Microsoft Microsoft Microsoft

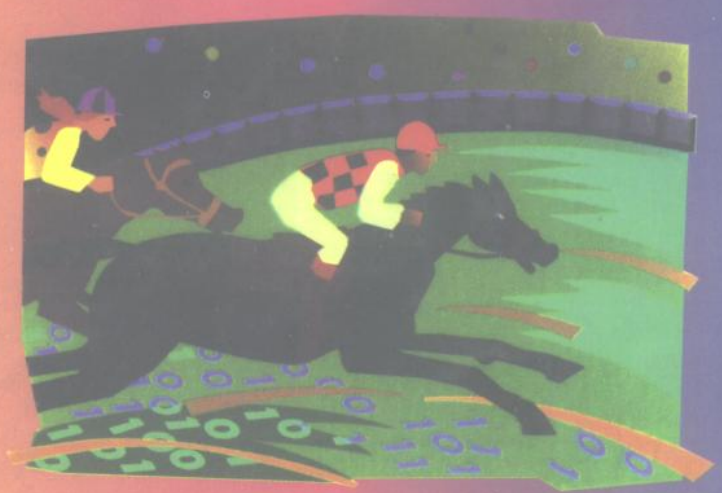
Microsoft®

Visual C++™ 2.0 for Win32®大全（四）

——运行库及io流类库参考手册

[美] Microsoft Corporation 著

丁国良 王嘉桢 张素琴 陈瑞 译



清华大学出版社

Microsoft®
Visual C++™ 2.0
for Win32® 大全(四)
——运行库及 io 流类库参考手册

[美] Microsoft Corporation 著
丁国良 王嘉祯
张素琴 陈 瑞 译

清华大学出版社

(京)新登字 158 号

Microsoft® Visual C++ ™ 2.0 for Win32® 大全(四)——运行库及 io 流类库参考手册
Microsoft Visual C++ Run-Time Library Reference

图书在版编目(CIP)数据

Microsoft® Visual C++ ™ 2.0 for Win32® 大全(四)：运行库及 IO 流类库参考手册/美国微软公司著；丁国良等译 .—北京：清华大学出版社，1996
ISBN 7-302-02106-6

Ⅰ. M… Ⅱ. ① 美… ② 丁… Ⅲ. ① C 语言，C++-程序设计-手册 ② 程序库-手册
Ⅳ. TP312C

中国版本图书馆 CIP 数据核字(96)第 02830 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**：26.5　**字数**：656 千字
版　次：1996 年 6 月 第 1 版　1996 年 6 月 第 1 次印刷
书　号：ISBN 7-302-02106-6/TP·986
印　数：0001—3000
定　价：52.00 元

目　录

第一部分　Microsoft 运行库函数

第二部分　io流类库

第一部分　Microsoft 运行库函数

引　言

Microsoft 运行库提供了在 Microsoft Windows NT 操作系统下编程所需的例程，这些例程可以使 C 和 C++ 语言没有提供的一些通用编程任务得以自动实现。

0.1 兼容性

Microsoft 运行库支持美国国家标准学会(ANSI) C 和 UNIX C(在本书中，凡涉及UNIX系统的说明也适用于 XENIX、其它类 UNIX 和 Windows NT 中的 POSIX 子系统)。在本书第二部分，每一个运行库例程的描述包含有同 ANSI、UNIX 和 Win32 应用程序接口(API)兼容部分的说明，本运行库包含的所有例程均与 Win32 API 兼容。

0.1.1 同 ANSIC 的一致性

本系统中，所有的 Microsoft 专用标识符(如函数、宏、常量、变量和类型定义)的命名习惯同 ANSI 一致。

Microsoft 专用函数名和全局变量名以单个下划线开头，在程序中，这些函数名和全局变量名只能被局部地取代。例如，当程序中包含有 Microsoft 运行头文件时，可以把一个局部变量定义为 _**open**，而取代相同的 Microsoft 专用函数名，但是不能把该标识符定义成程序中的全程变量或全程函数。

Microsoft 专用宏和常量的命名以两个下划线开头，或使用一个下划线开头后紧跟一个大写字母的形式。这些标识符的使用范围是绝对的，不能定义成其它类型的标识符。例如，不能将 Microsoft 专用标识符 _**UPPER** 定义成其它类型的标识符。

0.1.2 UNIX

如果准备将程序移植到 UNIX 系统中，就需要遵守下列准则：

- 不要删除 SYS 子目录下的头文件，只有在不准备将程序移植到 UNIX 系统中时，才可以将 SYS 子目录下的头文件移到他处。
- 许多例程是把表示路径和文件名的字符串作为参数的，如果准备将程序移植到UNIX系统中，就需要使用与 UNIX 系统兼容的路径分隔符。UNIX 系统在路径中只使用斜杠标记/表示路径分隔符，而 Windows NT 系统则使用斜杠/和反斜杠 \ 两种标记，所以本书在 #**include** 语句中使用与 UNIX 系统兼容的斜杠标记/。然而，对于 Windows NT 的外壳程序 CMD.EXE 来说，不能在命令中使用斜杠标记。
- UNIX 系统对于路径和文件名中的大小写字母是敏感的，而在 Windows NT 系统中的文件分配表(FAT)对于大小写不敏感。Windows NT 的安装 NT 文件系统(NTFS)在列目录时保持文件名中大小写形式不变，而文件查找和其它系统操作则不考虑字母的大小写。

0.1.3 向后兼容

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的结构类型，不能将旧结构类型复制到新结构类型中。同样，以 **struct** 为指针的旧的原型要在原型中使用旧的 **struct** 名。

为了与 Microsoft C professional development system 6.0 和早期的 Microsoft C 版本相兼容，该产品提供了 OLDNAMES.LIB 库，在编译时，可将原先的名字映射为新名字，例如将 **open** 映射为 **_open**，当使用下列命令组合编译时，必须显式地连接 OLDNAMES.LIB。

- /Z1(从目标文件产生缺省库名)和/Ze(缺省——使用 Microsoft 扩展)
- /link(连接器控制)，/NOD(非缺省——库搜索)和/Ze

有关编译命令操作的详细信息，请参考 Visual C++ 用户手册。

0.2 函数和宏之间的选择

Microsoft 运行库中的大部分例程是函数，它们由编译过的 C 语言代码，或汇编过的 Microsoft 宏汇编代码构成，但有些例程是作为宏来实现的，还有一些例程既有宏版本，又有函数版本。对于既有宏版本又有函数版本的例程，在使用时，就需要有所选择。如果希望使用宏版本，只要在程序中并入含有宏版本的头文件即可，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。如果希望使用函数版本，可以用下述两种方法实现。

- 可以通过将例程名括在括号内，强制编译程序使用函数版本。

```
#include <ctype.h>
a = toupper(a);          //use macro version of toupper
a = (toupper)(a);        //force compiler to use function version
                         //of toupper
```

- 使用 **#undef** 命令取消宏定义：

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

函数和宏是有一些区别的，如果需要在函数和宏之间选择，可以从以下几个方面做一权衡：

- 速度和代码大小　使用宏的主要优点是执行速度较快。在预处理期间，宏被扩展(即由其定义所替代)产生内联代码，而函数不管被调用几次，函数定义却只出现一次。与函数相比，宏只是增加了代码大小而没有调用函数时所产生的时间上的开销。
- 函数求值　函数可以对一个地址求值，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。
- 宏的副作用　宏在对其参数值进行多次计算时，可能会错误处理其参数。例如宏 **toupper** 定义如下：

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

在下例中，宏 **toupper** 会产生副作用。

```
#include <ctype.h>

int a = 'm';
a = toupper(a++);
```

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。

■ 类型检查　由于可以声明函数，而不能声明宏，因此，编译程序不能像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。

0.3 类型检查

编译程序对一些参数数目可变的函数进行有限的类型检查。这些函数是：

函数调用	参数类型检查
_cprintf, _cscanf, printf, scanf	第一个参数(格式化字符串)
printf, fscanf, sprintf, sscanf	前两个参数(文件或缓冲区和格式化字符串)
_snprintf	前三个参数(文件或缓冲区，数目，格式化字符串)
_open	前两个参数(路径和 **_open** 标志)
_sopen	前三个参数(路径，**_open** 标志和共享方式)
_execl, _execle, _execlp, _execlpe	前两个参数(路径和第一个参数指针)
_spawnl, _spawnle, _spawnlp, _spawnlpe	前三个参数(方式标志，路径和第一个参数指针)

编译程序对上述函数对应的宽字符也做同样的有限类型检查。

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, _cplusplus	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。
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。

续表

例　子	描　述

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。当两键名之间有 + 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''。
▶CEnterDlg;	箭头指向的代码表示这是由前面的例子变换而来，常常表明需要对此进行编辑。
Dynamic-Link Library (DLL)	第一次用到缩写时，将全文拼出。
Microsoft 特定→ **END Microsoft 特定**	某些特性具有专门使用限制时，比如 Microsoft C 实现定义的特性，而不是 ANSI 标准所规定的特性，则以箭头(→)标记，并以 END 结束标记。

第1章　运行库例程分类

本章按类别列出和描述了 Microsoft 运行库的所有例程，这里的描述只是运行库例程功能的概括介绍，若要详细了解每个例程的作用、语法及使用方法，请参考本书第 3 章“按字母顺序排列的运行库函数”。

运行库例程的主要类别有：

参数访问例程	字符分类例程	错误处理例程
浮点数支持例程	内存分配例程	串操作例程
缓冲区操作例程	数据转换例程	异常处理例程
输入和输出例程	进程和环境控制例程	系统调用例程
字节分类例程	目录控制例程	文件处理例程
国际化例程	查找和排序例程	时间管理例程

对于多字节字符例程和宽字符例程都同对应的单字节字符例程分在同一组类别中。

1.1　参数访问例程

当函数参数的个数可变时，宏 **va_arg**、**va_end** 和 **va_start** 提供了对函数参数的访问功能。在头文件 STDARG.H 定义的这些宏同 ANSIC 兼容，在头文件 VARARG.H 中定义的同 UNIX V 兼容。

参数访问宏

宏	作　用
va_arg	从参数表中检索参数
va_end	指针复位
va_start	将指针指向参数表起始处

1.2　缓冲区操作例程

缓冲区操作例程用于以字节为单位的内存区域中的工作。

缓冲区操作例程

例程	作　用
_memccpy	将所指定字符及其之前的字符或指定数目的字符从一个缓冲区拷贝到另一个缓冲区。
memchr	在缓冲区里指定数目的字符中搜索特定字符，返回指针指向该字符首次出现的位置。
memcmp	比较两个缓冲区中指定数目的字符。
memcpy	将指定数目的字符从一个缓冲区拷贝到另一个缓冲区中。
_memicmp	不考虑字符的大小写(大写字符与小写字符等价)，比较两个缓冲区中指定数目的字符。

续表

例程	作 用
memmove	将指定数目的字符从一个缓冲区拷贝到另一个缓冲区。
memset	用指定字符初始化缓冲区中指定数目的字节。
_swab	交换若干字节的数据,并将它们存放在指定位置。

当源数据区和目的区重叠时,只有 **memmove** 函数能够正确地拷贝源数据区的所有内容。

1.3 字节分类例程

下列每一个例程测试一个多字节字符中的某一指定字节是否满足测试条件,除了特别说明,否则测试结果取决于当前使用的多字节代码页。

注意:根据定义,ASCII 字符集是所有多字节字符集的一个子集。例如,日语片假名字符集既包括非 ASCII 字符,也包括 ASCII 字符。

下表中的常量均在头文件 CTYPE.H 中定义。

多字节字符字节分类例程

例 程	字节测试条件
isleadbyte	测试头字节;测试结果取决于当前地区的 IC_CTYPE 类别设置
_ismbbalnum	**isalnum** ‖ **_ismbbkalnum**
_ismbbalpha	**isalpha** ‖ **_ismbbkalnum**
_ismbbgraph	同例程 **_ismbbprint** 一样,但例程 **_ismbbgraph** 不包括空格字符(0x20)
_ismbbkalnum	不同于标点符号的非 ASCII 文本符号。例如,仅在代码页 932 中,该例程测试日文片假名字母
_ismbbkana	日文片假名(0xA1—0xDF),仅在代码页 932 中
_ismbbkprint	非 ASCII 文本或非 ASCII 标点符号。例如,仅在代码页 932 中,该例程测试日文片假名字母或日文片假名标点符号(0xA1—0xDF)
_ismbbkpunct	非 ASCII 标点符号。例如,仅在代码页 932 中,该例程测试日文片假名标点符号
_ismbblead	测试多字节字符的第一个字节。例如,仅在代码页 932 中,有效的范围是 0x81—0x9F,0xE0—0xFC。
_ismbbprint	**isprint** ‖ **_ismbbkprint**。该例程包括空格字符(0x20)
_ismbbpunct	**ispunct** ‖ **_ismbbkpunct**
_ismbbtrail	测试多字节字符的第二个字节。例如,仅在代码页 932 中,有效的范围是 0x40—0x7E,0x80—0xEC
_ismbslead	测试头字节(在字符串正文中)
_ismbstrail	测试尾字节(在字符串正文中)
_mbbtype	根据前一字节返回字节类型
_mbsbtype	返回串中的字节类型

MB_LEN_MAX 宏(在 LIMITS.H 中定义),扩展到任意一个多字节字符可能具有的

最大字节长度。**MB_CUR_MAX**(在 STDLIB.H 中定义)扩展到当前地区的任意一个多字节字符可能具有的最大字节长度。

1.4　字符分类例程

下列例程测试某一指定的单字节字符、宽字符或多字节字符是否满足测试条件(根据定义,ASCII 字符集是所有多字节字符集的一个子集,例如,日本片假名字符集包括 ASCII 字符,也包括非 ASCII 字符)。一般情况下,这些例程的执行速度要比用户编写的程序要快。例如,以下代码的执行速度要比 **isalpha**(c)慢。

```
if ((c >= 'A') && (c <= 'Z')) || ((c >= 'a') && (c <= 'z'))
    return TRUE;
```

字符分类例程

例　程	字符测试条件
isalnum, iswalnum, _ismbcalnum	测试字母数字字符
isalpha, iswalpha, _ismbcalpha	测试字母字符
__isascii, iswascii	测试 ASCII 字符
iscntrl, iswcntrl	测试控制字符
__iscsym	测试字母、下划线或数字
__iscsymf	测试字母或下划线
isdigit, iswdigit, _ismbcdigit	测试十进制数字
isgraph, iswgraph, _ismbcgraph	测试除空格外的可打印字符
islower, iswlower, _ismbclower	测试小写字符
_ismbchira	测试平假名字符
_ismbckata	测试片假名字符
_ismbclegal	测试合法的多字节字符
_ismbc10	测试日本 0 级多字节字符
_ismbc11	测试日本 1 级多字节字符
_ismbc12	测试日本 2 级多字节字符
_ismbcsymbol	测试非字母数字的多字节字符
isprint, iswprint, _ismbcprint	测试可打印字符
ispunct, iswpunct, _ismbcpunct	测试标点字符
isspace, iswspace, _ismbcspace	测试空白字符
isupper, iswupper, _ismbcupper	测试大写字符
iswctype	测试由 desc 参数确定的特性
isxdigit, iswxdigit	测试十六进制数字
mblen	返回有效的多字节字符长度,结果取决于当前地区的 **LC_CTYPE** 类别设置