仲萃豪
冯玉琳 编著
陈友君

程序设计方法学

北京科学技术出版社

程序设计方法学

仲萃豪　冯玉琳　陈友君　编著

北京科学技术出版社

程序设计方法学

仲萃豪　冯玉琳　陈友君　编著

*

北京科学技术出版社出版
（北京花园路3号3号院）
89920部队印刷厂印刷
北京市新华书店发行
各地新华书店经售

*

开本：787×1096毫米　印张：9 7/8 字数：212千字
1985年1月第一版　1985年1月第一次印刷
印数：1—22000定价：1.75元
统一书号：15274·011
本社书号：037

内 容 提 要

本书论述了程序设计方法学的科学原理和方法，把那些公认为成熟的、有实用价值的、最基本的内容收入本书；另外，适当地介绍了一些尚待进一步发展的研究课题。本书主要介绍了程序的基本结构和程序设计的基本知识，以及设计程序的各种方法，由此构成了结构程序设计的基本内容。最后采用了比较形式化的工具，介绍了程序推导、程序变换等有待深入研究的课题。本书主要章节后面附有习题。

本书可供专门从事计算机工作的科技人员阅读，也可作为大专院校计算机专业学生、研究生的教科书。

写在前面

程序设计方法学是一门非常年轻、发展又极其迅速的学科，仅近十多年时间内，已趋向成熟，因此我们编写了这本教程，旨在阐明程序设计方法学的科学原理和方法，并按教学要求，尽可能地把那些公认为成熟的、具有实际应用意义的、最基本的内容收入书中，还适当地介绍了一些尚待进一步发展的研究课题。

全书分为三部分，共十二章。

第一部分为1—4章。介绍程序的基本结构和程序设计的基本知识，并由此熟悉本教程所使用的程序设计语言T。

第二部分为5—9章。在讨论了GOTO问题之后，第6章讲自顶向下的逐步求精方法；第7、8两章分别介绍过程和模块的设计方法；第9章是递归程序的设计方法。以上两部分内容组成了结构程序设计的基本内容。

第三部分为10—12章，讨论了使用比较形式化的工具进行程序设计；第10章介绍程序的形式推导技术；第11章是程序变换技术；第12章论述了研制程序工具和环境的重要意义。这几章的内容都是有待进一步深入研究的课题。

书中附有大量习题，以作为各章内容的补充。书后的附录是程序设计语言T的文本。

本书侧重于程序设计科学，也涉及到一部分程序设计技术。使读者能掌握基本的程序设计理论和方法，从而能应用到实际工程中去。因此，本书内容的重点是逐步求精，模块程序设计和程序的形式推导这三章。

有关程序的推导和变换是最近几年发展起来的课题。实际上，推导技术是结构程序设计的继续。它把程序设计提高到更加数学化和科学化的地步。为此，我们将Gries的近著“程序设计科学”一书的内容加以总结，汇集为一章加以介绍。书中所列入的程序变换技术虽然还未用于实际，但它已成为本世纪程序设计自动化研究的中心课题，可以使我们思路开阔，有助于指导实际工作。预计不久的将来，它无疑会成为程序设计的重要方法之一。

除这些内容之外，程序设计方法学还应包括并发程序设计、多机通讯技术、函数型程序的代数变换、语言的公理化系统、语义学等，本书均未列入。一方面是为了使材料尽可能符合本教程的旨意；另一方面我们并不想使它成为一本理论性的教材，而希望是一本具有实际应用价值的、且以计算机科学为指导思想的教程。

本书可供高等学校计算机科学系高年级学生或低年级研究生作为教科书或教学参考书，也可供同等学历的软件工程技术人员学习之用。

我们假定读者已具备一般的程序设计语言知识和基本的数理逻辑知识，并有程序设计的训练，懂得集合的一般概念，学过大学一、二年级的数学课程。

讲授本教程约需60～80学时，各章节介绍的内容顺序可根据学生的情况加以改变，实例也不一定全部讲解。有条件的话，尽可能在计算机上作些实习。

学完本教程的读者一定会感到，展示在眼前的程序设计技术将使你眼花瞭乱。但是，我们要劝告读者，掌握这门技术，必须亲自动手，在自己设计的系统中实际应用一下，至

少要做一下练习，那时你才能真正领会和掌握所学到的东西。其中有几章的练习比较难，希望教师给以提示或进行集体讨论，我们将在适当的时候发表习题答案。

作者真诚地感谢樊哲民同志提供了本教程第11章的部分内容；感谢袁崇义同志仔细地审阅了本书的全部手稿，提出了很多意见和批评。本书曾在中国科技大学、科学院研究生院、北京大学、清华大学、西安交通大学、郑州大学等十多处院校讲演过，得到了所在单位教师的支持，提出了许多宝贵意见，这里一并表示感谢。

限于经验和水平，又因这门学科还正在发展中，书中难免有错误和不妥之处，恳切希望计算机科学界的同行和读者们不吝指教。

作者

1984年1月

目　　录

第1章 引 论

1.1 什么是程序设计方法学

回顾程序设计发展的历史，大体上可划分为三个不同的时期。

五十年代的程序都是用指令代码或汇编语言来编写的。这种程序的设计相当麻烦，编制和调试一个稍许大一点的程序，常常要花费一年半载的时间。在早期讲授程序设计的时候，总是用一台机器的指令系统或它的汇编语言作为抽象机，然后在它上面讲授如何设计计算公式、分叉、循环和子程序等程序设计知识。当时评论程序的好坏标准，就是看它能否作到指令条数少，存贮单元省，执行速度快。按这种教学方式，培养一个熟练的程序员要经过长期的训练和实习，这种局面严重影响了计算机的普及应用。

FORTRAN 语言的出现，改变了这种局面。1958 年，FORTRAN 语言刚被使用时，本身还存在很多缺陷，如编译时间长、质量差、错误多等等，但是使用高级语言大大简化了程序设计，缩短了解题周期，因此显示出强大的生命力。此后，编程序不仅是软件专业人员才能做的事了，一般工程技术人员花上几周时间学习，也能使用计算机解题。

随着计算机的应用日益广泛地渗透到各学科和技术领域，六十年代发展了一系列不同风格的、为不同对象服务的

程序设计语言。其中较为著名的有ALGOL、COBOL、LISP、SNOBOL、PL/1、APL和APT等十多种语言，而当时被采用的语言达三、四百种之多。高级语言的蓬勃兴起，使得编译理论和形式语言日趋完善，这是该时期的主要特征。但就整个程序设计方法而言，并无实质性的改进。

自六十年代末到七十年代初，出现了大型软件系统，如操作系统、数据库，这给程序设计带来了新的问题。大系统常常需要花费大量的资金和人力，可是，研制出来的产品却常常是可靠性差，错误多，维护和修改也很困难。一个大型操作系统有时需要几千人年的工作量，而所获得的系统又常常会隐藏着几百甚至几千个错误。当时，人们称这种现象为“软件危机”。

“危机”震动了软件界，这就促使人们开始重新研究程序设计中的一些最为基本的问题。例如，程序的基本组成部分是什么？应该用什么样的方法来设计程序？程序设计的主要方法和技术应如何规范化和工程化等等。

1969年，Dijkstra首先提出了结构程序设计的概念，它强调了从程序结构和风格上来研究程序设计。经过几年的争论、探索和实践，结构程序设计的应用确实取得了成效，用结构程序设计的方法编出的程序不仅结构良好，易写易读，而且易于证明正确性。

结构程序设计方法的推行，其意义远不止于当前的实用价值，更深远的意义在于它向人们揭示了研究程序设计方法的必要性。程序设计并不纯粹是一种技术性的活动，与任何学科一样，它自身也有一套基本的原理和方法。七十年代程序设计技术的发展是以方法论的研究为特征的。除结构程序

设计的方法日趋完善外，在数据类型抽象、程序的推导和综合、程序变换和程序自动化等方面，都取得了重要的进展。

程序设计方法的一个基本原则是抽象。为了解决一个复杂的问题，人的智力往往不可能一下子就触及到问题的细节方面。在分析了问题的要求之后，我们总是首先设计出一个抽象算法。在抽象数据上实施一系列抽象操作，这些数据和操作反映了问题的本质属性，而将所有细节都抽象掉了。然后，作为下一步，再考虑这些抽象数据和操作的具体实现。在抽象级，我们只要知道“做什么”，而在实现级，我们才考虑“如何做”。由于抽象技术的采用，使大问题分解成了相对独立的一些子问题，它们分别只涉及局部的环境和条件，可以独立地一个个地被解决，从而使整个问题得到圆满解决。

另外两个值得提出的基本原则是枚举原则和归纳原则。程序中的条件选取结构是枚举原则的应用，而循环重复结构则是归纳原则的应用。

抽象、枚举和归纳，这是人们通常进行思维的方法，也是进行程序设计的基本原则。

程序设计还涉及到程序性质的表示，程序的可靠性和可维护性，程序的效率等许多方面。为了改善整个程序设计的过程，使之更加科学化、规范化，这就需要有一整套关于如何进行程序设计的理论和方法。这就是程序设计方法学。结构程序设计是程序设计方法学的一个重要组成部分。

从计算机教育的角度来看，程序设计方法学将成为我们培养程序设计专门人才的必不可少的一门课程。有人认为，以前不学程序设计方法学，不是也能写出程序吗？美国康奈

尔大学D·Gries教授曾做过一次试验。他邀请了40～50名研究生和一些来自工业界的专业人员和程序员一起来解决一个问题（行向右对齐问题）。结果竟有半数人程序编得不正确，而Gries本人使用了方法论中的一些行之有效的技术，写出的程序不仅结论正确、结构清晰，而且比所有其他人的程序更有效，编写程序的时间也并不比其他人长。这个例子说明：虽然不学程序设计方法学也可以写程序，但是这样的程序设计既费劲，又易出错，这当然不是我们所希望的。因此，要下决心对程序员进行程序设计方法学的教育，以便从根本上摒弃原有的程序设计的观点和习惯。

在程序设计方法学的发展过程中，常常因对一些问题的看法不一致而引起争议。针对同一个问题，有时因为概念上理解的误差，往往就会带来不同的结论。为此，在进入本教材正文之前，有必要将几个有代表性的有争议的概念预先给以澄清。

• 关于好结构和效率

所谓好结构程序，指程序结构清晰，易于理解，也必然易于验证。好结构程序从效率上看，不一定是最好的程序。但是，它能提高程序的可靠性，便于检查和维护。结构程序设计的观点要求设计好结构程序。在七十年代初，曾发生过要不要GOTO语句的争论，这场争论的实质就在于讲究好结构还是讲究效率。在计算机硬件技术迅速发展和机器成本大幅度下降的今天，人们已普遍认为，除了系统的核心程序以及其他一些特殊要求的程序以外，在一般情况下，宁可降低一些效率，也要保证程序的好结构。

• 关于证明程序的正确性

证明自己写的程序正确，这应是程序员义不容辞的职责。所谓证明，就是使自己和别人确信一个断言真实性的论证。这种意义下的证明并非一定指从公理出发的形式推导，更非机械证明。正如Knuth所指出的，一个正确性证明，实际上是指一种理解，这种证明，可以是形式的，也可以是非形式的。

自从1967年Floyd提出赋给程序意义的概念以来，程序正确性证明的技术获得了很大进展，其主要标志是Hoare的公理化和Dijkstra的最弱前谓词推演。尽管这些研究目前尚难以工程化，但是，它们从不同侧面刻划了程序性质，对于程序设计方法学的发展，无疑起了很大的促进作用。

基于机械证明技术，现在也已发展有各种不同风格的自动证明系统或程序自动设计系统。这些系统的研制，对于程序设计自动化和程序设计方法学，显然是有益的贡献。但我们认为，它只能是一种辅助工具，盲目地追求庞大的纯机械证明系统，正如Perlis所说的那样，注定是要失败的。

• 关于小规模程序和大规模程序

与任何大型系统软件比较，任何一本程序设计方法学的教程中所举的示范性实例，都只是所谓小规模程序。人们之所以着重于小规模程序的训练，这是因为：只有学会有效地研制小的程序，才能期望有效地研制大的程序或程序系统。

这个观点曾得到Dijkstra的有力论证。假设一个程序是由n个小部分组成，每部分正确的概率是p，那末，整个程序正确的概率P一定满足

$$P<p^n$$

当 n 足够大时，如果我们希望 P 是个接近于 1 的有意义的值的话，p 就应该非常接近于1。

另一方面，Dijkstra也告诫人们，不要低估“量”的因素。为了进行大程序或程序系统的研制，仅仅凭借已有的小规模程序设计的经验是不够的。比如说，这样的程序或系统总是由许多人共同研制的，人员的组织和编排就是一个必须考虑的问题。值得指出的是，本教程所讲程序设计方法的那些原则，如模块，自顶向下的逐步求精等等，对于大程序或系统，不仅适用，而且显得格外重要。

• 关于程序设计是一门艺术还是一门科学

六十年代，一个程序员只要能正确理解程序设计语言的意义，经过一定的实践，掌握程序设计中的一些基本技巧，学会使用一些算法，就可以编制出适应当时要求的程序来。为此，人们称程序设计是一种艺术。然而如前所述，由于大型软件系统的发展，原有的程序设计基本技巧已不能满足要求。从七十年代起，人们开始总结出一套程序设计的基本原理和方法，摸索出了一套规律。从而，使它不再是一种纯粹技术性的工作，而逐渐上升为一门科学性的学科，用以指导程序设计。诸如，研制好结构的程序和进行程序正确性证明等，是颇有价值的。但是要编制出一个高质量的程序，在实现级的细节上和程序优化等方面，则还不能离开人的聪明才智及其积累的丰富经验。因此我们认为程序设计既是一门科学，又仍保持着艺术的特点。

本教程的内容侧重于讲述原理、方法和规律，但有时还要涉及到一部分技术性的问题。

1.2一个例子：求最大公约数

为了直观地了解如何从问题的要求出发，逐步设计出计算法，我们举一个例子。

设有不全为0的整数x、y，记gcd(x,y)为它们的最大公因数，即同时能整除x、y的公因数中之最大者，例如：

$$\gcd(3,5)=1$$

$$\gcd(8,-12)=4$$

$$\gcd(2,0)=2$$

函数gcd(x,y)具有如下性质：

$$\gcd(u,v)=\gcd(v,u) \tag{1.2.1}$$

$$\gcd(u,v)=\gcd(-u,v) \tag{1.2.2}$$

$$\gcd(u,0)=|u| \tag{1.2.3}$$

现要求计算两个不全为0的整数a、b的最大公因数gcd(a,b)。根据性质(1.2.1)、(1.2.2)，我们不妨假定a、b为非负，因为当a、b为负数时，只要求gcd(|a|,|b|)就是了。

根据(1.2.1)，a、b的位置是完全对称的，我们针对b讨论之：b可以大于0，也可以等于0。如果b=0，根据(1.2.3)，立即得到

$$\gcd(a,0)=a$$

问题就解决了。如果b>0，那么，我们用自然数变量x、y来代替a、b。也就是说，将求gcd(a,b)问题转化成求gcd(x,y)。然后，一步步地使y减少，但却保持gcd不变。那末我们始终有

$$\gcd(x,y)=\gcd(a,b)$$