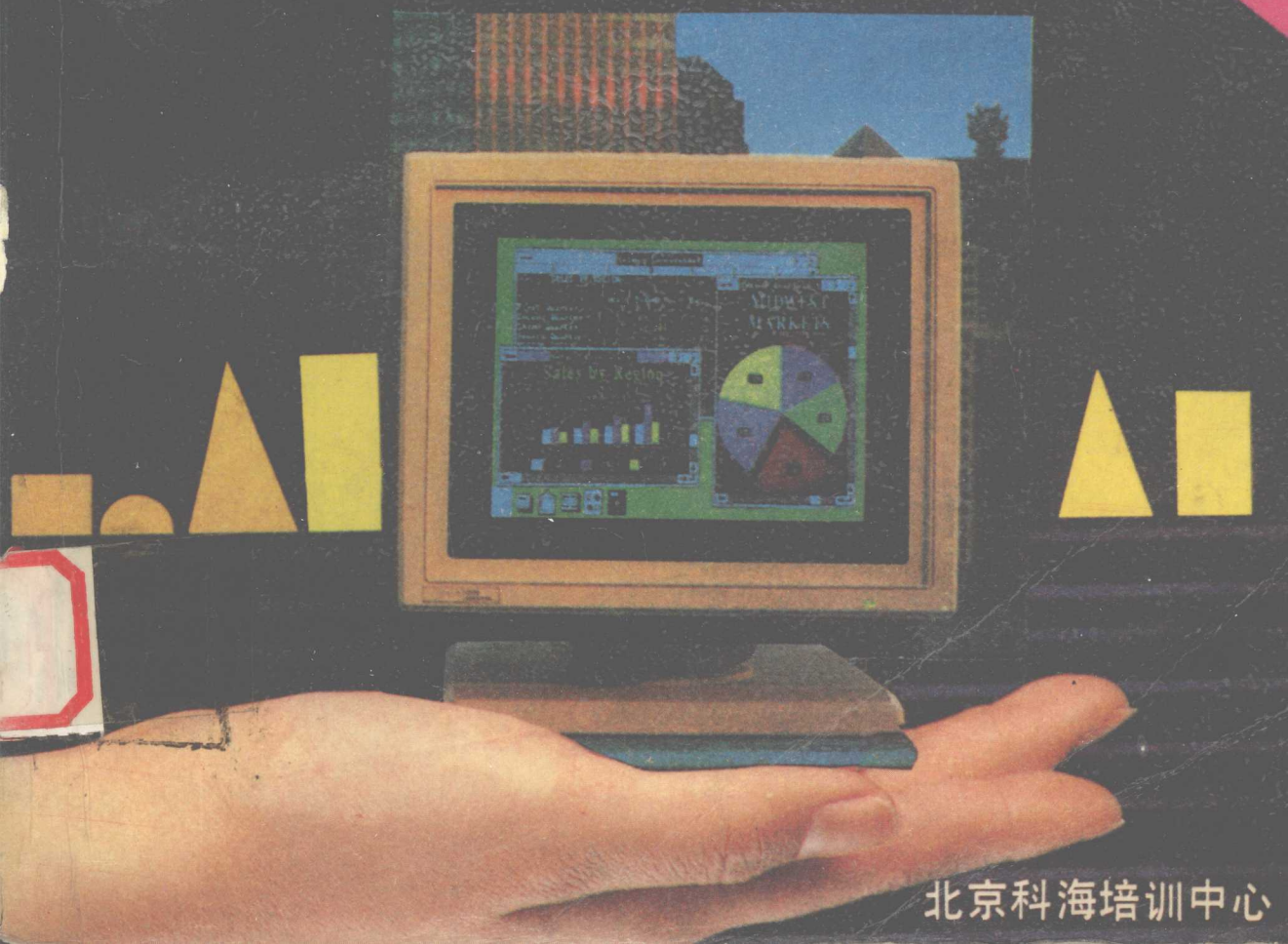
Turbo C++
及面向对象的编程方法

徐金梧　编译



北京科海培训中心

Turbo C ++
及面向对象的编程方法

徐金梧　编译

北京科海培训中心

一九九一年十一月

内容简介

C++是一种新的计算机编程语言，确切地讲，它以崭新的面目向广大编程者提供了一种更容易维护和开发的程序设计方法——面向对象的程序设计方法．它以一种全新的设计和构造软件的思维方法开拓了计算机编程方法历史上一个新的世纪，并将成为 90 年代主体语言．目前，国外大量的实用程序现已纷纷改变成C++版本，这部分体现了 C++强劲的功能和与无伦比的优越性。

本书以 Turbo C++为蓝本，通过大量实例阐明了面向对象的程序设计方法的基本特征，譬如，继承、多质性和封装．本书最具特色的部分是：怎样用面向对象的编程方法来编写实用的文本模式和图形模式下的弹出式窗口和下拉菜单；如何用鼠标器和键盘来编写用户界面；交互式屏幕对象．本书所提供的工具箱对你今后编写一个具有良好用户界面的实用程序是不可缺少的。

目　录

第一章　面向对象的编程方法简介 …… (1)
1.1　计算机语言的演变 …… (1)
1.2　OOP 是什么? …… (2)
1.3　OOP 观点 …… (3)
1.3.1　OOP 是一种极好的程序设计方法 …… (3)
1.4　OOP 的基本组成部分 …… (4)
1.4.1　类和对象 …… (4)
1.5　OOP 的两个特点 …… (4)
1.5.1　封装 …… (4)
1.5.2　继承 …… (5)
1.6　实例 …… (6)
1.6.1　从结构到对象 …… (6)
1.6.2　面向对象的编程实例 …… (9)
1.6.3　什么是继承 …… (10)
1.7　小结 …… (14)
第二章　面向对象编程方法的应用 …… (15)
2.1　文件阅读器 …… (15)
2.1.1　文件阅读器是如何工作的 …… (16)
2.2　基本框架 …… (17)
2.2.1　使用缓冲区 …… (18)
2.2.2　使用屏幕 …… (19)
2.3　完整的文件阅读器程序 …… (20)
2.4　改写文件阅读器程序 …… (26)
2.4.1　用对象来设计 …… (26)
2.4.2　阅读器对象 …… (27)
2.4.3　屏幕类 …… (28)
2.4.4　文件缓冲区类 …… (29)
2.4.5　使用继承 …… (31)
2.4.6　使用对象 …… (32)
2.5　面向对象的阅读器程序 …… (33)
2.6　代码评价 …… (43)
第三章　屏幕和键盘工具 …… (44)
3.1　屏幕对象的综述 …… (44)
3.2　PC 机文本屏幕 …… (44)

3.3 虚拟屏幕类 …… (46)
3.3.1 缓冲区初始化 …… (48)
3.3.2 屏幕的写操作 …… (49)
3.3.3、移动屏幕内存 …… (50)
3.4 其它屏幕工具 …… (51)
3.5 使用键盘接口 …… (52)
3.6 检验屏幕和键盘 …… (53)
第四章 面向对象的鼠标器工具箱 …… (65)
4.1 鼠标器工具箱综述 …… (65)
4.2 使用鼠标器 …… (65)
4.3 鼠标器对象 …… (66)
4.3.1 与鼠标器驱动程序通讯 …… (67)
4.3.2 说明和使用鼠标器对象 …… (68)
4.3.3 检验鼠标器 …… (68)
4.3.4 配置鼠标器 …… (69)
4.3.5 控制鼠标器光标 …… (69)
4.3.6 使用鼠标器坐标 …… (70)
4.3.7 移动鼠标器 …… (70)
4.3.8 使用鼠标器按键 …… (71)
4.3.9 使用鼠标器事件 …… (72)
4.4 鼠标器检验程序 …… (74)
4.5 图形模式下使用鼠标器 …… (76)
4.5.1 两个光标掩码 …… (76)
4.5.2 建立图形光标 …… (77)
4.5.3 检验图形光标 …… (77)
4.6 代码评价 …… (79)
4.6.1 抽象类 …… (79)
4.6.2 事件的处理 …… (80)
第五章 建立窗口和菜单平台 …… (87)
5.1 一个世界，许多对象 …… (87)
5.2 什么是屏幕对象 …… (87)
5.3 基本结构 …… (88)
5.3.1 秘决在于继承 …… (89)
5.4 从矩形开始 …… (90)
5.4.1 剪切矩形区域 …… (91)
5.4.2 相交的矩形 …… (92)
5.5 矩形屏幕的对象类 …… (93)
5.6 移动文本——类 Trso …… (94)
5.6.1 多个对象的连接 …… (96)

5.6.2 增加输出功能 …… (96)
5.6.3 复制和交换映象 …… (97)
5.7 使用 Trso 类 …… (98)
5.8 代码评价 …… (101)
5.8.1 增加清屏功能 …… (101)
5.8.2 支持文本光标 …… (101)
5.8.3 类的合并问题 …… (101)
第六章 建立框式屏幕对象 …… (108)
6.1 框式屏幕对象 …… (108)
6.1.1 检验坐标的状态 …… (110)
6.1.2 如何使用屏幕映象和阴影 …… (111)
6.1.3 如何使用属性和颜色 …… (111)
6.2 Tfso 类 …… (113)
6.2.1 交换对象和显示阴影 …… (115)
6.2.2 确定文本字符串的尺寸 …… (117)
6.3 显示窗口 …… (118)
6.4 屏幕对象的骨架 …… (121)
6.4.1 Tskel 的成员函数 …… (122)
6.5 代码评价 …… (122)
6.5.1 改变框 …… (122)
6.5.2 附加标题 …… (123)
6.5.3 限定成员函数的访问 …… (123)
6.5.4 在建立通用类时应注意的问题 …… (123)
第七章 交互式屏幕对象 …… (133)
7.1 支持多屏幕对象 …… (133)
7.2 交互式屏幕对象的等级 …… (133)
7.2.1 Iso 类 …… (134)
7.2.2 屏幕对象堆栈 …… (137)
7.2.3 Iso 的初始化 …… (140)
7.2.4 确定 Iso 的尺寸和位置 …… (140)
7.2.5 打开和关闭 Iso …… (141)
7.2.6 显示 Iso 对象 …… (142)
7.2.7 移动和延伸对象 …… (143)
7.3 事件处理系统 …… (143)
7.3.1 信息传递系统 …… (144)
7.3.2 如何处理事件 …… (144)
7.3.3 键盘事件 …… (145)
7.4 屏幕对象的管理机构 …… (146)
7.4.1 事件循环 …… (146)

7.5 建立文本窗口 …… (147)
7.6 支持文本屏幕 …… (148)
7.7 类 Wso 是如何工作的 …… (149)
7.8 覆盖成员函数 …… (151)
7.9 建立按键对象 …… (152)
7.10 代码评价 …… (155)
7.10.1 特征的组合 …… (155)
7.10.2 在窗口内输出 …… (155)
7.10.3 信息传递系统 …… (155)
7.10.4 功能的扩充 …… (156)
第八章 建立下拉菜单 …… (174)
8.1 建立下拦菜单 …… (174)
8.2 菜单综述 …… (175)
8.2.1 使用 msounit: 文本还是图形? …… (176)
8.2.2 建立菜单项 …… (176)
8.2.3 指定菜单操作 …… (177)
8.3 菜单表类 …… (178)
8.4 Mso 类 …… (179)
8.4.1 建立菜单 …… (181)
8.4.2 覆盖成员函数 …… (184)
8.5 菜单实例 …… (184)
8.6 建立下拉式菜单 …… (186)
8.6.1 Pmso 类 …… (187)
8.6.2 建立下拉菜单主框 …… (188)
8.6.3 下拉菜单的表示 …… (189)
8.7 下拉菜单系统实例 …… (190)
8.8 代码评价 …… (193)
8.8.1 图形菜单 …… (193)
8.8.2 多重父类 …… (193)
8.8.3 强制类型转换 …… (193)
8.8.4 矩形菜单 …… (194)
第九章 文本屏幕对象使用实例 …… (203)
9.1 可滚动的窗口 …… (203)
9.1.1 滚动条 …… (204)
9.1.2 滑块类 …… (206)
9.2 建立滚动条 …… (207)
9.2.1 使用鼠标器操作 …… (208)
9.2.2 与其它对象通讯 …… (208)
9.3 接收信息 …… (210)

9.4 通用的可滚动窗口类 …… (211)
9.5 可滚动窗口类 …… (213)
9.6 对话窗口 …… (217)
9.6.1 通用对话窗口类 …… (217)
9.6.2 信息窗口 …… (218)
9.6.3 带可选项的对话窗口 …… (220)
9.7 代码评价 …… (223)
9.7.1 柔性滚动窗口 …… (223)
9.7.2 可滚动窗口对象 …… (223)
9.7.3 对象的组合 …… (223)
9.7.4 对象间的通讯 …… (223)
第十章 图形屏幕对象 …… (234)
10.1 从文本模式到图形模式 …… (234)
10.2 图形类的综述 …… (235)
10.2.1 使用图形 …… (236)
10.2.2 图形模式下的矩形 …… (236)
10.2.3 初始化 Grgo 对象 …… (237)
10.2.4 使用像素 …… (237)
10.2.5 使用不同的窗口格式 …… (238)
10.2.6 写字符串 …… (239)
10.2.7 填充操作 …… (239)
10.2.8 绘制矩形 …… (239)
10.3 Gfso 类 …… (241)
10.3.1 初始化 Gfso 对象 …… (242)
10.3.2 绘制 Gfso 对象 …… (243)
10.3.3 移动 Gfso 对象 …… (243)
10.4 修正轮廓对象 …… (244)
10.5 Wso 类 …… (245)
10.5.1 Wso 的成员函数 …… (245)
10.6 图形模式工具 …… (245)
10.7 一个简单实例 …… (246)
10.8 一个菜单实例 …… (248)
10.9 代码评价 …… (251)
10.9.1 快速交换函数 …… (251)
10.9.2 提供较大的交换缓冲区 …… (251)
10.9.3 使用多种字体 …… (251)
10.9.4 增添填充模式 …… (252)
10.9.5 其它特性 …… (252)
第十一章 交互式图形对象 …… (262)

11.1 建立图形对象 …… (262)
11.2 派生选择按键 …… (262)
11.3 派生图案类 …… (264)
11.4 按键检验程序 …… (265)
11.5 带滚动条的窗口 …… (268)
11.5.1 图形模式下的滚动条 …… (268)
11.5.2 滚动条综述 …… (269)
11.6 滚动条类 …… (271)
11.6.1 初始化滚动条 …… (271)
11.6.2 滚动条鼠标器操作 …… (272)
11.6.3 滚动条的信息记录 …… (273)
11.7 滚动条窗口 …… (273)
11.8 可滚动窗口的实例 …… (274)
11.8.1 处理键盘输入 …… (276)
11.8.2 发送信息 …… (277)
11.9 主程序 …… (278)
11.10 代码评价 …… (281)
11.10.1 合并文本与图形滚动窗口 …… (281)
11.10.2 使用带有其它对象的滚动条 …… (281)
11.10.3 滚动条作为输入设备 …… (281)
附录 A Turbo C++简明指南 …… (290)
A.1 定义类 …… (290)
A.2 构造成员函数 …… (291)
A.3 建立和使用对象 …… (291)
A.4 参数 this …… (292)
A.5 使用构造函数和析构函数 …… (292)
A.6 建立动态对象 …… (294)
A.7 派生新的类 …… (295)
A.8 复态性 …… (296)
A.9 虚拟函数 …… (297)
A.9.1 在虚拟函数中使用参数 …… (298)
A.10 从内部看对象 …… (298)
A.11 对象之间的赋值相容性 …… (299)
A.12 对象作为参数传递 …… (300)
A.13 参数传递和多态性 …… (301)
A.14 使用类型转换的指针给对象 …… (302)
A.15 调用继承函数 …… (303)
A.16 确定哪个继承函数被调用 …… (303)
A.17 在派生类中使用构造函数 …… (305)

A.18 通过继承来隐藏数据 …… (305)
A.19 使用对象数组 …… (306)
A.20 使用组合对象 …… (307)
A.21 继承不同类型 …… (308)
A.22 在头部文件中定义类 …… (308)

第一章　面向对象编程方法简介

你也许会感到编程是一件烦人的事。当你编写一个实用程序后，往往会感到还有很多地方需要修改（至少对一个好的程序常常如此）。一个程序被使用得越多，用户要求对该程序作进一步修改和增添一些新功能的机遇也就越大。不幸的是，一旦程序编完后，再要对程序进行修改常常是十分困难的。使用 Turbo C^{++}和面向对象的编程技术（object-oriented programming，简称 OOP）能够大大地改善这种状态，因为对象允许我们设计更加容易扩充和可复用的代码。

在这一章中，将介绍有关 OOP 一些主要的概念。首先扼要地讨论一下 OOP 的历史背景，然后介绍 OOP 程序设计中的一些关键问题和 OOP 的基本术语，如类（classes）、对象（objects）实例变量（instance variables）和方法（methods），还将介绍 OOP 的两个主要特征：封装（encapsulation）和继承（inheritance）。在本章的末尾将通过一个简单的实例来看一下如何用 Turbo C^{++}实现面向对象的编程。

1.1　计算机语言的演变

自从计算机被引入以来，编程技术已经经历了一个漫长的道路。如图 1.1 所示，从人们使用象"ADD AX，5"，"JMP ERROR"这种低级指令和转向语句开始，编程技术迈进了混沌时代。无须细说，这种混沌时代并没有维持多长时间，多数编程已进入自 60 年代开始的结构化时代。在这个新的时代里，引入了很多编程语言，如 Pascal、C 和 Ada，并且引入了许多其它的工具来帮助编程者整理混乱的编程方法。

现在，我们正处在另一个新的时代--一个对象和面向对象的时代的门槛上。这将是一个与结构化具有同样革命性和重要性的新世纪。OOP 的目的十分明确：使编程更容易。随着用户要求程序功能越来越多，程序变得越来越复杂，对于一个好的工具来说，这种应变能力就显得越来越重要。这就是面向对象的语言，诸如 C^{++}这类语言为什么能如此迅速发展的真正原因。

1950–1960 浑沌时代	1970–1980 结构化时代	1990– 对象时代
jumps,gotos 非结构化变量 由程序弥散的变量	if–then–else 程序模块 记录 while 循环	对象 消息 方法 继承

图 1.1 从浑沌时代到对象时代的演变

1.2 OOP 是什么?

简单地讲，OOP 是用对象来编程。然而，对象又是什么呢？对象是记录概念的具有变革性的拓延，记录允许我们将各种数据组织在一个程序块中，而对象允许我们将数据和代码连结在一个独立的程序块中。简言之，一个对象是将各种数据和对这些数据进行操作的各种函数约束在一起的一种语言结构；你可以将对象想象为能够包含代码的 C 语言的结构形式，但是，请不要通过这个简单的比喻误解了对象的内在含义。对象远不止这一点，它具有更强的功能，对象可以通过继承这一特征来组织新的数据元素和函数。关于这一特征将在以后详细介绍。

由于对象包括了数据和代码，它们就象微型的、独立的程序。这就允许我们将它们用来构造程序块，建立更加复杂的对象。这很象晶体管和开关可以用来构造一个电路。

譬如，假设你需要建立一个字处理应用软件，你可以建立一系列对象，而不必编写一系列函数来处理各种所需的任务。图 1.2 表示一种可供选择的方案。这个程序分成几个对象，如低级屏幕对象、窗口对象、文件管理对象等等，这种表达形式的优点是什么呢？一个程序的主要操作部分很容易被分隔开。由于对象可以被设计成互相之间完全独立工作的，这就使得程序的维护更加容易。因为程序是由对象的组合构成的，而不是单个函数组成的，对函数的维护和修改是一件十分困难的事情。另外请注意：某些对象，如低级屏幕对象和键盘控制对象是用来建立其它对象的。这有助于隐藏低级屏幕对象的细节，并且允许我们以搭积木的方式来构造各种对象。

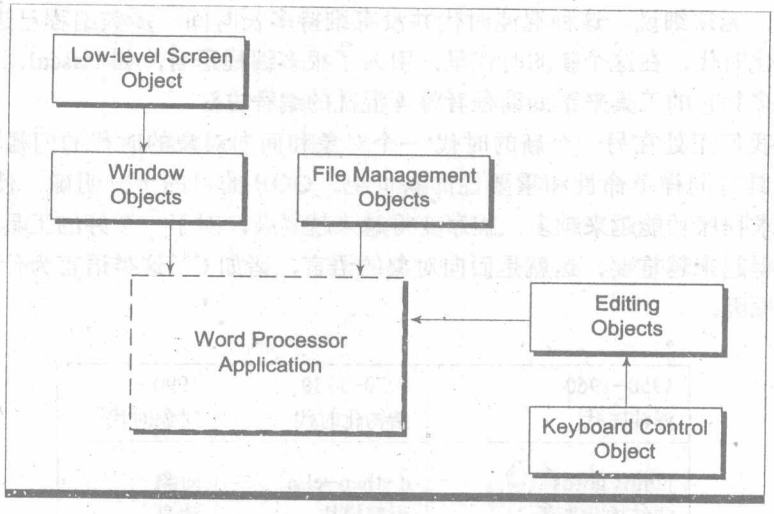


图 1.2 使用对象工作方式

1.3 OOP 观点

为了体现 OOP 提供的灵活性和功能强的优点，你必须改变编程方法。图 1.3 解释了传统的面向函数的编程方法与面向对象编程方法的主要区别在于：必须将代码与数据构造在一个程序块中，并且统一来管理。其它重要的区别是：

(1) 按传统的观点，函数是最重要的。一个程序中的所有代码都应围绕这些函数来设计。

(2) 按OOP的观点，对象是最重要的。程序是围绕这些对象来设计的，函数是第二位的。这种主角的改变体现在对象的用法上。我们用对象调用函数，而不是简单地将对象（数据）传递给函数。

(3) 程序可以避免包含多重逻辑分支（CASE语句）的大型函数，取而代之的是：多重对象来表示一个程序中不同的逻辑分支。

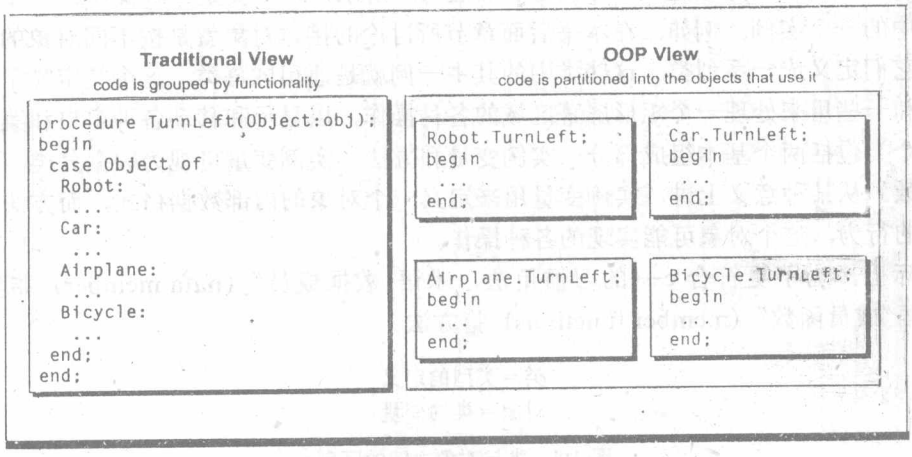


图 1.3 传统编程方式与 OOP 编程方法的比较

1.3.1 OOP 是一种极好的程序设计方法

近 10 年来，各种编程语言和工具都被设计成具有突出结构设计概念的优点。但是在很多场合，这些语言和工具未能提供足够的灵活性来处理现代软件要求所带来的复杂性。当然，结构化语言和编程工具有助于我们编写更加容易组织的代码，但是如果它们不易维护，那么这种结构化程序的好处又怎样体现呢？

现在的问题是：OOP 技术能有助于我们编写更加容易维护和修改的程序吗？答案是肯定的。另外，OOP 提供了使我们成为更好的程序设计者所需的支持。当你按照本书的方式去编程时，就会明白使用对象就象搭积木一样可以随心所欲地编写代码以适应各种不同的情况。例如，接口工具可以以文本模式和图形模式进行工作。(这可不是一件容易的事!)

1.4 OOP 的基本组成部分

虽然 Turbo C++是一种复杂的语言，但它只有四个也许是新的 OOP 基本组成部分。它们是：

(1) 类；

(2) 对象；

(3) 实例变量；

(4) 方法。

学习如何以面向对象方式编程的第一个障碍是弄清类与对象之间的区别。上一节我们引入了对象这个概念，术语"对象"实际上已经被使用了。

1.4.1 类和对象

类是一种模板，用来定义一个对象。如图 1.4 所示，一个类是一种类型，一个对象是一个类中的一个实例。例如，在本书后面章节所讨论的接口对象就是按不同对象的共同特征，将它们定义成一系列类。这些类中的其中一例就是通用屏幕类，这个类定义了一些数据元素和一些用来处理一个矩形屏幕区域的各种操作，以显示和移动各种窗口和菜单。

一个类包括两个基本组成部分：实例变量和方法。实例变量可视为数据分类，而方法表示函数。从某种意义上讲，实例变量用来定义一个对象的内部数据格式，而方法定义一个对象的行为，这个对象可能实现的各种操作。

实际上，为了更符合 c++的习惯用法，术语"数据成员"（data member）指实例变量，术语"成员函数"（member functions）指方法。

类＝类型的定义
对象＝类的变量

图 1.4 类与对象之间的区别

1.5 OOP 的两个特点

当用 C++面向对象编程时，必须弄清两个重要特征：

(1) 封装（Encapsulation）；

(2) 继承（Inheritance）。

简单地讲，封装是将数据和对这些数据进行处理所需的各种操作连接在一个根下的技术，继承是引用已经定义的各种类，并将它们扩充以满足新的功能的一种技术。下面我们将对封装和继承详细讨论。

1.5.1 封装

封装提供两个重要的特征：

(1) 使得数据和函数在一个根下；

(2) 使得数据具有隐藏能力。

在日常生活中，我们经常依赖于封装原理，尤其当与一些复杂的机器打交道时，例如，当你驾驶一辆小汽车时，并不需要知道它是前轮还是后轮驱动，而只要知道一踩油门，车能够开就够了。(当然还应当了解如何驾驶和使用刹车!)

使用 OOP 技术编程，也同样运用了这个原理。例如，我们编写数据库对象，并不想知道对象是怎样被存储在数据库系统中的，所要了解的唯一的一件事是：调用哪些程序可以访问这个对象。

通常，封装有三个目的。第一，防护一个对象的数据，以避免它们暴露在对象之外，因为这些数据只能被在这个对象内部定义的函数所访问。第二，封装使得运用这些数据更加方便，因为所有的操作只能通过已定义的接口。(再强调一下，通过由这个对象有关的各种操作。) 第三，封装可以用来隐藏数据是如何存储的，如何实现的等各种细节。

1.5.2 继承

继承可以从现有的类中派生出新的类：

(1) 在没有改变原始类的情况下，增加一些数据和代码；

(2) 重复使用代码；

(3) 改变一个类的性质。

让我们再来看一下小汽车这个例子它是如何继承的。假设一条装配线上只生产配置一些基本部件的同一类小汽车。你可以认为它们是一类低档的小汽车，譬如我们经常见到的大众型小汽车。现在需要增加一些设备，譬如 200W 的立体声音响，天鹅绒坐椅套，镀铬的轮箍，生产厂家可以将这种大众型小汽车改装成豪华型小汽车，又譬如标准的发动机可能换成大马力的高性能赛车发动机，这样，大众型就变成了赛车型。

我们可以使用这条基本的装配线来派生出其它的装配线，并由它们来生产各种想象中的小汽车。这些装配线继承了上一道工序，因为它们都是在这条基本的装配线上生产的同一类型汽车基础上加工出各种特殊类型的小汽车。

通过小汽车这个例子，我们应当明白这一点：同一条装配线派生出了其它几条装配线。这种共享资源的方法会给我们带来更好的工具和更有效益的产品，而面向对象的编程方法正是这样做的。我们建立各种类，它们可以复用，这样在编写一个新的应用软件时就不必每次从头做起。

继承的另一个有用的特征是：可以任意修改一个现有的类来产生与这个类有细微差别的新类。用这种方法，我们可以用继承的方法来产生多个对象，它们是在同一树枝下派生出来的。

再来考虑小汽车这个例子。我们可以用继承的方法从经济型小汽车派生出不同类型的小汽车，这些汽车有着不同的性能。例如通常经济型小汽车的油门踏脚板不十分灵活，这对于那些比较高档的小汽车来说是十分不合适的，应该换成另一种形式的油门踏脚板。

这后一种技术一改变不同对象所共享的部件的性能一应当引起我们足够的重视。在刚才这个例子中，实际上是利用了 OOP 的多质性 (polymorphism)，或者称为"多态性"。更换了油门踏脚板后，虽然表面上看它们没有什么区别，但是在功能上却有很大差异。

多质性的一个重要特点是：在一个族中的每个对象可以有同样名称的不同方法，这些

方法的代码可以完全不同。

为了了解多质性是如何实施的，我们假设一个通用的窗口类，它是作为在不同模式下（文本模式和图形模式）工作的两种特殊窗口类的基础。这个通用类可能有一成员函数Draw( )，它用来在屏幕上画出一窗口。为了在不同模式下显示这个窗口，我们需要在文本模式或图形模式中覆盖或修改该函数。在文本窗口对象中，窗口是使用字符，而在图形模式中却是象素。但不论哪种方式，最终的结果是显示一个窗口。

1.6 实例

前面我们已经介绍了 OOP 的基本原理，现在让我们看一下 C++是如何将这些概念变成代码的。下面通过一个简单的例子，解剖一下它是怎样从一个简单结构化程序演变为一个面向对象的应用软件的。这个例子将解释 Turbo C++所提供的有关 OOP 的关键特征。

1.6.1 从结构到对象

前面我们已经提到了编程方法的演变过程。从混沌时代经结构化时代到对象时代已经历了 30 年历史。了解为什么要用 C++来编写面向对象的程序的最好的方法是：了解一个程序是如何演变过来的。假设需要编一个程序，它用来管理公司雇员出差费用。我们先用结构化语言（譬如 C 语言）来编程。

首先，定义数据结构来支持程序中所要用到的各种信息。在这个例子中，下面这个结构可用来处理有关每个雇员出差的信息。

```
struct ExpenseRec { / * The trip expense structure * /
  char TripName[80], Date[80];
  float Mileage, Cost, Fringe;
};
```

然后，必须考虑一些函数用来提供用户接口和处理这个数据结构。完整的程序如下：

```
/* expense1.c: A structured version of the trip expense program */

#include <stdio.h>
#include <string.h>

struct ExpenseRec { /* The trip expense record */
  char TripName[80], Date[80];
  int Mileage, Cost, Fringe;
};

void CheckWithBoss(struct ExpenseRec ExRec
/* The boss is very tough. He won't approve every trip. */
{
  if (!stricmp(ExRec.TripName, "Hawaii") ||
    !stricmp(ExRec.TripName, "Tahiti"))
    printf("%s: You are fired for trying to sneak this one by\n",
           ExRec.TripName);
  else if (ExRec.Cost/4 < ExRec.Fringe)
    printf("The fringe expenses for the %s trip are too high\n",
           ExRec.TripName);
```

```
  else printf("The %s trip is ok\n", ExRec.TripName);
}

int I, ExCount;
struct ExpenseRec ExRec[20];  /* A list of employee records */

main()
{
  printf("How many trips did you take? ");
  scanf("%d", &ExCount);
  for (I=0; I<ExCount; I++) {
    printf("Please enter the name of your trip: ");
    scanf("%s", ExRec[I].TripName);
    printf("Which date did you leave (mo/day/year)? ");
    scanf("%s", ExRec[I].Date);
    printf("Enter the number of miles for your trip: ");
    scanf("%d", &ExRec[I].Mileage);
    printf("Enter the cost of your trip: ");
    scanf("%d", &ExRec[I].Cost);
    printf("Enter the cost of your meals and entertainment: ");
    scanf("%d", &ExRec[I].Fringe);
  }
  for (I=0; I<ExCount; I++) CheckWithBoss(ExRec[I]);
  return 0;
}
```

这个程序的主要任务是：记录有关出差的信息，然后调用 CheckWithBoss( )函数来看一下雇主是否批准这次出差。当然，这种使用结构形式的数据的优点在于：所有有关的数据放置在一个根下。但是，它们仍然没有将数据和操作这些数据的代码集中在一起。这就是我们考虑使用对象的原因，因为只有对象才允许我们将数据和操作连接在一个程序块中。

根据这一观点，很容易将这种费用记录转换成对象。如下所示：

```
struct Expense {                    / / The trip expense class
  char TripName[80], Date[80];
  int Mileage, Cost, Fringe;
  void AskQuestions (void) ;
  void CheckWithBoss (void) ;
};
```

这段程序看起来很象我们所用到的结构定义，但有一个很重要的区别：它包括了函数原型。这些函数原型指明了对象中所要用到的方法或成员函数。在这个例子中，对象 Expense 的成员函数是 AskQuestions( )和 CheckWithBoss( )。

在结构定义中的变量，如 TripName，Data，Mileaga 等是实例变量，或者用 C++的术语称为类的数据成员。数据成员可以是任何 C++的标准类型，如 int、chaf、float、double 或用户定义的类型，也包括其它类。

正象前面所提到的那样，在 Expense 结构中所说明的函数仅仅是函数原型。函数的代码是在这个结构定义的外部。例如，成员函数 AskQuestions( )：