21世纪大学计算机规划教材·工程应用型

C++程序设计

梁兴柱　王　婧　主编　　龚　丹　吕志峰　副主编

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY　http://www.phei.com.cn

21世纪大学计算机规划教材·工程应用型

C++程序设计

梁兴柱　王　婧　主编

龚　丹　吕志峰　副主编

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书为高等院校计算机及相关专业“高级语言程序设计”课程编写，全书共分三部分：第一部分是 C++程序设计基础，首先概述 C++语言的历史、特点和程序设计思想的发展，此后为数据类型与表达式、流程控制、数组、指针、引用、函数等内容；第二部分是 C++面向对象程序设计，逐一讲解 C++语言中抽象、封装、继承与派生和多态等机制；第三部分是 C++程序设计高级特性，包括输入/输出流、模板、异常处理及综合实例分析。

本书结构清晰，通俗易懂，注重应用，既适合作为计算机、电子信息等相关专业的本科或高职高专教材，也是具备一定开发经验的编程人员学习面向对象程序设计思想的参考书。

本书配套教学资源包括实例代码及各章习题参考答案等，可免责提供给任课老师。

图书在版编目 (CIP) 数据

C++程序设计 / 梁兴柱，王婧主编. —北京：电子工业出版社，2009.1
21 世纪大学计算机规划教材 • 工程应用型
ISBN 978-7-121-08030-2

Ⅰ. C… Ⅱ. ①梁…②王… Ⅲ. C 语言－程序设计－高等学校－教材 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2008）第 206065 号

策划编辑：童占梅
责任编辑：王 纲
印 刷：北京市天竺颖华印刷厂
装 订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编：100036
开 本：787×1092 1/16 印张：19.5 字数：499.2 千字
印 次：2009 年 1 月第 1 次印刷
印 数：4000 册 定价：29.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010)88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：(010)88258888。

前　言

计算机程序设计语言发展至今，C++仍然是应用最广泛的语言，这不仅仅因为 C++是由 C 语言发展而来，具有较长的历史背景和颇具规模的应用人群，更重要的是 C++语言是当前众多高级语言中抽象性最好、对面向对象程序设计思想支持最彻底的语言，而这一特点正是应对不断增加的软件工程的复杂度时所必须具备的。此外，C++语言完全兼容 C 语言，在各种硬件平台上可移植性好，并且作为一种高级语言，比汇编语言更易于掌握。因此，C++程序设计课程不仅是计算机专业的核心基础课程，而且已经作为一门公共基础课程在工科各专业中得到普及。

对于学习 C++语言是不是必须要先学习 C 语言的讨论，至今没有人能给出确切的答案，而事实上，完全不需要用这样的讨论来为初学者增加困扰，因为作为一个完整的程序设计语言体系，它所能解决的问题可简可繁，它所具备的特性有一般的、初级的，也有特殊的、高级的。本书在编排上就是按照这一规律，将 C++语言中最基本的语法知识集中在第一部分阐述，有人可能会认为它是对 C 子集的一个介绍，但是需要注意的是，这种理解有偏颇，因为诸如引用、内联等特性是 C 语言所不具备的。C++语言最引以为傲的部分，即抽象、封装、继承和多态等机制，在第二部分中为读者一一展开，从中可以逐步体会 C++语言面向对象程序设计思想是如何提高软件设计与开发效率的。最后一部分是 C++语言体系中解决工程问题不可或缺的一些高级特性，并有综合开发实例强化学习效果。

C++作为一门程序设计语言，其学习的最终目的是应用，本书编写的主旨亦为尽可能使学习者最快地理解 C++语言的精髓，掌握其技术，所以除必要的思想陈述和概念解释外，在行文中避免使用高深莫测的语句，而是使用设问及解答的方式，既引出相关知识和技术又表明其意义，缩短了课堂学习与编程实践的距离。因此，本书尤其适合于侧重应用能力培养的高等院校及应用型本科院校，也是当今全国范围内进行课程改革的大形势下，转变学生为学而学、学而不能用的尴尬局面的一次实践。

本书第 1，2，3 章由王婧编写，第 4，5，6 章由龚丹编写，第 7，8，9，10 章由梁兴柱编写，第 11，12，13，14 章由吕志峰、陈艳共同编写，刘振宇、张振蕊、邓琨也参与了部分编写工作。特别感谢高洪志和王建一老师审阅了本书，并提出了许多宝贵建议。

本书所有实例均使用 Visual Studio C++ 6.0 集成开发环境测试通过，配套的实例代码及各章习题参考答案等教学资源可从华信教育资源网（http://www.hxedu.com.cn）上免费下载。

编著者

于哈尔滨工业大学华德应用技术学院

目　录

第一部分　C++程序设计基础

第二部分 C++面向对象程序设计

第三部分 C++程序设计高级特性

第一部分 C++程序设计基础

第1章 概 述

C++语言是面向对象程序设计语言，学习和掌握 C++语言将会使读者在今后的软件开发中受益匪浅。本章首先向读者介绍 C++语言的发展过程及它与 C 语言的关系，然后逐一的向读者介绍C++语言的特点、开发过程及简单的C++程序，最后对 Visual C++ 6.0 的基本用法做了简单介绍。本章的学习使读者能掌握C++语言的特点及开发过程，并能设计简单的C++小程序。

1.1 C++语言的历史

1.1.1 C++语言的诞生

C++语言是从C语言发展而来的，而C语言的历史可以追溯到1969年。在1969年，美国贝尔实验室的 Ken Thompson 为 DEC PDP—7 计算机设计了一个操作系统，这就是最早的UNIX。接着，他又根据剑桥大学的 Martin Richards 设计的 BCPL 语言为 UNIX 设计了一种便于编写系统软件的语言，命名为 B。作为系统软件编程语言的第一个应用，Ken Thompson 使用 B 语言重写了其自身的解释程序。1972—1973 年间，同在贝尔实验室的 Denis Ritchie 改造了 B 语言，为其添加了数据类型的概念，并将原来的解释程序改写为可以在直接生成机器代码的编译程序，然后将其命名为 C。1973 年，Ken Thompson 小组在 PDP—11 上用 C 语言重新改写了 UNIX 的内核。与此同时，C 语言的编译程序也被移植到 IBM 360/370，Honeywell 11 及 VAX—11/780 等多种计算机上，迅速成为应用最广泛的系统程序设计语言。

然而，C 语言也存在一些缺陷，如类型检查机制相对较弱，缺少支持代码重用的机制等，造成用 C 语言开发大型程序比较困难。 为了克服 C 语言存在的缺点，贝尔实验室的 Bjarne Stroustrup 博士及其同事开始对 C 语言进行改进和扩充，将“类”的概念引入了 C 语言，构成了最早的 C++语言（1983）。后来，Stroustrup 和他的同事们又为 C++语言引进了运算符重载、引用、虚函数等许多特性，并使之更加精炼，于 1989 后推出了 AT&T C++ 2.0 版。随后美国国家标准化协会 ANSI（American National Standard Instiute）和国际标准化组织 ISO（International Standards Organization）一起进行了标准化工作，并于 1998 年正式发布了 C++语言的国际标准 ISO/IEC: 98—14882。各软件商推出的 C++语言编译器都支持该标准，并有不同程度的拓展。C++语言支持面向对象的程序设计方法，特别适合于中型和大型的软件开发项目，从开发时间、费用到软件的重用性、可扩充性、可维护性和可靠性等方面，C++语言均具有很大的优越性。同时，C++语言又是 C 语言的一个超集，这就使得许多 C 语言代码不经修改就可被 C++语言编译通过。

1.1.2 C++语言的发展

语言的核心特征是逐步完善起来的，这也许是C++语言不同于其他语言的独特之处。

（1）在“C with Class”阶段，研制者在C语言的基础上加进去的特征主要有：类及派生类、公有和私有成员的区分、类的构造函数和析构函数、友元、内联函数、赋值运算符的重载等。

（2）1985年公布的C++语言1.0版的内容中又增加了一些重要特征：虚函数的概念、函数和运算符的重载、引用、常量（const）等。

（3）1989年推出的2.0版形成了更加完善的支持面向对象程序设计的C++语言，新增加的内容包括：类的保护成员、多重继承、对象的初始化与赋值的递归机制、抽象类、静态成员函数、const成员函数等。

（4）1993年的C++语言3.0版本是C++语言的进一步完善，其中最重要的新特征是模板（template），此外解决了多重继承产生的二义性问题和相应的构造函数与析构函数的处理等。

（5）1998年，C++标准（ISO / IEC 14882 Standard for the C++ Programming Language）得到了国际标准化组织（ISO）和美国国家标准化协会的批准，标准C++语言及其标准库更体现了C++语言设计的初衷。名字空间的概念、标准模板库（STL）中增加的标准容器类、通用算法类和字符串类型等使得C++语言更为实用。

C++语言开发的宗旨是使面向对象程序设计技术和数据抽象成为软件开发者的一种真正的实用技术，所以C++语言的形成是一个发展和完善的过程，其研制和开发过程是以编译系统的有效实现为前提的，这或许正是C++语言能够成功的原因。

1.2 C++语言的特点

在程序设计语言的历史上，在所有比较成功的高级语言中，C++语言有着不少与众不同的地方，它的这些特点既是人们愿意选择它的原因，同时也是学好C++语言必须搞清楚的一些问题。

（1）C++语言是支持面向对象程序设计的最主要的代表语言之一。如前所述，C++语言包括了几乎所有的支持面向对象程序设计（OOP）的语法特征，基本上反映了20世纪80年代到90年代以来的所有程序设计和软件开发的新思想和新技术。其中包括：

① 封装和信息隐藏。把数据和对数据的操作一起封装到类和对象之中。

② 抽象数据类型。一种新的类的定义就是一种新的抽象数据类型，它可以用在不同的程序系统之中。

③ 以继承和派生的方式实现程序的重用机制，为程序的重用找到了一种可靠而方便的方式。

④ 通过函数与运算符的重载和派生类中虚函数的多重定义，实现多种情形下的多态特征，明显提高了程序的抽象水平。

⑤ 通过模板等特征实现了类型和函数定义的参数化，把抽象度又提升了一级。

C++语言这些典型的OOP特征，在OOP方法已被软件开发领域接受为新一代开发技术的今天，不仅已为许多软件开发者所接受，而且也为计算机的专家所接受。C++语言的3.0版本公布之后，一些计算机领域的教科书，对于其中的算法与数据结构的描述，已从过去大部分采用Pascal或类Pascal语言转变为采用C++语言，因为用类和模板描述算法和数据结构具有通用性，更加简洁合理。

（2）C++语言是程序员和软件开发者在实践中创造的，无论是语言的各个特征还是整个研制过程，都体现了面向实用、面向软件开发者的思想。

（3）C++语言是C语言的超集。从C++语言1.0版本问世以来，就确定了“与C语言100%兼容”的宗旨，虽然这样做也为C++语言带来了一些问题，但利大于弊，这也是C++语言成功的关键。

C++语言与C语言相比具有以下特点：

① 从C语言中继承了其独有的为程序员所喜爱的简明高效的表达式形式。

② 比较容易地解决了目标代码高质量、高效率的问题。

③ 吸引了20世纪80年代成长起来的一批高水平的C语言程序员，他们比较自然地转向C++语言。

④ 可以与20世纪80年代以来的大批C语言程序软件兼容，可以使它们在C++语言环境下继续维护使用。

为了有利于C++语言的发展，C++语言的设计者和开发商还注意解决与C语言相关的问题。一方面，在相当长的时间内，C语言与C++语言及其编译系统同时发售，促进了C语言程序员向C++语言的转化过程。另一方面，对于C语言的语法成分，做了许多有成效的取代工作，例如：

- 引入const常量和内联函数概念，取代宏（#define）定义。
- 引入reference引用概念，部分取代过于灵活而影响安全性的指针。
- 引入动态内存分配运算符（new），取代比较低级的有关库函数。
- 引入用于数据输入/输出（I/O）的流类，取代可读性差的C语言I/O库函数等。

C++语言采用全部包容C语言的策略，也带来了一些问题，例如：

- 同时支持两种程序框架——SP框架和OOP框架。
- 容易引起误解。C++语言是C语言的超集，是C语言的改善。实际上在设计方法、思想上和风格上，二者有着本质的差别。

要想今后深入学习C++语言，提高程序的安全性和健壮性，必须把这些问题搞清。

1.3 过程化程序设计

1.3.1 基于过程的程序设计

从程序设计方法的角度来说，程序的概念是组织成一定形式的操作序列。同一问题若组织成不同的操作序列，则反映了程序设计方法的不同。

编程问题是从一大类科学计算的问题开始的，这些科学计算问题所描述的数据大多不是很复杂。因而，依赖于早些时候的C语言中的内置数据类型，以及在空间上的简单复合就可以完成，其相关的数据操作也多半是数据复制、数据赋值等简单操作。所以程序设计主要体现在算法上，编程就是解决算法如何设计的问题。当算法很大时，就考虑将它按功能划分。程序组织围绕算法的切分而展开。这一类问题一般都是小规模的问题，一般的程序设计语言也都可以胜任，如图1-1所示。

图1-1中的过程结构是按照问题要求所编织的解或算法。它所使用的是语言中现成的基本数据类型。图1-1中反映了这样一个事实：问题模型反映为过程结构模型，实际上就是功

能模型。模型可以做得很精巧，但由于过程模块与数据的复杂关系没有清晰地分离出来，所以它一般都是“具体问题具体解决”，无法重复使用其中的“零部件”，而且问题变得庞大以后，其复杂性会无法收场。

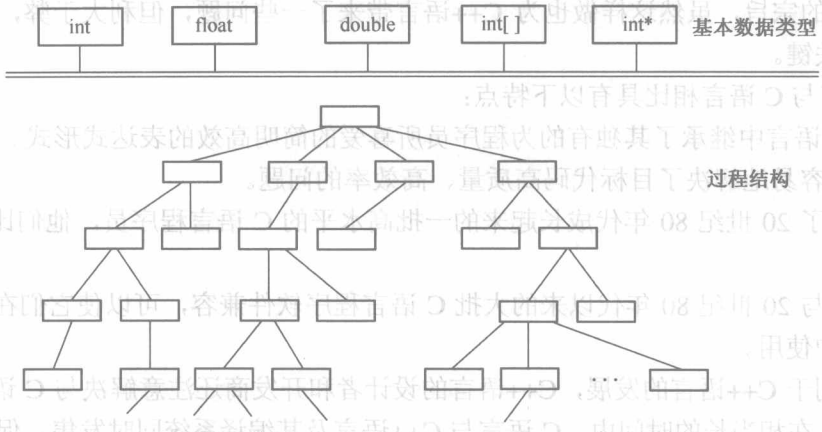

图 1-1 过程化编程结构框架

【例 1-1】有一些日期数据放在数据文件 day.txt 中，这些日期的年、月、日数值加起来若等于 15，则输出，然后按日期从小到大的顺序打印出来。

如文件 day.txt 中有：

03-06-18　　03-08-04　　05-01-01　　06-05-07　　08-01-06

则输出：

03 年 08 月 04 日

08 年 01 月 06 日

简单地看，可以把求解过程划分成三个部分：

第一部分是读入数据，将读入的数据放在一个年月日的复合数据结构的数组中；

第二部分是处理日期数据，数据处理完后，放到另一个同类数组中；

第三部分是输出处理，将数组中的数据按要求输出。

由于处理日期数据的算法“比较复杂”，所以把它划分为两个子部分，第一子部分是取年月日的数值，加起来判断其和等于 15 否。这部分工作要经历一个循环，对数组中的每个日期进行操作、比较，如果条件满足，就将其放入准备好的另一个数组中。

第二子部分是对结果数组进行排序操作。它要经历至少两重循环，通过频繁地比较大小和交换操作，达到数据按从小到大存放在该数组的目的。其中，大小比较是日期的大小比较，先比较年，再比较月，最后比较日。

将复杂的过程简单地按功能分层从而达到解决问题的目的，这种思想就是过程化程序设计的思想。过程化程序设计以一系列过程的划分和组织来观察、分析和解决问题。

1.3.2 结构化程序设计

学习程序设计方法的根本是要解决如何组织程序的问题，也即解决算法与数据的关系问题。当人们有了商业处理的需要时，觉得 FORTRAN 语言有点力不从心了，于是就发明了 COBOL 语言。商业处理就是大量数据的处理工作，人们不但从语言设计上来适应应用的需要，还从编程方法上来提高解决问题的能力。

N. Wirth 提出的结构化程序设计思想对程序控制结构做了本质的描述。他指出，程序控制结构有三重内容，如图 1-2 所示。

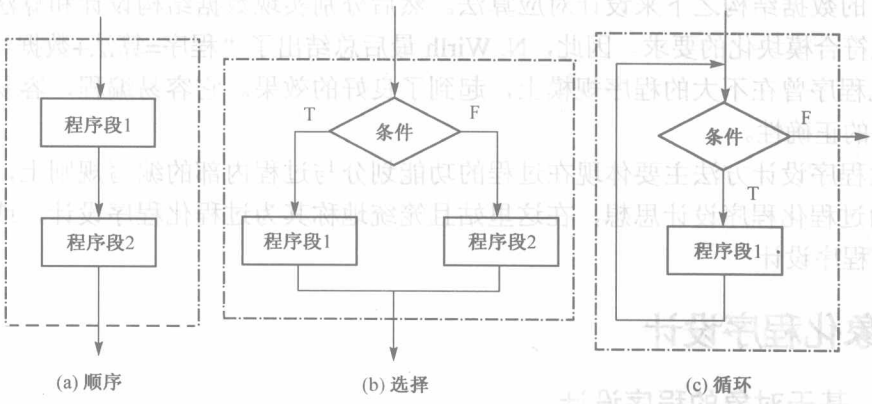

图 1-2　三种程序控制结构

1. 描述任何实体的操作序列只需用三种基本控制结构

描述任何实体的操作序列只需用顺序、选择、循环这三种基本控制结构，而且这三种基本结构对描述任何问题都是足够的。

可以将每个虚框看做是一个过程体，而每个过程体只有一个入口，一个出口。为避免过程体出现多个入口或出口，只要在现有的程序设计语言中避免使用 goto 语句，就可办到。

在图 1-2 中，图 1-2（a）先执行程序段 1，再执行程序段 2，同理，执行程序段 3，……（若还有其他程序段的话）。依次执行若干个程序段便构成了顺序结构。图 1-2（b）根据条件的真假，选择执行程序段 1 或程序段 2，执行完后便到达了本过程的结束点。该图构成了程序段的选择结构，选择结构说明了程序并非要一直无条件地执行程序段，它可以在中途依条件而改变执行路径。图 1-2（c）可以根据条件的真假，规定程序段是否执行，这个事件可以反复发生，甚至永远。程序能够代替人的大量重复计算劳动，都是源自于程序的这种能力。它构成了程序段的循环结构。

2. 程序设计中的各个过程体和组成部分应以模块表示

每个模块，其内聚性越强，外联性越少，则模块独立性越好。内聚性，即模块内部所涉及的功能越单一越好。这样一旦修改起来，就职责明确了，不会因为这个原因或那个原因都来找这个模块算账。

外联性，即模块之间的联系越少越好，联系意味着依赖，外联性少，模块的独立性就好，独立性意味着可以独自地修改本模块而与外界无关。因此就容易编程和修改。说一个小孩独立性差，就是说他（她）不会自己拿主意，受人影响太大，也是同一个道理。减少外联性还涉及对数据的分离与归类。将过程体中的数据分离出来，独立地用数据结构去描述其数据和处理，这都是模块划分的原则。

3. 过程化的程序设计方法

程序设计采用从上到下，逐步细分的方法展开，即过程化的程序设计方法。在细化的过程中，应充分运用前面的过程体控制结构和模块划分原则。当然，理论归理论，在

实际运用的时候，还要考虑到规模适中，不能为了模块化而将过程划分成只有一两条语句的模块。

在一定的数据结构之下来设计对应算法。然后分别实现数据结构设计和算法设计，这种方法总能符合模块化的要求。因此，N. Wirlh 最后总结出了“程序=算法+数据结构”的形式。结构化程序曾在不大的程序规模上，起到了良好的效果。它容易编程，容易维护，容易验证程序的正确性。

结构化程序设计方法主要体现在过程的功能划分与过程内部的编写规则上，因此它是一种规范的过程化程序设计思想，在这里姑且笼统地称其为过程化程序设计，或者称其为基于过程的程序设计。

1.4 对象化程序设计

1.4.1 基于对象的程序设计

伴随着人类对计算机的依赖性日益增强，程序规模不断扩大，模块数呈指数级递增，模块间的数据传递五花八门，同一程序中模块之间的关系错综复杂，结构化程序设计的规范已经不能保证程序的正确性、可维护性和重用性了。人们开始意识到不可能在语言中内置所有待解决问题的数据结构，必须让语言具有自建数据结构的能力。数据结构对于算法，对于程序是如此的重要，但当时大多数语言都没有专门支持对数据结构的直接描述。

在 C 语言中有一种结构类型 struct，可以在单纯空间上复合其他数据类型，描述数据的组织，从而，经验丰富的程序员可以通过捆绑相应的数据操作来达到可读、可维护的目的。但是，还是不能避免其数据操作的安全问题。在大规模程序设计中，问题尤其突出．软件发展似乎有一个不可逾越的极限，因此，在软件产业界曾一度有软件危机之说。

程序开发人员不必精通问题的每个细节，这就像使用电视机的人并非都要会安装完整的电视机。恰似电视机的外壳，把电视机的内部电路和外部使用一分为二，外部使用只需了解电视机的基本操作方法，内部电路提供电视机的各项功能，两者都需要一个共同的规范——电视机的按钮操作功能。

抽象数据类型就是想要描述这一共同的规范，它描述数据的组织和相关的操作，反映了问题的抽象模型。如果语言能够直接支持对抽象数据类型的描述，即自由定义数据类型，那么，问题就能化成以抽象数据类型为媒介的使用与实现独立的两部分，因而该语言的解决问题的能力一定就强。衡量一个语言的优劣，能否自定义，或者说扩充数据类型的能力是其重要指标。

C++有一个类（class）机制，这正是 C 语言欠缺的地方，同时也反映了语言中对获得抽象数据类型描述能力的急迫性。数据类型的本质是数据组织和其操作的捆绑性。当对应到具体编程时，用抽象数据类型来界定，就能把编程大军分为两个阵营：一个是专业性极强的、专门实现抽象数据类型的编程，好比安装电视机者；另一个是专门使用抽象数据类型的编程，好比使用电视机者。

然而，要能使抽象数据类型能够维护两大程序员阵营的编程利益，必须要在语言的设计中加入一些语言机制，这些语言机制采用了许多难以想象的技术，实现了数据封装、类型安全等，而且还必然要使代码更容易阅读和维护，否则没有人愿意用。抽象数据类型的使用，最终像使用基本数据类型那样简单，对应的实体就称为对象。因此，编程的意义就是算法在