21世纪高等教育计算机规划教材

COMPUTER

C++ 语言程序设计

C++ Programming

蒋爱军 刘红梅 王泳 吴维刚 编著

- 体系完整，内容全面
- 面向应用，实例丰富
- 循序渐进，深入浅出

人民邮电出版社
POSTS & TELECOM PRESS

21世纪高等教育计算机规划教材

COMPUTER

C++ 语言程序设计

C++ Programming

蒋爱军 刘红梅 王泳 吴维刚 编著

人民邮电出版社

北京

图书在版编目（CIP）数据

C++语言程序设计 / 蒋爱军等编著. -- 北京 : 人民邮电出版社, 2014.10
21世纪高等教育计算机规划教材
ISBN 978-7-115-33075-8

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

中国版本图书馆CIP数据核字(2014)第031392号

内 容 提 要

本书紧密结合C++语言的新标准，以C++语言为工具讲述面向对象程序设计方法。全书分为两个部分：第一部分介绍C++语言基础内容及结构化程序设计方法，包括基本类型、表达式、语句、函数、数组、指针等；第二部分介绍面向对象程序设计方法及C++语言中支持面向对象程序设计的主要机制，包括类、继承、多态、模板、命名空间、异常处理、标准库及泛型算法等。书中每章都包括丰富的代码和习题，供读者分析和练习。

本书既可作为计算机专业本科生程序设计课程的入门教材，也可以作为相关专业高年级学生面向对象程序设计的教材，还可供软件开发人员参考。

本书适合作为高等院校“C++语言程序设计”课程的教学用书，还可作为C++语言的自学或教学参考书。

◆ 编　　著　蒋爱军　刘红梅　王　泳　吴维刚
责任编辑　武恩玉
责任印制　彭志环　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
大厂聚鑫印刷有限责任公司印刷

◆ 开本：787×1092　1/16
印张：24　　2014年10月第1版
字数：669千字　　2014年10月河北第1次印刷

定价：49.80元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315

前　言

本书以 C++语言为工具，循序渐进地向读者介绍程序设计的基本方法与理念，重点介绍目前的主流程序设计方法——面向对象程序设计（Object-Oriented Programming）。

本书在体系结构的安排上，将程序设计的基本理念和 C++语言的基础知识有机地结合在一起；在选材上，充分考虑读者知识结构、能力结构的形成规律，使得知识点布局合理，内容难度、深度和广度安排恰当。

全书内容包括 14 章和 6 个附录。

在内容编排上，本书以程序设计的思想方法和程序设计语言的知识要点为线索，以 C++标准（International Standard ISO/IEC 14882）为依托，既注重内容的完整性，又注意对 C++语言介绍的精简性，对实际应用中很少使用的内容尽量不涉及，从而避免让读者过多地陷入语法细节；既注重理论知识的介绍，又强调实际的应用，力求提高读者利用面向对象程序设计方法和 C++语言解决实际问题的能力。

学习程序设计一要自己动手多编程序并上机调试，二要多阅读并评价别人编写的程序。因此，本书每一章都包含丰富的代码实例（本书中给出的程序代码均在 Microsoft Visual C++ .NET 2003 中编译通过），并在每一章都给出了一个具有应用背景的综合性编程实例，通过该实例深化应用该章的主要内容，讲解如何使用 C++语言解决具体问题，从而提高读者的编程与动手能力，为进行软件开发及学习其他相关课程打下良好基础；同时，每一章均提供具有针对性的典型习题，以帮助读者掌握该章内容。

本书注重培养读者对面向对象程序设计方法和 C++语言的实际运用能力，给出了大量的“提示”和“注意”类内容，旨在强调重要的知识点、提醒常犯的错误、引导读者深入思考。书中经常对不同程序设计方法进行比较探讨、对 C++语言特征上的优缺点进行描述，以期拓宽读者的专业视野。

本书作者从事程序设计课程教学多年，积累了一些经验，作为主要作者出版了多本译著和教材。其中，译著《C++ Primer（第 4 版）》及其习题解答在读者中反响较好，《C++程序设计实验教程》被教育部评为 2007 年度普通高等教育精品教材。本书正是在这些工作的基础上，根据作者多年的教学实践经验，并对国内外同类教材进行了深入的比较研究而形成的。

程序设计的世界非常广阔，没有一本教材能够囊括程序设计的所有相关知识，更多时候是要靠学习者的探索和发挥。但是，入门和兴趣是最重要的，本教材正是这样一本可以将读者引入 C++程序设计殿堂的书。

本书的形成得到了中山大学信息科学与技术学院相关老师的帮助，在此表示衷心的感谢！

由于作者水平所限，书中不当之处在所难免，恳请读者批评指正。

作　者

2013 年 6 月

目 录

第1章 程序设计与C++语言入门

C++语言是目前广泛应用的一门程序设计语言。本章将介绍与程序设计相关的基本概念，介绍C++程序的基本结构，讨论使用C++语言构造程序的基本方法和步骤。

1.1 程序及相关概念

1.1.1 计算机与用户（人）

电子计算机简称**计算机**（computer），是一种电子设备，也被称为“智力工具”，是一种能够接受输入、存储和处理数据并产生输出数据的设备。

遵循冯·诺依曼[①]体系结构的现代计算机由以下5个部分构成。

- 运算器——又称**算术逻辑单元**，简称**ALU**（arithmetic and logic unit），主要完成各种算术运算和逻辑运算；
- 控制器——对各部件加以控制，使得各部件能够协调工作；
- 存储器[②]——存储数据和程序；
- 输入设备——接收数据和程序；
- 输出设备——将处理结果呈现给用户。

各部分的关系如图1-1所示。

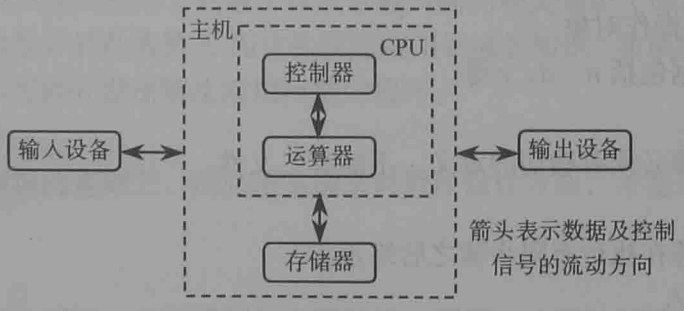

图1-1 计算机的基本组成部件

其中，运算器和控制器统称为中央处理器（central processing unit，**CPU**），而CPU和存储器通常又统称为主机。

冯·诺依曼体系结构的主要思想为：

- 计算机由五大基本部件构成（见图1-1）；

① 冯·诺依曼[Von Neumann (1903—1957)]，美籍匈牙利数学家，公认的现代计算机之父。

② 严格来说，这里的存储器指的是内存。一般而言，存储器包括内存（主存储器）和外存（辅助存储器）。

- 计算机内部采用二进制数表示指令和数据；
- 将程序（由一系列指令组成）和数据存放在计算机内部（内存）中，并让计算机自动执行。

计算机要能够工作，必须具备硬件和软件两个方面的条件。硬件就是计算机本身的构成部分，也就是上面提及的五大部件。软件就是计算机系统中的程序及相关支持文件。

计算机的使用者称为“用户”，也就是人。计算机是电子设备、是机器，无法与人进行直接交流。也就是说，如果我们需要用计算机计算“5+8”，直接对着一台计算机说“嗨，告诉我 5+8 等于多少!”是没有用的，而必须有一个**程序**（例如一个计算器程序），人通过这个程序来操纵计算机硬件，进行相关计算并获得计算结果。计算机是人们用来解决问题的通用工具，而程序则是解决某一特定问题的专用工具。因此，计算机用户实际上就是程序的用户，通过各种各样的程序来使用计算机完成工作。

1.1.2 算法

算法（**algorithm**）一词源于算术（algorism），原意指一个由已知推求未知的计算过程。后来人们把它加以推广，将解决问题的过程（方法和步骤）的描述统称为算法。

例如，判断任意给定自然数 n 是否为素数的算法为：

步骤 1：令 d = 2
步骤 2：令 r = （n 除以 d 的余数）
步骤 3：如果 r 等于 0 且 d 不等于 n，则断定 n 不是素数并终止；
　　否则，转步骤 4
步骤 4：令 d 值加 1 并转步骤 5
步骤 5：如果 d 大于 n 的平方根，则断定 n 是素数并终止；
　　否则，转步骤 2

其中，d 表示 divisor，除数；r 表示 remainder，余数。

根据上述步骤，对于任意给定的自然数，我们都可以判定它是不是一个素数。

1. 算法的特征

任意一个算法都应该具有如下基本特征。

- 以数据为主要操作对象

上述算法中的数据包括 n、d、r 等。

- 确定性

算法中的每一步骤必须有确切的定义，不能有二义性。

- 有穷性

一个算法必须能够在执行有限步骤之后终止。

- 0 个或多个输入

输入是指执行算法时从外界取得的数据。例如，上述算法的输入为 n。具有 0 个输入的算法在执行时不需要从外界获取数据。

- 一个或多个输出

算法的输出就是算法所求解问题的“解”，用以反映对输入数据进行处理之后的结果。上述算法的输出为对 n 是否为素数的判定。没有输出的算法是无意义的。

- 有效性

算法中的每一步骤都必须能够有效地执行且得到确定的结果。对上述算法而言，若 d 为 0，则步骤 2 不能有效执行（当然，事实上该算法中的 d 不可能为 0）。

2. 算法的表示

算法有各种各样的表示方法，上文中给出的素数判定算法是用自然语言描述的，但同一算法也可以用其他方式描述，常用的有流程图、N-S 图、PAD 图、伪代码、UML 活动图等。这里所说的算法表示指的是人与人之间交流算法思想而采用的表示方式。实际上，程序是对算法最精确的表示，通常称为算法的实现。

1.1.3　程序

如果要判断给定自然数是否为素数，可以用纸笔作为工具执行 1.1.2 小节中的算法得到结果，也可以用计算机作为工具完成上述计算。二者在本质上没有什么区别，但执行的效率是有区别的。一般而言，对于较为复杂的计算问题，计算机的执行效率远比人类要高。这也是人们经常选用计算机来完成某些工作的原因。

如果要以计算机为工具解决某个问题（例如判断某自然数是否为素数），则必须将解决问题的步骤（即算法）告诉计算机，即使用程序将算法表示成计算机能够理解的形式（这一过程通常称为算法的实现），然后让计算机执行程序来完成指定的任务。

程序（**program**）是由计算机执行的指令序列，用来表示程序员要求计算机执行的操作。也就是说，程序是算法在计算机系统中的表示[①]。

1.2　程序设计

1.2.1　程序设计的基本概念

程序设计（**programming**）是指编制程序的活动，也就是用计算机能够理解的形式表达算法的过程。

一般而言，进行程序设计的人员必须具备 4 个方面的知识。

- 应用领域的知识

这是构造问题解决方案的基础。例如，要解决素数判断问题，就必须了解素数的概念（即除了 1 和它本身之外没有其他约数的自然数）以及素数判断的相关知识（即不能被 2 至 $\sqrt{n}$ 之间的任意整数所整除的自然数 n 就是素数），而这些就是应用领域的知识。如果程序设计员不具备应用领域的知识，当然就不可能开发出解决应用问题的程序。

- 程序设计方法

在具有应用领域知识的基础上，还必须掌握某种程序设计方法，才能运用适当的思维方式构造出问题的解决方案。

- 程序设计语言

要使用计算机来解决问题，必须将解决方案转换为程序，才能被计算机理解和执行，因此程序设计人员需要掌握程序设计语言（如 C++语言）这一工具。

- 程序设计环境与工具

程序设计环境与工具可以提供许多可重用的基本程序让程序员在设计程序时使用。因此，开发程序（尤其是大型程序）时，通常需要利用程序设计环境和工具，以便提高程序开发的效率及程序的质量。例如，当我们用 C++语言开发程序时，可以选 Microsoft 公司的 Visual C++为工具，

① 后文会谈到，程序也是实体在计算机系统中的表示。实体用于对算法进行有效的组织。

也可以选 Boland 公司的 C++ Builder 为工具。

1.2.2 程序设计过程

程序设计过程可以分为 4 个阶段：分析阶段、设计阶段、实现阶段和测试阶段。

分析阶段的主要任务是理解问题，弄清楚要解决的问题是什么，即所开发的程序需要做什么。例如，要开发一个素数判断程序，则分析阶段的任务就是明确该程序需要做什么。如该程序可能要“判断用户从键盘输入的一个自然数是否为素数”，也有另一种可能，即该程序要“判断某个文件中存放的所有自然数是否为素数”，这就是两个有区别的需求，与此相对应的解决方案也有所不同。分析阶段就是要明确程序到底要“做什么”。

在分析阶段对问题加以明确定义之后，就进入设计阶段。设计阶段的目标是针对问题的要求开发出相应的解决方案。也就是要开发出解决问题的逻辑步骤序列，即通常所说的算法。一般而言，得出问题的解决方案之后，还需要对方案进行验证和确认，确保该方案的确能解决对应的问题。

确定了问题的解决方案之后，就需要用某种程序设计语言对方案进行严格的描述，这一过程称为实现。实现阶段的主要任务就是将算法转换为程序。

获得程序之后，需要对程序进行测试[①]，通过测试之后的程序才能发布给用户使用。

任何程序都是人的智力产品，而任何人都有可能犯错误，故很难保证一个实用程序是完全正确的。所以，程序发布之后，用户在使用的过程中也有可能发现程序中的错误，这时就需要对程序进行修改；此外，随着时间的推移，用户对程序的功能可能会产生新的要求，为了满足用户的新要求，往往也需要对程序进行修改，这一类修改工作我们通常称为对程序的“维护”。因此，程序在发布之后就进入了维护阶段。

程序设计初学者常犯的一个错误是：一拿到问题就开始编写代码，也就是直接进入实现阶段，忽略了程序设计过程中的分析和设计阶段。这会严重影响程序设计的质量，甚至导致开发项目的失败，尤其对大型程序的开发更是如此。分析阶段、设计阶段要与实现阶段分开，在开始学习某门具体程序设计语言之前，可以针对一些简单问题设计算法，以此作为学习程序设计的入门方法。

1.2.3 程序设计方法

程序设计方法是指用什么方法来组织程序内部的数据和逻辑。自从 1946 年世界上诞生第一台电子计算机 ENIAC 开始，程序设计方法及程序设计语言就在不断地发展。随着计算机及通信、网络等相关技术的不断发展，计算机的应用越来越普及，程序的规模也越来越大。程序设计的目标不再是片面的高效率，而是对程序的可理解性、可扩充性、可靠性、可重用性等因素的综合考虑，从而使程序设计方法和程序设计语言得到了极大的发展。

1. 早期程序设计

在计算机诞生之初，受硬件技术的限制，计算机的内存空间极为有限，运算速度也较慢。因此在开发程序时，程序员更多地注重程序的执行效率，而程序的可理解性、可扩充性等质量因素往往只能作为次要因素考虑，甚至为了追求效率而被牺牲掉。在这段时期，基本上没有成型的程序设计方法，程序员主要依赖个人技巧和天分进行编程，致使编程成为一种“艺术”，编出来的程序往往在可理解性、可维护性和通用性等方面都比较差。

2. 结构化程序设计

随着计算机硬件技术及相关信息技术的发展，计算机的应用范围越来越广，要开发的程序也

① 严格来说，对大型程序的开发，在每个阶段都需要进行相关测试，以保证程序的质量。

越来越大，越来越复杂，单纯依靠个人的编程技巧已难以编制出能满足应用要求的程序。为适应这一需要，20 世纪 60 年代出现了**结构化程序设计**（structured programming，SP）方法，又称为**过程式程序设计**（procedural programming）方法。

SP 方法的主要思想是：自顶向下、逐步求精、模块化编程，以及采用单入口/单出口的控制结构构造程序。所谓“自顶向下”，是一种问题分解技术，指的是将复杂问题分解为一系列复杂性相对较低的子问题，逐个解决这些子问题，整个问题也就得到了解决；“逐步求精”指的是对问题进行连续分解，直至最后的子问题小到易于解决，最终可用 3 种基本控制结构（顺序结构、选择结构、循环结构）来表示；“模块化编程”是指将较大的程序划分成若干子程序，每个子程序称为一个模块，较复杂的模块可以继续划分为更小的子模块，从而使得程序具有层次结构。

假设我们要编一个程序解决如下问题：判断用户从键盘输入的一个自然数是否为素数。采用 SP 方法，设计过程如下：

首先将该问题分解为 3 个子问题，对应算法的如下 3 个步骤。

步骤 1：输入自然数 n

步骤 2：判断 n 是否为素数

步骤 3：输出判断结果并终止

其中步骤 1 和步骤 3 比较简单，基本上可直接解决；而步骤 2 则需要进一步细化，如可细化为如下子步骤：

步骤 2.1：令 d＝2

步骤 2.2：令 r＝（n 除以 d 的余数）

步骤 2.3：如果 r 等于 0，则将结果置为“是”并转步骤 3；
　　否则，转步骤 2.4

步骤 2.4：令 d 值加 1 并转步骤 2.5

步骤 2.5：如果 d 大于 n 的平方根，则将结果置为“否”并转步骤 3；
　　否则，转步骤 2.2

至此，就解决问题的算法而言已经比较详细，只要掌握了某门程序设计语言，就可以编制出相应的程序。

结构化程序设计方法将程序看作对数据的一系列处理过程，将数据和对数据的处理过程分开，**以过程为中心**，从程序的功能出发进行分解，其结果是一个程序由若干过程组成，每个过程完成一个确定的功能。C++语言中用**函数**（function）来表示过程。

3. 面向对象程序设计

结构化程序设计方法的思想符合人们处理问题的一般习惯，为处理复杂问题提供了有力的手段，结构化程序相比于非结构化程序更容易理解和修改，因此结构化程序设计方法得到了广泛应用。

但是，结构化程序设计将数据和对数据的处理分开，程序中的某一数据可能会被许多过程处理，该数据发生变化将影响到所有相关过程（例如，如果对某数据的结构进行了修改，则所有使用到该数据的过程通常也必须修改）。因此，当程序规模较大、数据量较大时，数据和处理的这种分离状态将使得程序难以被人理解，从而难以维护。针对这一情况，出现了**面向对象程序设计**（object-oriented programming，OOP）方法，并于 20 世纪 80 年代开始逐渐成为主流程序设计方法。

OOP 的基本概念是**对象**（object）和**类**（class）。所谓“对象”指的是客观存在的单个事物[又称为**实体**（entity）][①]，例如一个人、一辆汽车、一架飞机、一个银行账户等，都是对象。对象一

① 对象可以是现实世界的事物，如一辆汽车；也可以是思维世界的事物，如堆栈这种数据结构。

般都具有属性和行为，属性用来描述对象的特征和状态，例如汽车具有发动机、传动系统、燃料系统等属性，银行账户具有账号、户名、存款余额等属性；行为用于改变对象的状态，例如汽车具有启动、加速、刹车等行为，这些行为可以改变汽车的状态。比如，启动行为可以让汽车的发动机从非工作状态转变为工作状态；银行账户的存款、取款等行为可以改变账户的存款余额。对象就是属性和行为的统一体。

对象可以归类，例如张三是一个人，李四也是一个人，则这两个人是同类对象，可以归入“人”这个“类”。类描述了同类对象的共性，例如“汽车”是一个描述某类交通工具的类，而你的汽车和我的汽车都是汽车的实际例子，因此都具有发动机等属性，也都具有启动、刹车等行为。因此，类是描述同类对象共同具有的属性和行为的模型，对象则可以看作根据这个模型制造出来的实际事物，因此又称为类的“实例”。

在程序当中，对象的属性通常用数据来表示，而对象的行为通常用操作来表示。面向对象程序设计以结构化程序设计为基础，最大的改变是将数据和对数据的操作（即数据处理）当作一个整体对象来对待，从而使得在对数据的结构进行改变时，所涉及程序的修改仅限于有限的范围（即对象的范围，具体而言是该对象所属的类的范围）。

OOP 方法以“对象”为中心进行程序设计，程序就是实体在计算机系统中的表示。采用 OOP 方法设计的程序中包含一系列对象，由这些对象的相互协作完成程序的功能。对象是类的实例，因此，面向对象程序的基本构造单位是类。OOP 方法包括封装（即信息隐藏）、继承、多态性等基本特征。

- **封装**（encapsulation）

封装是面向对象方法的重要原则，有两个重要作用：一是把对象的属性和行为结合在一起成为不可分割的独立单位；二是使得对象内部的实现细节可以尽可能隐藏起来不被外界所见，外界只能通过对象所提供的对外接口与对象发生联系。

封装的例子在现实生活当中非常常见。例如，当我们购买一台冰箱时，获得的就是一个冰箱的封装体，生产厂家将冰箱内部的压缩机、控制电路等都隐藏起来，只提供一些控制按钮（接口）给我们，而我们通过这些按钮来使用这台冰箱。

封装的优点包括：

实现信息隐藏。例如，冰箱的内部细节不会暴露给外界。

更容易使用。用户只需了解怎样访问接口就能直接使用，无须考虑实现细节。例如，使用冰箱的人只要知道哪个按钮控制哪项功能，根本无须了解冰箱的内部工作原理，就能顺利操作。

更容易变更内部实现。可以在不考虑用户的情况下变更（只需保持接口不变），因为内部的实现细节对用户的使用没有直接影响。例如，为了更省电，可以改变冰箱的内部设计，但只要保持按钮的设置不变，则不会对用户的使用方式产生任何影响。

C++语言用类来支持封装，类封装了由同类对象所共享的属性和行为。C++中提供访问控制方式 private 和 protected 来隐藏内部信息，并提供 public 访问控制方式来定义公共接口。

- **继承**（inheritance）

客观事物中普遍存在着一种关系：一般和特殊的关系，也就是所谓的“是一种（is a）”关系。例如，梨是一种水果，莱阳梨是一种梨；汽车是一种交通工具，吉普车是一种汽车。针对这种一般和特殊的关系，如果有了关于一般概念的定义，我们在定义新的特殊概念时就不必一切从头做起。例如，有了水果的概念之后，在定义梨的概念时，就不必将梨所具有的水果的一般特征（如多汁、可以生食等）一一列举，而只需简单地说明：梨是一种水果，然后对梨不同于其他水果的特征加以描述即可。在这里，我们定义梨的概念时，使用了已经存在的水果这一概念，这就是**重用**（reuse）。重用了“水果”这一概念之后，定义“梨”这一概念的工作就简化

了，因为只需要描述梨这种水果不同于其他种类水果的特点；同时，对于已经知道了“水果”这一概念的人而言，理解梨这一新概念也就简单了，因为只需要了解梨与其他种类水果的不同之处。

在现实生活中重用的例子极为常见。例如，以前的电视机功能比较单一，只要能收看电视节目就可以了，而随着计算机的普及，许多家庭有了将电视机与计算机相连的需求，以便利用电视机的大屏幕播放视频文件。为了满足这种需求，生产电视机的厂家就推出了新的产品，这种新产品往往就是在原有产品的基础上增加新的部件，使得它可以与计算机连接。这样就不必一切从头做起，从而可以大大加速新产品的推出速度。

面向对象程序设计中继承机制的主要作用就是支持软件重用。利用继承机制，程序员可以通过对现有的类进行扩充（增加新的成员）而定义新的类，用这种方式定义的新类称为**派生类**（derived class），被继承的类称为**基类**（base class）。派生类自动具有基类中定义的所有成员，在此基础上可以定义一些新的成员来满足新的需要。这样一来，定义基类成员的代码得到了重用，从而大大简化了定义派生类的工作量，提高了程序开发效率。

C++语言直接支持继承机制。

- **多态性**（polymorphism）

多态由两个希腊词组成：“poly”意为“多”，“morph”是代表形态的后缀，合起来表示“多种形态”。它通常指一件东西具有很多形态。在面向对象程序设计中，多态是指“同一名字，多种含义”，或者“同一接口，多种实现”，通常指函数和方法（即对象的行为）具有相同的名字，但有不同的行为特征。

在日常生活中我们使用许多东西时，往往只关注它们的功能和使用方法，至于产品的实现方法并不会太关注。例如，当我们使用全自动洗衣机时，只会关心当我们按下开始按钮时，洗衣机是否能按我们事先设定的模式把衣服洗好，至于洗衣机内部采用哪种电机、哪种控制电路则并不关心。可能有采用不同电机的洗衣机，但它们的使用方法是相同的。同样的使用方法（接口），不同的内部实现，这就是多态性。

在面向对象程序设计中，程序员可以为同一函数名定义多种不同的函数实现（称为函数重载），也可以在派生类中对基类中定义的行为提供不同的实现（称为重定义），以适应不同的使用场合，由此实现多态性。

在 C++语言中，通过函数重载、模板、虚函数结合继承等机制来实现多态性。

本书后续章节将进一步介绍 OOP 的上述特征。

1.3　程序设计语言

程序设计语言（programming language）又称编程语言，就是编制程序时使用的语言，用于实现人与计算机之间的交流。程序设计语言是人造语言，一般具有明确的语法和语义，不像人类所使用的自然语言（如汉语、英语等）那样容易产生歧义。

程序设计离不开程序设计语言的支持。任何问题的解决方案（算法）最终都必须用某种程序设计语言加以实现，才能获得可由计算机执行的程序。

同一算法可以采用不同的程序设计语言、编制不同的程序来实现。例如，1.1.2 小节给出的素数判定算法既可以用 C++语言编程实现，也可以用 Java 语言编程实现。不同的程序设计语言有不同的特征，可能支持不同的程序设计方法。例如，C++语言支持面向对象程序设计方法，而 C 语言则仅支持结构化程序设计方法，不能直接支持 OOP。

随着计算机硬件技术和程序设计方法的发展，出现了数以百计的程序设计语言，根据其与人

类自然语言的接近程度，可以分为高级语言与低级语言[①]。

1.3.1 机器语言

在计算机出现初期，程序员使用**机器语言**（machine language）编制程序。机器语言就是计算机的指令集，是唯一能被计算机直接理解和执行的语言。用机器语言编写的程序称为**可执行程序**（executable program），又称**机器代码**（machine code）。机器指令由二进制数字串表示，即每一条指令都是一个由若干个 0 和 1 构成的串。这样的机器语言显然难以记忆和使用，因此使用机器语言编程是一件麻烦而困难的事情。

1.3.2 汇编语言

使用机器语言的程序既难以编制又难以阅读，因此人们便用一些容易记忆和阅读的助记符来表示机器指令中的操作。例如，用 ADD 表示加，SUB 表示减，JMP 表示转向等，这种以一系列助记符为主体构成的语言称为**汇编语言**（assembly language）。汇编语言与机器语言基本上一一对应，用汇编语言编制的程序通常用一个程序自动转换为机器语言程序，完成这一转换工作的程序称为**汇编程序**（assembly program）。

1.3.3 高级语言

机器语言和汇编语言都是面向计算机的语言，通常称为**低级语言**（low-level language）。使用低级语言编程必须涉及机器硬件细节，其表达方式与人类的思维方式相去甚远，编程烦琐而困难；而且低级语言与计算机的 CPU 紧密相关，每种 CPU 的指令集各不相同，相应的低级语言也各不相同，因而用低级语言编写的程序不便于移植[②]。为了改进低级语言的不足，满足计算机广泛应用的需求，人们设计出了**高级语言**（high-level language）。高级语言的表达方式比较接近人类的自然语言（英语），比低级语言使用方便、表达简洁。用高级语言编写的程序称为**源程序**（source program），又称**源代码**（source code）。高级语言一般与具体计算机无关，使用高级语言编制的程序可以在多种计算机上运行。到目前为止，人们已设计出百余种高级语言。

1.3.4 编译型语言与解释型语言

使用高级语言编制的源程序必须转换为机器代码（二进制代码）才能被计算机所理解和执行，这一代码转换过程通常也用程序来完成。根据完成转换过程的不同方式，高级语言可以区分为如下两大类。

- **编译型语言**（compiled language）

使用这类高级语言编制的源程序首先要完整地转换为可执行程序，然后才能执行。完成代码转换工作的程序称为**编译器**（compiler）。C++语言就是典型的编译型语言。

- **解释型语言**（interpreted language）

使用这类高级语言编制的源程序不需要事先完整地转换为可执行程序，而是在执行时逐条语句进行转换（称为解释），解释一条语句就执行一条语句。完成代码转换工作的程序称为**解释器**（interpreter）。BASIC 语言就是典型的解释型语言。

① 除此之外，还有其他的分类方式，如可分为过程式语言和说明性语言。

② 移植就是将在某种计算机上编制的程序放在另一种计算机上运行。

1.3.5　C++语言

C++语言由贝尔实验室的 Bjarne Stroustrup 设计，是目前广泛应用的一种面向对象程序设计语言。1998 年形成了 ISO C++标准，并于 2003 年进行了一次修订，成为目前的 C++，并不断发展。

C++语言以结构化程序设计语言 C 为基础，包含 C 语言的所有内容，并在类型检查、代码重用、数据抽象等方面进行了扩充。除此之外，还增加了对 OOP 的支持机制。因此，C++语言是一种混合型面向对象语言[①]，使用 C++语言既可以开发面向对象程序，也可以开发结构化程序。

1.4　C++程序的结构

任何 C++程序都必须包含一个特殊的函数：main 函数，又称**主函数**（main function）。主函数是 C++程序的执行起点，任何 C++程序都是从主函数开始执行的。

一般而言，C++程序中的主要成分是完成程序功能的语句，这些语句被组织成函数或类。前文已提及，C++语言是一种混合型面向对象程序设计语言，既可用于开发面向对象程序，亦可用于开发结构化程序。因此，用 C++语言编制的程序就有两种基本形式，分别以类和函数为构成单位。

除此之外，C++程序中通常还包括 3 种成分，那就是**空白**、**注释**（comment）和**预处理指示**（preprocessing directive），又称为**预编译指示**（pre-compile directive）。

空白就是程序中出现的空格、空行、制表符等，用于形成程序的格式。例如，同一层次的语句列对齐、内层语句相对于外层语句向右缩进等。恰当地使用空白可使程序结构清楚、层次分明、易于阅读，从而提高程序的可理解性。

1.4.1　注释

注释就是在程序语句上添加的注解，一般源程序中都应该有注释。注释的作用是提高源程序的可理解性，只对阅读程序的人有用。对执行程序的计算机而言，注释毫无作用，因为编译器对源程序进行编译时将忽略其中的所有注释，可执行程序中根本不包含注释的内容。虽然注释对计算机无用，但我们在编制程序时却不能忽视注释，因为很多情况下我们需要阅读源程序。例如，要对某个程序进行修改，我们就必须首先读懂原来的程序，才有可能修改它；对于大型程序而言，往往需要多个开发人员合作开发，因此我们的程序能被人理解就更为重要。初学者很容易犯的一个错误就是：整个程序完全没有注释，或者形式上有注释，但注释根本无法说明程序的编程思路。这样的程序即使是自己编的，但过一两个星期以后也很难保证能看懂。

C++程序中的注释有两种形式：一种以双斜线“//”为标记，称为“行注释”，表示从“//”开始直到本行末尾的内容都是注释；另一种以“/*”开头，以“*/”结尾，称为“块注释”，表示从“/*”开始直到“*/”之间的内容都是注释。行注释的形式比较简单，无需注意头尾标记的匹配，使用起来不容易出错；而块注释无需每行都带双斜线标记，比较适合大块的注释文本，但如果注释文本与程序语句夹杂在一起就不适合采用。

1.4.2　预处理指示

编译器在对源程序进行编译之前会首先对其进行预处理（例如去掉其中的所有注释），这一过程又称为预编译。预处理指示就是在预编译过程中处理的指令，最常用的预处理指示是#include，

① 相对于“纯”面向对象语言（如 Eiffel）而言。

用于将指定头文件的内容插入程序中的当前位置。

我们在编程时，经常需要用到由其他人开发的代码，这就好像一个厨师要做炒鸡蛋时，需要用到鸡蛋、盐、味精等原材料和调味品，但厨师一般不会自己去制造这些东西，也没有这个必要。常规的做法是，厨师使用从市场上购买的这些材料来制作自己的菜式。在现实生活中，这种使用已有产品来构造自己产品的例子比比皆是。例如，生产汽车的企业往往会使用其他企业生产的发动机，制造飞机的企业也大多不会自己去制造每一枚螺丝钉。编程也一样，并不需要每一行代码、每一个功能细节都由自己来完成。例如，输入数据和输出结果是一般程序都有的功能，进行输入/输出时，需要与计算机硬件打交道，对相同输入/输出设备进行控制的程序是类似的，如果每个程序员都自己编写控制键盘或显示器的程序，就会造成大量人力资源的浪费；相反，如果由少数程序员开发公共的控制程序，而其他大多数程序员使用这些公共程序来完成自己特定的输入/输出，则可以大大节省资源且提高开发效率。

使用C++语言编制程序时，如果要用到他人编制的代码，一般形式是由代码提供者给出相关的头文件（也是一种源代码文件），然后用预处理指示#include将其包含到自己的程序中。C++标准中包含一个内容丰富的标准库的定义，C++标准库是使用C++语言编程时最经常用到的公共程序。要使用C++标准库，就需要用#include将相应的头文件包含进来。例如，<iostream>就是最经常使用的头文件，用于进行标准输入/输出。

预处理指示将在第4章中进一步介绍。

1.4.3 以函数为单位的程序结构

以函数为单位的C++程序由一个主函数和若干个其他函数构成。这样的程序结构主要对应于结构化程序设计方法。下面是这种程序的实例。

```
// ****************************************************************
// prime.cpp
// 功能：判断用户从键盘输入的一个自然数是否为素数
// ****************************************************************

#include <iostream>              // 使用其中的 cout 和 cin
#include <cmath>                 // 使用其中的平方根函数 sqrt

using namespace std;             // 使用名字空间 std

bool primeNumber(unsigned);      // 函数原型

// 主函数
int main()
{
   unsigned value;    // value 记录用户输入的自然数
   cout << "Enter a natural number:" << endl;
   cin >> value;

   if (primeNumber(value))
      cout << value << " is a prime number."
           << endl;
   else
      cout << value << " is not a prime number."
           << endl;

   return 0;
}

bool primeNumber(unsigned n)
// 判断 n 是否为素数
// 前置条件：
//    n 已赋值
// 后置条件：
//    如果 n 是素数，则函数返回值为 true
```