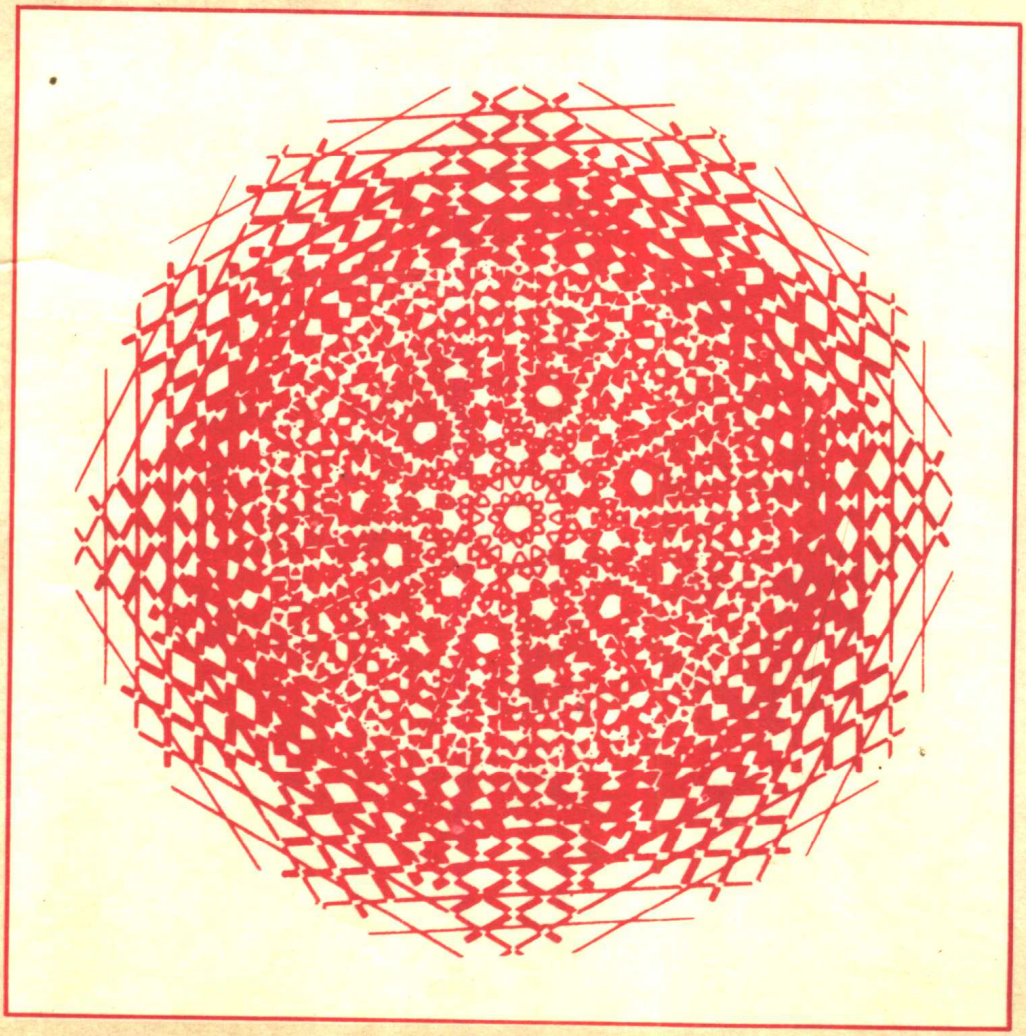
Borland C++ 2.0 程序设计指南

●邓集锋　寇应展　赵粤生　王建新　编著



電子工業出版社

Borland C++2.0程序设计指南

邓集锋　寇应展
赵粤生　王建新　编著

電子工業出版社

(京)新登字 055 号

内 容 提 要

本书介绍的 Borland C++ 2.0 版本是最新的面向对象的程序设计软件包。全书共分四大部分，分别介绍了 Borland C++ 集成开发环境、程序设计技术、实用程序、高级特征及应用分析，并结合软件开发的经验剖析了实现任务类的典型例子。本书内容丰富，覆盖面广，具有新颖性、实践性、实用性，是学习 C++ 程序设计的实用参考书。适合广大计算机软件开发、教学与应用的技术人员、教师、学生和计算机高级程序设计人员阅读与参考。

Borland C++ 2.0 程序设计指南

邓集锋 凌应展
赵粤生 王建新 编著

责任编辑 陆伯雄 祖振升

*

电子工业出版社出版(北京市万寿路)
电子工业出版社发行 各地新华书店经售
电子工业出版社计算机排版室排版
北京科技印刷厂印刷

*

开本：787×1092 毫米 1/16 印张：28.5 字数：730 千字
1992 年 9 月第 1 版 1992 年 9 月第 1 次印刷
印数：10100 册 定价：17.50 元
ISBN-5053-1880-2/TP·447

前　言

面向对象的程序设计是计算机程序设计的崭新模式，它提供了一种与结构化程序设计完全不同的计算机程序设计、思考和研究的新方法，使计算机解决问题的方式更接近于人类活动本身。

C++是面向对象程序语言的杰出代表。一方面，C++ 作为 C 语言的超集，保持与 C 语言的高度兼容性，继承了 C 语言高效率和强有力的特性。另一方面，它又支持面向对象程序设计，而且与 Ada、Modula－2、Smalltalk 等面向对象语言相比，它具有更完善的体系结构和更加强大的功能。C++ 倡导软件开发的新思维。从应用的角度看，它可以充分地挖掘硬件的潜力，在降低开发成本的同时，提供了强有力的软件开发工具。正是由于具有这些优良特性，C++ 可以广泛用于各种系统软件和应用软件的开发实践中。可以预料，C++ 将对操作系统软件工具、图像处理、数值分析、人工智能、数据库管理系统等等产生深远的影响。

Borland C++ 与 Turbo C 高度兼容，并提供 C++ 编程的全部功能。它全部实现了 AT&T C++ 2.0 版，支持 C++ 1.2 版流，同时支持 ANSI C 的标准及 K&R C 定义，并且具有一定程度的功能扩展，可以使用混合语言和混合内存模型编程以充分发挥 PC 机的潜能。Borland C++ 编译器可编译 C 及 C++ 源代码，具有很高的编译效率和很强的代码优化能力。Borland C++ 提供使用 8087 协处理器的浮点运算例程，同时可以在没有协处理器时进行仿真运算。另外，它还支持鼠标操作和多文件编辑以及内部汇编语言。

Borland C++ 适用于 IBM PC 系列的计算机（包括 XT、AT 和 PS/2 及与 IBM 全兼容的计算机）。它要求 DOS 3.0 以上的版本，至少 640K 内存，80 列显示器，PC 机至少配备一个硬盘和一个软盘驱动器。

本书围绕 C++和 Borland C++两条线索，从实用性和可操作性两个角度出发，在体例上共分四个部分（共计十七章）和一个附录。章节安排及主要内容如下：

第一部分介绍 Borland C++集成开发环境。第一章从程序设计语言发展史的角度探讨 C++与 Borland C++的问世，对于软件工程学的划时代意义。第二、三章具体细致地介绍 Borland C++集成开发环境的多项内容，第四章用一个 C++程序实例说明 C++源代码如何编辑、编译、连接和运行。

第二部分阐述 C++与 Borland C++程序设计的实用技术。第五章为读者提供一个低级台阶，着重讨论 C++的前身 C 语言的基本要素，C++相对于 C 语言所作的独立于类的功能扩充（包括强类型、内联函数、动态分配内存、重载函数等），以及 C++预处理程序。第六章全面阐述 C++ 程序设计的基本要素——类、构造函数与析构函数、数据成员与成员函数、单一继承与多重继承、虚函数和多态性，以及 I/O 流。本章是全书的核心内容之一。第七章介绍 C++ 高级程序设计的一些要求，内容涉及 Borland C++存储管理、多语言混合编程问题、屏幕及图形功能、浮点库技术等。

第三部分介绍 Borland C++ 实用程序。从操作入手，分别讨论了交互式编辑器（第八章），命令行实用程序（第九章），用户定做程序（第十章）以及其它常用的实用程序（第十一章）。这一部分内容与 Borland C++ 的集成开发环境起着相同的作用，但利用这些实用程序，

可以提高运行效率。

第四部分着重讨论 Borland C++ 的高级特征，结合应用项目的开发实例深化 C++ 概念和思想。高级特征包括：VROOMM 技术(第十二章)，内部汇编语言(第十三章)，流控制的特殊格式（第十四章）等。第十五章介绍 Windows 应用程序开发，介绍 Borland C++ 可以与 Windows 作兼容性开发的新特征。第十六章 C++ 应用程序分析是全书的第二个核心内容，包括：任务类的建立及程序实现，运用任务类完成多任务处理功能的两个实例分析。第十七章提供了 Borland C++的两个有用的资料。

本书作为 C++ 程序设计和 Borland C++ 的基础读物，主要面向广大试图掌握一种算法语言的普通读者。甚至很少接触程序设计算法语言的读者，可以通过第五章的学习快速进入C 语言，而具备C 语言或 Pascal 语言程序设计基础的读者则可以跳过第五章的部分内容直接阅读第六章以后的章节。但对于不太熟悉集成开发环境的读者有必要阅读第二、三、四章，并结合上机操作体会其中的技巧。全书第三部分和第四部分则是专为高级的 C 语言及 C++ 语言程序员安排的。第十六章通过 C++应用实例的分析，帮助读者深化 C++ 程序设计的基本概念和开发技巧。

本书由邓集锋(第一、十二、十三、十四、十六章、十七章第 1 节及附录)、寇应展(第六、八、九、十、十一章）、赵粤生(第五章)、王建新(第二、三、四、十五章)四位同志分工合作完成。第七章和第十七章第 2 节由邓集锋、寇应展、赵粤生、王建新等共同完成。全书由清华大学计算机系唐瑞春博士审校。

在本书的编写过程中，曾得到北京理工大学管理学院甘仞初院长的热情鼓励，电子工业出版社杜振民主任、发行部余刚主任和北京图书馆黄高年老师多次关心本书的编写工作，管理学院许多朋友大力协助与配合（其中田林哲与付亮两位同志参与了图形汉字处理程序的设计工作），在此谨表谢忱。

由于编者水平有限，加之时间仓促，书中难免存在一些缺点乃至错误，恳请读者批评指正。

编　者

1991 年 6 月于北京

目　录

第一部分　集成开发环境

第一章　C++ 概述

Borland Intl. 公司开发了一系列 Turbo 编译器——从 Turbo Pascal 到 Turbo C，进而到 Turbo C++及 Borland C++。1990 年 5 月面市的 Turbo C++1.0 版是其中颇具革命意义的成员；Borland C++2.0 版则是前者的更新版本，于 1991 年 2 月问世。不难理解，这二者与以前的系列产品又是一脉相承的。

Turbo C++ 与 Borland C++更像 Turbo C 的支撑版本。它们既支持 AT&T C++2.0 版，又支持实时程序设计语言 C 的面向对象版本。上述两点恰恰体现了 Turbo C++的革命性。为了保证与 Windows 3.0 兼容，BorlandC++ 2.0 版在 Turbo C++ 的基础上提供了强有力的资源开发工具 Whitewater。

考察一种新的算法语言，都要提出这样的问题：为什么要开发这种语言？它比现有语言有何优越之处？我们在分析 C++时，同样要问：C++究竟是什么？它在哪些方面超越了 C 语言？

本章从程序设计发展史的角度引出面向对象程序设计(Object Oriented Programming，简称 OOP) 的思想，着重介绍 Borland Intl. 公司在这方面的成功探索，指出 OOP 思想从理论到实践的成熟将导致软件工程学的一场革命。

1.1　概念及发展演变

在实际深入剖析 C++之前，有必要考察 C++与其前身——C 语言及其它程序设计语言的关系。本节旨在介绍 C++的起源、用途以及开创程序设计新思维的重要意义。

众所周知，C++思想的提出是基于 70 年代发明的 C 语言。Matin Richards 最早开发出 BCPL 语言，接着 Ken Thompson 在继承 BCPL 的许多优点的基础上发明了比较实用的 B 语言。B 语言直接促进了 C 语言的诞生，这一任务由 Dennis Ritchie 完成，并在 DEC PDP-11 计算机配备的 UNIX 操作系统中最终实现。C 语言凭借其灵活性和高效性，自 80 年代初以来在程序设计界占领了广泛的市场，程序员运用它成功地开发了许多重要的软件产品。

但是，C 语言并不是万能的。随着软件工程规模的扩大，C 语言的缺陷逐渐显露出来。比如，当程序量超过 50 000 行、开发人员达数十个时，系统维护工作量变得相当大，而且系统的整体性难以保证。正因为如此，1980 年，美国 AT&T 贝尔实验室的研究员 Bjarne Stroustrup 为 C 语言扩充了一系列新的功能，他命名为“C with class”。1982 年，正式定名为“C++”。

Bjarne Stroustrup 对 C 语言的功能扩充是革命性的，这是因为 C++ 支持面向对象的程序设计。由于借鉴了面向对象语言 Simula 67 面向对象的性质，因此，C++实质上是这两种强有力的程序设计语言的有机融合。

程序设计从其发展历史来看，大致经历三个阶段，每个阶段对应一种典型的程序设计范例。

(1) 无序态程序设计范例

早期的程序设计范例称作无序态范例。在这种模式中，程序员的所有注意力直接集中在问题求解本身，很少考虑解决问题的方法。其代表性语言是 BASIC 的低级版本，它甚至没有子例程结构。程序员因考虑结构性或可扩展性等问题而手忙脚乱，不得不把精力放在特殊问题上。因此，在无序态范例中只能编写小型程序(几百行以内)。在这一范例中，由于程序大小不同，代码间参数传递的关系变得复杂，无疑将增加调试与测试的难度。

(2) 过程化程序设计范例

无序态范例经过改进以后发展为过程化程序设计范例，主要以 FORTRAN 和 COBOL 这类语言为代表。在这种范例中，程序员可把一个问题分成许多个函数。理论上每个函数可看作一条语句。函数本身被抽象化了。于是，在源代码和数据之间形成一道屏障。所有注意力集中于代码抽象，而非数据抽象。

(3) 结构化程序设计范例

在这个阶段，代码抽象超出函数的范围。结构化程序设计范例不同于无序态范例的函数控制，而类似于过程化程序设计，按照规则把每种控制结构(如 for ，while ，if 等)当作子功能，以便作深入的数据抽象。例如，可以把一个 for 循环看作独立的程序块，因为全部通道口都以同一种方式进出，不允许跳进循环体去执行一些指令然后又跳出来。因此，支持结构化程序设计范例的算法语言必须像 PASCAL 语言所规定的那样，有一套完整的控制结构。其次，这一范例不再忽视数据部分。函数之间的参数传递有一套规则，且只允许存取以输入参数形式接收的数据。此外，不提倡用全局变量作为通信参数来传递。这些限制反映出过程化范例的弱点，因为如果函数可存取或修改全局变量，就难以抽象成一个实体。

大多数结构化语言鼓励程序员定义新的数据类型以便更加准确地描述客观对象。C 语言用 enum 或 typedef 等语句定义简单的数据类型，而复合类型需要某种结构(structure)来定义。事实上，用“结构”来定义单个实体，可以保证数据的一致性，更加接近模型化之后的客观对象。

模块化程序设计是结构化程序设计风格的延伸，此时函数作为模块被分解成更加独立的可编译源文件。每一模块包含一组唯一被自己存取的数据与函数。一组独立的数据与函数可被所有模块存取。所有相似的函数归在一个模块之中，所有定义的函数并不都对模块外部透明。限制全局定义函数的个数可以减少模块与模块间通信参数的个数，从而降低整个系统的复杂性。C 语言正是使用模块化程序设计范例的典范。首先，单个函数使用的数据相对于其它模块是不可见的，从而实现数据隐藏或封装。其次，用户自定义数据类型与原类型不相同，为了支持数据抽象，必须允许程序员为新类型定义内部操作符。但是定义新数据类型及其相关的全部操作符是一个复杂的过程，加之根据基本内部类型定义新类型掩盖了用户定义类型之间的关系，不及从现有用户定义类型中派生出新类型那么简单明了。由于派生类的出现，继承和多态性等概念相伴产生了。这已不是 C 语言所能完成的任务了。面向对象的语言 C++ 具备上述抽象、封装、继承和多态性等特征，从而取代了 C 语言，在程序设计的发展史上树起一块崭新的里程碑。

程序设计范例的演变过程及其具有代表性的计算机算法语言如表 1-1 所示。

表 1-1 程序设计范例的发展及计算机算法语言

结构化程序设计范例	C++ Smalltalk Simula 67 ——具有面向对象的性质 Ada Object C Pascal COBOL FORTRAN BASIC 的高级版本(如 TURE BASIC)
过程化程序设计范例	C FORTH Micro-assembler
无序态程序设计范例	BASIC 的低级版本 Assembler

那么,从用途方面讲,C++ 有哪些突出之处呢?

前面已经指出,发明 C++ 的基本目的是帮助维护大型程序(指超过 50 000 行、开发人员达数十个的大型项目)。不过,随着 C++ 本身的发展、完善和扩充,它的这种面向对象的性质可被有效地应用到任何编程任务中。越来越多的程序员用它开发编辑器、数据库、个人文件系统以及通信与接口程序等。由于保持了的高效率,用它来开发高性能、高质量的系统软件也为时不远。

C++ 之所以能完成如此多方面的编程任务,是由于:一是可方便地构造一个由相关对象组成的层次等级树;二是程序易于维护、移植和扩充。

1.2 面向对象的程序设计

C++ 紧密结合了面向对象程序设计(OOP)的思想,在 C 语言的基础上增加了面向对象的特点。当然,C++ 并非这种结合的唯一方式。例如,目标 C(Object C)语言也具有面向对象的特点,不过它属于与前者不同的结合方式:这二者在原理上比较接近,实现方法却不相同。所以,在理解 C++ 概念之前,必须先掌握 OOP 的基本思想。

人们对于 OOP 概念的理解,大多数集中在"OOP 是如何如何优秀"之类的问题上。也就是说,近几年来在程序设计理论方面的探索中,"面向对象"的思想是最有价值、最热门的话题。从程序设计的角度看,面向对象代表一种通过摹仿人类建立现实世界模型的方法(包括概括、分类、抽象和归纳等)进行软件开发的思想体系。对这一概念的理解,Pinson 与 Wiener 提出:面向对象的计算机语言必须具有四种特殊的对象化属性:"抽象,封装,继承和多态性"(见 Pinson 与 Wiener 著,《An Introduction to Object—Oriented Programming and Smalltalk》,1988)。这一定义叙述简单明了,不过还需要作些解释。

(1) 抽象和封装——对象

对象(Object)是 OOP 最重要的性质之一。从概念上讲,对象就是既含数据又含对数据代码的操作的一个逻辑实体。在对象中,有些代码或数据是为该对象所特有的,亦即不能为对象之外的任何东西直接存取。这样,可以有效地防止程序中其它不相关部分无意修改或不正确使用该对象的私有部分。

实际上,对象是类(Class)的实例。类用于描述对象的群体特性。另一方面,类是进行抽象

和封装的基石。

抽象(Abstraction)是指具体事物一般化的过程。对具有特定属性及行为特征的对象进行概括，从中提炼出这一类对象的共性，并从通用性的角度描述共有的属性及行为特征。抽象包括两方面的内容：一是数据抽象，即描述某类对象的公共属性；一是代码抽象，即描述某类对象共有的行为特征。正是通过类，方便地实现了代码抽象和数据抽象，这种结合的方式就是所谓的封装(Encapsulation)机制。抽象和封装可以提高软件的模块化程度，增强代码的重用性。

需要明确的是：对象只是一个用户定义类型的变量；定义一个对象意味着建立了一个新的用户定义数据类型。

(2) 派生和继承

派生(Deriving)是指由基本类导出子类的过程，这个基本类的子类称作派生类。派生是分层次等级进行的。继承(Inheritance)是指一对象获取另一对象之性质的过程。它与按层次分类的思想是面向对象的主要性质。在使用"类"之前，必须针对每个对象定义其所有性质；由于使用了"类"，就只需定义使对象区别于"类"中其它对象的性质。由于派生类的存在，那些与基本类共享的性质能方便地继承下来，这种继承机制使得对象成为基本类的实例。

(3) 多态性

多态性是面向对象的另一个突出性质。其含义是同一个函数名可用于多个相互之间既有差别又相关联的目的。使用多态性，就是为了让函数名变为说明某种行为的通用类。对于不同的处理数据，相应地执行通用类的某一具体实例。例如，对三种不同类型的数据(如 int，char，float)执行进栈、出栈操作。利用多态性，可建立三组进栈 push( )和出栈 pop( )函数，由编译程序根据不同的参数选择相应的函数。若不用多态性，就必须建立三组不同名称的进栈、出栈函数，比如：

```
pushint( );      popint( );
pushchar( );     popchar( );
pushfloat( );    popfloat( );
```

显然，这样做增加了开销。

面向对象的程序设计是一种崭新的程序设计范例，可将其视为一类方法或一种风格。对程序设计而言，重要的是要抓住其核心部分，为此人们采用一系列方法。在程序设计方法学中，比较而言，"面向对象"是一门新课题，富有挑战性，因而也最有发展潜力。

程序设计范例是人们用于解决程序设计问题的思想方法的总和。程序员从中选取最为优秀的方法编写大型程序的代码。一个程序员的观点只代表一种解决问题的方式，程序员有时需要分析自己选用的范例，根据多年编程实践所积累的经验来判断这些范例是否最理想。C++为程序员考察这些面向对象的程序设计范例提供了有益的启示。

面向对象语言必须支持面向对象的程序设计范例。当然，这里所说的"支持"是一个相对概念。人们曾经讨论过"汇编语言支持结构化程序设计，乃是因为结构化代码又可以写成汇编语言"之类的观点。然而，一般说来，汇编语言并不是一种结构化语言。

C++之所以是一种面向对象的语言，是因为它支持面向对象的程序设计范例。作为 C 语言的超集，C++也支持某些早期的范例。因此，C++属于综合性的算法语言。其它许多语言也有这种性质，如 Apple 公司的 Object Pascal，Borland 公司的 Turbo Pascal 5.5，Neon 公司的 Forth Objects 以及 Logo 语言的变种 Objective Logo。事实上，绝大多数程序设计语言存在或正在使用面向对象的版本(参见 Peter Wenger 著文，《Learning the Language》，1989)。

1.3 从C到C++

C++是C的超集。C++保存了C的所有组成部分，这样，C编译的程序可以在C++环境下运行，基于K&R C标准与ANSI C标准的程序在C++环境下产生同样的出错信息和警告信息。经Turbo 2.0编译未出错的程序由Turbo C++或Borland C++编译时将正常通过。

为实现以上目的，Turbo C++及Borland C++根据扩展名区分C与C++源程序：当IDE(集成开发环境)中Options|Compiler菜单的C++ Always选项无效时，程序源文件扩展名为.C；C++的则是.CPP。

C++与ANSI C共享一套通用的规则。ANSI C的大部分扩展内容来源于C，如函数说明格式和强类型使用。C++则扩展到包容ANSI C的特征以保证最大限度的兼容性。二者的共同点是主要的，差别是次要的，列述如下。

1.3.1 函数原型

原K&R C定义通常根据变量大小说明一个变量，根据返回类型说明一个函数。一旦函数说明出错，它将被指派成缺省类型int(即整型)。这是因为：返回整型值适合于CPU寄存器；而舍弃寄存器左侧值不会引起处理器错误。

C++与ANSI C都强调高级的类型检查。程序员因而可以避免混淆没有作兼容分配的存储类型(C中隐含类型转换，同一表达式中可使用字符型、整型与浮点型)，使用起来更加方便。

C++允许在定义函数之前，先说明其类型，这就是所谓的"函数原型"。K&R C忽视了这一点，ANSI C和C++则特别强调原型的使用。其格式是，全部参数名和类型出现在同一行上。例如：

```
unsigned func(signed a,int b);
```

若不想说明函数名和函数类型，可用省略号。例如：

```
void f1(int a,...);
void f2(...);
```

f1( )表示函数使用一个整形参数和其它任意类型参数；f2()表示函数接受任意类型的参数。

1.3.2 关键字void

C++沿用ANSI C的关键字void。最初，函数缺省返回类型为整型，如函数说明fn()与int fn( )是等价的。

void的原意指没有返回值的函数。如void fn( )(缺省类型仍为int)。编译器据此确定函数是否正确调用了。void的扩展含义指无参数的函数。例如，在ANSI C中，int fn(void)表示无参数的函数；int fn()则等价于int fn(...)，表示接受变长变元参数的函数。在C++中，int fn()等价于int fn(void)，不过后者增强了程序的兼容性。

指针变量可定义为pointer to void (空指针)。它不能作递增、递减或重指运算，但可与其它任意类型作兼容性分配。这是一个"万能"指针，用于程序没有确定指针所指的内容时存储地址。

此外，void可用作转换符，表示运算结果被有意舍弃。例如：

```
int fn(char x);
fn('z');
```

若忽略函数的返回值，应将第二行改写成：