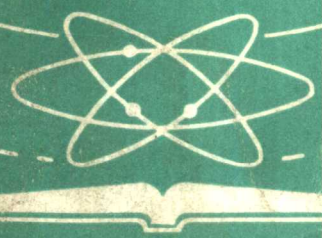
高等学校教材

程序设计方法学基础

中国人民解放军国防科学技术大学

陈火旺　罗朝晖　马庆鸣编著



湖南科学技术出版社

高等学校教材

程序设计方法学基础

中国人民解放军国防科学技术大学
陈火旺　罗朝晖　马庆鸣编著

湖南科学技术出版社

内 容 简 介

本书讨论程序设计方法学的一般理论基础以及程序设计的基本原则和方法。全书包括三部分。第一部分：程序逻辑基础（最弱前置谓词，Hoare逻辑和时态逻辑）。第二部分：模型的建立与程序开发(维也纳形式化开发方法——VDM)。第三部分：抽象数据类型的规范和大型程序的模块组装。本书集中了七十年代中期以来一些最重要的研究成果。可作为高等理、工科院校计算机科学、计算机软件专业高年级本科生或研究生教材，也可作为专业教师、软件工作人员的参考书。

高等学校教材

程序设计方法学基础

中国人民解放军国防科学技术大学

陈火旺　罗朝晖　马庆鸣编著

责任编辑：周翰宗

*

湖南科学技术出版社出版

（长沙市展览馆路3号）

湖南省新华书店发行　湖南省新华印刷二厂印刷

*

1987年4月第1版第1次印刷

开本：787×1092毫米　1/16　印张：15.25　字数：379,000

印数：1—6,500

ISBN 7—5357—0105—1／TP・2

统一书号：15204・196　定价：3.70元

87年秋理工《3124—6》

出版说明

根据国务院关于高等学校教材工作分工的规定，我部承担了全国高等学校工科电子类专业课教材的编审、出版的组织工作。从一九七七年底到一九八二年初，由于各有关院校，特别是参与编审工作的广大教师的努力和有关出版社的紧密配合，共编审出版了教材159种。

为了使工科电子类专业教材能更好地适应社会主义现代化建设培养人才的需要，反映国内外电子科学技术水平，达到“打好基础、精选内容、逐步更新、利于教学”的要求，在总结第一轮教材编审出版工作经验的基础上，电子工业部于一九八二年先后成立了高等学校《无线电技术与信息系统》、《电磁场与微波技术》、《电子材料与固体器件》、《电子物理与器件》、《电子机械》、《计算机与自动控制》，以及中等专业学校《电子类专业》、《电子机械类专业》等共八个教材编审委员会，作为教材工作方面的一个经常性的业务指导机构。并制定了一九八二～一九八五年教材编审出版规划，列入规划的教材，教学参考书、实验指导书等217种选题。在努力提高教材质量，适当增加教材品种的思想指导下，这一批教材的编审工作由编审委员会直接组织进行。

这一批教材的书稿，主要是从通过教学实践、师生反映较好的讲义中评选择优和从第一轮较好的教材中修编产生出来的。广大编审者，各编审委员会和有关出版社都为保证和提高教材质量作出了努力。

这一批教材，分别由电子工业出版社、国防工业出版社、上海科学技术出版社、西北电讯工程学院出版社、湖南科学技术出版社、江苏科学技术出版社、黑龙江科学技术出版社和天津科学技术出版社承担出版工作。

限于水平和经验，这一批教材的编审出版工作肯定还会有许多缺点和不足之处，希望使用教材的单位、广大教师和同学积极提出批评建议，共同为提高工科电子类专业教材的质量而努力。

电子工业部教材办公室

目　次

导　论

本书是从方法论角度来研究程序设计的。我们首先注重程序设计的理论基础，然后在此基础上研究程序设计的基本原则和方法。这是一门关于程序设计的科学。我们并不试图从细节上讨论一些具体问题的程序设计方法，这些在计算导论、高级语言、数据结构、编译方法、数据库、操作系统和结构程序设计等课程中大家已接触过很多了。我们希望通过这门课程来提高我们大家——未来的程序员——在程序设计方面的修养和素质。因此，我们把注意力集中于更一般化、更理论化的"方法论"（或"方法学"），而不是具体的"技巧"或"方法"。

以前不少人把程序设计看成是一种单纯技巧性的活动，甚至于把"程序员"的工作看作是一种"下贱的职业"[6]。社会实践正在改变着人们的这种偏见。越来越多的人认识到，与其它技术科学一样，程序设计也有它自身的一套规律、原理和方法。二十多年来许多计算机科学家在这方面作出了卓著的贡献。今天，"程序设计方法学"已成为计算机科学的重要组成部分，是软件工程学的重要柱石。方法学的研究成果表明，程序员的职责不再只是单纯编制程序，而且还是应用科学理论（尤其是数学理论）创造新方法与工具，架设沟通理论与工程之间的桥梁。程序员可以在这广阔的领域里施展他的创造性智慧与才能。在这里，我们着意为"程序员"的职业"正名"。

这篇导论分两节。第一节介绍本书基本结构。第二节从某一侧面介绍程序设计方法学的简单历史。

0.1 本书的结构

本书包括三部分。第一部分：程序逻辑。第二部分：模型的建立与程序开发。第三部分：抽象数据类型的规范和大型程序的模块组装。

第一部分（程序逻辑基础）介绍基于最弱前置谓词和循环不变式的程序语义定义和程序正确性证明方法，Hoare公理方法，以及基于时态逻辑的并行程序性质的描述与验证技术。这一部分是基础性的，适合于对小型程序的描述（规范）、分析和论证。

第二部分（模型的建立与程序开发）介绍以VDM（维也纳开发方法）为背景的形式化软件开发方法，在多值逻辑的基础上，讨论了模型抽象与规范描述，以及模型的分解、细化和程序开发。这一部分是面向大型程序设计的形式化方法。

第三部分也是面向大型程序设计的。根据"分而治之"的原则，在讨论模块概念（如Ada的"程序包"）的基础上引伸讨论了抽象数据类型的概念，研究了抽象数据类型的代数规范技术和代数语义方法，以及通过Pebble语言介绍了大型程序的模块组装技术。

第一部分共有六章。第一章引入了用一对谓词（前置谓词和后置谓词）描述（串行）程序的

规范，直观地介绍了有关程序正确性证明的基本概念。第二章应用最弱前置谓词与循环不变式定义了一个小语言$\mathcal{L}$的语义。第三章以语言$\mathcal{L}$为工具讨论面向目标的程序设计方法。第四章简介公理学方法，Hoare系统。第五章讨论时态逻辑的一般概念、并行程序模型以及程序性质的时态逻辑描述。第六章给出并行程序的时态逻辑证明系统。

第二部分共有三章。第七章在指出模型抽象与软件开发过程的关系的基础上给出了偏函数逻辑(一种三值逻辑)的一般基础。第八章讨论了运算抽象和数据抽象的方法。第九章研究模型具体化与程序开发：数据精化和运算分解。

第三部分共有四章。第十章介绍大型程序的模块化方法，Ada的程序包和抽象数据类型的概念。第十一章讨论抽象数据类型的代数规范技术。第十二章研究抽象数据类型的代数语义，给出了初始语义和终止语义的模型。第十三章通过Pebble语言介绍大型程序的模块组装技术。

我们选择上述材料的原则是：基础性强、相对稳定、比较成熟。程序设计方法学是一门发展中的技术科学，它远没有象其它古老技术科学(如机械学、电气学等)那样成熟。在某种意义上说，它的研究对象、范围迄今尚无定论。本书只编录那些我们认为比较重要的材料，不敢求全。考虑到这本书主要是面向大学生的，有些重要材料(如指称语义、结构操作语义或非过程式的程序设计风格等)也未列入，这些最好放到研究生阶段。

这本教程是准备80学时讲授用的。第一部分可安排30学时，其它两部分各25学时。有的章节作业量较大。如果时间紧的话，打星号的章节可以不讲。

本书的基本材料主要取自：

① Gries, D. The Science of Programming. Springer—Verlag, 1981.

② Loeckx, J. et al. The Foundation of Program Verification. John wiley & Sons, 1984.

③ Manna, Z., A. Pnueli. Verification of Concurrent Programs, Part I, The Temporal framework. Tech. Rept. of Dept. of Comp. Sci., Stanford Univ. Report No. STAN-CS, 1981. Part II: Temporal Proof Principles. Report No. STAN-CS-81-843. Part III: A Temporal Proof System. Report No. STAN-CS, 1982.

④ C. B. Jones, Software Development: A Rigorous Approach, Prentice-Hall International, London, 1980

⑤ C. B. Jones, "Specification and Design of (Parallel) Programs", IFIP83, 1983

⑥ H. Barringer, J. H. Chang, and CB. Jones, "A Logic Covering Undefinedness in Program Proofs", Acta Informatica 21, 1984

⑦ C. B. Jones, "Systematic Program Development", Software Development Techniques (eds. N. Gehani and A. D. McGettrick), 1985

⑧ D. Bjφner, C. B. Jones(edc.), The Vienna Development Method: The Meta-Language, LNCS61, 1978

⑨ Barnes, J. G. P., Programming in Ada, Addison-Wesley, 1982.

⑩ Burstall, R. M. and Lampson, B., A Kernel Language for Abstract Data Types and Module, Symposium on Semantics of Data Types, Sophia Antipolis, G. kahn et al. eds., LNCS 173, springer, 1984, 1—50

⑪ Burstall, R. M., Programming with Modules as Typed Functional Programming, Proc. of Int. Conf. on Fifth Generation Computer Systems, ICOT, 1984, 103—112.

⑫ Gugoen, J. A., Thatcher, J. W. and Wagner, E. W., An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types, in Current Trends in Programming Methodology IV, R. T. Yeh, ed., Prentice-Hall, 1978, 80—144

⑬ Guttag, J. V., Horowitz, E. and Musser, D. R., The Design of Data Type Specifications, in Current Trends in Programming Methodology IV, R. T. Yeh, ed., Prentice-Hall, 1978, 60—79.

0.2 程序设计方法学发展回顾

六十年代的中、后期，计算机应用已相当广泛，特别在军事领域更为普遍。计算机的软、硬系统也空前庞大与复杂。而那时的软件设计技术却远远落后于这种发展需要。人们也没有认识到从宏观上对程序设计方法进行研究的重要性。许多人只满足于写出可运行的程序，"不拘一格"，将就凑合。因此，许多大型软件常常质量低劣，可靠性不高，可维性极差，但又价格昂贵，供不应求。这种严重的局面阻碍了计算机和计算机应用的发展，成为当年所谓的"软件危机"。

1968年，北大西洋公约组织(NATO)在西德召开了一次软件工程会议[7]，分析了危机的局面，研究了问题的根源，第一次提出了用工程学的办法解决软件的研制和生产问题。这次会议在软件发展史上可算得是一个重要的里程碑。1969年IFIP(国际信息处理协会)即成立了"程序设计方法学工作组"——WG2.3，云集了许多当代著名的计算机科学家，来专门研究程序设计方法学，这个国际机构对尔后程序设计方法学的发展起了很大的促进作用。

在开创程序设计方法学的工作上，1968年Dijkstra的goto有害论[8]打响了第一炮。这是对传统方法的挑战。在当时就像捅了马蜂窝一样，引起了轩然大波。他指出，goto是程序复杂性难以控制的主要原因。过后不久，他在他的著名论文"结构程序设计"[9]中提出了结构设计的原则与方法。他建议程序设计只使用几个结构良好的语句(不包含goto)，希望通过程序的静态结构上的良好性(即良构性)保证程序的动态运行的正确性。在他的论文中他还给出了关于正确性的一系列原则。

几乎与此同时，另一位著名计算机科学家Wirth在他的论文"自顶而下逐步求精"[10]中提出了用"自顶而下"、"分而治之"的原则进行大型程序设计的方法。其主要思想是从所欲求解的原问题出发，运用科学抽象的办法，把它分解成若干相对独立的小问题，依次细分，直至各个小问题获得解决为止。

这些思想对七十年代初期程序设计方法学的研究有着深刻的影响。形成了那个时期的主要研究方向和基本研究内容。

与"结构程序设计"并行发展的是关于"程序正确性证明"的研究方向。这是另一个引人注目的课题。并且这两个课题之间有着甚为密切的关系。

1967年Floyd提出用断言方法[11]证明框图程序的正确性。他在框图的每条边上附上一个断言(谓词公式)，其意义是，每当程序执行到达这条边时，此断言应为真。对于循环，即框图上的一个回路，他定义了一个切点(cut point)，并在切点上置一个断言ρ，然后他证明，若回路开始执行时切点上的ρ为真，那末，当循环回复到该点时ρ将仍然为真。循环不变式的思想就是这个时候产生的，断言ρ就是循环不变式。

1969年Hoare在Floyd的基础上，定义了一个小语言和一个逻辑系统[12]。此逻辑系统含有程序公理和推导规则，目的在于证明程序的部分正确性。此系统是一阶谓词逻辑的扩充。这就是著名的Hoare逻辑。Hoare是第一个从正确性证明角度定义程序语言的，这对形式语义学的研究有着重要的影响。他的工作也为公理学语义的研究奠定了基础。

七十年代上半叶，公理学语义的研究引起了不少人的兴趣。人们企图把种种基本程序结构公理化，从赋值语句到各种形式的循环，从过程调用到伙伴程序(coroutine)，甚至于goto语句也被公理化。1973年，Hoare和Wirth把Pascal的大部分公理化[13]。1975年，一个基于公

理和推导规则的自动验证系统首次出现[14]。1979年出现了用公理化思想定义的程序设计语言——Euclid[15]。

1973年，Manna把部分正确性证明和终止性证明归于一体[16]。1976年Dijkstra提出了最弱前置谓词和谓词转换器的概念[17]。1980年Gries[1]综合了以谓词演算为基础的证明系统，称为"程序设计科学"，首次把程序设计从经验、技术升华为"科学"。这些都是研究程序正确性证明有意义的发展。

为了验证并行程序的正确性，从1974年以来，不少人把注意力集中于模态逻辑[20]。旨在描述并行程序性质的模态逻辑系统的研究，如时态逻辑[3][18][19]，是多年来一直倍受关注的新课题。人们发现，用模态算子"必然"和"可能"可以比较方便地刻画诸如安全性、事件性和事件优先性等程序性质。而它们有些用普通逻辑是相当困难的。

程序正确性证明从一开始就争论纷纷。有些人反对，不少人怀疑。主要理由有二：一是形式证明太复杂。那怕是一个正确性显而易见的小程序也得花九牛二虎之力证明大半天，人们无法接受。并且，谁又知道，证明本身是否无误呢！其次是，程序写好之后再搞证明不是"马后炮"吗？如果错误已经铸成，证明何能补救？著名计算机科学家Perlis等人在1979年写了一篇重要论文[21]，列举很多事实来批评程序正确性证明是不切合实际的。他的指出，数学证明是一个社会实践过程，历史上不乏那种数学"证明"，它们在若干年甚至几十年之后才被发现，原来搞错了。数学定理证明如此，程序正确性证明当然也不能例外。针对上述第二条理由，后来许多人倾向于主张，边写程序边考虑证明。即程序设计与正确性证明同时并行考虑。例如，1980年Gries的"程序设计科学"就是持这种观点的。

早在六十年代中期就开始的形式语义学的研究一直是计算机科学的一个极其重要的研究方向。七十年代这方面的工作有很大的进展。这对程序设计方法学的研究有深刻的影响。

以Strachy和Scott为代表的指称语义学[22]的工作在七十年代有了新的突破。指称语义是建立在完全偏序的不动点理论的基础上，通过语法、语义方程，建立语法范畴与数学对象的对应关系。著名的形式化软件开发方法VDM(维也纳开发方法)[23]的理论背景就是指称语义。Scott的"标准"语义[22]形式化开发技术更是指称语义的直接应用。

基于抽象数据类型，运用泛代数、范畴论的代数语义学十多年来也有重要的发展。最有代表性的是1975年以来Likov，Guttag等人的工作[24,25][26]。抽象数据类型的代数规范是通过给出产生类型客体(元素)的构造算子和有关运算的公理来描述的。在论证这种规范满足协调性和完全性的基础上，通过寻找适当的模型代数，可以定义一个抽象类型的不同层次的语义，如初始语义、终止语义等。然后就可以用普通的代数方法论证规范的正确性和实现的正确性。

古老的操作语义学在六十年代Mccarthy[27]和Landin[28]等人的卓越工作基础上，七十年代末以来又重新引人注目。Plotkin[29]等人在结构操作方面的研究开创了操作语义学的新局面。现在人们正基于这一理论开展形式化软件开发技术的探讨。

在形式语义学基础上，从形式规范到程序，或从程序到程序的推导、变换技术是七十年代中期以来在程序设计方法学研究方面的重要工作。程序综合和递归程序变换是这一时期最有意义的成果。如Burstall和Darlington的"递归程序变换系统"[30]，Bauer的"逐次变换的程序开发系统"[31]都是在这一时期完成的。

形式化软件开发方法的研究是程序设计方法学的主要组成部分。目前影响较大并且已付诸实际应用的首数"维也纳开发方法"(VDM)。它是由IBM维也纳研究所创造的(1973)。它

采用了许多六十年代和七十年代发展起来的形式技术和数学方法。VDM方法已经在程序设计语言，数据库，操作系统，办公自动化和一些特殊应用系统等领域中得到应用。有些应用已推广到工业部门。据报导，用VDM方法开发一个Ada编译程序比用传统方法在效率上（依人/年计）要提高一倍。VDM是一种面向模型的软件开发方法，通过建立模型，实行模型变换和细化，最后实现问题解。但开发过程目前尚欠合适的支撑系统。

大型软件系统的复杂性使得以“模块”为中心的大型程序设计方法成为七十年代程序设计方法学的一个十分重要的研究课题。这一课题与程序正确性证明，形式语义的研究又密切相关。面临复杂系统的大型程序设计，原先的“简单类型”，“子程序”和“过程”这些概念已显得不足，需要一种表示能力更强、更灵活，结构更清晰，关系更简单的“程序单位”。许多人从各个不同的侧面把“模块”的概念精确化和理论化，而这其中Parnas和Morris等人在“抽象数据类型”方面的工作最有影响[32][33]。抽象数据类型是程序设计方法中一种极为重要的方法，人们把它誉为是程序设计方法学发展史中的一个新的里程碑。

现在许多程序设计语言中引进了一些高级措施，以利程序员能在更高的抽象级上进行形式思维。而这一点对于研究程序设计方法来说是十分本质的。例如，CLU,Mesa, Modula2, Ada, ML和Pebble中的“族”，“模块”，“色”等等，都是直观概念“模块”的延伸、发展和理论化。应用它们，程序员能够更好地发挥抽象思维的才智，降低程序复杂性，从而更有把握地保证大型程序的正确性。“抽象数据类型”在程序语言中是继“运算抽象”和“控制抽象”之后，把数据与运算统一于一体的一种抽象。通过这种抽象使我们能够把现代的数学理论更好地应用于软件设计方法的研究。

近年来Burstall等人采用构造性数学的“相关类型”概念，在抽象模块程序基础上提出了带类型的泛函程序方法[34]，以支持大型程序设计。这种方法把一个含抽象模块的大型程序表示成一个表达式，而表达式中的算符就是模块。这是抽象化，形式化方法的很好应用。

最后我们想提一下，逻辑程序设计和函数程序设计代表着一种新的研究方向。七十年代中期发展起来的Prolog[35]是以谓词逻辑的子集（Horn子句）为基础的一种形式系统。在这种系统中，编制程序和描述程序性质是一致的。一个Prolog程序的执行过程就是执行逻辑上消解算法的过程。由于Prolog的基础与关系数据库甚相似，因此，它将是一种研究专家系统和人工智能的有力工具。

Backus在1978年提出的，旨在建立一种新的计算体制，以克服Von Neumann计算体制的局限性的函数程序概念[36]，对于程序设计和计算机体系研究（如数据流机）有着重大的影响。在这种体制中，程序设计的过程就是构造函数的过程。因而，程序正确性证明可用代数演算的方式来实现。

有些人预言，逻辑语言Prolog和函数语言可能成为下一代计算机的核心语言。因此，使用这类语言的程序设计方法可能成为一个非常重要的研究课题。

1985年丹麦计算机科学家Bjørner在其论文“信息学——一种新宇宙观”[37]中预言，从科学发展的角度看，我们正从“信息技术 时代”转变到“信息科学时代”。计算科学（Computation）将是信息科学的主要组成部分。所谓“计算科学”，他认为应包括三个方面，即，计算机科学，程序设计方法学和计算工程。程序设计方法学，也称程序设计科学，是研究和创造构造程序的过程的学问，是研究关于问题分析，环境模拟，概念获取，需求定义（包括数据和处理过程的描述）以及把这种描述变换、细化和编码成为机器可接受的表示的一般方法。

第一部分

程序逻辑基础

这部分将讨论串行程序和并行程序的程序性质的描述和验证。前四章讨论串行程序，后两章讨论并行程序。

对于串行程序，我们将用一对谓词（前置条件和结果条件）描述程序的功能，称为程序规范。运用最弱前置谓词的概念，讨论程序正确性证明的方法。然后给出一些原则和方法，研究如何使用这些原则和方法构造符合规范要求的正确程序。

对于并行程序，我们将以时态逻辑为工具，讨论并行程序的一系列的重要性质，如不变性（安全性）、活动性和优先性等等。然后给出证明系统和证明原理。

第一章给出了（串行）程序的规范描述，定义了程序执行状态，直观地介绍了程序正确性证明的要旨。第二章引进了最弱前置谓词的概念，并用它定义了一个小语言的语义。第三章以这个小语言为工具讨论程序设计的基本原则和方法。第四章介绍了Hoare的公理化证明系统。

第五章介绍了时态逻辑的一般概念，给出了共享存贮器的并行程序模型，然后讨论了并行程序性质的时态逻辑描述。第六章引进了时态逻辑证明系统，然后讨论证明原理。

第一章　程序规范与程序正确性

在进行程序设计之前，第一步必须明确“做什么”。所谓“做什么”是指对所欲求解的问题的描述。这种描述是程序设计的依据。关于“做什么”的描述通常称为**程序规范**(Program specification)。我们这里所说的程序规范仅指功能描述，不包括诸如处理速度，执行时间、响应周期等与时间有关的性能指标。程序规范通常是软件工程第一阶段——需求分析的结果。

在工程上，现今多数是用自然语言（如汉语、英语）辅以数学公式来表示程序规范的。由于自然语言有二义性，这种表示常常不够准确。但若规范不准确，或有纰漏，则按此设计的程序就不可避免含有错误，这将导致工程后阶段的困难，甚至失败。因此，越来越多的人探索用形式化的方法描述程序规范。甚至试图建立从形式规范到生成程序的转换规则。下面将给出的规范描述虽不是完全形式化的，但还是比较形式的。我们将采用清晰、无二义的断言（布尔表达式）来表示规范，描述程序性质，进行程序正确性验证，以及讨论由断言构造程序的一些基本方法。

1.1　断言与规范

从逻辑上说，断言就是关于事物性质的陈述，这种陈述可真、可假。例如：“3是个质数”是个断言，这个断言是“真”的；而“4是个质数”也是个断言，但这个断言是“假”的。我们说“x是个质数”也是个断言，但这仅是一种断言形式，也称谓词，其真假值依变量x所取值之不同而改变。以后我们用true表示“真”，用false表示“假”。

当断言用作程序注解或作为正确性命题的一部分时，常用花括号括起来。例如，{y是m，n的最大公约数}。也通常用特定符号名表示一些具体函数或关系，例如，用gcd(m，n)表示“m，n的最大公约数”。因而断言{y是m，n的最大公约数}就可形式地表示成{y = gcd(m，n)}。

使用逻辑算符可以从简单断言构成复合断言。本书所使用的逻辑运算符号为

¬ （非）	⇒ （蕴涵）	∃ （存在）
∧ （与）	≡ （等价）	∀ （全称）
∨ （或）		

其它某些符号将在引用时再定义。由于假定本书的读者已具有普通逻辑的知识基础，因此，一般的演算规则就不再一一介绍了。

假若我们要写一个计算商和余数的程序。这个程序的规范可以表示为：“设被除数x_1是个

非负整数，除数x_2是个正整数，计算x_1除以x_2的商y_1和余数y_2”。又假定我们所处理的对象都是整数，则这个规范可等价地表示为：“初始条件：$\{x_1 \geqslant 0 \wedge x_2 > 0\}$，计算满足$\{x_1 = x_2 * y_1 + y_2 \wedge 0 \leqslant y_2 < x_2\}$的$y_1$和$y_2$”。这个陈述中含有两个断言，前一个是初始断言，后一个是结果断言。

一般而言，一个程序**规范**可表示为由两个谓词构成的二元组(φ, ψ)。其中φ描述了所欲求解问题必须满足的初始条件，这个条件限定了输入参数的性质，称为**初始断言**或**前置断言**。断言ψ描述了问题的最终解必须具备的性质，称为**结果断言**，或**后置断言**。

例如，对于前例，若令φ为“$x_1 \geqslant 0 \wedge x_2 > 0$”；$\psi$为“$x_1 = x_2 * y_1 + y_2 \wedge 0 \leqslant y_2 < x_2$”。该问题的规范即为$(\varphi, \psi)$。

所谓程序断言是对程序性质的陈述。最重要的一个程序断言形如

$$\{\varphi\}S\{\psi\},$$

其中φ和ψ是两个谓词，它们联合起来构成一个规范(φ, ψ)，S是一个程序（或语句）。φ称S的前置断言，ψ称S的结果断言。断言$\{\varphi\}S\{\psi\}$称为S关于(φ, ψ)的**正确性断言**。它的意义为：

“若S开始执行时φ为真，则S的执行必终止且终止时ψ为真”。

因此，断言$\{\varphi\}S\{\psi\}$为真意味着，S是满足规范(φ, ψ)的一个程序。或说，程序S是规范(φ, ψ)的一个**实现**。

程序设计的任务是：对给定的规范(φ, ψ)，构造一个程序S，使得断言$\{\varphi\}S\{\psi\}$为真。

对于前述求商和余数的例示来说，现代多数计算机都具备这种指令，至少在高级语言中都具备相应的标准函数，因此，用不着编制什么程序。但为了解释断言在程序注解方面的作用，我们采用减法运算代替整除运算，这样可以构造如下的一个小小程序S：

```
y2:=x1; y1=0;
while  y2≥x2 do
begin
y2:=y2-x2; y1=y1+1
end;
```

对于这个程序S，为了表明它的正确性（即断言$\{\varphi\}S\{\psi\}$的真确性），我们将把断言（用花括号括起来）作为注解插在程序中间的某些位置上，逐句逐句加以验证。我们可以设想编译程序将对这些断言产生相应的检查码。在程序执行中每当控制流程到达这些位置时，相应的断言即被检查。若为真，程序继续正常执行；若为假，卸出程序变量的现行值并作流产处理。这样做对于程序调试是极有好处的。一旦正确性有了保证，为了提高目标程序的执行效率，可令编译程序重新产生不含断言检查的代码。上述小程序插进适当的断言后即变为：

```
{0≤x1∧0<x2}——前置断言                                    (1)
y2:=x1; y1:=0;                                          (2)
{0≤y2∧0<x2∧x1=x2*y1+y2}                                 (3)
Whiley2≥x2  do                                          (4)
  begin{0≤y2∧0<x2≤y2∧x1=x2*y1+y2}                       (5)
    y2:=y2-x2; y1:=y1+1;                                (6)
    {0≤y2∧0<x2∧x1=x2*y1+y2}                             (7)
  end;                                                  (8)
{0≤y2<x2∧x1=x2*y1+y2}——结果断言                          (9)
```

其中第(1) 行和第(9) 行上的断言分别为程序的前置断言和结果断言，即，程序是在断言(1)为真的条件下执行的，终止时结果断言(9) 必须为真。这是程序规范的要求。

这个程序的核心部分是一个循环。描述循环性质的最好办法是寻找一个断言，它在循环开始时为真，而且每当循环体执行一轮后它仍然保持为真。稍经思考后我们可能发现，这个断言应为

$\{0 \leqslant y_2 \wedge 0 < x_2 \wedge x_1 = x_2 * y_1 + y_2\}$,

这就是出现在第(3) 行和第(7) 行上的断言。这个断言通常称为循环不变式。第(5) 行上的断言是由这个不变式加上循环条件$\{y_2 \geqslant x_2\}$并合而成的。

现在让我们跟踪这个程序。在前置条件(1) 为真的前提下，经第(2) 行的赋值句执行后，第(3)行上断言显然是真的。第(5)行上的断言只有在循环条件$\{y_2 \geqslant x_2\}$为真的情况下才可能被检查，因而它的真确性也是显然的。第(6) 行的赋值句改变了y_2和y_1的值，由于对y_2，y_1的老值断言(5) 是成立的，于是，不难直观地检验对于它们的新值断言(7) 也必将成立。当第(4) 行的循环条件$y_2 \geqslant x_2$为假时循环即告终止，此时由$\{y_2 < x_2\}$和断言(7)立即可得断言(9)，即整个程序的结果断言。从而说明上述的程序是正确的。

现在我们面临着许多问题，诸如，如何构造断言使它们能准确地反映不同位置上的程序性质？有了断言之后，如何证明它们的正确性？尤其是，是否有准则，用它们可以从规范(φ, ψ)构造出程序S，使得断言$\{\varphi\}S\{\psi\}$为真？我们希望从规范(φ, ψ)出发，边构造程序边产生中间断言，并证明断言的正确性，直至最后的程序S满足$\{\varphi\}S\{\psi\}$。要回答上述这些问题需要逻辑，需要一种关于程序设计的逻辑。这种逻辑是在普通逻辑基础上建立的，这就是后续章节将要讨论的问题。

1.2 程序的执行状态

程序的执行可视为一系列的“状态”变迁。程序执行中某一时刻的状态是指在该时刻各程序变量所持的值。譬如说，在状态σ下，整型变量m持值3而n持值4，从机器角度而言，当机器处于状态σ时，名为m和n的存贮单元分别含值3和4，因此，状态σ可以看成是从程序变量的标识符集到值域的函数。这种函数关系可以用一个有限集表示，例如，{(m, 3), (n, 4)}。把状态看成函数就可以用通常的函数引用来表示某个程序变量在某种状态下的值，例如，σ(m) = 3意味着在状态σ下m的值为3。

把状态作为函数只对那样的一种程序语言是有效的，即用该种语言编写程序时，任何不具初值的标识符会自动地按其类型赋予一个特定初值。例如，Similar67和Pascal就是这样的程序语言。但有些语言，如Fortran，Algol60却并不是这样的。用这些语言编写的程序不自动对不具初值的标识符赋特定值，这些标识符只有在有关的赋值语句或输入语句执行之后才持有值。在这种情况下就存在这样的状态，在此状态下有些标识符的值没有定义。因此，一般而言，把状态定义为部分函数更为恰当一些。我们用“⊥”表示“无定义”，σ(k) = ⊥意味着在状态σ下k的值没有定义。(有些语言，同一标识符在不同“环境”中含义不同。)

显然，在程序运行时，程序中表达式的值、断言的真假都是依状态的变化而变化的。例如，令σ = {(m, 3), (n, 4), (b, false)}，那末，表达式m<n∨b在状态σ下为真，而m<n∧b在状态σ下为假。

现在我们把状态函数σ扩展到定义一个表达式在某一状态下的值。对于任何状态σ和表达