

面向对象的程序设计

Borland
C⁺⁺

4

费向东
李炳法
邹贻明
郭新明

审稿
等编

新

成都科技大学出版社

Borland C++ 4.0

——面向对象的程序设计

费向东	李炳法	审稿
邹贻明	郭新明	等著

成都科技大学出版社
中国·成都

成都科技大学出版社出版发行

峨影印刷厂·印刷

开本:787×1092 1/16

印张:40 字数:600千字

印数:1—500册

版次:1994年7月第1版

印次:1994年7月第一次印刷

ISBN 7—5616—2867—6/TP·86

定价:39.00元

内 容 提 要

本书以 Borland International 公司最新推出的 Borland C++ 4.0 为载体,由浅入深围绕面向对象的主要特征为主干线,循序渐进地介绍 C++ 语言和面向对象程序设计技术。考虑到最近对 Windows 编程兴趣的迅速增加,本书还讨论了如何利用 OWL 编写各种复杂的 Windows 应用程序。为了方便广大读者学习和理解各种面向对象的编程技巧,全书贯穿了大量例程,都经过仔细调试,可以上机编译验证。适用于具有一定 C 语言编程经验的广大软件工作者,可作为大专院校学生和研究生参考书。

需要购买、批发本书的单位和个人,请直接与四川大学计算机系联系。

地 址:成都市望江路 29 号四川大学计算机系

邮 编:610064

电 话:028-5583875-2955

2516

联系人:郭新明 邹貽明

零售邮寄一律免邮寄费

前 言

今天,面向对象的编程(OOP)是一个很热门的话题。对于许多人来说,C++本身就是OOP的代名词,就象许多人认为LISP就是人工智能一样。然而实际上,OOP并非一种具体的语言,把它看作一种特殊方法的结晶也许更适当。我们完全可以把OOP应用于多种程序语言中,如Pascal,Ada,BASIC以及汇编语言等,尽管那会带来相应的困难。

本书强调C++中的OOP,使用特定环境——Borland C++——作为其载体。考虑到最近对Microsoft Windows编程兴趣迅速增加,本书挑选了一些Windows编程的技术做适当讲解。读者在编写Windows程序时,应尽量理解OOP的实质。

本书是一本手册书,书中贯穿了大量编程实例。所有的例程和代码都可以被调入并编译,以验证OOP所具有的许多新特征。对于其功能和效率,我们给了大量注解。要求本书读者熟悉C语言编程。高水平的读者也能在每章后面部分找到自己感兴趣的内容。

在80年代,C语言编程技术风靡全世界。C语言代码可以被高效地移植到各种计算机平台上。软件编写速度提高,以及软件工程的引入使软件平均代码量增加,由此带来的复杂度增加了软件开发时间。今天困扰许多公司的难题便是开发时间和效率的问题。AT&T发展了C++语言作为ANSI C的扩展,试图以此为C语言增加面向对象编程的优点,同时保留C语言本身的优点,如简洁、运行效率高等。

发展C++的目的之一是使编程更容易,要做到这一点,C++肯定要比C复杂。C++中所增加的特性都是为了帮助减少难度。显然,单纯采用C++并不能自动提高软件质量,也不能自动简化软件。要获得C++的优点,你必须采用一种新的编程方法——面向对象的编程,简称OOP。

几年前,计算机科学家们注意到程序员在一定时间内不论采用何种语言均可编写和调试几乎同样行数的代码。工作量相同,但结果并不相同。编写100行C语言代码与100行汇编语言代码难度相差不大,但这100行C语言代码显然能完成较多的任务。在这种思想引导下,研究人员试图发展一种能数倍提高单个程序员能力的高级语言,以降低项目开发的时间和费用。

早在70年代,编程语言研究者们对于“对象”(object)的概念已经比较熟悉了。一个对象便是一个若干代码与数据的集合,以模拟一个物理实体(Entity)或抽象实体。对象是一个高效编程单元,原因有二:它们代表常用单元的直接抽象,同时对其调用者而言其复杂性是不可见的。开发的第一批对象是与计算机紧密相联的对象,如整数、数组和栈。一些语言(如Smalltalk)设计得与其它语言不同,其所有内容都是对象。

面向对象的编程方法着重于对象之间的关系,而不重其完成的细节。关系是联接对象的纽带,通常会发展成一棵关系树,其中一些对象类型由其它对象类型发展而来。隐藏对象的实现使用户更注重对象间的关系而非单个对象的内容。这一点非常重要,代表了它与非面向对象语言(如C语言)的本质区别,后者编程着重于函数和函数调用。

对于C++,语言本身几乎没有对象。生成对象的任务完全落到用户身上。Borland C++则集成了一定数量的对象类型,但要真正应用仍需开发相当数量的对象类型。集成的对象类型通常叫做“类系”(class hierarchies)。发展类系是OOP的一项中心任务。

本书基于 Borland C++ 4.0 讲解发展类系的新方法。在此之前,本书讲解 C++ 的基本特征。C++ 的主要 OOP 特征在各章节中分别讲解。

本书可分为两部分,第一部分“C++”,对 C++ 语言作通常意义的讲解,特别注意了 Borland C++ 这一面向对象语言的特征。本书并不是一本完整的 Borland C++ 手册,只讲解如何应用其面向对象语言特征。

第二部分“OWL”,从第十章开始,讲解如何使用 OWL,带有大量实例。每一实例均经过测试,可以编译调试。

学习编程可以没有计算机,但通过上机却有益于学习。对于本书的学习,你不但要了解大量的实例代码,还应该修改它,加以完善,并进行编译调试。这需要以下系统环境:

- IBM PC AT 或兼容机
- MS-DOS 3.31 或后继版本
- Microsoft 兼容鼠标
- VGA 或更高级显示适配器
- Borland C++ 4.0
- OWL 2.0(第十章到十二章要求)
- Windows 3.1 或后继版本

第十到十二章的例子程序的工程文件是用 Borland C++ 4.0 建立的,如果使用低版本编译器,可能不能成功。

不需要 Microsoft SDK for Windows。如果你仅有 Turbo C++ 或 Borland C++ 4.0 以前版本,一到七章大部分代码可以顺利通过,但对第八、九两章的程序需要适当修改。

第十到十二章的 Windows 程序要求有 Borland C++ 4.0,但如果只有 Borland C++ 3.1 和 Windows 3.1,对源程序适当修改后也可通过。

参加本书编写工作的有:

张天庆、朱敏、谭胜勇、罗以宁、余建坤、廖果、王炎、刘军、谭多望、邹贻明、郭新明。

目 录

第一章 基础

§ 1.1 Borland C++工程文件的结构	(1)
§ 1.1.1 头文件	(1)
§ 1.1.2 一个完整的例程	(2)
§ 1.2 变量	(6)
§ 1.2.1 作用域	(6)
§ 1.2.2 数据类型	(8)
§ 1.2.3 存储类	(9)
§ 1.2.4 CONST 修饰符	(9)
§ 1.2.5 Volatile 修饰符	(11)
§ 1.3 语句	(11)
§ 1.3.1 表达式语句	(12)
§ 1.3.2 if 语句	(13)
§ 1.3.3 Switch 语句	(14)
§ 1.3.4 标号语句	(15)
§ 1.3.5 While 语句	(15)
§ 1.3.6 do while 语句	(16)
§ 1.3.7 for 语句	(17)
§ 1.3.8 break 语句	(17)
§ 1.3.9 continue 语句	(18)
§ 1.3.10 goto 语句	(19)
§ 1.3.11 return 语句	(19)
§ 1.4 函数	(20)
§ 1.4.1 传送参数给函数	(20)
§ 1.4.2 传递 const 参数	(21)
§ 1.4.3 使用默认参数	(21)
§ 1.4.4 从函数返回值	(22)
§ 1.4.5 返回常量	(22)
§ 1.4.6 返回值的问题	(23)
§ 1.4.7 使用函数修饰符	(24)
§ 1.5 指针和引用	(25)
§ 1.6 在指针和引用中用 const	(27)
§ 1.7 高级部分	(28)
§ 1.7.1 使用嵌入汇编语言	(28)
§ 1.7.2 名字管理	(30)
§ 1.7.3 一起使用 C 和 C++	(30)
§ 1.7.4 由传值返回大结构	(31)
§ 1.7.5 使用枚举	(33)

§ 1.7.6	使用内存管理	(34)
§ 1.7.7	C 调用顺序	(36)
§ 1.7.8	Pascal 调用顺序	(37)
§ 1.7.9	中断处理函数	(38)
§ 1.7.10	带变参表的函数	(39)
第二章 对象和类		
§ 2.1	定义类	(41)
§ 2.1.1	类标识符	(42)
§ 2.1.2	类体	(42)
§ 2.2	类的使用	(43)
§ 2.2.1	封装	(43)
§ 2.2.2	类的存取控制	(44)
§ 2.2.3	私有类成员	(45)
§ 2.2.4	公有类成员	(46)
§ 2.2.5	保护类成员	(46)
§ 2.2.6	用于类对象的存储类	(48)
§ 2.2.7	类作用域	(48)
§ 2.2.8	空类	(48)
§ 2.2.9	嵌套类	(48)
§ 2.2.10	类实例	(50)
§ 2.2.11	不完全类说明	(51)
§ 2.3	使用数据成员	(51)
§ 2.3.1	静态数据成员	(52)
§ 2.3.2	私有静态数据成员	(54)
§ 2.3.3	作为数据成员的对象	(55)
§ 2.3.4	用作数据成员的引用	(56)
§ 2.3.5	用作数据成员的指针	(57)
§ 2.3.6	指向类数据成员的指针	(57)
§ 2.3.7	指向对象数据成员的指针	(59)
§ 2.4	使用成员函数	(59)
§ 2.4.1	简单成员函数	(60)
§ 2.4.2	静态成员函数	(61)
§ 2.4.3	Const 成员函数	(62)
§ 2.4.4	Volatile 成员函数	(63)
§ 2.4.5	inline 成员函数	(63)
§ 2.4.6	带 const 自引用指针的成员函数	(64)
§ 2.4.7	带 volatile 自引用指针的成员函数	(65)
§ 2.4.8	特殊的类函数	(67)
§ 2.4.9	构造函数	(67)
§ 2.4.10	析构函数	(73)
§ 2.4.11	Friend 关键字	(75)
§ 2.4.12	友元的特性	(76)
§ 2.5	高级部分	(77)

§ 2.5.1 指向成员函数的指针	(77)
§ 2.5.2 数组和类	(79)
§ 2.5.3 成员函数调用分析	(84)
§ 2.5.4 类模板	(86)
§ 2.5.5 函数模板	(93)

第三章 继承

§ 3.1 可重用性	(96)
§ 3.2 继承	(96)
§ 3.3 继承的功效	(97)
§ 3.4 C++ 继承的局限	(97)
§ 3.5 对继承的一个不同看法	(98)
§ 3.6 单一继承	(99)
§ 3.6.1 何时继承	(99)
§ 3.6.2 什么不能被继承	(99)
§ 3.6.3 基类的访问说明符	(100)
§ 3.6.4 设计用于继承的类	(101)
§ 3.6.5 向基类传递参数	(102)
§ 3.6.6 构造函数的调用顺序	(103)
§ 3.6.7 析构函数的调用顺序	(104)
§ 3.6.8 “种”类	(104)
§ 3.6.9 派生类的类型转换	(107)
§ 3.6.10 分辨作用域	(107)
§ 3.6.11 特征扩展	(110)
§ 3.6.12 特征限制	(113)
§ 3.6.13 一个使用单一继承的例子	(114)
§ 3.6.14 函数闭包	(116)
§ 3.7 多重继承	(122)
§ 3.7.1 用多个基类说明一个类	(123)
§ 3.7.2 调用基类构造函数	(123)
§ 3.7.3 使用虚基类	(124)
§ 3.7.4 同时使用虚基类和非虚基类	(126)
§ 3.7.5 调用析构函数	(126)
§ 3.7.6 使用类型转换	(126)
§ 3.7.7 保持基类函数连续	(128)
§ 3.7.8 对多重继承使用作用域分辨	(129)
§ 3.7.9 保持对内存的跟踪	(131)
§ 3.8 高级部分	(132)
§ 3.8.1 运行时的考虑	(132)
§ 3.8.2 在一个对象中	(132)
§ 3.8.3 一个继承调试程序	(135)

第四章 重载

§ 4.1 为什么应该重载	(139)
§ 4.2 函数重载	(140)

§ 4.2.1	重载非成员函数	(140)
§ 4.2.2	重载成员函数	(141)
§ 4.2.3	在一个类层次结构中重载函数	(142)
§ 4.2.4	重载不是覆盖	(144)
§ 4.2.5	作用域分辨	(144)
§ 4.2.6	变量匹配	(144)
§ 4.2.7	重载构造函数	(145)
§ 4.2.8	一些特殊情况	(147)
§ 4.2.9	通过重载的用户转换	(148)
§ 4.2.10	重载静态成员函数	(151)
§ 4.3	运算符重载	(151)
§ 4.3.1	用作函数调用的运算符	(152)
§ 4.3.2	重载的运算符用作成员函数	(153)
§ 4.3.3	运算符成员函数的注释	(155)
§ 4.3.4	重载的运算符用作友元函数	(155)
§ 4.3.5	赋值运算符	(157)
§ 4.3.6	函数调用运算符()	(159)
§ 4.3.7	下标运算符	(161)
§ 4.3.8	运算符重载的限制	(162)
§ 4.3.9	运算符的作用域分辨	(162)
§ 4.4	高级部分	(163)
§ 4.4.1	名字管理的规则	(164)
§ 4.4.2	重载 new 和 delete	(166)
§ 4.4.3	前缀和后缀运算符	(169)

第五章 多态性

§ 5.1	早期联编与滞后联编	(171)
§ 5.2	C++是一种混合语言	(172)
§ 5.3	虚函数	(172)
§ 5.4	函数重载	(173)
§ 5.5	改进类的用户界面	(174)
§ 5.6	抽象类	(175)
§ 5.7	虚函数的局限	(178)
§ 5.8	虚友元	(178)
§ 5.9	虚运算符	(179)
§ 5.10	一个多态性的例程	(182)
§ 5.11	作用域解析使多态性失效	(185)
§ 5.12	虚函数与非虚函数	(186)
§ 5.13	Vptr 与 Vtab 结构的内存布局	(187)
§ 5.14	虚函数没有必要被重载	(187)
§ 5.15	是否说明为虚函数	(189)
§ 5.16	虚函数也能作为私有成员	(190)
§ 5.17	提高部分	(191)
§ 5.17.1	多态性机制	(192)

§ 5.17.2	具有单一继承的多态性	(192)
§ 5.17.3	具有多重继承的多态性	(196)
§ 5.17.4	嵌入的虚函数	(199)
§ 5.17.5	在基类中调用多态性函数	(202)
§ 5.17.6	虚函数与分类层次	(204)
§ 5.17.7	在一个构造函数中调用虚函数	(206)

第六章 异常处理

§ 6.1	处理异常情况的老方法	(209)
§ 6.2	处理异常情况的 OOP 方法	(209)
§ 6.3	抛出异常情况	(210)
§ 6.4	抛出初始化的对象	(210)
§ 6.5	捕获异常情况	(212)
§ 6.6	捕获未定类型的异常情况	(214)
§ 6.7	使用多重捕获区块	(214)
§ 6.8	使用 catch/throw 而非 setjmp/longjmp	(215)
§ 6.9	查找恰当的异常情况处理程序	(216)
§ 6.10	绕开堆找	(216)
§ 6.11	作为类对象的异常情况	(217)
§ 6.12	用引用处理异常情况	(218)
§ 6.13	异常情况类型的层次	(219)
§ 6.14	处理带有异常情况的常见错误	(220)
§ 6.15	异常情况和资源占用	(220)
§ 6.16	使用函数闭包	(221)
§ 6.17	异常情况和构造函数	(223)
§ 6.18	带有子对象的对象中的异常情况	(225)
§ 6.19	标准 C++ 异常情况	(226)
§ 6.20	OWL 中的错误处理	(228)
§ 6.21	TXOWL 异常情况	(230)
§ 6.22	TXCompatibility 异常情况	(230)
§ 6.23	TXGdi 异常情况	(230)
§ 6.24	TXInvalidMainWindow 异常情况	(231)
§ 6.25	TXInvalidModule 异常情况	(231)
§ 6.26	TXMenu 异常情况	(232)
§ 6.27	TXout Of Memory 异常情况	(232)
§ 6.28	TXPrinter 异常情况	(233)
§ 6.29	TXValidator 异常情况	(234)
§ 6.30	TXWindow 异常情况	(234)
§ 6.30.1	窗口编辑中的警告	(236)
§ 6.30.2	异常情况处理和回叫函数	(236)
§ 6.31	多执行路线代码中的异常情况处理	(237)
§ 6.32	异常情况接口说明	(237)
§ 6.33	unexpected() 函数	(239)
§ 6.34	Terminate() 函数	(241)

§ 6.35 调试对 Terminate() 的调用	(242)
----------------------------------	-------

第七章 流

§ 7.1 老式 ANSI C 的缺陷	(244)
§ 7.2 C++ 流	(245)
§ 7.3 作为一般筛选器的流	(245)
§ 7.4 标准流 I/O 与内部数据类型	(246)
§ 7.4.1 I/O 与 char 与 char* 类型	(248)
§ 7.4.2 I/O 与整型和长整型	(249)
§ 7.4.3 I/O 与浮点型和双精度型	(250)
§ 7.4.4 I/O 和用户类	(251)
§ 7.5 操作程序(/操作符)	(253)
§ 7.5.1 使用基于数字的操作程序	(254)
§ 7.5.2 设置和清除格式标志	(255)
§ 7.5.3 改变字段宽度的填充	(256)
§ 7.5.4 使用格式化操作程序	(256)
§ 7.5.5 用户自定义操作程序	(258)
§ 7.5.6 带参数的操作程序	(261)
§ 7.5.7 操作程序与用户流类	(263)
§ 7.6 文件 I/O 和流	(266)
§ 7.6.1 使用文本文件作输入	(266)
§ 7.6.2 流的错误检查	(268)
§ 7.6.3 使用文本文件作输出	(269)
§ 7.6.4 使用二进制文件作输入	(271)
§ 7.6.5 使用二进制文件作输出	(273)
§ 7.6.6 文件拷贝	(276)
§ 7.7 内存格式	(278)
§ 7.8 将打印机作为一个流	(280)
§ 7.9 高级部分	(281)
§ 7.10 Streambuf 的层次结构	(282)
§ 7.11 ios 层次结构	(283)
§ 7.11.1 类 filebuf	(283)
§ 7.11.2 类 fstream	(288)
§ 7.11.3 类 fstreambase	(290)
§ 7.11.4 类 ifstream	(294)
§ 7.11.5 类 ios	(296)
§ 7.11.6 类 istream	(307)
§ 7.11.7 类 istream—withassign	(310)
§ 7.11.8 类 ostream	(311)
§ 7.11.9 类 ostream—withassign	(317)
§ 7.11.10 类 istrstream	(318)
§ 7.11.11 类 ofstream	(320)
§ 7.11.12 类 ostream	(323)
§ 7.11.13 类 ostream—withassign	(327)

§ 7.11.14	类 ostream	(328)
§ 7.11.15	类 streambuf	(332)
§ 7.11.16	从 streambuf 中派生出一个类	(339)
§ 7.11.17	类 stringstream	(343)
§ 7.11.18	使用类 stringstream	(344)
§ 7.11.19	类 stringstreambase	(345)
§ 7.11.20	类 stringstreambuf	(347)

第八章 基于对象的集成库

§ 8.1	类目录	(353)
§ 8.2	Abstract Array 类(抽象数组类)	(355)
§ 8.3	Array(数组类)	(359)
§ 8.4	使用 Array 类(数组类)	(360)
§ 8.5	重用存储槽	(361)
§ 8.6	结合类	(362)
§ 8.7	定义使用结合的对象	(364)
§ 8.8	使用结合类	(365)
§ 8.9	从结合中派生类	(367)
§ 8.10	无序的单位组类(Bag 类)	(368)
§ 8.11	基本期类	(372)
§ 8.12	Base Time 类(基本时间表)	(374)
§ 8.13	B-树类	(376)
§ 8.13.1	树的入门	(377)
§ 8.13.2	二叉树	(377)
§ 8.13.3	性能	(378)
§ 8.13.4	B-树	(378)
§ 8.14	聚集类	(391)
§ 8.15	集成器类	(392)
§ 8.16	Degree 类	(397)
§ 8.17	字典类(Dictionary 类)	(399)
§ 8.17.1	一个字典例子	(400)
§ 8.17.2	使用字典集成外部重复	(402)
§ 8.18	双向表类	(403)
§ 8.19	Error 类(错误类)	(407)
§ 8.20	散列函数类(HashTable 类)	(410)
§ 8.21	表类(List 类)	(417)
§ 8.22	对象类(Object 类)	(420)
§ 8.23	优先队列类(priorityQueue 类)	(424)
§ 8.23.1	使用 PriorityQueue 类	(426)
§ 8.23.2	将一个 PriorityQueue 转换成一个 GIFO 队列	(429)
§ 8.24	队列类(Queue 类)	(430)
§ 8.24.1	Set 类(集合类)	(433)
§ 8.24.2	一个处理串(strings)的集合(set)类	(434)
§ 8.25	一个更数学化的 Set 类	(435)

§ 8.26	Sortable 类	(438)
§ 8.27	已分类数组类(SortedArray 类)	(439)
§ 8.28	栈类(stack 类)	(441)
§ 8.29	String 类(串类)	(443)
§ 8.30	使用 String 类	(445)
§ 8.30.1	从 String 派生一个类	(447)
§ 8.30.2	时间类(Time 类)	(450)
§ 8.31	使用 Time	(450)
§ 8.32	从 Time 派生类	(452)
§ 8.33	重复器	(455)
第九章 基于模板的容器库		
§ 9.1	FDS 容器和 ADT 容器	(457)
§ 9.2	FDS 容器	(457)
§ 9.3	FDS 存储范例	(458)
§ 9.4	FDS 容器	(458)
§ 9.5	FDS 容器向量窗口器	(458)
§ 9.5.1	简单直接向量	(459)
§ 9.5.2	计数直接向量	(460)
§ 9.5.3	有序直接向量	(461)
§ 9.5.4	简单间接向量	(465)
§ 9.5.5	有序间接向量	(466)
§ 9.6	FDS 表容器	(468)
§ 9.6.1	简单直接表	(468)
§ 9.6.2	有序直接表	(470)
§ 9.6.3	间接表	(473)
§ 9.6.4	有序间接表	(474)
§ 9.7	ADT 容器	(475)
§ 9.8	用容器管理内存	(476)
§ 9.9	ADT 数组	(476)
§ 9.10	ADT 有序数组	(480)
§ 9.11	ADT 堆栈	(482)
§ 9.12	ADT 直接堆栈	(482)
§ 9.13	ADT 间接堆栈	(485)
§ 9.14	ADT 队列和解散队列	(486)
§ 9.15	ADT 元素单元组和集合	(489)
§ 9.16	多相 ADT 容器	(492)
第十章 OWL 类		
§ 10.1	OWL 调试板的使用	(494)
§ 10.2	OWL 应用程序的入口	(494)
§ 10.3	定制主窗口	(496)
§ 10.4	OWL 对话框	(498)
§ 10.4.1	对话框类型的选择	(499)
§ 10.4.2	灰色三维对话框	(499)

§ 10.4.3	Borland 风格的对话框	(500)
§ 10.4.4	对话框中的位图	(501)
§ 10.4.5	位图子控制的 OWL 数字化方案	(502)
§ 10.4.6	对话框数据的读或写	(503)
§ 10.5	OWL 子控制	(508)
§ 10.6	数据的有效性	(508)
§ 10.7	Validator 选项	(509)
§ 10.7.1	TFilterValidator 的使用	(510)
§ 10.7.2	TRangeValidator 的使用	(510)
§ 10.7.3	TPxPictureValidator 的使用	(511)
§ 10.8	TStringlookupValidator 的使用	(512)
§ 10.9	定制的控制器的使用	(513)
§ 10.10	BWCC 定制控制器的使用	(513)
§ 10.11	Glyphs	(513)
§ 10.12	自定义的控制器	(515)
§ 10.13	自定义的无线按钮	(521)
第十一章 多文档界面的应用		
§ 11.1	窗口层次	(529)
§ 11.2	定制 MDI 客户区	(530)
§ 11.3	框架和客户绘画消息	(530)
§ 11.4	改变预定义窗口类许多的属性	(531)
§ 11.5	绘画实背景色	(533)
§ 11.6	绘制抖动背景颜色	(535)
§ 11.7	拖动并放下	(537)
§ 11.8	键盘控制	(538)
§ 11.9	主消息循环	(539)
§ 11.10	TApplication::ProcessAppMessage() 函数	(541)
§ 11.11	允许键盘控制操作的窗口	(541)
§ 11.12	具有子窗口控制的 MDI 子窗口	(543)
§ 11.13	改变 MDI 窗口的背景	(543)
§ 11.14	处理 MDI 子窗口下的子窗口控制	(546)
§ 11.15	子窗口控制的两种结构	(547)
§ 11.16	两种构造函数之间的差异	(547)
§ 11.17	一个完整例子	(547)
§ 11.18	将对话框用作 MDI 子窗口	(553)
§ 11.19	MDI 应用程序的菜单	(557)
§ 11.20	替换主菜单	(558)
§ 11.21	合并菜单	(562)
§ 11.22	OLE 2.0 的菜单合并	(562)
§ 11.23	OWL 合并菜单的方式	(563)
§ 11.24	改变菜单项的状态	(567)
§ 11.25	命令的启用	(567)
§ 11.26	为菜单命令增加检验标记	(569)

第十二章 列表框的变异

§ 12.1	Tab stops	(571)
§ 12.2	设置 Tab stops	(571)
§ 12.3	一个短小的例子	(572)
§ 12.4	制表停止溢出	(575)
§ 12.5	用多个列表框模拟制表停止	(575)
§ 12.5.1	移动列表框选择条	(576)
§ 12.5.2	一个完整的例子	(578)
§ 12.5.3	用户可定制的显示格式	(585)
§ 12.5.4	改变列宽度	(586)
§ 12.5.5	重新安排列	(586)
§ 12.5.6	删除不必要的列	(586)
§ 12.5.7	跟踪字符串范围	(588)
§ 12.5.8	增加列表框字符串	(588)
§ 12.5.9	删除列表框字符串	(589)
§ 12.5.10	清除列表框	(589)
§ 12.5.11	一个完整的例子	(590)
§ 12.6	多列列表框	(597)
§ 12.6.1	列溢出	(598)
§ 12.6.2	一个短小的例子	(598)
§ 12.7	自己绘制列表框	(601)
§ 12.7.1	LBS—HASSIRINGS 风格	(601)
§ 12.7.2	显示肖像	(602)
§ 12.7.3	WM—MEASUREITEM 消息	(602)
§ 12.7.4	WM—DRAWITEM 消息	(603)
§ 12.7.5	一个例子	(605)
§ 12.7.6	一起使用肖像和字符串	(610)
§ 12.7.7	绘制肖像和文字	(611)
§ 12.7.8	WM—COMPAREITEM 消息	(613)
§ 12.7.9	一个完整的例子	(613)
§ 12.8	没有制表溢出的列数据	(617)
§ 12.8.1	截短长字符串	(617)
§ 12.8.2	绘制文字	(618)
§ 12.8.3	一个完整的例子	(619)
§ 12.8.4	排序列表框的列	(624)

第一章 基础

本章讨论在 ANSI C 基础上扩充而来的 C++ 主要特征,但并不涉及对象和其它面向对象的结构。本章主要是为了解释那些较易理解、不需进一步讨论的 C++ 特点。对于更多的基本内容,我们建议读者查阅 Borland C++ 程序员指南。

与纯的面向对象语言相对,C++ 被认为是一种不纯的面向对象语言,其原因是它建立在常规的过程语言之上。其它一些语言(例如 Eiffel 或 Smalltalk)是纯的面向对象语言,但并不是说纯的更好。衡量好坏往往要看语言如何应用以及要解决的是什么任务。

C++ 源代码在运行前必须先通过编译。编程的过程应该是多次重复编辑、编译、联接及运行这几步。尽管重复这一过程相当慢,但代码的生成却非常快,并且效率较高。C++ 在表达能力、运行速度和代码尺寸之间选择了一个极好的平衡点。进一步而言,Borland C++ 集成开发环境(IDE)自动地把开发周期的所有步骤集成在一起,极大地加快了开发速度。

因为 C++ 设计为 C 的扩展改进,它包含了传统 ANSI C 的全部特性。事实上,任何一种软件在新版本中即使是对一些很基本的内容也作了相应修改。本章中所讲述的各节都是 C 语言的基本内容。C 程序员在本章可能找不到什么新内容,但并不意味着可以跳到第二章,因为其中涉及一些内容的改动。

§ 1.1 Borland C++ 工程文件的结构

就结构而言,Borland C++ 的工程文件与 ANSI C 工程文件很类似。你可以在头文件中定义结构(和类)。C++ 头文件的后缀是 *.hpp,用以区别 ANSI C 头文件的后缀 *.h,因为 C++ 头文件中可能包含 ANSI C 下不能编译的内容。真正的 C++ 源程序可以放在一个或多个文件中,编译后联接在一起,这与 ANSI C 的编译联接相同。

§ 1.1.1 头文件

类标识符必须先定义后使用,通常在类定义时做这件事。要使用类的公有成员的话,不仅要求类标识符在其作用域内,还要求类被完整定义。因为类都有其定义,而且在不同的模块中都可能调用它,所以常把类定义放在头文件中,而在每一个引用到这些类的模块中包含这一头文件。

一 多次引用问题

考虑典型程序中会遇到的有大量类和头文件的情况,可能要在同一个文件中多次引用某个头文件。考虑下列文件:

```
// file HEADER.HPP
class EssentialClass { /* ... */ };

// file DESKTOP.HPP
#include "HEADER.HPP"
class DeskTop { /* ... */ };
```