Microsoft Microsoft Microsoft

Microsoft VISUAL C++ 运行库参考手册

[美] Microsoft 公司
张素琴 蒋维杜 李景淑 译

清华大学出版社

Microsoft Visual C++运行库

参 考 手 册

[美] Microsoft 公司 编

张素琴 蒋维杜 李景淑 译

清华大学出版社

(京)新登字 158 号

Microsoft Visual C ++ Run-Time
Library Reference

本书英文版由 Microsoft 公司属下的 Microsoft 出版社(Microsoft Press)出版。版权为 Microsoft 公司所有(Copyright© 1993 by Microsoft Corporation)。本书中文版经 Microsoft Press 授权清华大学出版社独家出版,1994。未经出版者书面允许,不得用任何手段复制本书内容。本书封面贴有微软公司防伪标签,无标签者不得销售。

图书在版编目(CIP)数据

Microsoft Visual C++运行库参考手册/美国 Microsoft 公司著;张素琴等译. —北京:清华大学出版社,1994
ISBN 7-302-01635-6

Ⅰ.M… Ⅱ.①美… ②张… Ⅲ.应用软件-程序设计-手册 Ⅳ.TP311.11-62

中国版本图书馆 CIP 数据核字(94)第 12502 号

出版者:清华大学出版社(北京清华大学校内,邮编 100084)
印刷者:清华大学印刷厂
发行者:新华书店总店北京科技发行所
开　本:787×1092 1/16　印张:32.25　字数:762 千字
版　次:1994 年 12 月第 1 版　1994 年 12 月第 1 次印刷
书　号:ISBN 7-302-01635-6/TP・699
印　数:0001—8000
定　价:56.00 元

引　言

Microsoft 运行库由 550 多个函数和宏组成。这些函数和宏是为 C 和 C++语言编程而设计的。运行库不但提供了高效、快速的例程以执行通用的编程任务(例如串操作),而无须用户再花费时间和精力进行编写,而且运行库还提供了执行操作系统函数(如打开、关闭文件)的可靠方法。运行库的重要性还在于它提供了 C 和 C++语言本身不具备的基本函数,包括输入输出函数、内存分配函数、进程控制、图形函数以及其它许多类型的函数。

本书描述了 Microsoft Visual C/C++所有的运行库例程,包括 Microsoft C/C++ 7.0 及早期版本的所有库例程和一些新的例程。

0.1　Microsoft 运行库

Microsoft 运行库所包含的例程和功能支持美国国家标准学会(ANSI) C 和 UNIX C 兼容性,支持 MS-DOS 和 Microsoft Windows 操作系统程序设计以及高级图形设计。

为便于操作系统和编译器之间传送程序,对每个运行库例程的描述包含关于兼容性的说明段。具有全部兼容性的例程包括以下各项:

ANSI MS-DOS QWIN UNIX WIN WIN DLL

(本书中,凡涉及 UNIX 系统的说明也适用于 XENIX 和其它类 UNIX 系统)如果某个例程与先前标准中的任一个不兼容,它的兼容性说明段就将该标准划去。例如,下面说明段表示与 ANSI 和 UNIX 的不兼容性:

~~ANSI~~　MS-DOS QWIN ~~UNIX~~ WIN WIN DLL

0.1.1　ANSI C 的兼容性

Microsoft C 和 C++的编译器是支持 ANSI C 标准的,而运行库例程也是为与 ANSI C 兼容而设计的。与 ANSI C 兼容的函数,其兼容性说明段标志为 ANSI.

1. 类型检查

ANSI C 的主要改进是在函数原型(声明)中允许参数类型。在函数原型中给定信息后,编译器可以在其后对函数引用时进行检查,以保证使用正确的参数个数、类型以及正确的返回值。

为了利用编译器的类型检查能力,对伴随运行库的包含文件作了相应扩充。除了库例程需要的定义和声明外,包含文件还包括带有参数类型表的函数声明,并加进了几个新的包含文件。这些文件的名字是为最大限度地与 ANSI C 标准和 UNIX,XENIX 名字兼容而选取的。

2. 下划线和 OLDNAMES.LIB

所有 Microsoft 专用运行函数、常量、变量、类型定义、结构和宏都与 ANSI 兼容(如 **_open**, **_VRES16COLOR**, **_cpumode**, **_HEAPINFO**, **_heapinfo** 和 **_isascii**)。Microsoft 专用运行函数、常量、变量、类型定义、结构以单个下划线开头;Microsoft 专用运行宏以两个下划线开头。

为了与 Microsoft C/C++ 7.0 版本及更早期的版本兼容,该产品提供了 OLDNAMES.LIB 库。其中包括了别名记录,以将原先的名字映射为新名字。例如,将 **open** 映射为 **_open**。

必须将由 Microsoft C/C++ 7.0 及其早期版本产生的目标文件与 OLDNAMES.LIB 同时连接。不过,缺省情况下,编译器产生库搜索记录,唯一需要显式连接 OLDNAMES.LIB 的是下列情况之一:

- 用缺省的/Ze 选项(使用 Microsoft 扩充)和/Zl 选项(从目标文件产生缺省库名)的组合编译时;
- 用缺省的/Ze 选项(使用 Microsoft 扩充)和/link 选项(连接器控制)、/NOD 选项(不进行缺省库搜索)组合编译时。

有关 CL 命令行选项的详细信息,请参考《命令行使用用户手册》第 1 章。

> **注意**:编译程序将同时具有旧名和新名的同一个结构看成两个不同类型;不能将旧类型复制到新类型中。同样,以结构为指针的旧的函数原型要在原型中使用旧的结构名。因此,使用时必须注意一致性,即例程旧名必须匹配参数的旧名,同样,新的例程名字也要与新的参数名保持一致。

0.1.2 在 MS-DOS 和 Microsoft Windows 操作系统环境下的程序设计

Microsoft 运行库例程的设计保持了 MS-DOS,Windows 操作系统与 UNIX 或 XENIX 系统间最大兼容性。运行库提供了许多操作系统接口例程,可以使你充分利用 MS-DOS 和 Windows 的长处。与 MS-DOS 和 Windows 操作系统兼容的函数,其兼容性说明段标志分别为 MS-DOS 和 WIN。特别对于 Windows 来说,兼容性说明还包括了动态连接库(DDL)兼容性信息。

0.1.3 QuickWin

Microsoft 运行库当前版本包括了一些 QuickWin 函数,其功能是将与 Windows 不兼容的 MS-DOS 程序作为像简单的文本方式 Windows 应用程序进行编译。在/Mq 编译选项下编译过的 MS-DOS 程序具有 Windows 用户界面,包括一个标准菜单条,标准在线帮助(对于 QuickWin 特征)和客户(或应用)窗口,该客户窗带有一个用于 C 输入/输出流 **stdin**, **stdout**, **stderr** 的子(文档)窗口,还可以加入其它自定义的子窗口。QuickWin 应用程序支持 Windows 剪接板,可以使用标准 C 函数在 QuickWin 应用程序的窗口中进行读写,作用与流相似。与 QuickWin 兼容的函数其兼容性说明段标志为 QWIN。

试读结束,需要全本PDF请购买 www.ertongbook.com

0.1.4 UNIX C 兼容性

Microsoft 运行库中的绝大多数函数，与名字相仿的 UNIX 例程是兼容的。为保证特殊兼容性，扩充了数学库函数，以提供与 UNIX 系统 V 数学函数相同的处理方式，与 UNIX和 XENIX 兼容的函数，其兼容性说明段标志为 UNIX。

0.1.5 扩展的图形库

Microsoft 运行库包括了一百多个图形例程。该库的核心包括几十个低级图形例程，其作用是在程序中选择显示方式、设点、画线、改变颜色、画矩形与椭圆等形状。还可以在不同坐标系的不同大小窗口中显示实型数据，如浮点数。

图形库包括表示图形和字形。表示图形库为用户程序提供了增加表示图形属性的工具。这些例程可以将数据显示成不同的图形，包括饼图、直方图、折线图和散列图。字形库可使用户程序在图形图象或图表中显示不同式样和大小的文字。字形操作例程可以和任何显示文字的图形例程连用，包括表示图形。

0.2 本书内容安排

本书是对 Microsoft Visual C++提供的运行库的使用指南。它包括两部分，第一部分“概述”和第二部分“运行库函数”。

第一部分“概述”，介绍 Microsoft 运行库，描述了使用库的一般规则，综述了库例程的主要种类。这部分包括以下各章：

• 第 1 章，“如何使用运行库”，给出理解和使用库例程的一般规则，并提及与某些例程有关的特殊规定。建议读者最好在使用运行库之前先阅读本章内容；另外，任何时候对库发生疑问时都可能参考本章。

• 第 2 章，“运行库例程分类”，按类别列举全部库例程，讨论了适于每一类例程的情况，以易于读者按任务类型查找例程。对所需要的例程，可在第二部分的相关页目寻找详细的描述。

• 第 3 章，“全程变量和标准数据类型”，描述了库例程使用的变量和类型。全程变量和标准类型也在使用它们的例程中给予了描述。

第二部分，“运行库函数”，按字母顺序逐个详细描述了库例程。读者一旦熟悉了运行库的规则和处理过程，就会经常参考、使用这部分内容了。

> **注意**：有关 Microsoft 产品支持的信息，请参见《Visual 工作平台用户手册》中“Microsoft 提供的服务”一章。

0.3 其它有关书目

为方便读者，下面列出一些书目，这些书目包括的课题，也许对读者很有帮助。除了

Microsoft 出版的书之外，对同样的课题，Microsoft 公司并不保证这些书的内容绝对合适。

• Barkakati, Nabajyoti. *The Waite Group's Microsoft C Bible*. Indianapolis, IN: Howard W. Sams, 1988.

Microsoft C 运行库的专题指南。Microsoft Quick C 产品也有类似手册。

• Campbell, joe, *C Programmer's Guide to Serial Communications*. Indianapolis, IN: Howard W. Sams & Company, 1987.

在串行通信领域用 C 语言编程的综合指南。

• Christian, Kaare, C++ *Programming*. Redmond, WA: Microsoft Press, 1992.

介绍了面向对象程序设计的概念，C++基本原则，Microsoft C/C++ 7.0，以及用 Microsoft 基本类库在 Windows 环境下的程序设计。

• Harbison, Samuel P., and Guy L. Steele, Jr. *C: A Reference Manual*, 3d ed. Englewood Cliffs, NJ: Prentice Hall, 1991.

C 语言和标准库的综合介绍。

• Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*, 2d ed. Englewood Cliffs, NJ: Prentice Hall, 1988.

本书的第一版是 C 语言的经典定义，第二版包括了基于 ANSI C 标准的新信息。

• Lafore, Robert. *Microsoft C Programming for the IBM*. Indianapolis, IN: Howard W. Sams & Company, 1987.

本书的第一部分介绍 C 语言，第二部分重点讲述 PC 环境的特殊内容，如基本输入/输出系统(BIOS)调用、内存和视频显示。

• Mark Williams Company. *ANSIC C: A Lexical Guide*. Englewood Cliffs, NJ: Prentice Hall, 1988.

对 ANSI C 标准的按部就班地介绍。

• Plauger, P. J., and Jim Brodie. *ANSI and ISO Standard C: A Guide for Programmers*. Redmond, WA: Microsoft Press, 1992.

由 ANSI 和 ISO 授权的 C 程序设计语言标准委员会的秘书和主席编写的一本书，是对 ANSI 和 ISO C 实现的参考。

• Plum, Thomas. *Reliable Data Structures in C*. Cardiff, NJ: Plum Hall, 1985.

使用 C 语言数据结构的中级水平的讨论。

• Plum, Thomas and Jim Brodie. *Efficient C*. Cardiff, NJ: Plum Hall, 1985.

关于提高 C 语言程序效率的技术方面的指导。

• Press, William H., Brian P. Flannery, SaulA, Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. New York: Cambridge University Press, 1988.

使用 C 语言的数值技术方面的综合介绍。

• Schustack, Steve. *Variations in C: Building Professional Applications with Microsoft C*. Second Edition. Redmond, WA: Microsoft Press, 1989.

用C语言开发商业应用软件的一般指南。

• Ward, Robert. *Debugging C*. Indianapolis, IN: Que Corporation, 1986.

调试C程序的理论和实践的高级指南。

• Wilton, Richard. *Programmer's Guide to PC and PS/2 Video Systems: Maximum Video Performance from the EGA, VGA, HGC, & MCGA*. Redmond, WA: Microsoft Press, 1987.

对所有PC和PS/2视频方式的高级指南。

0.4 文档约定

本书使用了下列印刷约定：

例　子	描　述
STDIO.H	大写字母串表示文件名、段名、寄存器和用于操作系统命令级的术语。
char, **_setcolor**, **__far**	黑体字表示C和C++关键字、操作符、语言专用字符，以及库例程名。 在讨论语法时，黑体字表示文本必须按给出的形式输入。 许多函数和常量以单个或两个下划线开始。下划线是名字的一部分，是强制性的。例如，欲使编译程序识别**__cplusplus**常量，必须输入开头的两个下划线。
expression	斜体字表示在该位置由使用者提供信息，如文件名。斜体字也偶尔用于正文中的强调部分。
[*option*]	双重方括号内的项是可选的。
# ***pragma pack***{**1** **2**}	花括号和垂直线表示在两个或更多项中选择一项。只要双重方括号[ ]没有包围该花括号，就必须选择一项。
#include<io.h>	这种字形用于范例、用户输入、程序输出以及文本形式的错误信息。
CL [*option*…] *file*…	三个点(省略号)跟在一项之后，说明可以出现同一形式的多个项。

while ( )
{
⋮
}
	三点排成一列或一行，表示有意省略了部分程序。

CTRL+ENTER	小号大写字母表示键盘上的键名。当两键之间有+号时，表明必须同时按下这两个键。 回车键，称为ENTER，有时在键盘上标志为弯箭头。
"argument"	在文本中第一次定义的一个新术语，用引号括起来。
"C string"	有些C结构，如串，需要引号。语言本身要求的引号使用""或''形式，而不是“”或‘’。
Color Graphics Adapter (CGA)	第一次使用缩写形式时，通常给出完整的拼写形式。

第一部分

概　　述

第 1 章

如何使用运行库

本章提供了使用 Microsoft 运行库例程的基本信息，同时还介绍了适用于特殊例程的专门规定，如路径名、文件名约定等。使用运行库例程前应先阅读本章，另外，在库的使用过程中有什么疑问也需要参考本章内容。

1.1 调用库例程

使用库例程的方法很简单，只要在自己的程序中调用它就行了，就好比它已在程序中定义好了。例如，编写下面的程序，命名为 SAMPLE.C：

```
# include <stdio.h>
void main (void)
{
printf ("cats and dogs\n");
}
```

该程序调用运行库中的 **printf** 例程，打印信息“cats and dogs”。调用库例程时，通常要涉及两组文件：

- 由 include 引入的头文件，其中包含库例程需要的声明、常量、类型定义。
- 编译过的库例程的库文件。

头文件和库文件都是随同 Microsoft Visual C++ 引用的，头文件用于编译时，库文件用于连接时。

用户程序的源代码中，通过# **include** 命令引入必需的头文件。本书的第二部分“运行库函数”中，在介绍各个库例程时，指出了其需要的头文件。由于 **printf** 库例程需要 STDIO.H 头文件，程序 SAMPLE.C 包含了下面的语句行：

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

该语句使编译器将 STDIO.H 的内容插入到源文件 SAMPLE.C 中。

源文件被编译后，需要将得到的目标文件(.OBJ)与适当的库文件(.LIB)连接起来，以产生可执行文件（.EXE）。目标文件中包含了程序调用的所有例程(包括库例程在内)的名字。如果某个例程没有在该程序中定义，连接器就到库文件中寻找其代码，并将这段代码并入到执行文件中。

一般说来，标准库例程的代码放在“缺省库”中，该库是在安装 Microsoft Visual C++ 时由用户产生的。由于连接程序会自动搜索缺省库，在连接程序时不必指明该库的名字，下面是将 sample 程序与缺省库连接起来的命令：

```
link sample ,,,;
```


试读结束，需要全本PDF请购买 www.ertongbook.com

如果所调用的例程未在缺省库中，必须告诉连接程序包含该例程的库文件名。假如你的程序中使用了 Microsoft 图形例程，则需要用下面的命令行连接程序，将 GRAPHICS.LIB 包含进去：

```
link sample ,,, graphics.lib;
```

欲了解关于连接方面的详细内容，请参考《命令行使用用户指南》(在 Microsoft Visual C++的文档集中)也可以查询编译程序安装说明。

1.2 使用头文件

前一节中已提到过，使用库例程时必须嵌入头文件。本节讲述需要头文件的特殊原因。

1.2.1 引入必要的定义

许多运行库例程使用了头文件中定义的宏常量或类型定义。为了使用例程，必须嵌入含有这些定义的头文件，下表中给出一些例子：

■ 宏：	如果库例程是以宏的方式实现的，则该宏的定义存在于某一头文件中。例如，**toupper** 宏是在头文件 CTYPE.H 中定义的。
■ 常量：	许多库例程引用了头文件中定义的常量。例如，例程 **open** 使用了 **OCREAT** 常量，该常量是在头文件 FCNTL.H 中定义的。
■ 类型定义：	有些库例程的参数类型是结构，或返回值类型是结构。例如，流输入/输出例程使用了类型 **FILE** 的结构，该结构是在 STDIO.H 中定义的。

1.2.2 引入函数声明

运行库头文件还包含了运行库中每个函数的函数声明，它们是按 ANSI C 推荐的标准声明的。有了这些声明，编译程序就可以对所调用的每个库函数进行类型检查，以保证使用正确的返回类型和参数。函数声明有时称为函数原型，因为对函数的所有调用而言，声明起了原型或模板的作用。

函数声明列出了函数的名字、返回值类型、参数的个数和类型。例如，下面是头文件 MATH.H 中对 **pow** 库函数的声明：

```
double pow (double x, double y);
```

该声明中，**pow** 的返回值类型为 **double**，两个类型是 **double** 的参数。有了这个声明，编译程序就可以对用户程序中每次引用的 **pow** 进行类型检查，以保证这种引用向 **pow** 传递两个 **double** 类型的参数，并返回 **double** 类型的值。

编译程序只能对出现在函数声明之后的函数引用进行类型检查。因此，函数声明通常置于源文件的开头，在函数的所有调用之前。函数返回值类型缺省时为 **int** 类型，因此，对那些返回值类型不是 **int** 的函数来说，函数声明尤为重要。例如，**pow** 函数返回值类型为 **double**，如果没有这个声明，编译程序就将返回值当作 **int** 处理，这会导致意外的结果。

使用头文件为自定义的函数提供声明也是一种惯用的方法。当你不愿意手写这些声

明时，可以用 /Zg 编译选项，自动生成这些声明，该选项使编译程序为当前源文件中定义的每个函数产生 ANSI 标准的函数声明，将这一输出重定向到一个文件中，然后，将该文件插到紧挨着你的源文件的开头处。

用户程序中可以包含同一函数的多个声明，只要这些声明不冲突，这对于原来没给出参数类型表的函数声明的程序是很有用的。例如，若原来用户程序中包含下列声明

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

那么可以在后面部分再包含这样的声明：

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

因为，尽管这两个声明并不一样，它们却是兼容的，不会发生冲突。第二个函数声明只是比第一个多提供了关于函数参数的信息。不过，当不同的声明给出的参数数目不同，或类型不同时，会产生冲突现象。

有些库函数允许有可变数目的参数。例如，**printf** 函数可以有一个参数或多个参数，对这些函数，编译程序只执行有限的类型检查，这点将对下列库函数有影响：

- 调用 **_cprintf**，**_cscanf**，**printf** 和 **scanf** 时，只检查第一个参数（格式串）的类型。
- 调用 **fprintf**，**fscanf**，**_snprintf**，**sprintf** 和 **sscanf** 时，只检查前两个参数（文件或缓冲区和格式串）的类型。
- 调用 **_open** 函数时，只检查头两个参数（路径和 **_open** 标志）的类型。
- 调用 **_sopen** 函数时，只检查前三个参数（路径，**_open** 标志，共享模式）的类型。
- 调用 **_execl**，**_execle**，**_execlp** 和 **_execlpe** 时，只检查前两个参数（路径名和第一个参数指针）的类型。
- 调用 **_spawnl**，**_spawnle**，**_spawnlp** 和 **_spawnlpe** 函数时，只检查前三个参数（模式标志、路径名和第一个参数指针）的类型。

1.3 路径和文件名

许多库例程是把表示路径和文件名的字符串作为参数的。如果你想把程序移植到 UNIX或 XENIX 系统中，必须记住，UNIX 系统中使用的路径和文件名的约定与 MS-DOS 不同。如果你不想把程序移植到 UNIX 系统中，可以跳过这一节。

1.3.1 大小写有关

MS-DOS 系统不区别大写和小写字母，因此，SAMPLE.C 和 sample.c 指的是同一个文件。不过，UNIX 系统中大小写是有关的。在 UNIX 中，SAMPLE.C 和 sample.c 指不同的文件。为了将程序移植到 UNIX 系统，必须在 MS-DOS 系统中都起作用的大小写方式中选择一种能在 UNIX 中正确工作的路径和文件名。例如，下列命令在 MS-DOS 中是一样的，但在 UNIX 中只有第二种命令才起作用：

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

1.3.2 子目录约定

在 UNIX 系统中，某些头文件通常存放在子目录 SYS 中，Microsoft Visual C++也遵循了这一约定，以便于将程序移植到 UNIX 系统中，如果你不打算移植程序，可以把 SYS 下的头文件放到别的地方。

1.3.3 路径分隔符

UNIX 系统在路径中用斜杠标记(/)，而 MS-DOS 则用反斜杠(\)。为了便于将程序移植到UNIX中，最好尽可能使用与 UNIX 兼容的路径分隔符。

1.4 在函数和宏之间选择

本书中交替使用了术语“例程”和“函数”，不过，术语“例程”实际上包括函数和宏，由于函数和宏具有不同的性质，用户必须留心到底使用了哪种形式。参考资料中对库函数的描述(本书第二部分)指明了各例程是用函数还是用宏实现的。

Microsoft 运行库中的大部分例程是函数，它们由编译过的 C 代码，或汇编过的 Microsoft 宏汇编(MASM)代码构成。尽管有些库例程是用宏来实现的，但可以象函数那样使用，可以向库宏传递参数并调用它们，就象调用函数一样。

使用宏的主要优点是执行时间短。在头文件中，每个库宏是用 **#define** 预处理指令定义的。在预处理时，宏被扩展(即由其定义所替代)，产生内联代码。因此，与函数调用相比，宏调用没有开销。从另一方面看，每次使用宏，都会在你的程序中插入相同的代码，而函数不管被调用几次，函数定义却只出现一次。这样，函数和宏提供了速度和代码大小的权衡。

除了速度和代码大小的不同，宏和函数还有以下重要区别：

(1) 有些宏在多次计算参数值时，处理参数会带来不正确的副作用(见后面例子)。并不是每一个宏都会有这种副作用。想要确定一个宏是否按要求处理副作用，可以到适当的头文件中检查其定义。

(2) 函数名可以对应一个地址，但宏名不能。因此，在需要函数指针的情况下，不能用宏。例如，可以声明一个指向函数的指针，但不能声明指向宏的指针。

(3) 可以声明函数，但不能声明宏。因此，编译程序不能象检查函数参数类型那样，对宏的参数进行类型检查，但是，如果程序中传递给宏的参数数目不正确，编译程序可以检查出来。

下面的例子说明了一些宏是怎样产生非期望副作用的，其中使用了例程 **toupper**。

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

该例中，在将参数 a 传递给宏 **toupper** 时，将 a 的值增加了 1。该宏是在 CTYPE.H 中定义的：

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

该定义使用了条件操作符(?:)。在条件表达式中,对参数 c 计算了二次:一次是检查该参数是否是小写字符,一次是产生结果。该宏将参数 a++计算了两次,使 a 的值增加了 2,而不是 1。结果是,**islower** 操作的值与**_toupper** 操作的值不同。

与其它一些例程一样,**toupper** 既有宏版本,又有函数版本,头文件 CTYPE.H 不仅声明了 **toupper** 函数,还定义了 **toupper** 宏。对这种例程选择宏版本或函数版本很容易。如果你想使用宏版本,只要并入含有宏版本的头文件就行了。这是因为,例程的宏定义常常出现在函数声明之后,因此宏定义一般具有优先性。所以,如果你的程序包含了 CTYPE.H,然后调用 **toupper**,编译程序会使用 **toupper** 宏:

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

可以通过将例程名括在括号内,来强制编译程序使用函数版本:

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

因为 **toupper** 名字后没有直接跟左括号,编译程序就不能将它译成宏名字,而必须用 **toupper** 函数。

另外一个使用函数版本的方法,是用 **#undef** 命令取消宏定义:

```
#include<ctype.h>
#undef toupper
```

由于宏定义不再存在,后面所有对 **toupper** 的引用都会使用函数版本。

第三个保证编译程序使用函数版本的方法,一般很少介绍,其途径就是显式声明函数:

```
#include<ctype.h>
int toupper(int c);
```

由于该函数声明出现在 CTYPE.H 中的宏定义之后,这将强制编译程序使用 **toupper** 函数。

1.5 入口处栈检查

对某些库例程,编译程序在入口处执行栈检查(栈就是用于临时存储的内存区域)。在这种例程的入口处,需要对栈进行检查以便确定是否有足够空间存储该例程使用的局部变量。如果空间足够,则调整栈指针分配空间;否则,产生"栈溢出"运行错误。如果取消了栈检查,编译程序则认为有足够的栈空间;这时若空间不够,很可能会覆盖数据段中的内存,却得不到任何警告信息——结果会导致无法预料的程序操作。

一般说来,只有那些需要大局部变量(大于 150 字节)的函数才进行栈检查,因为栈和数据段之间的空间足以处理需求量小的函数。如果多次调用栈检查函数,会稍稍降低执行速度。下列库函数需要进行栈检查:

_execvp	**scanf**	**system**
_execvpe	**_spawnvp**	**vprintf**
fprintf	**_spawnvpe**	**_write**
fscanf	**sprintf**	
printf	**sscanf**	

可以用/Gs 和/Ge 编译选项(见《命令行使用用户指南》第 1 章,"CL 命令参考"),或 **check_stackc** 编译指示(见《C 语言参考》第 7 章"预处理器宏指令及其应用")来触发和取消栈检查,这两本书都包括在 Microsoft Visual C++的文档集中。

1.6 处理错误

许多库例程返回值是指示错误条件的。为了避免异常结果,用户程序必须常常检查错误值,并处理所有可能的错误条件。本书第二部分,在对每个库例程的描述中列出了例程的返回值。

有些库函数没有设置错误返回。包括无返回的函数和返回值范围不可能包含错误值的函数。

为了帮助用户进行错误处理,一些函数设置了全程变量 **errno**。如果例程的描述中说明该函数设立了 **errno** 变量,可以用两种方法使用 **errno**:

(1) 将 **errno** 的值与头文件 ERRNO.H 中定义的值比较。

(2) 用 **perror** 或 **strerror** 库例程处理 **errno**。**perror** 例程向标准出错流 **stderr** 打印一条系统错误信息。**strerror** 例程将同一信息存于字符串中以备以后使用。

当用户使用 **errno**,**perror** 和 **strerror** 时,必须记住 **errno** 的值反映的是上一次调用中所设置的值。为防止混淆,必须经常测试返回值,以证实是否真正出错。一旦确定出错,立即使用 **strerror** 或 **perror**。否则,**errno** 的值可能会被中间的其它函数调用改变。

数学库例程通过调用 **_matherr** 或 **_matherrl** 库例程设置 **errno** 值;这两个函数都在本书第二部分库函数中介绍。如果你想用与这些例程不同的方法处理数学错误,可以自己写例程,并命名为 **_matherr** 或 **_matherrl**,你的例程必须遵循 **_matherr** 描述中列出的规则。

ferror 库例程用于检测输入/输出流的错误。该例程检查是否对指定流设置了错误指示符。该指示符的值在关闭或反绕流时自动被清除,也可以调用 **clearerr** 库例程来清除。

feof 库例程检测指定流的文件结束标志。低级输入和输出中的文件结束条件可以用 **_eof** 例程检测,或以 **_read** 操作读出的字节数为 0 做标志。

_grstatus 库例程在调用某些图形库操作后检测是否出错。详细内容请参考有关 **_grstatus** 函数的描述。

1.7 操作系统考虑

本节中列出的库例程,在不同的操作系统版本中行为不同。有关每个例程的详细信

息，请查阅本书第二部分中关于该例程的描述。

例　程	限　定
_locking	这些例程只在 MS-DOS 3.0 版及更高版本中才有效。
_sopen	
_fsopen	
_dosexterr	该例程对 MS-DOS 3.0 或更高版本中系统调用 0x59(获得扩充错误)进行出错处理。
_dup	**_dup** 和 **_dup2** 例程在低于 MS-DOS 3.0 的版本中可能会引起意外结果。如果
_dup2	用 **_dup** 或 **_dup2** 例程产生 **stdin**，**stdout**，**stderr**，**_stdaux** 或 **_stdprn** 使用的双重文件柄，那么，用一个文件柄调用 **_close** 函数后，用另一个文件柄进行 I/O 操作会引起错误。这种异常情况不会出现在 MS-DOS 3.0 或更高版本的 MS-DOS 中。
_exec	在低于 3.0 的 MS-DOS 版本下使用 **_exec** 和 **_spawn** 的函数族时，参数 arg0(或
_spawn	子进程中的 argv[0])的值不提供给用户；相反，用空字符串("")代替该值，存于该位置。在 MS-DOS 3.0 及更高版本中，arg0 参数包含了完整的命令路径。

Microsoft Visual C++定义了一些全程变量以指明当前操作系统版本，可以根据这些值来确定你的程序在哪个版本的操作系统中运行。更为详细的信息见第 3 章"全程变量和标准数据类型"。

1.8 浮点支持

Microsoft 数学库例程要求浮点支持，以实现实数(带有小数部分的数)的计算，这种支持可以由伴随编译软件的浮点库或 8087，80287，80387 协处理器提供。需要浮点支持的函数如下：

acos	**ceil**	**expl**	**_gcvt**
acosl	**_ceill**	**fabs**	**_hypot**
asin	**_clear87**	**_fabsl**	**_hypotl**
asinl	**_control87**	**_fcvt**	**ldexp**
atan	**cos**	**_fieeetomsbin**	**_ldexpl**
atanl	**_cosl**	**floor**	**log**
atan2	**cosh**	**floorl**	**logl**
atan2l	**_coshl**	**fmod**	**log10**
atof	**_dieeetomsbin**	**fmodl**	**_log10l**
_atold	**difftime**	**_fmsbintoieee**	**modf**
Bessel	**_dmsbintoieee**	**_fpreset**	**_modfl**
_cabs	**_ecvt**	**frexp**	**pow**
_cabsl	**exp**	**_frexpl**	**powl**

试读结束，需要全本PDF请购买 www.ertongbook.com