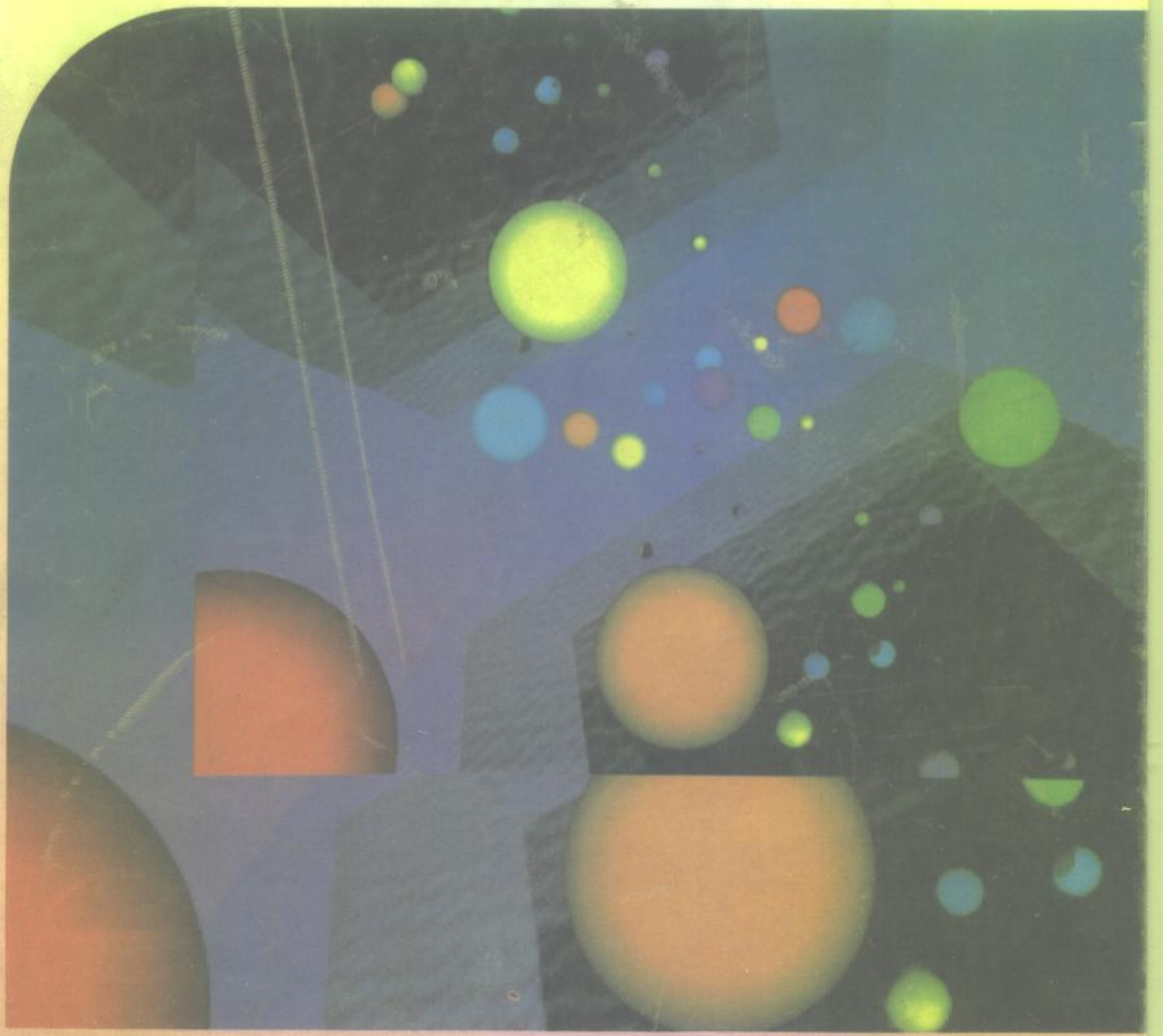
计算机软件应用系列

Visual C++ 程序开发指南（三）

——类库使用参考

● 陈佳海　等 编著



科学出版社

计算机软件应用系列

Visual C++程序开发指南(三)
—— 类库使用参考

陈佳海　等 编著

科学出版社

1995

（京）新登字 092 号

内 容 简 介

本书介绍了如何在 Visual C++程序中使用 C 函数库、MFC 类库以及全局变量和全局宏，同时简单地讨论了如何用 Visual C++设计 DOS 和 Windows 应用程序。本书对类库、全局变量和全局宏的介绍非常全面、透彻，可作为 Visual C++和 Windows 程序员的备查手册。

计算机软件应用系列

Visual C++程序开发指南(三)

——类库使用参考

陈佳海 等 编著

责任编辑 刘晓融 留 霞

科学出版社出版

北京东黄城根北街16号

邮政编码：100717

化学工业出版社印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1995 年 6 月第 一 版 开本：787×1092 1/16

1995 年 6 月第一次印刷 印张：28

印数：1—3 150 字数：644 000

ISBN 7-03-004934-9/TP·484

定价：36.40 元

前　言

面向对象程序设计(OOP)是目前流行的程序语言概念，Visual C++是汇集 Microsoft 公司技术精华的主流产品。《Visual C++程序开发指南》是《计算机软件应用系列》丛书中的一套，共四册。这套书从各个方面说明了应如何在 Visual C++环境下以 OOP 的概念设计 DOS 和 Windows 应用程序。

作为一个成熟的软件工程师，面对铺天盖地的 Word，Excel，PowerPoint，FoxPro，Visual Basic 等应用软件，不熟悉 Visual C++是很不明智的，因为借助 Visual C++，软件工程师就可以开发出很多动态链接库，从而为广大用户提供全方位的支持，其商业前景更是无可估量的。

Visual C++是一个面向对象、界面友好的 DOS 和 Windows 程序开发环境。在这个集成环境中开发应用程序时，不一定要手工设计用户界面，而只需选取菜单命令，Visual C++系统就会生成一个可实际运行的 Windows 应用程序框架。此后，程序员就可利用基于 Windows 的 C++编辑器，借助 AppWizard 建立面向对象的应用程序。

除了 AppWizard 外，Visual C++还包含 C++应用程序生成器、基于 Windows 的编辑器、基于 Windows 的面向图形的编程工具、使 C++代码和 Windows 消息及类成员函数相联系的交互式工具、可用来编写 Visual C++文本程序的 QuickWin 库以及预编译的头文件和源文件。当然，类库丰富更是 Visual C++的最大特色。

尽管目前市面上已有一些 Visual C++方面的书籍，但学习 Visual C++的人常常会有下列困惑：

(1) 对 Visual C++缺乏一致的认识，遇到问题总是缺这少那，缺乏一个值得信赖的良师益友。

(2) 只了解 OOP 的直观语法，而对于 OOP 有何好处、为什么 OOP 可取代结构化语言并成为未来程序设计风格的主流，都不太明白。

(3) 无法理解 OOP 对未来程序的管理与维护究竟有何重要性。

(4) 在 Visual C++环境下设计 OOP 程序时，对 Visual C++提供的资源不太清楚。

(5) 不知道如何用 Visual C++和 Windows 解决实际问题。

针对这些问题，这套书从各个方面介绍了 Visual C++集成环境、OOP 概念、类库以及基于 DOS 和 Windows 的 C++应用程序开发方法，并提供了丰富的类模板和大量的应用程序实例。

本套书的内容基本属于中等难度，适用于初、中级读者。不熟悉 OOP 技术和 C++语言的读者可参阅第一册的第一至十三章，这几章主要介绍 Visual C++集成环境和 OOP 概念。经常从事应用程序开发的读者大都比较关心 Visual C++类库和通用类的构造方法，可参阅第二册和第三册的第七至九章，其中第二册主要介绍通用类的构造方法，并从我们所熟悉的数据结构着手，讨论如何设计和实现自己的通用类，这为进行大程序开发的用户提供了设计开发环境的手段；第三册第七至九章则主要介绍 Visual C++基础类库，讨论如何使用 Visu-

al C++类库。比较熟悉 OOP 概念而对 C++程序设计技术不太精通的读者可参阅第四册，其中包含大量的程序设计实例。

本书是这套书的第三册，参与编写的人员有：陈佳海、蒋志国、罗心方、胡威、赵云朋、李兵、方良、刘志义、李蕾、李建、刘实燕、万崇福、李纪鸿、刘石华、赵越、郭芳芳、陈楚南、梁友国、郑旭光。全书由吴佳教授和陈忠敏研究员审校。此外，朱小宝为本书的编排付出了辛勤的劳动。在此对以上同志深表感谢。

限于水平和时间，书中的错误和不妥之处，敬请读者批评指正。

目　录

第一章　如何在C++中使用C库

本套书第一册的第一章到第十三章介绍了Microsoft Visual C++编译器，讲述了面向对象的程序设计技术并向用户展示了C++程序设计语言。特别是第一册的第七章至第十二章中讲述的有关C++中支持的数据抽象、继承和多态性，它们是构成面向对象的程序设计的基础。

从本章开始主要讲述如何建造和使用C++类库。因为还没有标准C++库，标准的C库依然是C++程序员的重要函数资源。本章讲述如何在C++程序中使用C库(标准库以及用户自己的库)；同时也简单说明了ANSI标准C库中的函数，并给出了C++程序中的这些函数的应用实例。第二章讲述如何建立和设计C++类库。

1.1　C与C++的链接

假设有一个函数库已被编译为对象代码，并且该库以对象代码的形式保存在库中。在程序中当用户从这种库中调用一个函数时，编译器用程序的对象代码中的一个不可分辨的符号标记该函数名称。为创建一个可执行程序，用户必须使用一个链接器并保证链接器为该函数代码搜索正确的库。如果链接器在库中找到函数的对象代码，它将把这些代码与程序的对象代码组合起来以创建可执行文件。为了在C++程序中使用C函数，必须完成这个链接过程。下面将讲解如何使用C++的链接语法完成链接。

1.1.1　类型安全链接

为了理解C++程序如何与C函数链接，首先必须了解C++如何分辨函数名。在C++中可以重载一个函数名以便用不同参数声明同一函数。为了帮助链接器工作，C++编译器通过一个编码方案为每个重载函数建立唯一的名字。编码算法的主要思想是通过组合以下各部分来产生唯一的名称：

① 函数名

② 被定义的函数中的类名

③ 函数接受的参数类型表

用户不必了解编码算法细节，因为不同编译器的算法不同。但是明确了类中的某一个函数都产生一个独一无二的名字将有助于用户理解链接器如何确定调用一个函数的哪个版本。

1.1.2　函数名编码的影响

函数名编码是如何影响C++程序的？分析下面的C程序就可以了解名称编码的一个好处：

```
// 说明错误参数类型的影响
```

```
#include <stdio.h>

void print(unsigned short x)
{
    printf("x = %u",x);
}

void main(void)
{
    short x = -1;
    print(x);
}
```

print 函数需要一个 unsigned short 整型参数，但 main 却以一个 signed short 整型参数调用 print。用户可以从程序输出中看到类型不匹配的结果：

```
x = 65535
```

即使 print 通过－1 被调用，输入值仍为 65 535，因为程序运行在以 16 位表示 short 整数的系统上。值－1 被表示为 0xffff(所有 16 位都为 1)，如果将其作为一个 unsigned 数值，则 0xffff 表示 65 535。在 C＋＋中，可以通过重载函数来解决这个问题，也就是一个 print 版本处理 unsigned short 类型，另一个版本处理 short。通过这一修改，C 程序的 C＋＋版本应为

```
// C++版本通过重载函数解决这个问题

#include <stdio.h>

void print(unsigned short x)
{
    printf("x = %u",x);
}

void print(short x)
{
    printf("x = %d",x);
}

void main(void)
{
    short x = -1;
    print(x);
}
```

运行这个 C＋＋程序的输出为

```
x = -1
```

这次的输出结果是正确的。因为 C＋＋编译器用函数名编码产生了一次对 print 的一个版本的调用，使用了 signed short 参数。C＋＋的这一能力(根据参数类型区分重载函数)被称为类型安全链接，因为不能以错误参数类型调用函数。

1.1.3 C 的链接命令

只要了解 C＋＋能对所有函数名编码，就可以理解从 C＋＋程序调用 C 函数的主要问题。如果 C＋＋程序调用 C 函数 strlen 以获取字符串长度：

```
// C++ 程序

#include <stddef.h>                    // 定义 size_t

size_t strlen(const char * s);         // strlen 的原型
```

```
//...
char str[] = "Hello";
size_t length = strlen(str);
```

当编译和链接包含C库的C++代码程序时，即使将程序与含有strlen代码的C库相链接，链接器仍认为函数strlen不可识别。因为C++编译器用函数原型strlen(const char *)创建了一个与strlen截然不同的名字，而strlen只是C库用来存储此函数对象代码的名字。

为了成功地将C对象代码与C++程序链接，应防止C++编译器为C函数名编码。C++链接命令提供了换码功能，以确保成功地链接使用strlen的C++程序。例如，将strlen作为一个C函数应这样声明：

```
// 为函数 strlen 指定 C 链接
extern "C" size_t strlen(const char * s);
```

1. 链接命令的其他形式

为了说明带有"C"链接的大量函数，可以使用链接命令的组合形式：

```
// 链接命令的组合形式
extern "C"
{
    int printf(const char * format,...);
    void exit(int status);

/* 其他函数... */
}
```

一般地，头文件包含这样的链接命令，因为函数是在头文件中声明的。

2. C与C++程序分享头文件

由于C和C++的关系非常相近，用户经常需要在C++程序中使用C函数。在C++程序中，必须用一个extern"C"链接说明C函数。用户只需要简单地组合链接命令就可以完成，但那样会使C编译器不能接受，因为extern"C"不是标准的C结构。可以使用C预处理器的条件编译命令来解决这个问题：

```
/* C与C++分享头文件 */

#ifdef __cplusplus
extern "C" {   /* if it's C++,use linkage directive */
#endif

/* 声明C函数 */
void clearerr(FILE *f);
FILE * fopen(const char * name,const char * mode);
int fclose(FILE * f);

/* ... */

#ifdef __cplusplus
}
#endif
```

__cplusplus宏在每个C++编译器中都预定义。在编译C++程序时，处理C和C++程序的编译器定义了这一符号。通常，这些编译器提供使用#ifdef __cplusplus结构的头文件声明带有extern"C"的C库函数(仅需在C++程序中包含这个头文件)。本书中的程序假设

像 STDIO. H 这样的标准 C 头文件，可与 C++程序链接运行。

如果用户使用的 C 头文件不能与 C++编译器配合使用，而且又不能改变 C 头文件，仍可通过加入 #include 指令标识 extern"C"链接：

```
extern "C"
{
#include <stdio.h>
}
```

如果必须这样的话，最好的办法是使用 extern"C"以及 #include 创建一个新的头文件，如前例所示。这将避免使用链接指令绕乱 C++程序，链接指令应该属于头文件。所有使用 C 函数的源文件可以通过一个相容的链接指令来完成这一工作。

3. 链接伪指令的限制

可以为重载函数的一个实例指定链接指令。如果说明 Complex 和 binary_coded_decimal 类的 sqrt 函数如下：

```
class complex;
class binary_coded_decimal;

extern complex sqrt(complex&);
extern binary_coded_decimal sqrt(binary_coded_decimal&);
```

如果进行如下声明，也可以使用标准 C 的 sqrt 函数：

```
extern"C" double sqrt(double);
```

这样，用户就拥有 C++类的 sqrt 的两个实例，其中有一个来自 C 库。但是，只能在 C 中定义 sqrt 的一个版本。

用户不能放置链接标识限制符，因为不能把它们放在一个局部作用域中，所有链接标识符必须在文件作用域中出现。注意，C++编译器认为以下语句是错误的：

```
// 这是一个错误：在函数作用域中不能有链接标识符

main()
{
extern "C" size_t strlen(const char *);          // 错误标志
//...
}
```

可将链接标识符移入文件作用域以改正这个“错误”：

```
// 放置链接标识符的正确位置是文件作用域(在函数体之外)

extern "C" size_t strlen(const char *);

void main()
{
//...
}
```

1.1.4 与其他语言的链接

虽然用户只看到 extern "C"链接指令，但 C 和它两边的引号说明链接机制可将 C++程序与其他语言的函数相链接。实际上，链接标识机制就是用以完成这一目的的，但 C++编译器只能支持两种语言：C 和 C++。正如用户所见的 C 链接的实例，C++是所有 C++程序的缺省链接。

1.2 如何使用 ANSI 标准 C 库

现在我们已经知道了如何在C++程序中使用 C 函数。若 C 编译器的标准头文件可以同 C++编译器配合工作，则可以简单地通过在 C++程序中包含适当头文件来使用 C 函数。作为一个 C 程序员，一定知道 C 通过 C 库能完成许多功能，诸如 I/O，内存管理和数学功能。每个 C 程序都使用函数，例如 printf，scanf 和 gets 等，这些函数均在 stdio.h 头文件中定义。除了 I/O 函数，标准 C 库中还有更多的函数，如字符串处理和以不同格式返回日期和时间的函数等等。以下章节概述了 ANSI C 库的功能并演示它们在 C++程序中的应用。

1.2.1 ANSI C 库功能概述

从第一册第三章的 ANSI 标准 C 的介绍中，可以知道这一标准不仅定义了 C 程序设计语言，还定义了库的内容以及函数的头文件和原型。由于大多数 C 编译器将会与 ANSI 标准 C 一致，标准 C 库就成为一个寻找 C++程序中使用的函数的好地方。

ANSI 标准为 C 库指定了 140 多个函数，但其中许多函数在多数 C 编译器中是无效的，因为它们是用来处理国际字符集的。表 1.1 按功能列举了标准 C 库中的大多数函数。以下将更深入地说明每个函数的种类以及如何在 C++程序中应用每个函数。

表 1.1 标准 C 库的功能

函数分类	函数名称
标准 I/O	clearerr, fclose, feof, ferror, fflush, fgetc, fgetpos, fgets, fopen, fprintf, fputc, fputs, fread, freopen, fscanf, fseek, fsetpos, ftell, fwrite, getc, getchar, gets, printf, putc, putchar, puts, remove, rename, rewind, scanf, setbuf, setvbuf, sprintf, sscanf, tmpfile, tmpnam, ungetc, vfprintf, vprintf, vsprintf
过程控制	abort, assert, atexit, exit, getenv, localeconv, longjmp, perror, raise, setjmp, setlocale, signal, system
内存分配	calloc, free, malloc, realloc
可变长度参数表	va_start, va_arg, va_end
数据转换	atof, atoi, atol, strtod, strtol, strtoul
数学函数	abs, acos, asin, atan, atan2, ceil, cos, cosh, div, exp, fabs, floor, fmod, frexp, labs, ldexp, ldiv, log, log10, modf, pow, rand, sin, sinh, sqrt, srand, tan, tanh
字符分类	isalnum, isalpha, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper
字符串和缓冲区操作	memchr, memcmp, memcpy, memmove, memset, strcat, strchr, strcmp, strcoll, strcpy, strcspn, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, strtok
搜索和排序	bsearch, qsort
时间和日期	asctime, clock, ctime, difftime, gmtime, localtime, mktime, strftime, time

1.2.2 标准 I/O 函数

标准 I/O 函数是在 stdio.h 头文件中定义的，其中包括 C 库中一些常用的函数(如 printf 和 scanf)。几乎所有 C 程序都至少使用这些函数中的一个。在 C++中用户可以继续使用它们。但是，正如第一册第八章所述，多数 C++编译器包括了 iostream 类库，它提供了一套更

为简洁的面向对象的用于C++程序的I/O函数。

1.2.3 进程控制函数

标准C库的种类包括：

(1) 管理错误条件的信号处理函数。

(2) 结束进程的实用程序，它与操作系统通信并建立数字和当前格式，依赖于程序定制的现场。

这些函数在以下头文件中定义：

(1) LOCALE.H 声明 localeconv 和 setlocale。

(2) SIGNAL.H 声明 raise 和 signal。

(3) SETJMP.H 声明 longjump 和 setjmp。

(4) STDLIB.H 声明 abort，atexit，exit，getenv，perror 和 system。

(5) ASSERT.H 声明 assert。

以下介绍这些函数的概要及用途。

1. 环境变量

进程(process)是指一个正在执行的程序，当一个程序运行时，就创建了一个进程。进程环境包括执行进程必要的信息。环境因操作系统的不同而不同。在UNIX和MS-DOS中，环境包括一个以空字符结束的字符串数组，每个字符串定义一种格式的符号：

```
VARIABLE= value
```

等式左边出现的符号是一个环境变量。在UNIX系统中，环境变量使用命令printenv和env中的一个。在MS-DOS中，在DOS提示符下键入SET可以看到一个环境变量的列表。

环境变量用于向进程传递信息。例如，在UNIX中，全屏幕编辑器vi使用TERM环境变量确定文本应显示在何种终端上。用户在环境变量TERM中指明终端类型，编辑器vi通过TERM值进行设置。用户程序同样可以利用环境变量。例如，在UNIX系统中，环境变量TZ指明了用户的时区。利用getenv函数可以获取调用函数getenv的环境变量的值，getenv是STDLIB.H头文件中定义的一个实用例程。

2. 使用 setjmp 和 longjmp 进行异常处理

在C中，可以使用setjmp和longjmp处理异常情况。也就是用setjmp保存进程的状态(或环境)，然后通过longjmp以及保存的状态跳转到程序先前执行到的那一点(状态和环境的含义相同，指在进程执行过程中，恢复某点运行所需的信息)。ANSI C需要编译器定义一个数组类型，即jmp_buf，它保存着重建一个调用环境所必需的信息。这个数据类型定义在SETJMP.H头文件中。为了理解setjmp和longjmp的使用技术，请分析如下程序：

```
/*------------------------------------------*/
/* 以实例说明如何通过 setjmp 和 longjmp 处理异常
   */

#include <setjmp.h>
jmp_buf last_context;

void process_commands(void);
```