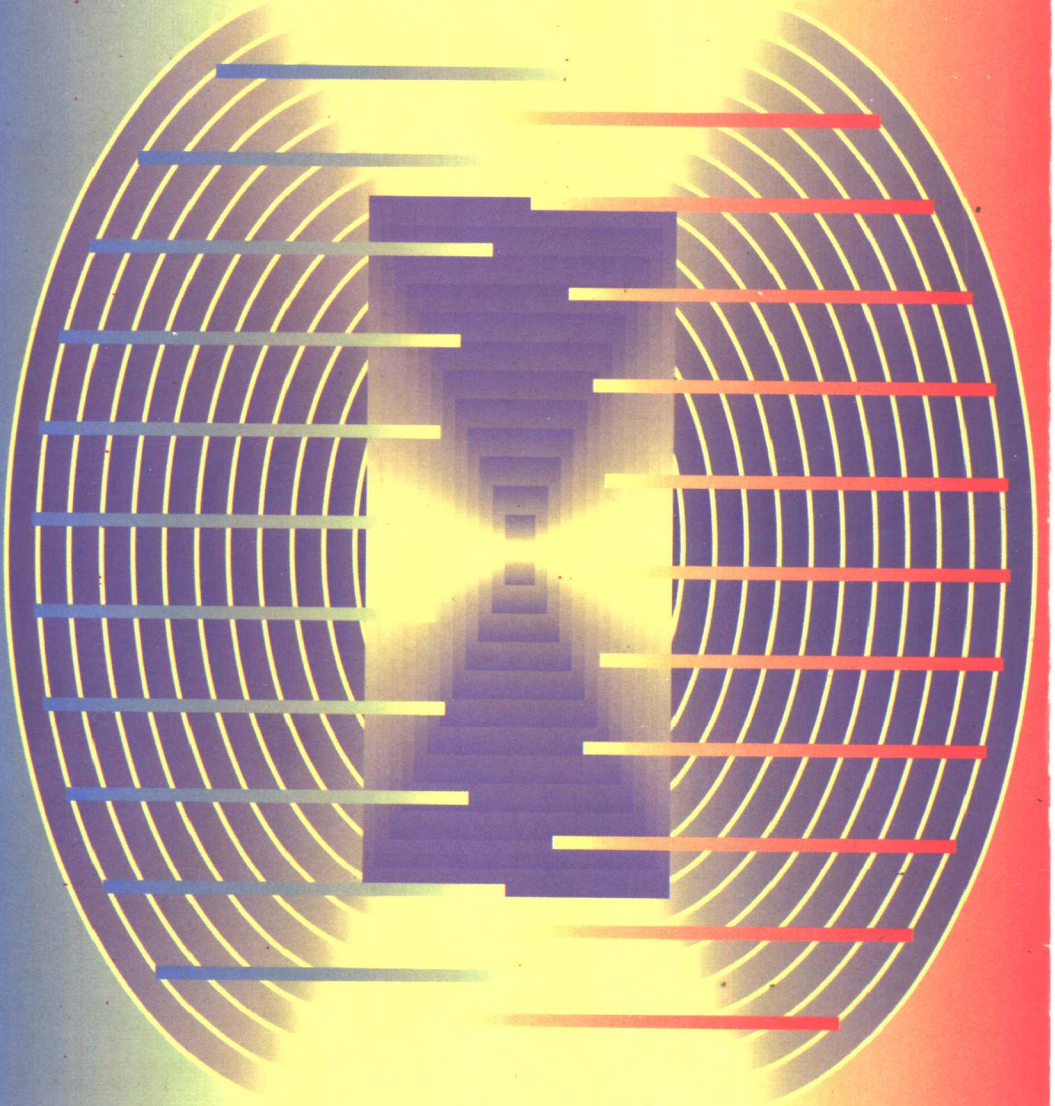
# 面向对象
# 程序设计基础

李师贤 李文军 周晓聪 编著



高等教育出版社

面向对象程序设计基础

出版社

# 面向对象程序设计基础

李师贤　李文军　周晓聪　编著

高等教育出版社

（京）112号

## 内 容 简 介

本书以循序渐进、深入浅出的方式引导读者步入面向对象程序设计的大门。本书假设读者无任何程序设计基础，从基本数据类型与基本控制结构开始，逐步过渡到函数、类与对象、复合数据类型、继承、多态性、类属、输入/输出流等复杂机制，最后还介绍了提高程序可靠性的程序断言与异常处理机制、提高程序可重用性的规范方法等高级内容。书中提供了丰富、典型的例题，并且每一章均附大量精选的练习思考题与上机实习题。

本书不仅介绍面向对象程序设计语言，而且强调面向对象程序设计方法，注重程序设计理论与实践相结合。

本书内容丰富、条理清晰、文笔流畅，适合作为计算机专业程序设计课程的入门教材，也可作为其他专业高年级学生学习面向对象程序设计的教材，同时可供广大软件开发人员参考。



**图书在版编目（CIP）数据**

面向对象程序设计基础/李师贤等编著.—北京：高等教育出版社，1998
ISBN 7-04-006410-3

Ⅰ.面… Ⅱ.李… Ⅲ.面向对象程序设计-教材 Ⅳ.TP311

中国版本图书馆 CIP 数据核字（98）第 12486 号

*

高等教育出版社出版
北京沙滩后街 55 号
邮政编码：100009 传真：64014048 电话：64054588
新华书店总店北京发行所发行
北京二二〇七工厂印刷

*

开本 787×1092 1/16 印张 24.25 字数 600 000
1998 年 8 月第 1 版 1998 年 8 月第 1 次印刷
印数 0 001—6106
定价 19.30 元

# 目　录

# 引　言

我们编写本书的目的是为了给程序设计初学者提供一本清晰的入门教材，该教材以**面向对象程序设计**（Object－Oriented Programming，简称 OOP）为核心，并选用 C++语言作为工具。

程序设计是计算机专业非常重要的基础课程，学好程序设计的入门课程对于软件工程、编译原理、形式语言与自动机、形式语义学等后续课程有很大帮助。程序设计课程包括两个相辅相成的方面：程序设计方法与程序设计语言。自从第一台计算机诞生以来，程序设计方法与程序设计语言一直在不断地发展，20 世纪 70 年代是结构化程序设计的时代，而 80 年代以来面向对象程序设计逐渐占据了程序设计的主流。“面向对象”不再是软件开发中的一个时髦名词，而是对软件开发人员一种很基本的要求。

在过去的 10 多年中，程序设计入门教材一般选用 Pascal 语言或 C 语言，并贯穿结构化程序设计思想。随着面向对象技术逐步成为程序设计的主流，从一开始就向学生传授面向对象程序设计思想是十分必要和有益的。面向对象程序设计与结构化程序设计是解决问题的两种不同的思维方式，前者注重事物的结构，而后者注重事物表现的行为。学习面向对象程序设计并不要求学生先完全掌握结构化程序设计。在程序设计技术迅速发展的今天，面向对象程序设计方法与面向对象程序设计语言可以并应该作为初学者的入门选择。

本书正是为满足初学者的这种需要而编写的，书中为第一次学习程序设计的读者介绍了面向对象程序设计方法与 C++程序设计语言。本书的内容大致可分为两部分，一部分从数据结构和控制结构两个角度介绍程序设计语言的一般概念，另一部分围绕类与对象介绍面向对象程序构造的基本思想。本书选用 C++作为程序设计语言是因为 C++是一种广泛使用的面向对象程序设计语言，它为面向对象程序设计思想提供了较好的支持，同时也保留了 C 语言的灵活性特点。在介绍某些语言特征时，本书针对 C++语言在面向对象程序设计理论上的不完善之处，参考了其他语言的相关概念进行了解释（例如类属、程序断言与异常处理等机制参考了 Eiffel 和 Ada 语言的有关概念）。

与其他面向对象程序设计教材相比，本书有以下特色：

**一、以循序渐进、深入浅出的方式引导读者步入面向对象程序设计的大门**

传统的面向对象程序设计教科书都假设读者已经掌握了结构化程序设计方法以及 C 语言或 Pascal 语言，因而难以作为程序设计入门教材。本书则从程序设计中最基本的概念开始，详细介绍基本数据类型与控制结构，逐步过渡到函数、类、继承、多态性、类属等复杂机制，最后还包括提高程序可靠性的断言机制与异常处理机制、提高程序可重用性的规范方法等高级内容。例如，对于初学者而言，变量是一个比较难理解而又是十分重要的概念，本书在开始时对变量的概念作直观描述以求给读者一个感性认识，在引入指针概念之前，

又对变量的名字、属性、关联和值等要素进行了深入讨论。

### 二、OOP != C++

“!=”是 C++语言的运算符，表示“不等于”。我们想用这个式子表达两重含义：一方面，学习面向对象程序设计不仅仅是学习使用 C++语言，在 C++语言之外还有许多其他的优秀语言支持面向对象程序设计，如 Smalltalk、Eiffel、Objective C 以及最近在 Internet 上广泛应用的 Java 语言；另一方面，以 C++语言编写的程序未必是面向对象的，正如彩色电视机上播出的可能是黑白节目一样，用一种优秀的程序设计语言也可能写出非常糟糕的程序。学习面向对象程序设计并不只是学习面向对象程序设计语言。本书不仅介绍面向对象程序设计语言，而且引导读者采用面向对象的程序设计方法去解决实际问题。在介绍 C++语言的各种机制时，我们力求使得读者不仅会使用它们，而且可以理解这些机制。只有这样，读者才可能容易地转向其他程序设计语言。

### 三、尽早引入类的概念以开拓面向对象程序设计思维方式

类是构造面向对象程序的基本单元。类中的操作是以函数来表达的，在本书中函数更多地是以类中某种操作的形式出现的，而不是作为构造程序的基本单位。提早引入类的概念有助于初学者采用面向对象而不是传统结构化的思维方式去解决实际问题，有助于构造良好的程序结构，为日后处理大型程序打好基础。因而本书在介绍了所必需的基本数据类型与基本控制结构后，即引入函数的概念，紧接着就引入类的概念。

### 四、本书不打算成为一本 C++语言手册大全

C++语言是一种优秀的面向对象程序设计语言，然而 C++语言提供的程序设计机制十分复杂，本书不打算作为一本 C++语言手册大全。我们的目标是从 C++语言十分复杂的概念中整理出最核心的概念介绍给初学者，放弃那些明显为了效率的原因而引入的机制（如联合、内联成员等）或与面向对象程序设计原则有抵触的机制（如 goto 语句等）。

### 五、本书各章均精选典型习题

本书每一章均附有精选的典型习题，以帮助读者掌握该章节内容。习题中包括一些讨论题和实习题，供有兴趣的读者进一步学习。

采用本书学习面向对象程序设计有两条有效途径。途径之一是亲自动手多编程序并上机调试。程序设计是一门实践性很强的学科，本书中的例题、习题中的程序均可作为实习内容。此外在学习过程中如果遇到关于 C++语言机制的疑难问题，也可尝试自己动手编写一些小程序上机解决。途径之二是阅读别人编写的程序，并从程序的可靠性、可理解性、可重用性、可维护性等方面对程序进行评价。这样既可从好的程序中吸取精华，又可从差的程序中吸取教训。

此外，多思考与多提问也是一种好的学习方法。学习一种新的语言机制时不仅要学会使用它们，还应思考与提出一些问题：

为什么要引入这些机制？

这些机制在内部是如何实现的？

是否有其他的解决方式？

不同解决方式相比较有何异同？

通过这些问题，我们不仅学会了这些语言机制，也真正理解了这些语言机制。

本书既可供计算机专业本科学生作为程序设计课程的入门教材，也可供非计算机专业高年级学生作为面向对象程序设计课程的教材。任课教师可根据自己的教学目标、学生基础、实际经验以及学校特点对书中的部分内容进行调整。例如，抽象数据类型、多重继承与重复继承、运算符重载、程序断言、异常处理等内容均可根据需要进行扩展或裁剪，其中有些知识将在后续课程中作进一步深入探讨。

安排并指导学生上机实习对于学好本课程具有重要意义。任课教师既可在每一章布置不同的上机实习题，也可安排二、三道典型上机实习题贯穿全书内容。例如，不断改进上机实习题 6-1 中的 MEGA_INTEGER 类可覆盖前六章及函数与运算符重载、输入／输出流、异常处理等知识单元，而比较 8.6 节给出的多态数据结构与 9.2 节给出的包容数据结构则覆盖了继承机制、运行时多态性、类属机制、文件流等知识单元。

对于计算机专业本科学生，建议本课程课内学时数为 72 学时（其中含习题课 10 学时，除第一章与第十二章外每两章安排一次习题课），学生上机学时数共 40 个学时。对于非计算机专业学生，课内学时至少 54 学时，上机学时数不少于 36 学时。

如果安排 72 个课内学时、40 个上机学时，教学计划可参考以下学时数安排（各章后的括号中列出的分别是课内学时数与上机学时数，其中课内学时数不包括习题课）：第一章（2／2）、第二章（4／2）、第三章（6／4）、第四章（6／4）、第五章（6／4）、第六章（8／6）、第七章（8／4）、第八章（6／4）、第九章（4／4）、第十章（6／4）、第十一章（4／2）、第十二章（2／0）。

南京大学计算机科学系许满武教授认真审阅了书稿，并提出了宝贵的意见；中山大学计算机科学系张治国、乔琳、吴向军诸位老师多次参加了书稿的讨论。编者从中获益良多，在此谨向他们致以衷心的感谢。

由于编者水平所限，书中谬误之处在所难免，恳请广大读者不吝批评指正。

编　者

# 第一章 程序设计与C++语言初步

## §1.1 计算机程序

### 1.1.1 算法

计算机出现前，人类已经积累了许多解决问题的经验。解决问题时并不一定使用计算机，如果使用计算机，只不过在解决问题的时间、空间、精度等方面提供更大的方便而已。

给定两个正整数 $p$ 和 $q$，如何求出 $p$ 和 $q$ 的最大公约数 $g$? 对于这样一个问题，数学家欧几里德（Euclid）给出了一个解决方案，这个方案由三个步骤完成，如例 1.1.1 所示。

**例 1.1.1** 求解最大公约数的欧几里德算法。

步骤 1: 如果 $p < q$，则交换 $p$ 和 $q$。

步骤 2: 令 $r$ 是 $p / q$ 的余数。

步骤 3: 如果 $r = 0$，则令 $g = q$ 并终止;

否则令 $p = q, q = r$ 并转向步骤 2。

按照上述步骤，我们可以逐步地计算出任意两个正整数的最大公约数。计算工具可以是纸和笔，也可以是先进的计算机。这种用来解决问题的由有限多个步骤组成的具体过程我们称之为**算法**（algorithm）。

从以上的简单算法，我们可以看出一个算法具有的基本特征。算法是由一些能够机械执行的操作组成的，欧几里德算法中包含了除法、比较、转向等操作。算法可以具有多个输入和输出，欧几里德算法的输入是正整数 $p$ 和 $q$，输出是 $p$ 和 $q$ 的最大公约数 $g$。算法对于任何输入都应该是可终止的，欧几里德算法交替执行步骤 2 和步骤 3，执行步骤 3 时可能重新转向执行步骤 2，也可能终止算法，我们可以证明最多执行 $1+2q$ 步之后算法一定会终止。算法的主要操作对象是数据，欧几里德算法中除了输入 / 输出数据外，还包括保存中间计算结果的数据 $r$。

### 1.1.2 实体

除数学问题外，现实生活中的许多**行为**（behaviour）也可以用算法来表示。例如，在银行帐户中存款或从银行帐户中取款这两种行为可以表示为例 1.1.2 和例 1.1.3。

**例 1.1.2** 在银行帐户中存款。

输入: 存款金额 $m$ 和当前余额 $b$。

输出: 新余额 $b'$。

步骤: 令 $b' = b + m$ 并终止。

**例 1.1.3** 从银行帐户中取款。

输入： 取款金额 $m$、当前余额 $b$ 和透支限额 $v$。

输出： 已取金额 $m'$ 和新余额 $b'$。

步骤 1： 如果 $m > b + v$ 则令 $m' = 0, b' = b$ 并转向步骤 2;

否则令 $m' = m, b' = b - m$ 并终止。

步骤 2： 提示超额透支并终止。

这种表示方式忽略了很重要的一点，即存款行为与取款行为所作用的数据均包含帐户当前的余额，存款与取款行为是密切相关的。

按照我们日常的思维习惯，例如银行帐户这一类的个别事物通常被看作是一个**实体**（Entity），其中包括了帐号、户名、地址、密码以及当前余额等信息，都可完成存款、取款等行为。而不同的帐户实体所具有的信息可能是不相同的，例如每一个帐户都有唯一的帐号、自己的姓名、与别人不同的密码、当前的存款余额等。通常一个实体应具有一个名字、一组表示该实体特征的数据以及若干作用在这些数据上的行为。实体具有的数据表示了它的状态，而这些状态可由实体的行为来改变。

银行帐户可描述为实体，如例 1.1.4 所示，存款与取款只是表现在这一实体上的行为。

**例 1.1.4** 银行帐户实体。

实体： 银行帐户。

属性： 帐号、户名、地址、密码、当前余额 $b$、透支限额 $v$。

行为： 1) 存款

输入： 存款金额 $m$。

输出： 无。

步骤： 令 $b = b + m$ 并终止。

2) 取款

输入： 取款金额 $m$。

输出： 已取金额 $m'$。

步骤 1： 如果 $m > b + v$，则令 $m' = 0$ 并转向步骤 2;

否则令 $m' = m, b = b - m$ 并终止。

步骤 2： 提示超额透支并终止。

从实体出发来把握事物或从行为出发来把握事物，其实是我们理解事物的两种不同思维方式：前者是从事物内在结构的角度出发，而后者则是从事物外部表现出来的行为出发。我们通过观察与分析事物表现出来的行为来探讨事物的内在结构，掌握了事物的内在结构后又可用来解释或预测事物的行为。在处理大型问题时，从实体出发比从行为出发更容易把握问题的复杂性。

### 1.1.3 程序

有了实体后，如何利用计算机来解决问题呢？**程序**即是实体在计算机中的体现。然而实体中的属性与行为是如何转换为程序而在计算机中工作的呢？因此必须首先了解计算机的工作原理。计算机内部只能表示 0 或 1 这样的二进制数，因而所有由计算机处理的数据和处理这些数据的程序都必须表示为二进制数。

### 1. 数据在计算机内部的表示

计算机中的数据可以分为数值型和字符型两大类。数值型数据包括整数、浮点数、有符号数、无符号数等，字符型数据包括字母、数字、标点等符号。无论是数值型还是字符型数据，在计算机内部都是通过二进制编码来表示的。由于二进制数书写困难且易产生错误，所以我们经常用八进制或十六进制数来表示，因为三者之间有比较直接的转换方法，如表 1.1.1 所示。

**表 1.1.1 不同数制的对应表示**

| 十进制 (decimal) | 二进制 (binary) | 八进制 (octal) | 十六进制 (hexadecimal) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 2 | 10 | 2 | 2 |
| 3 | 11 | 3 | 3 |
| 4 | 100 | 4 | 4 |
| 5 | 101 | 5 | 5 |
| 6 | 110 | 6 | 6 |
| 7 | 111 | 7 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

在计算机内部表示整数的方法有原码、补码和反码等表示方法。如果数值有符号，通常用最高有效位表示（如 0 表示+，1 表示－）。实数的表示则有定点和浮点两种形式。在计算机内部表示数值型数据要特别注意容量问题，即给定字长和表示方法，可以表示的最大数或最小数分别是多少。例如,用两个字节（共 16 个二进制位）表示无符号整数，则最小可表示的值是 0，最大可表示的值是 $2^{16}-1$，即 65 535。如果这两个字节表示有符号整数，则最小可表示的值是 $-2^{15}$，即 －32 768；最大可表示的值是 $2^{15}-1$，即 32 767。

字符型数据主要有两种编码方法：由美国国家标准局（ANSI）制定的 ASCII(American Standard Code for Information Interchange)码与 IBM 公司提出的 EBCDIC(Extended Binary Coded Decimal Interchange Code)码。EBCDIC 码主要用于 IBM、UNIVAC 等大、中型计算机，ASCII 码则普遍用于微型计算机及 UNIX 工作站（见附录 A）。汉字在计算机内部的表示采用最高位均为 1 的双字节，我国也制订了相应的标准——GB 2312《信息交换用汉字编码字符集——

基本集》。

**2. 程序在计算机内部的表示**

不管是哪一种计算机，它们各自都有一套基本的指令系统，其中的每一条指令都是由二进制编码表示的，这种表示程序的低级语言称为机器语言。

然而使用机器语言编写程序是很麻烦的事，程序员必须记住许多由二进制编码构成的指令，这样的程序既难写又难读，因而人们采用一些易记的符号表示这些代码，例如用 MOV 表示赋值指令的二进制编码、JMP 表示跳转指令的二进制编码等，这种比机器语言更容易读写的语言称为汇编语言。汇编语言中的符号与机器指令基本上是一一对应的，因而将一个用汇编语言编写的程序转换为计算机可以理解的机器语言程序并不困难，这一转换工作是由一个程序完成的，称为汇编程序。

使用汇编语言编写程序时，经常有些程序段需要重复使用。为了表示这些公用的程序段，引入了宏的概念。程序员将常用的程序段设计为宏，一个宏通常完成一个明确的功能，并能够多次在程序中使用。支持宏的汇编语言称为宏汇编语言，相应的转换程序称为宏汇编程序。

无论是机器语言、汇编语言还是宏汇编语言，都是以计算机领域的术语来表达对问题的解的，这与我们日常思考问题或表达问题解的方式相距甚远。由于每种计算机的基本指令系统不尽相同，对于同一算法或实体编写的程序也有所不同。高级语言正是为满足这些需要而产生的。用高级程序设计语言编写程序更接近自然语言，因而可以更好地表达我们的思维过程。计算机并不能直接理解高级语言，与汇编语言类似，我们需要一个程序将高级语言编写的程序转换为计算机能够理解的机器语言，这种程序称为编译程序或解释程序。一种高级语言普及后，在各种计算机上都建立这种语言的编译程序或解释程序，从而在一种计算机上编写的程序就很容易放在另一种计算机上运行，提高了程序的可移植性。

### 1.1.4 程序设计

程序设计是一种编写计算机程序的活动。由于计算机是一种专门用程序来解决特定问题的“通用”工具，因而程序设计是为解决某一特定问题而构造一种专用工具的活动。

要完成程序设计工作必须具备四个方面的知识：一是应用领域的知识，二是程序设计方法，三是程序设计语言，四是程序设计环境与工具。

应用领域知识是构造实体中属性与行为的基础。例如，要解决最大公约数问题，就必须具备数论方面的知识，了解最大公约数的基本概念与特性——对两个正整数 $p$ 和 $q$（假设 $p > q$），$p$ 和 $q$ 的最大公约数等于 $p-q$ 和 $q$ 的最大公约数，即 $\gcd(p, q) = \gcd(p-q, q)$；要解决银行帐户的存款与取款问题，必须具备有关帐户的知识——如帐户都有帐号、户名、密码、透支限额等属性，还有取款不可超过透支限额等约束。如果程序员不具备应用领域的知识，对程序设计方法与程序设计语言再熟练也只不过是“巧妇难为无米之炊”而已。

有了应用领域的知识，我们还必须按照某些明确的步骤，运用适当的思维方式才能构造出实体、设计相应的数据结构或算法，这就是程序设计方法。程序设计方法大致上可以分为两类：**自顶向下**（top - down）或**自底向上**（bottom - up）。许多程序设计方法通常是这两种方式的结合。

使用计算机解决问题时，必须将实体转换为程序来表示，这就需要掌握程序设计语言这门工具。除少数系统软件外，程序一般采用高级语言来编写。高级语言种类繁多，各种语言都有自己的特色，我们无法逐一学习，只能在学习一种语言的程序设计时不仅学会如何使用该语言，而且应该理解这些语言机制，这样才能比较容易地转向其他语言。

对于大型程序设计，我们经常还需要程序设计环境与工具的帮助。程序设计环境与工具提供了许多可重用的基本程序供我们在设计程序时利用，这些基本程序通常以类库或函数库的形式提供，但这些库通常依赖于操作系统（例如 DOS、Windows、OS/2、UNIX 等），并且不同软件公司（例如 Microsoft、Sunsoft、Borland 等）提供的库也有所不同。

本书主要介绍程序设计方法与程序设计语言两方面的知识，应用领域的知识需要读者在实际开发中不断积累，而程序设计环境与工具的知识则需要读者根据自己使用的开发环境与工具选择合适的参考手册另行学习。

程序设计方法是独立于具体程序设计语言的一种技术，由于程序设计通常离不开程序设计语言，所以许多人容易混淆程序设计方法与程序设计语言。另一方面，程序设计方法与程序设计语言是相辅相成的：采用某种程序设计方法编写程序需要相应的程序设计语言作为工具，而程序设计语言的设计主要是为了支持某种程序设计方法。

## §1.2 程序设计的演变

自从第一台计算机诞生以来，程序设计方法与程序设计语言不断发展。早期，由于计算机硬件条件的限制，运算速度与存储空间都迫使程序员追求高效率，编写程序成为一种技巧与艺术，而程序的可理解性、可扩充性等因素被放到第二位。随着计算机硬件与通信技术的发展，计算机应用领域越来越广泛，应用规模也越来越大，程序设计不再是一两个程序员可以完成的任务。在这种情况下，编写程序不再片面追求高效率，而是综合考虑程序的可靠性、可扩充性、可重用性和可理解性等因素。正是这种需求刺激了程序设计方法与程序设计语言的发展。

### 1.2.1 早期程序设计

早期出现的高级程序设计语言有 FORTRAN、COBOL、ALGOL、BASIC 等语言。FORTRAN 是 FORmula TRANslator 的缩写，是最早广泛使用的高级语言之一，主要应用在科学计算领域；COBOL 是 COmmon Business Oriented Language 的缩写，主要应用在商业事务处理领域，其中的许多指令是为了会计或工资一类应用而设计的；ALGOL 是 ALGOrithmic Language 的缩写，是一种通用的算法语言；BASIC 是 Beginner's All - purpose Symbolic Instruction Code 的缩写，主要面向初学者，BASIC 语言简单易学，但不是很先进。

这一时期，由于追求程序的高效率，程序员过份依赖技巧与天分，不太注重所编写程序的结构，这一时期可以说是无固定程序设计方法的时期。一个典型问题是程序中的控制随意跳转，即不加限制地使用 goto 语句，这样的程序对别人来说是难以理解的，程序员自己也难以修改程序。

### 1.2.2 结构化程序设计

随着程序规模与复杂性的不断增长，人们也不断探索新的程序设计方法。Bohm 和 Jacopini 证明了只用三种基本的控制结构（顺序、选择、循环）即可实现任何单入口／单出口的程序；Dijkstra 建议从一切高级语言中取消 goto 语句从而引起一场关于 goto 语句的大辩论；Mills 提出程序应该只有一个入口和一个出口。这些工作导致了结构化程序设计方法的诞生。

结构化程序设计方法的主要技术是自顶向下、逐步求精，采用单入口／单出口的控制结构。自顶向下是一种分解问题的技术，与控制结构无关；逐步求精指结构化程序的连续分解，最终成为三种基本控制结构的组合。结构化程序设计的结果是使一个结构化程序最终由若干个过程组成，每一过程完成一个确定的功能。

Pascal 语言是由 Niklaus Wirth 根据结构化程序设计方法开发出来的语言，它是因纪念 17 世纪法国数学家 Blaise Pascal 而命名的。其特点是提炼出程序设计共同的特征并能将这些特征编译成高效的代码，因而成为结构化程序设计的有力工具，在教育界深受欢迎。

C 语言也是一种广为流行的结构化程序设计语言，它是由贝尔实验室的 Brian Kernighan 和 Dennis Ritchie 开发的。C 语言最初的目标是为描述和实现 UNIX 操作系统提供一种工作语言，UNIX 操作系统中 90%的代码是用 C 语言编写的。随着 UNIX 操作系统的成功和广泛应用，C 语言也成为一种普遍使用的程序设计语言。C 语言具有灵活方便、目标代码效率高、可移植性好等优点。美国国家标准局于 1987 年制定了 C 语言的标准，称为 ANSI C。

由美国国防部支持开发的 Ada 语言是结构化程序设计的总结，它是为了纪念第一位程序员 Ada Augusta 而命名的。Ada 语言包括了许多新的机制，如模块化、数据抽象、类属参数、异常处理、并发处理等。

### 1.2.3 面向对象程序设计

到今天，结构化程序设计已无处不在，几乎每种程序设计语言都具备支持结构化程序设计的机制。然而，随着程序规模与复杂性的增长，程序中的数据结构变得与这些数据上的操作同样重要。在大型结构化程序中，一个数据结构可能被许多过程处理，修改此数据结构将影响到所有这些过程。在由几百个过程组成的成千上万行结构化程序中，这相当麻烦并且容易产生错误。

面向对象程序设计建立在结构化程序设计基础上，最重要的改变是程序围绕被操作的数据来设计，而不是围绕操作本身。面向对象程序设计以类作为构造程序的基本单位，具有封装、数据抽象、继承、多态性等特点。

Simula 语言是面向对象程序设计的先驱，它首先提出类的概念；Smalltalk 语言及程序设计环境的成功导致了面向对象程序设计的兴起；Eiffel 语言是完全根据面向对象程序设计方法开发出来的纯面向对象语言，具有强类型、多重继承、类属机制、断言机制、异常处理、自动内存管理等特点，进一步完善了面向对象程序设计。

C++语言则是目前应用最广泛的面向对象程序设计语言。C++语言是对 C 语言的扩充（或称为 C 语言的超集），它继承了 C 语言高效、灵活的特点，完善了 C 语言的类型检查、代码重用、数据抽象机制，扩充了对面向对象程序设计的支持。目前 C++语言尚未完成标准化工作，基本上以 AT&T 贝尔实验室公布的文本为标准。C++语言是一门还在发展的语