文件名和路径名的约定，以及MS-DOS和XENIX在这方面的差异。有下面几点需要考虑：

1. 对大写与小写的区分
2. 子目录的表示法
3. 路径名的各成分间的分界符

C 语言是要区分大写字母和小写字母的，而 MS-DOS 则不加区分。在 MS-DOS 中存取文件和目录时，不能依靠大小写来区分它们的名字。比如“FILEA”和”fileA”这两个文件名在 MS-DOS 中等价，即代表同一个文件。

对可移植性的考虑也可能影响你如何选择文件名和路径名。比如你计划把代码搬到 XENIX 系统中去，就得考虑 XENIX 的命名规则。与 MS-DOS 不同，XENIX 是要区分大小写的，因此下面的两条命令在 MS-DOS 中等价但在 XENIX 中不等价：

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

由于大小写在 MS-DOS 中都行得通，要产生可移植的代码就得选择能在 XENIX 中使用的名字。

在名叫“sys”的子目录下面存放一些嵌入文件也是 XENIX 的一种约定。本手册接受了这一约定，在某些嵌入文件的说明中包含了“sys”这个子目录。如果你不关心可移植性，就可以不理睬这一约定自行存放嵌入文件。如果关心可移植性，使用子目录“sys”会使在 MS-DOS 和 XENIX 间进行移植容易一些。

MS-DOS 和XENIX 操作系统在使用路径名中的分界符上有区别。XENIX 用向前的斜线（/）来分隔路径名的成分，而MS-DOS 常用向后的斜线（\）。不过当路径名出现在所希望的地方时，MS-DOS 内部也能识别用作分界符的向前的斜线（/）。因此只要没有二义性并且

是在希望出现路径名的地方，在C程序中可以用向前或向后的任一种斜线作为MS-DOS的路径名中的分界符。

注：　在C的字符串中，向后的斜线（或称反斜线“\”）是一个转义符。它示意其后是某个特定的转义序列。如果一个普通的字符跟在反斜线之后，那么就忽略反斜线而只打印出那个字符。因此要在C的字符串中表示一个反斜线就必须用序列“\\”。（参阅《Microsoft C语言参考手册》中对转义序列的详细讨论。）

上述规则对大多数库函数都适用：当用到路径名时既可用反斜线也可以用向前的斜线作分界符。如果考虑到XENIX的可移植性，就应该使用向前的斜线（/）。

注意以下的例外情况：下面这些函数所接受的字符串参数不能提前知道是否是路径名（即它们可以是，也可以不是路径名）。在这些情况下，参数被当成C字符串处理，并且用到以下的特殊规则：

- 在exec和spawn这两族函数中，用户把将被运行的子进程名传递给程序，然后再把表示子进程所需的参数的串传递给它。作为子进程运行的程序，其路径名中可用向前的斜线或反斜线作分界符，因为这时能确定期望的是一个路径名；但是建议在作为子进程的参数的路径名中都使用反斜线作为分界符，因为这个子进程可能需要一个非路径名的串作为其参数。
- 在系统调用中，用户向MS-DOS传递一个由它执行的命令，这个命令中可能包含也可能不包含路径名。

在上述情形下，应该只用反斜线作路径名的分界符，向前的斜线（/）将不予接受。

当用户在对exec和spawn的调用中向子进程传递路径参数或在系统调用中使用路径名时，必须使用两条反斜线（\\）来表示一个路径名分界符。

例1

```
result = system ("DIR B:\\TOP\\DOWN") ;
```

例2

```
spawnl (P_WAIT, "bin/show", "show", "sub", "bin\\tell", NULL) ;
```

在例1中，在调用system时表示路径名“B:\TOP\DOWN”必须使用双反斜线。注意不是所有对system的调用都用到路径名。例如

```
result = system ("DIR") ;
```

就不包含路径名。

在例2中，函数spawnl被用来运行在子目录BIN下面的文件SHOW.EXE。因为第二个参数必须是一个路径名，所以用了向前的斜线（也可以用反斜线）。传递给SHOW.EXE的前两个参数是show和sub；第三个参数是一个表示路径名的字符串，由于这个参数不一定是路径名，就必须用双反斜线（\\）来表示在bin和tell之间的那一个反斜线。

2.8　二进制和正文方式

大多数C程序都使用一个或几个数据文件做为输入或输出。在MS-DOS中，数据文件通常按正文（TEXT）方式处理。在正文方式下，回车换行的组合（CR-LF）在输入时被转换成一个换行符（LF），而换行符在输出时被转换成回车换行符的组合。

在有些场合中，你也许想使处理的文件不做上述转换。在二进制方式下，这个转换就不生效。

你可以用下述方法控制程序中文件的转换方式。

● 选择一些文件作为二进制方式处理，使剩下的大多数文件的缺省方式为正文。这个选择可在打开文件时进行。如果在函数 fopen 的文件存取类型串中指定字母“b”，它将按二进制方式打开文件。如果用的是函数open，则可在其 oflag这个参数位置上指定O_BINARY，使它按二进制方式打开文件。详细情况请参看本手册的第二部分。

● 若要把大多数文件按二进制方式处理，可将二进制方式作为缺省方式。全局变量 _fmode 控制缺省转换方式。当_fmode 被置为 O_BINARY 时，缺省方式为二进制；否则缺省方式为正文方式，但stdaux和stdprn 除外，因为缺省时它们总是按二进制方式打开。初始时，_fmode 的缺省值为正文方式。

有两种办法可以改变_fmode 的值，首先，你可以把文件 BINMODE.OBJ（同 Microsoft 编译软件一起提供的）连接起来。同 BINMODE.OBJ 连接起来就将_fmode 的初值改为 O_BINARY，使得除 stdin，stdout和stderr 外的所有文件都用二进制方式打开。（这个选项在《Microsoft C 编译器用户指南》中介绍。）

其次可以直接修改_fmode 的值。这只要在你的程序中将其赋值 O_BINARY 即可。这样做的效果将 BINMODE.OBJ 连接在一起相同。

你还可以用正文方式打开某些文件从而使缺省方式（现在是二进制）无效。对于函数 fopen，当文件存取类型串中指定了“t”时，它就按正文方式打开文件。如果用 open 函数，把它的 oflag 参数置为 O_TEXT 就按正文方式打开文件。详细情况请参看第二部分中的描述。

● 在缺省情况下，文件 stdin，stdout和stderr 按正文方式打开，而stdaux和stdprn按二进制方式打开。若要按二进制方式处理 stdin，stdout或stderr，或者按正文方式处理 stdaux或stdprn，请用 setmode 函数。这个函数也可用于改变已打开的文件的转换方式。函数setmode有两个参数，即文件号和转换方式，它根据转换方式参数来设置文件的转换方式。

2.9 有关MS-DOS版本的问题

有些库函数的使用情况受你所用的MS-DOS 的版本的影响。下面列出这些函数并加以说明：

dosexterr，locking，sopen：这几个函数只在MS-DOS 3.0 及更新的版本中起作用。函数 sopen 使被它打开的文件具有共享属性。如果你希望某文件具有共享属性就应该用 sopen 而不是 open 去打开它。函数 locking 将一个文件全部或部分封锁起来，不让其他用户存取。函数 dosexterr 为 MS-DOS 3.0 及更新的版本中的系统调用 59H 提供出错处理。

dup，dup2：在有些情况下，在比 MS-DOS 3.0 旧的版本中使用函数 dup和dup2 会导致意料之外的结果。在这些旧MS-DOS 版本中，当使用函数 dup或dup2为stdin、stdout，stderr，stdaux和stdprn复制文件号时，关闭（close）任何一个文件号都将给使用另一个文件号进行的 I/O 操作造成错误。在 MS-DOS 3.0 及更新的版本中，close 函数得到了正确的处理，不会导致后面的操作出错。

exec, spawn：在比 MS-DOS 3.0 旧的版本中使用这两个函数时，用户不能存取参数 arg0或argv〔0〕的值；在相应参数位置上存放的是空串。在 MS-DOS 3.0 及更新的版本中，用户可以使用 arg0 或argv〔0〕的值。

要写出能在 MS-DOS 的任何版本中都能正确运行的程序，你可以使用变量_osmajor和_osminor（这两个变量在第三章第 3.5 节“全局变量和标准类型”中讨论）来测试当前的操作系统的版本号，并根据测试结果采取相应的行动。

范例

在下例中，我们通过对全局变量_osmajor 的测试来决定是应该用 open（在比 MS-DOS 3.0 旧的版本中）还是 sopen（在 MS-DOS 3.0 及更新的版本中）来打开文件 TEST.DAT：

```
unsigned char _osmajor;
...
if(_osmajor<3)
    open("TEST.DAT", O-RDWR);
else
    sopen("TEST.DAT", O-RDWR, SH-DENYWR);
```

2.10 支持浮点运算

C 库中的数学函数要在浮点运算的支持下进行实数计算。它可以由编译软件中的浮点运算库或者8087或80287协处理器来提供（关于在程序中使用浮点运算库请参阅《Microsoft C 编译器用户指南》）。需要浮点运算支持的函数名表列如下：

acos	_clear87*	exp	frexp	sin
asin	_control87*	fabs	gcvt	sinh
atan	cos	fcvt	hypot	sqrt
atan2	cosh	fieeetomsbin	ldexp	_status87*
atof	dieeetomsbin	floor	log	strtod
bessel**	difftime	fmode	lof10	tan
cabs	dmsbintoieee	fmsbintoieee	modf	tanh
ceil	ecvt	_fpreset	pow	

---

*使用了编译命令选项/FPa后不能使用

**bessel 函数不是指一个函数，而是指下列6个函数：j0、j1，jn，y0，y1和 yn。

除此之外，printf 族的函数（cprintf，fprintf，printf，sprintf，vfprintf 和 vsprintf）用于输出浮点值时，也需要有浮点输入输出的支持。

C 编译器会检查程序中是否用到浮点值，只有在需要时才装入浮点运算函数。这为那些不需要浮点运算的函数节省了相当数量的空间。

当你在 printf 或 scanf 族函数（cprintf，fprintf，printf，sprintf，vfprintf，vprintf，vsprintf，escanf，fscanf 或 sscanf）的格式串中用到浮点类型符时，一定要在参数表的相应

位置安排浮点值或指向浮点值的指针使其与格式串中的浮点类型符对应。浮点类型参数的出现使编译器察觉用到了浮点值。如果用格式串中的浮点类型符来打印一个整型参数，编译器就不会知道应该用浮点函数来解释此参数，因为它并不真正去读 printf 或者 scanf 中的格式串。例如，下面的程序在运行时将产生错误：

```
main( )     /*这个例子会出错*/
        {
        long f=10L;
        printf("%f", f);
        }
```

上例中，由于以下原因浮点 I/O 函数没有装入：

- 对 printf 的调用中没有指定任何浮点参数
- 在程序的其他任何地方没有用到浮点值，其结果是产生以下错误：

Floating point not loaded（浮点函数未装入）

下面是对上述 printf 调用的更正：

```
printf("%f", (float)f);
```

上面的调用中将整型数强制转换成浮点类型。

2.11 在库函数中使用巨型数组

在使用小型（small）、紧凑型（compact）、中型（medium）和大型（large）内存模式的程序中，Microsoft C 允许使用超过64K 物理内存的数组，只要把它们说明为huge 即可。（请参看《Microsoft C 编译器用户指南》第八章“使用内存模式”中对内存模式及关键字 near、far、huge 的讨论）。但是一般不能向库函数传递巨型（huge）数据。在小型和中型模式中，缺省的数据指针大小为 near（16位），只有 halloc 和 hfree 两个例程接受巨型数据指针。在紧凑模式下程序中使用的紧凑模式库、以及大型和巨型模式下程序使用的大型模式库中，只有下列函数的参数中可以用到巨型数据：

bsearch	halloc	lsearch	memcmp	memset
fread	hfree	memccpy	memcpy	qsort
fwrite	lfind	memchr	memicmp	

利用上述函数，你可以对巨型数据进行读、写、查找、排序、复制、初始化、比较，而且可以动态分配和释放巨型数组。上述函数中的任何一个可以毫无困难地在紧凑、大型、巨型模式程序中传递巨型数据指针。