Visual C++ 4.0
编程指南

知音难觅·好书难求

珍藏版

万跃华 主编

西北工业大学出版社

Visual C++ 4.0 编程指南

万跃华 主编

徐荣桥 朱晓芸 万跃华 编著

西北工业大学出版社

1997年4月 西安

(陕)新登字 009 号

【内容简介】 本书首先概述了C++语言、OOP技术、Windows 95下程序设计的主要内容，然后通过具体的例子，详细介绍了在Windows 95环境下Visual C++4.0各可视化工具包括AppWizard、ClassWizard、Resource Editors和Wizard-Bar的功能和具体使用方法。读者通过本书，不但可以了解风靡全球的C++语言和OOP方法，更重要的是学会利用这些工具，避开Windows 95应用程序中的大量细节，快速地开发为自身专业服务的Windows 95应用程序。

本书内容详实，图文并茂，示例说明性好，讲述由浅入深，适合于计算机应用、软件开发人员，软件用户及大专院校师生和广大计算机爱好者使用。

本书各章所附源程序均收录在一张软盘中，需要者，请与西北工业大学出版社编辑部联系。

Visual C++ 4.0编程指南

万跃华 主编
责任编辑 刘 红
责任校对 何格夫
*

(710072 西安市友谊西路127号 电话8493844)
全国各地新华书店经销
陕西省咸阳市印刷厂印装
ISBN 7-5612-0944-4/TP·128
*
开本:787×1092毫米 1/16 印张:28.625 字数:702千字
1997年4月第1版 1997年4月第1次印刷
印数:1—6 000册 定价:39.00元

前　言

Microsoft Visual C++ 4.0(简称 VC 4.0)是 C 和 C++应用程序的集成开发环境。它支持多平台和交叉平台的开发。VC 4.0 包括用来开发 Windows 95 或 Windows NT 应用程序的工作框架(Framework)和基本类库(Microsoft Foundation Class Library 4.0,简称 MFC 4.0)等主要内容。使用它们可以方便地开发 Windows 应用程序,并与 VC 4.0 支持的其他平台保持源程序级的兼容,即只要重新编译就能把应用程序从一个平台移植到另一个平台。

VC 4.0 提供了可视化的集成环境,称为 Microsoft Developer Studio。VC 4.0 的组件,一组功能强大齐全的基于 Windows 95 或 Windows NT 的开发工具,都在 Developer Studio 上运行,使得用户能在一个环境中完成应用程序的编辑、编译、测试、细化。这组工具包括:Text Editor、Resource Editor、Project Build Facilities、Optimizing Complier、Incremental Linker、Source Code Browse Window、Integerated Debugger 和 Book Onlines 等。这些工具都在 Windows 环境下运行,具有一致的外观和操作方式,只要熟悉一种就能触类旁通。

VC 4.0 还提供了功能强大、操作方便的辅助工具:

● AppWizard:它能创建一组完整的,基于 MFC 类库的源文件和必需的资源文件,称为应用程序的骨架。在这个基础上开发应用程序可以节省大量的时间和精力。

● ClassWizard:它能创建、编辑基于 MFC 的类及其成员和消息映射,完成许多烦琐的细节工作,使得捆扎(bind)Windows 消息变得相当容易。利用它还能方便、正确地重载基类成员函数、映射对话框数据等。

● WizardBar:它是 ClassWizard 的快捷方式,利用它,只要用鼠标轻轻一点,就能完成重载基类成员函数、捆扎 Windows 消息、定位成员函数等等。

本书的目的就是要介绍如何使用 VC 4.0 和 MFC 4.0 编写 Windows 95 应用程序。其中对 VC 4.0 各种工具的主要功能和使用方法作了详细的介绍。本书适合于计算机应用、软件开发人员,软件用户及大专院校师生,广大计算机爱好者使用。

本书采用循序渐进的方式,第一、二章主要介绍 C++语言、OOP 技术和 Windows 95 编程基础(对此比较熟悉的读者可以跳过);从第三章开始,以一个具体的实例来详细介绍 VC 4.0的使用,通过实例程序功能的不断增强,VC 4.0 的众多工具也得到越来越详细的介绍。

本书由万跃华主编,浙江工业大学万跃华,浙江大学徐荣桥、朱晓芸编著。

在本书编著过程中,得到了作者所在单位领导和许多同事的帮助,并提出了很多宝贵意见和建议,作者在此深表谢意。

鉴于编者水平所限,谬误和疏漏在所难免,敬请读者批评指正。

编　者

1996 年 9 月于杭州

目　录

第一章　面向对象程序设计及C++语言概述

§1-1　面向对象程序设计概述

面向对象程序设计(Object-Oriented Programming,简称OOP)是软件系统设计与实现的新方法。它利用代码的可扩充性和可重用性来提高软件开发的速度,通过数据封装和隐藏、设计与实现分离来控制维护软件的复杂性和软件维护的开销。

面向对象程序设计的中心概念之一是抽象数据类型。它包含一个类型和与之相关的操作。在大多数面向对象语言中,用"类"来描述抽象数据类型。类定义了以类型为基础执行所有操作的接口。为一个类定义的数据和操作称为类成员,如果在类内定义的数据和操作只能在类范围内被访问,则称它为私有成员(private),如果能在该类范围外被访问,则称为公共成员(public)。

对象(object)是被一个特定类说明的变量,也被称为该类的一个实例(instance)。类的一部分成员(一般是数据成员)可表示对象的状态、属性,而另一部分成员(通常是操作)则是对象为用户提供的接口,通过这些接口,可以对该对象实施各种操作,如改变对象状态,测试或传递消息等,而且这些操作对用户是完全封闭的,用户不需要了解各种操作的具体细节。

在典型的面向对象语言中,类中操作接口的定义可以和实现细节的定义分开,即允许系统的设计和实现分开,这在面向对象的程序设计中是非常重要的,在构成重复使用性和控制软件维护的开销也是很重要的。

代码的维护在面向对象程序设计中可以得到很好的控制,这是因为改变数据结构或算法(即实现类的代码)仅局限于实现这个类(或类的一部分)的代码区域内。

在面向对象程序设计中,还通过类的继承来引出类层次,以实现代码的重用性和可扩充性。继承的概念来自现实生活。类可以从另一个类中继承数据和操作来达到代码的重用性。

总之,使用面向对象程序设计可缩短开发时间和减少开发费用,能控制软件维护的开销,因此面向对象程序设计成为当今最流行的程序设计方法是不可逆转的潮流,掌握面向对象程序设计方法是每一个软件设计人员最基本的素质。

§1-2　C++语言概述

程序设计语言随软件技术的发展而快速发展,反过来它又推进软件产业的发展。60年代初期由FORTRAN、COBOL和ALGOL60编制的软件奠定了软件产业的基础,但是由于这些语言类型单调,程序控制过多依赖程序员的技巧,使得很难编出更大的程序。但是由于硬件不停地发展,它完全能胜任运行更大的程序,因此人们希望以规范的软件结构消除复杂软件内部的混乱。于是70年代兴起了结构化程序设计,出现了结构化的程序设计语言PASCAL和C,而且老版本语言纷纷改版,出现了FORTRAN77和COBOL-74。然而虽然70年代中期兴起的软件

工程曾一度缓解了软件危机，但是随着硬件的进一步发展和更大软件的需要，结构化程序设计方法越来越显得力不从心。终于在90年代初期出现了面向对象技术，给处于危机中的软件产业带来了希望之光，并以其众多的优点成为当代最成功的软件技术。现在软件产业的各主导语言纷纷向面向对象扩充，以求在新的竞争中立于不败之地。C++就是在这种情况下登上了程序设计语言的舞台的。

C++语言是 AT&T 公司在1985年正式推出的，其开发者是该公司贝尔实验室的系统程序员 Bjarne Stroustrup。

C++语言保留了 C 语言灵活且强有力的处理硬件接口和低层系统设计能力，保留了 C 语言的紧凑性和强有力的表达式能力。由于采取与 C 完全兼容的策略，使得 C++语言能利用 C 语言所取得的成果，并使大量的 C 程序员愿意使用 C++语言。

另外，更重要的是 C++语言是 C 语言面向对象的扩充，它除了 C 语言编制过程式程序的所有语法机制外，增加了类、重载、继承、虚函数等支持面向对象程序设计和高层问题抽象的平台。

§1-3 快速掌握 C++语言

本节主要介绍 C++语言中的新的内容，但并不涉及 C++中面向对象的特性。

一、注释行

C++中的注释行与其他语言的注释行一样，只是为了阅读和修改。在编译时，注释行被跳过，因此对编译器来说，注释行是可有可无的，而对程序设计者而言，注释行是必不可少的，因为它对于了解和维护软件起着举足轻重的作用。

C++中使用“//”(两个除号)表示从该字符起至该行结束为注释内容。C 语言中的“/*”和“*/”仍能使用。大部分编译器允许“//”嵌套在老式 C 语言注释符中。

二、变量类型

表1-1为 C++中预定义的基本类型。

表1-1 C++变量类型

类型名	字节数	别名	范围
int	*	signed, signed int	依赖系统
unsigned int	*	unsigned	依赖系统
__int8	1	char, signed char	−128～127
__int16	2	short, short int, singed short int	−32 768～32 767
__int32	4	signed, signed int	−2 147 483 648～2 147 483 647
char	1	signed char	−128～127
unsigned char	1	无	0～255
short	2	short int, signed short int	−32 768～32 767
unsigned short	2	unsigned short int	0～65535
long	4	long int, signed long int	−2 147 483 648～2 147 483 647

续　表

类型名	字节数	别　名	范　围
unsigned long	4	unsigned long int	0～4 294 967 295
enum	*	无	与int一样
float	4	无	3.4E+/－38(7位小数)
double	8	无	1.7E+/－308(15位小数)
long double	10	无	1.2E+/－4 932(19位小数)

注：①在Windows 95和Windows NT中，long double类型被直接映射成double类型；②signed和unsigned能在任意整型变量前修饰，char类型在缺省时为signed；③int或unsigned int依赖于系统，在MS－DOS和16位的Windows下，它是2个字节，在32位Windows(Windows 95或Windows NT)下，它是4个字节。

三、名字作用域

C++中的名字(包括变量名、函数名、对象或类型名)都只能在某一区域内使用，这个区域称为名字的作用域。作用域决定了非静态变量的生存期，也决定了名字的可见性。作用域可分成以下五种：

(1) 局部作用域。在块内声明的名字只能在块内该名字的声明点之后使用。函数形参的作用域是函数体的最外层。

(2) 函数作用域。语句标号是唯一具有函数作用域的名字，它们可以出现在函数体内的任意位置。

(3) 文件作用域。在任何块外声明的名字或类具有文件作用域，在它们的声明点以后的任何地方都可以使用它们。具有文件作用域的非静态对象一般称为全局变量。

(4) 类作用域。类成员名具有类作用域，它们只能通过类成员选择运算符(.，－>，.*和－>*)来访问。非静态类成员只属于该类对象。

(5) 原型作用域。在函数原型中声明的名字只在函数原型中有效。

可以在外块内声明一个名字以隐藏块外相同的名字，如图1－1所示，变量i在块内被重新声明，它就隐藏了块外的i。

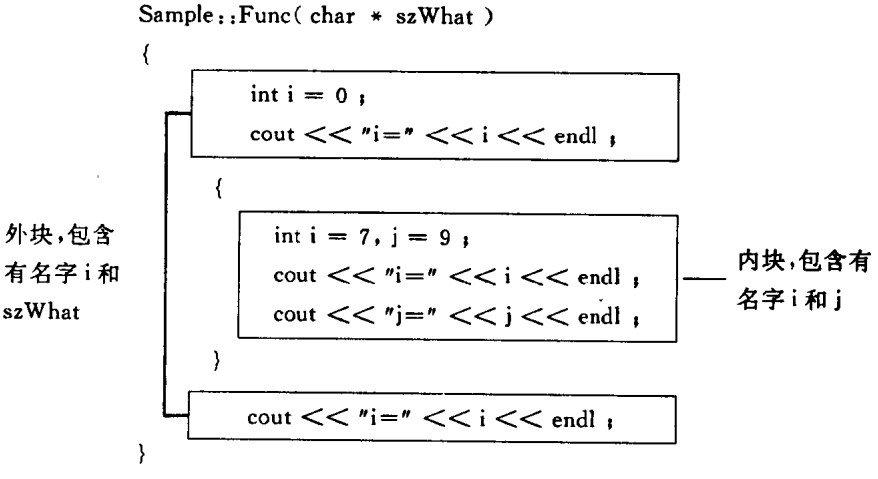

图 1－1　名字隐藏

图1－1所示代码的输出结果为：

```
i=0
i=7
j=9
i=0
```

同样，如果具有文件作用域的名字在某一块内被再次声明，则在该块内全局名字被隐藏，但是可以使用作用域分辨符(::)在块内访问全局名字。例如：

```
#include <iostream.h>
int i=7 ; // i has file scope, declared outside all blocks
void main()
{
    int i = 5 ;// have block scope, hides the i with file scope
    cout << "Block-scoped i has the value: " << i << endl ;
cout << "File-scoped i has the value: " << ::i << endl ;
}
```

上述代码的结果：

```
Block-scoped i has the value : 5
File-scoped i has the value : 7
```

在同一作用域内，声明一与类名相同的函数、对象或变量、枚举元素能隐藏类名，如果要使用类名则要加上前缀 class 关键词，如：

```
class Account
 {
  public :
    Account( double InitialBalance )
        { balance = InitialBalance ; }
    double GetBalance()
        { return balance ; }
  private :
    double balance ;
}
double Account = 15.37 ; // Hides class name Account
void main()
 {
    class Account Checking( Account ) ; // Qualifies Account
                                        // as class name
    cout << "Opening account with balance of: "
         << Check.GetBalance() << endl ;
}
```

注意：这种重用表示符的编程风格不值得提倡。

四、const 说明符

用 const 说明符说明一个对象后，该对象的值在其作用域内被冻结，即其值不能被修改。

例如：

```
const int i = 5 ;
i = 10 ; // Error
i++ ; // Error
```

在C++中，可以用const说明符来代替#define定义一组常数。如果用const定义，在编译时能进行类型检查，这是const比#define优越的地方。用const说明的变量也可以出现在要求常量表达式的地方，如在C++中可以用如下方式定义数值：

```
const int maxarray = 255 ;
char store_char[maxarray] ; // Legal in C++, illegal in C
```

在C中，常量在缺省情况下是外部连接，因此只能出现在源文件中，而在C++中，常量缺省情况下是内部连接，因此允许出现在头文件中。

const也可以用来说明指针：

```
char * const aptr = mybuf ; // const pointer
* aptr = 'a' ;              // Legal
aptr = yourbuf ;           // Illegal
```

const也可以说明指针所指的对象：

```
const char * bptr = mybuf ; // pointer to const date
* bptr = 'a' ;              // Illegal
bptr = yourbuf ;           // Legal
```

使用指向常量的指针作为函数参数，可以防止在函数体内修改该指针所指的对象。

用const说明一个类成员函数(类及其成员函数的概念将在§1－4中介绍)，则该函数是"只读"的，它不能修改该类的任何数据成员，也不能调用任何没有用const说明的其他成员函数。说明时把const置于表示函数参数表结束的右括号")"后面，例如：

```
// Example of a constant member function
class Date
{
  public :
     Date( int mn, int dy, int yr ) ;
     int GetMonth() const ; // A read-only function
     void SetMonth( int mn ) ; // A write function
                                  // can not be const
  private:
    int month ;
}

int Date::GetMonth() const
{
    return month ;     // Doesn't modify anything
}

void Date::SetMonth( int mn )
{
```

```
    month = mn ;      // Modify data member of class Date
}
```

五、new 和 delete

在C++中,在变量定义处建立"程序变量"(分配内存),而当在程序正文作用域范围内不再使用该程序变量时,则消除此名(释放内存)。从分配内存到释放内存的过程称为变量的生存期。C++的两个新操作符 new 和 delete 允许程序直接控制变量的生存期。new 操作符的语法如下(用方括号括起来的部分是可选的,全书同):

```
[::]new [placement] type-name [(initializer)]
[::]new [placement] ( type-name ) [(initializer)]
```

new 操作符为 type-name 对象分配内存,返回一个合适的非零指针。如果内存分配失败,则在缺省情况下返回 NULL,但是也可以改变这个缺省值。使用 new 分配的内存要用 delete 将其释放。

上述操作符语法中各参数的含义:

placement　如果重载(重载的概念在§1-5中详述)new,则此参数为其提供一个额外的参数。

type-name　要分配内存的对象的类型。如果其类型是由多个类型名组合而成的,则可以使用括号来改变组合的顺序。

initializer　对象的初始化值。new 操作符不能初始化数组,而且只有当类具有缺省的构造函数时,new 才能创建该类对象的数组。

为一个整型变量分配内存:

```
int * pi = new int ;
```

为一个字符型变量分配内存,并用字符 'a' 初始化该变量:

```
char * pc = new char( 'a' ) ;
```

利用 Date 的构造函数来创建一个 Date 对象,并返回指向该对象的指针(Date 类的声明见四):

```
Date * pmc = new Date( 7, 1, 1997 ) ;
```

创建一个字符数组:

```
char * pstr = new char[sizeof( str )] ;
```

创建一个二维数组,大小为 dim * 10。当创建多维数组时,除第一个维数外,其余均须常量:

```
char ( * pchar ) [10] = new char[dim][10] ;
```

创建一个一维数组,该数组元素是7个指向返回整型数的函数指针:

```
int ( * * p ) () = new ( int ( * [7] ) () ) ;
```

使用 new 操作符动态分配的内存如果没有用 delete 释放掉,则这些内存不再返回到内存堆中,并且对程序员来说这部分内存就被丢掉了。虽然这种丢失对许多小型的应用程序来说可不被注意,但是对大型程序,这种坏的程序设计可导致系统失败。delete 操作符的语法如下:

```
[::]delete pointer
[::]delete [] pointer
```

其中 pointer 必须是由 new 分配的指针。如果 pointer 指向一个数组，则使用第二种形式，即在 pointer 前加方括号"[]"。例如：

```
int * pi = new int ;
...
delete pi ;
char * pchar = new char[255] ;
...
delete [] pchar ;
```

六、cin 和 cout

像 C 一样，C++不包含预先定义的输入/输出函数。输入和输出是通过 stdio 或 iostream 外部库来处理的。

标准输入/输出流库提供了一个类型稳定、灵活而有效的方法，用以处理字符输入和整型、浮点数及字符串的输出。

流库定义标准 cout(为输出重定向)和 cin(为输入重定向)。

例：流 cout 及其运算符<<。

```
#include <iostream.h>
main ()
{
    float r1 = -27.3 ;
    float r2 = 26.9 ;
    char ch = 'A' ;
    int i = -560 ;
    double d = 1.23456789123 ;
    cout << "\nr1 * r2 = " << r1 * r2 ;
    cout << "\nch = " << ch ;
    cout << "\ni = " << i ;
    cout << "\nThe value of d is " << d ;
}
```

该程序的输出是：

```
r1 * r2 = -734.37
ch = 65
i = -560
The value of d is 1.234567
```

注意：字符串作为整型数输出，如果请求 ch 的字符值，则应使用下列语句：

```
cout << "\nch=" << chr( ch ) ;
```

例：流 cin 的运算符>>。

```
#include <iostream.h>
main ()
{
```

```
    int i ;
    float r ;
    char str[80] ;
    cout << "\nEnter an integer: " ;
    cin >> i ;
    cout << "\nEnter a real: " ;
    cin >> r ;
    cout << "\n"Enter a string: " ;
    cin >> str ;
    cout << "\nThe string input is: " << str ;
}
```

七、C++运算符及其优先级

C++保留了所有C的丰富的运算符,表1-2和表1-3列出了所有的运算符及其优先级。

表 1-2 C++运算符操作符功能

操作符	功能	操作符	功能	操作符	功能	操作符	功能
::	作用域运算符	new	分配内存	>	大于	*=	乘/赋值
[]	数组下标	delete	释放内存	>=	大于等于	/=	除/赋值
()	函数引用	,	逗号运算符	==	相等	%=	取余数/赋值
(type)	类型强制转换	*	乘	!=	不等	<<=	左移/赋值
.或->	成员选择	/	除	&	位与	>>=	右移/赋值
++	自加1(前/后级)	%	取余数	^	位异或	&=	位与/赋值
--	自减1(前/后级)	+	加	|	位或	^=	位异或/赋值
*	取指针内容	-	减	&&	逻辑与	|=	位或/赋值
&	取地址	<<	左移	||	逻辑或	?:	条件测试
!	逻辑非	>>	右移	=	赋值	&	引用
~	按位非	<	小于	+=	加/赋值		
sizeof	取类型长度	<=	小于等于	-=	减/赋值		

八、引用

引用是一个保存对象地址的变量,但是在语法上和对象一样操作。在C++中,任何对象的地址能被转换成给定指针的类型,也能转换成引用类型。例如:

```
//Example of reference operator
void AddIt( int & x )
{
    x += 2 ;
}
```

```
main ()
{
    int x ;
    x = 3 ;
    AddIt( x ) ;
    printf( "%d", x ) ; // the value of x is now 5
}
```

表 1－3　C++运算符优先级

	运算符	结合方式	运算符	结合方式	
高	++(后缀) --(后缀) [] -> .	从左到右	+ -	从左到右	高
			<< >>	从左到右	
			< <= > >-	从左到右	
	++(前缀) --(前缀) ! ~ & * sizeof new delete (type)	从右到左	== !=	从左到右	
			&	从左到右	
			^	从左到右	
			|	从左到右	
			&&	从左到右	
			||	从左到右	
	.* ->*	从左到右	?:	从右到左	
			= *=,/=, %= |-,|,<<-, >>=,&=,^=,|=	从右到左	
	* / %	从左到右			
低					低

九、无名共用体

无名共用体是没有名字的共用体。它不允许有成员函数，也不允许有私有和保护成员(类的私有、保护成员概念将在§1－4中详述)。全局的无名共用体必须是静态的。一个局部的无名共用体必须是静态的或自动的，但不能是外部的。

无名共用体的成员必须有唯一的名字，它们能被直接访问，例如：

```
static union
{
    int i ;
    float f ;
    union
    {
      char c ;
      unsigned char uc ;
    } ;
} ;

void func()
{
    i = 1 ;
    f = 3.14 ;
    c = 'c' ;
    uc = 'u' ;
}
```

十、C++中的函数

1. 函数原型

下例说明了C++中新的函数原型和老式C的函数的差别：

```
#include <stdio.h>
void function _1( int a, int b )        //新的函数原型
{
    printf( "\na=%d, b=%d", a, b ) ;
}

void function _2( a, b )                //旧的C函数说明
int a, b
{
    printf( "\na=%d, b=%d", a, b ) ;
}
```

2. inline 函数

一个函数说明能冠以说明符inline，例如：

```
inline int max( int a, int b )
{
    return a > b ? a : b ;
}
```

虽然inline说明符强制C++编译器在函数调用处直接插入max函数的代码可增加执行速度(因为节省了函数调用开销)，但是在代码长度方面可能有不良的后果。

3. 缺省自变量