《程序设计方法学》课程教学设计和教学模式研究

1 概述

程序设计方法学(Programming Methodology)是讨论程序的性质以及程序设计的理论和方法的一门学科， 是研究和构造程序的过程的学问，是研究关于问题的分析，环境的模拟，概念的获取，需求定义的描述， 以及把这种描述变换细化和编码成机器可以接受的表示的一般的方法。

程序设计方法学有两种含义:一种是以程序设计方法为研究对象的学科，它不仅研究各种具体的方法，而且着重研究各种具体方法的共性，涉及规范的全局性方法，以及这些方法的显示背景和理论基础;另外一种含义是针对某一领域或某一领域的一类特定问题，所用的一整套特定程序设计方法所构成的体系。作为一门学科(第一种含义)，程序设计方法学可对程序设计人员选用具体的程序设计方法起指导作用，而具体的程序设计方法对程序设计工作的质量以及所设计出大程序的质量影响巨大。因此，对程序设计方法学的研究是非常重要的。作为一套完整特定的程序设计方法所构成的体系(第二种含义)，如逻辑式程序设计方法学、函数式程序设计方法学、对象式程序设计方法学等。它们有各自的利弊得失，与具体领域、具体问题以及具体环境相关。两种含义之间的关系是：第二种含义是第一种含义的基础，第一种含义是在第二种含义的基础上的总结、提高，并上升到原理、原则和理论的高度。这两种含义的程序设计方法学都非常重要。

任何设计活动都是在各种约束条件和相互矛盾的需求之间寻求一种平衡，程序设计也不例外。在计算机技术发展的早期，由于机器资源比较昂贵，程序的时间和空间代价往往是设计关心的主要因素；随着硬件技术的飞速发展和软件规模的日益庞大，程序的结构、可维护性、复用性、可扩展性等因素日益重要。另一方面，在计算机技术发展的早期，软件构造活动主要就是程序设计活动。但随着软件技术的发展，软件系统越来越复杂，逐渐分化出许多专用的软件系统，如操作系统、数据库系统、应用服务器，而且这些专用的软件系统愈来愈成为普遍的计算环境的一部分。这种情况下软件构造活动的内容越来越丰富，不再只是程序设计活动了，还包括数据库设计、用户界面设计、接口设计、通信协议设计和复杂的系统配置过程。程序设计(Programming)是指设计、编制、调试程序的方法和过程。它是目标明确的智力活动。由于程序是软件的本体，软件的质量主要通过程序的质量来体现的，在软件研究中，程序设计的工作非常重要，内容涉及到有关的基本概念、工具、方法以及方法学等。

《程序设计方法学》是计算机科学与技术专业的专业选修课。通过本课程的学习，学生可以掌握程序正确性证明的基本理论和方法。通过本课程的教学，使学生能运用所学到的程序规范、程序正确性证明理论、程序设计方法熟练地编写出具有良好结构的、规模较大的、正确性好的程序，为进一步学习软件形式化技术、今后从事高安全和可靠的计算机系统研究和开发工作打下良好的基础。按计算机专业 2014 版大纲要求，该课程设置有三个主要要求：掌握面向程序设计和验证目的的程序规范设计方法、掌握程序正确性证明的基本理论和方法、了解基本的程序语言范型和最新的程序设计和验证技术及其发展。

本课程是一门理论性很强的课程，要求学生具有较强的数理逻辑推导能力和抽象思维能力，需要培养学生利用数学的思想和方法去理解程序设计和验证，并用这一思想去指导软件的开发。本课程的重点和难点是让学生掌握利用数学方法和思想去研究和表达程序设计，这也是本课程的精髓所在。学生首先要转变对程序设计的理解，本课程的宗旨并不是要利用相关的程序设计语言来编写程序，而是利用数理逻辑以及形式化的方法从理论的角度表达和验证程序设计的思想和方法。

2 知识体系与接口

2.1 相关课程一览表

100384-5	高级语言程序设计	（第 1、2 学期）	4 学分
100386-7	高级语言程序设计实验	（第 1、2 学期）	2 学分
100388	离散数学	(第 3 学期)	3 学分
100391	软件开发方法	（第 4 学期）	2 学分
100435	软件开发方法课程设计	（第 4 学期）	1 学分
101031	程序设计方法学	（第 6 学期）	2 学分
101023	软件工程	（第 6 学期）	3 学分
100414	软件形式化技术	（第 7 学期）	2 学分

2.2 本课程知识点解析和教学要求

第一章 概述

1.1 程序设计方法学的产生

程序设计方法学的主体应用大致经历了：手工作坊式和程序设计→结构化程序设计方法→模块化程序设计方法→面向对象程序设计方法，此外，还有逻辑型程序设计方法、函数型程序设计方法、并行程序设计方法等。

1950 年代—1960 年代初，手工艺式的程序设计方法，高德纳把程序称为艺术品。

1960 年代末—1970 年代初，出现软件危机：一方面需要大量的软件系统，如操作系统、数据库管理系统；另一方面，软件研制周期长，可靠性差，维护困难。编程的重点：希望编写出的程序结构清晰、易阅读、易修改、易验证，即得到好结构的程序。

1968 年，北大西洋公约组织（NATO）在西德召开了第一次软件工程会议，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研制和生产的问题，本次会议可以算做是软件发展史上的一个重要的里程碑。

1969 年，国际信息处理协会（IFIP）成立了“程序设计方法学工作组”，专门研究程序设计方法学，程序设计从手工艺式向工程化的方法迈进。

1968 年，结构化程序设计方法的研究。Dijkstra 提出了“GOTO是有害的”，希望通过程序的静态结构的良好性保证程序的动态运行的正确性。

1969 年，Wirth 提出采用“ 自顶向下逐步求精、分而治之” 的原则进行大型程序的设计。其基本思想是：从欲求解的原问题出发，运用科学抽象的方法，把它分解成若干相对独立的小问题，依次细化，直至各个小问题获得解决为止。

1967 年，Floyd 提出用“ 断言法” 证明框图程序的正确性。

1969 年，Hoare 在 Floyd 的基础上，定义了一个小语言和一个逻辑系统。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，这就是著名的 Hoare 逻辑。他的工作为公理学语义的研究奠定了基础。

1973 年，Hoare 和 Wirth 把 PASCAL 语言的大部分公理化。

1975 年，一个基于公理和推导规则的自动验证系统首次出现。

1979 年，出现了用公理化思想定义的程序设计语言 Euclid。

1976 年，Dijkstra 提出了最弱前置谓词和谓词转换器的概念，用于进行程序的正确性证明和程序的形式化推导。

1980 年，D.Gries 综合了以谓词演算为基础的证明系统，称之为“程序设计科学”。首次把程序设计从经验、技术升华为科学。

1974 年，人们利用模态逻辑验证并行程序的正确性。

这一节重点介绍程序设计方法学产生的历史背景、程序设计思想的进步、程序设计方法学的研究内容、学

科定位、与其他课程之间的关联、研究和应用的意义等。

讨论： 程序设计方法学与结构程序设计的关系是什么？

1.2 程序设计验证的一般途径

这一节重点介绍程序设计方法学对软件的研制和维护的指导作用。软件工程要求程序设计规范化，建立新的原则和技术。而一种新的方法的出现，又要求制订出相应的规范。方法和工具是同一问题的两个侧面。工具的研究以方法学为基础，而工具的研制成功又会影响程序设计。程序设计方法学还涉及程序推导、程序综合、程序设计自动化研究、并发程序设计、分布式程序设计、函数式程序设计、语义学、程序逻辑、形式化规格说明和公理化系统等课题。以结构化程序设计语言中 goto 语句的处理、程序的结构、自顶向下逐步求精的程序设计方法为例，引导学生理解对程序设计验证方法和工具的应用重要性和一般途径。

1.3 数理逻辑基础

这一节简单回顾离散数学中数理逻辑部分的相关知识：要求掌握形式逻辑中命题的表示与计算、命题公式的变形与化简、等价命题证明、重言式证明和命题的推理。重点巩固推理规则（恒等式、永真蕴含式、代入规则、规则 P：引入前提、规则 T：应用永真蕴含推导过程）和不同的证明方法（证明前件为假的无义证明法、证明后件为真的平凡证明法、直接证明法、间接证明法、CP 规则、分情形证明、反证法、数学归纳法、归谬法）。教学中建议以用文本替换和数组的解释为例说明命题的表示和推理计算。本课程与离散数学课程在符号逻辑（证明论、模型论、递归论和公理化集合论）、命题演算和谓词演算知识点上存在必然联系。在教学中，需要结合起来学习和巩固所学知识点。

第二章 程序规范

2.1 程序规范的描述

对欲求解的问题的描述。关于“做什么”的描述通常称为程序规范（PS，Program Specification）。这里的 PS 仅指功能的描述，不包括诸如处理速度、执行时间、响应周期等与时间有关性能指标。PS 是软件工程的需求分析的结果。PS 的含义是映射，是输入到输出的映射，它反映了程序对数据的作用。从这个意义上而言，程序也是映射，是输入到计算的映射，即每一输入都对应一串计算步骤。进一步地：给出规范后，程序开发就是建立一个程序，使之计算刚好能实现规范的映射；程序验证是证明程序正确地实现了规范，即证明规范和已有程序之间的一致性。注意程序规范与程序代码规范不同，后者注意强调代码的整洁性、完整性、一致性和统一性，主要出于可读性和可维护性的要求。

课程作业（1）：若干个程序的程序规范描述，并用于课堂讨论。

2.2 断言与规范

本节主要介绍断言和程序规范的关系。要求学生能够根据程序验证要求写出相应的断言。

一般地，一个程序规范可表示为由两个谓词构成的二元组（P，Q）。其中，

P 描述了所欲求解的问题必须满足的初始条件，它限定了输入参数的性质，称为初始断言或前置断言；

Q 描述了问题的最终解必须满足的性质，称为结果断言或后置断言。

程序断言是对程序性质的陈述

最重要的一个程序断言为：{P}S{Q}　其中，（P，Q）是程序 S 的程序规范

S 是一个程序（或语句）

断言{P}S{Q}称为 S 关于（P，Q）的正确性断言

若，S 开始执行时 P 为真　则，S 的执行必终止，且终止时 Q 为真

在理解上述概念基础上，回答以下三个问题：（1）如何构造断言使它们能准确地反映不同位置上程序的性质？（2）有了断言，如何证明它们的正确性？（3）能否有准则，可以从规范（P，Q）构造出程序 S，使{P} S {Q}为真。

2.3 非形式化证明

程序正确性断言{P}S{Q}的意义："若 S 的执行开始于一个满足 P 的状态，那么这个执行必将在有限的时间内终止于一个满足 Q 的状态"。所谓一个状态时满足 P（或 Q），即指在此状态下 P（或 Q）为真。

设（P，Q）是一个规范，S 是依照这个规范要求设计的程序，且是由语句 s1，s2，…，sn 组成的一个枚举型程序。即其执行等于组成它的各个语句的逐一顺序的执行，其中的每个语句都只有一个入口和一个出口，且没有 GOTO 语句。

令 P1,Q1,P2,Q2,…,Pn,Qn 是 2n 个谓词，且 P=P1，Q=Qn。如果所有断言 {Pi} Si {Qi}，i=1，2，…,n,为真，并且每个蕴涵：Qi⇒Pi+1， i=1，2，…,n 成立，就称（P1，Q1），（P2，Q2），…，（Pn，Qn）是{P}S{Q}的一个证明。

用程序断言进行非形式化证明的实例解析。

第三章　结构化程序设计

3.1 结构化程序设计的综合描述

程序设计方法学是在结构化程序设计基础上逐步发展和完善起来的。1968 年 Dijkstra 在 ACM 通信上提出"GOTO 语句是有害的"，向传统的程序设计方法提出了挑战，引起人们对程序设计方法学的普遍重视。关于 GOTO 语句的争论。Dijkstra 的建议：GOTO 语句太容易把程序弄乱，应从高级语言中去除。通过采用 3 种基本结构就可以完成各种程序设计，使程序可以自顶向下阅读而不会返回。Eknuth 的观点：不加限制地使用 GOTO 语句，特别是使用回跳的 GOTO 语句，会使程序结构难于理解，应尽量避免。为提高程序执行效率，同时不破坏程序的良好结构，有控制地使用一些 GOTO 语句是必要。

3.2 定义结构化程序

本节主要介绍结构化程序设计的思维方法、7 种基本程序结构、流程图程序、基本程序、结构化程序。要求学生学会流程图程序的做法。

结构化程序设计的思维方法：自顶向下、逐步求精、不变式设计。

程序设计方法学有 7 种基本程序结构：函数、序列、If-then、If-then-else、Do-until、While-do、Do-while-do。

流程图程序：程序流程图也称程序框图，是用一种标准的图形助记符表示编程思路的辅助手段，是一个描述程序的控制流程和指令执行情况的有向图。将一个程序用流程图的形式表示，即为流程图程序。正规程序：一个流程图程序如果满足：（1）只有一个入口和一个出口（2）对于每一个结点，都有一个从入口到出口的通路（即每个结点都可达），则这个流程图程序称为正规程序。

基本程序：一个正规程序，如果不包含多于一个结点的正规子程序，则称为基本程序。基本程序是正规程序，基本程序可以包含正规子程序，基本程序包含的正规子程序只有一个结点。

结构化程序：一个基本程序的函数结点用另外一个基本程序替换，就产生一个新的正规程序，称为复合程序。用以构造程序的基本程序集合称为基集合，例如{序列,if-then-else,while-do}或{序列,if-then-else,do-until}等。由基本程序的一个固定的基集合，构造出来的复合程序称为结构化程序。

3.3 结构化定理

本节的重点是结构化定理、程序函数、程序的函数等价的概念和结构化定理的证明。要求学生掌握流程图程序转化为结构化程序的方法。

结构化定理:任意一个正规程序，都可以用{序列,if-then-else,while-do}三种基本控制结构来等价表示。

程序函数：已知一个正规程序 P，对于每一个初始输入数据 X，若程序终止，那么有确定的最终输出数据 Y。如果对于每一个给定的 X，值 Y 是唯一的，那么所有的有序对的集合 {（X,Y）}定义了一个函数，这个函数称为程序 P 的程序函数，记为[P]。

程序的函数等价：如果程序 P1 和 P2 具有相同的程序函数，则称它们是函数等价的。如果程序 P1 和 P2 具有相同的执行流程，则称它们是执行等价的。程序的执行等价蕴涵函数等价。

结构化定理证明：（1）对程序 P 中的函数和谓词结点编号，对每条射线赋上其所到达的最近的结点编号，若为出口线，则赋为 0；若为入口线则赋为第一个结点编号。（2）对程序 P 中每个函数和谓词结点增加计

数器 L，形成 gi（I＝1,2,…,n)。(3）利用已经得到的 gi（I＝1,2,…,n)，构造 while-do 循环，对 L 从 1 到 n 进行循环测试的嵌套 if-then-else。

课堂练习：将流程图程序转化为结构化程序，并进行讨论：结构化定理对结构化程序设计和验证的要求和意义。

第四章　最弱前置谓词和程序语言的语义

4.1 最弱前置谓词（Weakest Pre-predicate）的概念

定义：假定 S 是一个语句，Q 是一个谓词，它描述 S 执行后所确定的某种关系。从 S 和 Q 定义另外一个谓词，记为 wp(S,Q),它表示：“所有这样的状态的集合，S 从其中任一状态开始执行必将在有限的时间内终止于满足 Q 的状态”。我们称 wp(S,Q) 是语句 S 关于 Q 的最弱前置谓词。

说明为何要引入最弱前置谓词，它对程序验证的意义：

$$\{P\}\ S\ \{Q\} \quad \Leftrightarrow (P \Rightarrow wp(S,Q))$$

4.2 基本语句的语义

程序语言的基本语句语义：空语句（Skip)、赋值语句、选择语句、循环语句。

对于空语句：$wp(skip,Q) =_{df} Q$

对于赋值语句：$wp(\text{“y := e”},Q) =_{df}\ domain(e) \wedge Q_e^y$

对于顺序复合语句：$wp(\text{“s1;s2”},Q) =_{df} wp(\text{“s1”},wp(\text{“s2”},Q))$

对于选择语句：

$$wp(IF,Q) =_{df} domain(BB) \wedge BB \wedge c1 \Rightarrow wp(s1,Q) \wedge c2 \Rightarrow wp(s2,Q) \wedge \ldots \wedge cn \Rightarrow wp(sn,Q)$$

$$=_{df}\ domain(BB) \wedge (\exists j : 1 \le j \le n : cj) \wedge (\forall i : 1 \le i \le n : ci \Rightarrow wp(si,Q))$$

对于循环语句的语义：$wp(DO, Q) =_{df} (\exists k: k \ge 0 : H_k(Q))$　其中

$$H_k(Q) = H_0(Q) \vee wp(IF, H_{k-1}(Q)),\ k>0$$

定理 1（Theorem 1）　考虑 IF 命令，假设谓词 P 满足：

(1)　$P \Rightarrow BB$

(2)　$P \wedge ci \Rightarrow wp(si,Q),\ 0 \le i \le n$

则：　$P \Rightarrow$　wp（IF，Q）

课堂练习：引入最弱前置谓词，对几种基本程序语句的语义描述。

4.3 不变式和界函数

循环不变式：

用 ρ 作为循环 DO 对于 Q 的前置条件，它满足：

{ρ}
do BB → {BB ∧ ρ} IF {ρ}　od;
{¬BB∧ ρ }
{Q}

ρ 在循环的每轮迭代的执行前后均保持为真，因此称 ρ 为该循环的一个不变式（Invariant，简称 Inv.）

界函数 τ：

为说明循环的终止性，引入一个整型函数 τ，它是程序变量的函数，　用以指明循环至多执行 τ 次迭代。它满足：τ 的值随迭代次数的增加而递减，每次迭代 τ 的值至少递减 1，且: 只要能够继续迭代，　τ 的值就一定保持 τ>0。因此，只要表明存在这样的函数 τ，循环就必定能够终止，称 τ 为循环的界函数（Bound Function，简称 BF）

定理 2（不变式定理）　考虑一个 DO 循环，令谓词 ρ 满足：(1)　$\rho \wedge ci \Rightarrow wp(si, \rho),\ 1 \le i \le n$，令整数函数 τ 满足（ τ1 是新标识符）　(2)　$\rho \wedge BB \Rightarrow \tau > 0$ (3)　$\rho \wedge ci \Rightarrow wp(\text{“}\tau 1 := \tau ; si\text{”}, \tau < \tau 1),\ 1 \le i \le n$　则：　$\rho \Rightarrow wp(DO, \rho \wedge \neg BB)$

课堂讨论：结合实例说明不变式定理在循环语句验证中的作用和意义。

本节重点：关于{P}DO{Q}的验证项目：

根据定理 2，为证明{P}DO{Q}，关键在于寻找谓词 ρ 和界函数 τ，使得下面的项目一一成立：

ρ 在循环执行前为真，

对每个 i，1≤i ≤n， {ρ ∧ci} si { ρ}

ρ ∧ ¬ BB ⟹ Q

ρ ∧ BB ⟹ τ>0

对每个 i，1≤i≤n，{ρ ∧ci} τ1 := τ ; si”{τ<τ1}

课程作业（2）：用定理 2 形式化地证明若干个 DO 循环的验证项目。

第五章 程序设计验证的基本方法

重复递归引理内容：引理 1 while-do 的正确性定理：已知预期函数 f 和循环程序 P，while p do g ，则 f=[P]的充要条件是：对所有 X∈D（f）， 程序 P 终止，且 f=[if p then g；f]。引理 2 do-until 的正确性定理：已知函数 f 和循环程序 P：do g until p ,则 f=[P]的充要条件是：程序 P 终止，且 f=[g;if ～p then f]。引理 3 do-while-do 的正确性定理：已知函数 f 和循环程序 P：do1 g while p do2 h ,则 f=[P]的充要条件是：程序 P 终止，且 f=[g;if p then h;f]。

结构化程序正确性证明的代数方法：给定一个程序 P 的预期程序函数 f，若 X∈D（f），程序 P 是终止的，且通过正确性定理求解程序 P 的程序函数 f’，若与预期函数 f 相等，则得证。

结构化程序正确性证明步骤: （1）程序 P 是终止的；（2）通过正确性定理求解程序 P 的程序函数 f’，与预期函数 f 相等。对于相对简单直观的程序，可以直接使用代数方法计算程序函数。对于复杂的序列和条件程序,循环程序的证明，可以采取跟踪表方法求解程序函数。

循环不变式产生的方法: 对于程序部分正确性证明的不变式断言法，这一方法的关键是建立一个正确的不变式断言，对一般程序来说，不变式断言的建立主要依靠程序员对程序的理解，尚无系统的方法。但对结构化程序来说，如果已知它的程序函数，则可以根据不变式状态定理，来确定它的一个循环不变式。

不变式状态定理: 假设 f=[while p(x) do g(x)]，x0 是初始值，则循环不变式 q(x)为:f(x) = f(x0)。

不变式状态定理证明：（1）在第一次进入循环时， f(x0) = f(x0)，因此 q(x)成立。（2）试证假设在每一次进入循环前 q(x) 成立，即 f(x0)=f(x)，则执行循环后 q(x)也成立，即证明 p(x) ^ q(x) => q(g(x))=(f(g(x))=f(x0))，由 p(x)为真，以及正确性定理 f={(x,y)|p(x) —>y=f*g(x)|~p(x) —>y=x}可知，即 f(x0)=f(x)＝y=f*g(x)=f(g(x))。

5.1 程序设计的基本原则

Principle 1: 一个程序和它的正确性证明应同时设计，并且最好以考虑证明为先导。

Principle 2：使用理论来提供正确性的理解，而在适当的地方用常识和直观方法。但当出现困难和复杂情况时，仍然依据形式理论作保证。当然一个人只有既有常识又掌握理论才能有效地在形式化和常识之间进行协调。

Principle 3：要了解将用程序来处理的对象的性质。

Principle 4：不要忽视任何看起来十分明显的原则，只有通过有意识地运用这些原则才会获得成功。

Principle 5：认识一个原则和应用一个原则是两回事。

Principle 6:在着手编程之前，首先要弄清楚问题是要求“做什么”的，并将其抽象化为前置条件和后置条件，即给出精确的程序规范，且勿仓促编码。

Principle 7:程序设计是一种面向目标的设计活动。即程序设计是围绕着目标 Q 进行的，这意味着 Q 起着比 P 更重要的作用。

课堂讨论：证明与测试的区别？

调试的方法虽然可以帮助人们发现程序中的错误，但是却不能证明程序中没有错误，并且只能对很有限的几组初始值进行试算。它不能证明程序是正确的。程序正确性证明不仅可以证明顺序程序的正确性，而且利用这些方法还可以证明非正确性程序及并行程序的正确性。

学习和研究形式证明技术是培养能力的一种有效的途径。测试只能证明程序有错，并不能证明程序是正确的。

5.2 选择语句的设计

选择语句的设计策略：对于规范（P，Q），（1）寻找语句 s1，它执行后能使 Q 成立；（2）再寻找满足 P∧c1⇒wp(s1,Q)的条件 c1；（3）形成带卫哨语句 c1→ s1；（3）继续寻找带卫哨语句 c2→ s2,..., cn→ sn；（4）直到 P⇒c1∨c2 ∨... ∨cn 成立为止。

课堂练习：选择语句的实例设计。

5.3 循环语句的设计

循环语句的设计策略有两种：卫哨先行、走向终止。

对于卫哨先行策略：首先寻找循环条件卫哨 c，满足 ρ ∧¬c ⇒ Q; 然后形成循环体，使得界函数 τ 的值递减，并保持不变式 ρ 为真。

对于走向终止策略：首先寻找导致循环终止（即递减 τ）的语句 si，然后寻找保持不变式 ρ 的相应条件 ci 形成带哨语句 ci → si，重复这个过程，直到形成足够多的带哨语句足以证明 ρ ∧¬BB ⇒ Q 成立为止。

课堂练习：循环语句的实例设计。

5.4 循环不变式的设计

气球理论(The Balloon Theory)：把 Q 看成一个瘪了气的气球，逐渐放宽对 Q 的限制，即削弱 Q，即将气球吹大，直至包含循环的初始状态。吹气的过程是设计 ρ 的过程，而漏气的过程是设计循环体的过程。因此，将设计不变式的方法归结为削弱谓词 Q 的过程。

循环不变式的构造方法：

（1）削弱 Q 就 ρ：删除 Q 的一个合取分量，替换 Q 中的常量为变量，扩大 Q 中变量的变化范围。适合于输入变量的值不因程序的执行而变化的情况。

（2）联合 P 和 Q 就 ρ：适用于输入变量的值因程序的执行而变化的情况。

5.5 界函数的设计

一般地，通过分析循环不变式 ρ，能够较容易地找到界函数 τ。多数情况下，循环迭代是依照一个或几个变量的值的变化而进行的，此时，界函数可依照这些变量的变化范围和变化规律而设定。

界函数的设计策略：先用语言描述不变式和问题的一个简单的性质，然后将其形式化为数学表达式。对于任意可数的良序集，都可以找一个从该集到自然数集的一一对应的满射。所以在程序中找到一个良序集，如字典序，就可以通过这种映射得到界函数。

课程作业（3）：写出某一程序的规范，并设计其中的循环不变式、界函数，用不变式定理验证其正确性。

第六章 程序正确性证明

6.1 概述

（1）非形式化的程序验证是靠程序员阅读理解程序完成的。（2）程序测试是指测试者特意挑出一批输入数据，通过运行程序，检查每个输入数据所对应的运行结果是否符合预期要求。（3）Dijkstra 说过“程序测试只能证明程序有错，不能说明程序正确”。除非进行穷举行测试。（4）正确性证明是论证程序达到预期目的的一般性陈述，而该论证与程序输入数据的特定值无关，能够代表穷举性测试。

程序正确性理论：程序功能的精确描述（1）程序规约：对程序所实现功能的精确描述，由程序的前置断言和后置断言两部分组成。（2）前置断言：程序执行前的输入应满足的条件，又称为输入断言。（3）后置断言：程序执行后的输出应满足的条件，又称为输出断言。

程序正确性定义:衡量一个程序的正确性，主要看程序是否实现了问题所要求的功能。若程序实现了问题所要求的功能，则称它为正确的，否则是不正确的。

程序规范 Q{S}R 是一个逻辑表达式，其取值为真或假，其中取值为真的含义是指：给定一段程序 S，若程

序开始执行之前 Q 为真，S 的执行将终止，且终止时 R 为真，则称为 “程序 S，关于前置断言 Q 和后置断言 R 是完全正确的”。

部分正确：若对于每个使得 Q(i)为真，并且程序 S 计算终止的输入信息 i，R(i,S(i))都为真，则称程序 S 关于 Q 和 R 是部分正确的。

程序终止：若对于每个使得 Q(i)为真的输入 i，程序 S 的计算都终止，则称程序 S 关于 Q 是终止的。

完全正确：若对于每个使得 Q(i)为真，并且程序 S 的计算都将终止的输入信息 i，R(i,S(i))都为真，则称程序 S 关于 Q 和 R 是完全正确的。

一个程序的完全正确，等价于该程序是部分正确，同时又是终止的。

程序正确性的证明方法：（1）证明部分正确性的方法——A. Floyd 的不变式断言法；B. Manna 的子目标断言法；C. Hoare 的公理化方法。（2）终止性证明的方法——A. Floyd 的良序集方法；B. Knuth 的计数器方法；C.Manna 等人的不动点方法（3）完全正确性的方法——A. Hoare 公理化方法的推广；B. Burstall 的间发断言法；C. Dijkstra 的弱谓词变换方法以及强验证方法

部分正确性证明方法：不变式断言法、子目标断言法、公理化方法。

6.2 不变式断言法证明程序的部分正确性

本章重点内容：不变式断言法证明程序部分正确性证明步骤：（1）建立断言：建立程序的输入、输出断言，如果程序中有循环出现的话，在循环中选取一个断点，在断点处建立一个循环不变式断言。（2）建立检验条件，将程序分解为不同的通路，为每一个通路建立一个检验条件，该检验条件为如下形式： $I \wedge R \Rightarrow O$, 其中 I 为输入断言，R 为进入通路的条件，O 为输出断言。（3）证明检验条件：运用数学工具证明步骤 2 得到的所有检验条件，如果每一条通路检验条件都为真，则该程序为部分正确的。

课堂练习：用不变式断言法证明实例程序的部分正确性。讨论：如果得不到程序的正确性证明，如何进一步处理？

课堂讨论：美国斯坦福大学的 Z.Manna 教授在 Floyd 的不变式断言法的基础上，提出了部分正确性证明的子目标法。子目标法与不变式法的主要区别是什么？（1）在断言设置中，不变式法的断言描述程序处理过程中的中间变量与初始值之间的关系，而子目标断言中描述的是中间变量与终值间的关系；（2）其归纳推理方向不同。不变式断言法沿程序执行方向正向推理，而子目标法沿程序执行的反方向逆向归纳推理。

课程作业（3）：用不变式断言法法证明某一程序的部分正确性。

6.3 良序集方法证明程序的终止性

本章重点内容：用良序集方法证明程序终止性步骤：（1）选取一个点集去截断程序的各个循环部分，并在每个截断点 I 处建立中间断言 Qi (x,y)。（2）选取一个良序集(W, ≺)，并在每个截断点处定义一个终止表达式 E i (x,y)，表达式在 W 上取值。（3）证明对每一个从程序入口到截断点 j 的通路 aj 有 I(x) ^ Raj (x,y) => Qj(x,r(x,y))证明对每一个从截断点 i 到截断点 j 的通路 aij，有 Qi (x,y) ^ R a i j (x,y) => Q j (x,r a i j (x,y))即证明中间断言是“正确的”，是“良断言”。（4）证明 Qi (x,y) = > E i (x,y) ∈ W，即终止表达式是良函数。（5）证明对于每一个从截断点 i 到截断点 j 的通路 aij，有 Qi (x,y) ^ Raj (x,y) =>[E j (x,y) < E j (x,y)]。

课堂练习：用良序集方法证明实例程序的终止性。

课堂讨论 1：如果得不到程序的终止性证明，如何进一步处理？

课堂讨论 2：D.E.Kunth 于 1968 年提出了证明程序终止性的计数器方法。该方法直观易懂，其证明过程分为三步：（1） 为程序中的每个循环附加一个新的变量（如 i，j 通常被称为计数器），进入该循环前计数器置 0，每执行一次循环计数器累加 1；（2） 为每个循环设置一个中间断言，它表明相应的计数器不会超过某个固定的界限；（3）进一步证明第（2）步设置的中间断言是不变式断言。

课程作业（4）：用 Floyd 良序集方法证明某一程序的终止性。

6.4 Hoare 公理学方法

1969 年 C.A.R.Hoare 提出了一种新的程序验证的方法。使用了简洁的符号，提出了定义程序语义的公理系统。Hoare 系统是一个关于形如[P]S[Q]的断言的逻辑系统。包括 [P]S[Q] 部分正确性断言、{P}S{Q} 完全

正确性断言。这个方法是针对 WHILE 型程序提出的。

不变式演绎系统—Hoare 系统引理：

一个程序 S 关于其输入断言 P(x)和输出断言 Q(x,z)的部分正确性可以表示为: [P(x)] S [Q(x,z)]，因而，证明程序 S 的部分正确性,就可以归结为证明这一归纳表达式为真。

本章重点内容：Hoare 公理及其推理规则：

(1)赋值公理

$[P(x,g(x,y))]\ y \leftarrow g(x,y)\quad [P(x,y)]$

(2) 条件规则

$$\frac{[P\wedge R]\ F1[Q],\ [P\wedge\neg R]\ F2\ [Q]}{[P]\ \text{if R then F1 else F2}\ [Q]}$$

或者:

$$\frac{[P\wedge R]\ F1[Q],\quad P\wedge\neg R \Rightarrow Q}{[P]\ \text{if R then F1}\ [Q]}$$

(3) While 规则

$$\frac{P \Rightarrow I,\ [I\wedge R]\ F\ [I],\ I\wedge\neg R \Rightarrow Q}{[P]\ \text{while}\quad R\ \text{do F}\ [Q]}$$

(4) 并置规则

$$\frac{[P]\ F1[P1],\ [P1]\ F2\ [Q]}{[P]\ F1;\ F2\ [Q]}$$

(5) 结论规则

$$\frac{P \Rightarrow\quad R,\ [R]\ F\ [Q]}{[P]\quad F\quad [Q]}$$

或者:

$$\frac{[P]\ F\ [R],\ R \Rightarrow Q}{[P]\quad F\quad [Q]}$$

派生规则：

(1) 赋值规则

$$\frac{P(x,y) \Rightarrow Q(x,g(x,y))}{[P(x,y)]\ y\leftarrow g(x,y)\ [Q(x,y)]}$$

(2) 重复赋值规则

$$\frac{P(x,y) \Rightarrow Q(x,g_n(x,g_{n-1}(x,\ldots,g_2(x,g_1(x,y))\ldots)))}{[P(x,y)]\ y\leftarrow g_1(x,y);\ y\leftarrow g_2(x,y);\ldots;\ y\leftarrow g_n(x,y);\ [Q(x,y)]}$$

(3) 变形并置规则

$$\frac{[P]\ F1\ [Q],\ [R]\ F2\ [S],\ Q \Rightarrow R}{[P]\quad F1;\ F2\ [S]}$$

证明程序部分正确性的公理学方法就是依据以上公理和推理规则进行的，一般有两种形式：

（1） 正向“证明” ：从某些公理出发，经使用规则，直到最后获得结果。

a) **根据给出的不变式断言，建立一些引理。**

b) **根据引理和赋值公理，对程序中 S 的每一个赋值语句 F_i 导出相应的不变式语句$[R_i]\ F_i\ [Q_i]$。**

c) **再根据这些不变式语句和上述的推理规则，逐步组成越来越长的程序段。**

d) **一直到推演出：[P(x)] F [Q(x,z)] 为止。**

课堂练习：正向“证明”实例。

（2） 反向“证明” ：先用规则，把总目标分成若干子目标，最后寻根于公理。

a) **从不变式语句 [P(x)] F [Q(x,z)] 出发**

b) **利用有关的规则将它逐步分解**

c) **一直到将所有的语句推演为逻辑表达式(即检验条件)**

d) **然后证明这些逻辑表达式成立**

e) **这样就证明了程序的部分正确性.**

课堂练习：反向“证明”实例。

课堂讨论：如何将 Hoare 公理化方法的推广到完全正确性证明？

课程作业（5）：用 Hoare 公理化方法证明某一程序的部分正确性。

知识扩展：BAN 逻辑：用于分析密码协议安全性的一种形式化工具。

BAN 是一种基于信念的模态逻辑。在推理过程中参加协议的主题的信念随消息交换的发展，不断地变化。应用 BAN 逻辑时，首先需要进行“理想化的步骤”，将协议的消息转换为 BAN 逻辑的公式，再根据具体情况进行合理的假设，有逻辑的推理规则根据理想化的协议和假设进行推理，推断协议能否完成预期的目标。

BAN 逻辑试图回答以下的问题：（1）协议所欲达到什么样的目标；（2）协议基于的假设是什么？（3）协议是存在冗余？（4）协议中的加密消息是否可以明文传递而不影响协议的安全性？

几个基本假设：

（1）密文块不能被篡改，也不能用几个小的密文块组成一个新的大密文块。

（2）一个消息中的两个密文块被看作是分两次分别到达的。

（3）总假设加密系统是完善的，即只有掌握密钥的主体才能理解密文消息，因为不知道密钥的主体不能解密密文而得到明文。攻击者无法从密文推断出密钥。

（4）密文含有足够的冗余信息，使解密者可以判断他是否应用了正确的密钥。

（5）消息中含有足够的冗余信息，使主体可以判断该消息是否来源于自身。

（6）BAN 逻辑还假设参与协议的主体是诚实的。

作为一种多类型模态逻辑（many-sorted modal logic），BAN 逻辑主要包含下面 3 种处理对象：主体（principal）、密钥（keys）和公式（formula）。其中的公式，也称为语句或命题（statement）。BAN 逻辑仅包含合取这一命题联接词，用逗号表示；合取连接词满足交换律和结合律。依照 BAN 逻辑的惯例，P，Q 和 R 表示主体变量，K 表示密钥变量，X 和 Y 表示公式变量。A，B 表示两个普通主体，S 是认证服务器。K_{ab}，K_{as}，K_{bs} 等表示具体的共享密钥；K_a，K_b，K_c 等表示具体的公开密钥；K_a^{-1}，K_b^{-1}，K_c^{-1} 等表示相应的秘密密钥；N_a，N_b 和 N_c 等表示临时值；$h(X)$ 表示 X 的单向散列函数。下面逐一介绍 BAN 逻辑构件（construct）的语法和语义、BAN 逻辑的推理规则（又称逻辑公设）和 BAN 逻辑的形式化分析步骤。

（1）$P \mid\equiv X$ 或 P *believes* X　主体 P 相信公式 X 是真的；

（2）$P \triangleleft X$ 或 P *sees* X　主体 P 接收到了包含 X 的消息，即存在某主体 Q 向 P 发送了包含 X 的消息；

（3）$P \mid\sim X$ 或 P *said* X　主体 P 曾经发送过包含 X 的消息；

（4）$P \mid\Rightarrow X$ 或 P *controls* X　主体 P 对 X 有管辖权；

（5）$\#(X)$ 或 *fresh*(X)　X 是新鲜的，即 X 没有在当前回合前作为某消息的一部分被发送过，这里 X 一般为临时值；

（6）$P \overset{K}{\longleftrightarrow} Q$　K 为 P 和 Q 之间的共享密钥，且除 P 与 Q 以及他们相信的主体之外，其他主体都不知道 K；

（7）$\overset{K}{\longrightarrow} P$ 或 $\overset{K}{\longmapsto} P$　K 为 P 的公开密钥，且除 P 和他相信的主体之外，其他主体都不知道相应的秘密密钥 K^{-1}；

（8）$P \overset{X}{\rightleftharpoons} Q$　X 为 P 和 Q 的共享秘密，且除 P 和 Q 以及他们相信的主体之外，其他主体都不知道 X；

（9）$\{X\}_K$　用密钥 K 加密 X 后得到的密文；这是 $\{X\}_K$ from P 的简写；

（10）$\langle X\rangle_Y$　表示由 X 和秘密 Y 合成的消息。

BAN 推理规则

BAN 逻辑共有推理规则 19 条，下面将第 n 条推理规则简记为 $R_n(n = 1,2,\cdots,19)$。在以下的推理规则中，$\vdash$ 是元语言符号，$\Gamma \vdash C$ 表示可以由前提集 Γ 推导出结论 C。

1. 消息含义规则(message-meaning rules)

消息含义规则共有 3 条,分别如下:

$$R_1 \quad P \mid\equiv Q \overset{K}{\longleftrightarrow} P, P \triangleleft \{X\}_K \vdash P \mid\equiv Q \mid\sim X$$

$$\frac{P \ believes \ Q \overset{K}{\longleftrightarrow} P, P \ sees \ \{X\}_K}{P \ believes \ Q \ said \ X}$$

对于公开密钥情形,有类似的推理规则,它的两种表示方法如下:

$$R_2 \quad P \mid\equiv \xrightarrow{K} Q, P \triangleleft \{X\}_{K^{-1}} \vdash P \mid\equiv Q \mid\sim X$$

$$\frac{P \ believes \ \overset{K}{\longmapsto} Q, P \ sees \ \{X\}_{K^{-1}}}{P \ believes \ Q \ said \ X}$$

对于共享秘密情形,有类似的推理规则:

$$R_3 \quad P \mid\equiv P \overset{Y}{\rightleftharpoons} Q, P \triangleleft \{X\}_Y \vdash P \mid\equiv Q \mid\sim X$$

2. 临时值验证规则(nonce-verification rule)

临时值验证规则只有 1 条,它的两种表示方法如下:

$$R_4 \quad P \mid\equiv \#(X), P \mid\equiv Q \mid\sim X \vdash P \mid\equiv Q \mid\equiv X$$

$$\frac{P \ believes \ fresh(X), P \ believes \ Q \ said \ X}{P \ believes \ Q \ believes \ X}$$

R_4 的含义是,如果 P 相信消息 X 是新的,且 P 相信 Q 曾发送过 X,则 P 相信 Q 相信 X。

3. 管辖规则(jurisdiction rule)

管辖规则只有 1 条,它的两种表示方法如下:

$$R_5 \quad P \mid\equiv Q \mid\Rightarrow X, P \mid\equiv Q \mid\equiv X \vdash P \mid\equiv X$$

$$\frac{P \ believes \ Q \ controls \ X, P \ believes \ Q \ believes \ X}{P \ believes \ X}$$

R_5 的含义是,如果 P 相信 Q 对消息 X 有管辖权,且 P 相信 Q 相信 X,则 P 相信 X。

4. 接收消息规则(seeing rules)

接收消息规则共有 5 条,它们的两种表示方法如下:

$$R_6 \quad P \triangleleft (X, Y) \vdash P \triangleleft X$$

$$\frac{P \ sees \ (X, Y)}{P \ sees \ X}$$

$$R_7 \quad P \triangleleft < X >_Y \vdash P \triangleleft X$$

$$\frac{P \ sees \ \langle X \rangle_Y}{P \ sees \ X}$$

$$R_8 \quad P \mid\equiv P \overset{K}{\longleftrightarrow} Q, P \triangleleft \{X\}_K \vdash P \triangleleft X$$

$$\frac{P \ believes \ Q \overset{K}{\longleftrightarrow} P, P \ sees \ \{X\}_K}{P \ sees \ X}$$

$$R_9 \quad P \mid\equiv \xrightarrow{K} P, P \triangleleft \{X\}_K \vdash P \triangleleft X$$

$$\frac{P \ believes \ \overset{K}{\longmapsto} P, P \ sees \ \{X\}_K}{P \ sees \ X}$$

$$R_{10} \quad P \mid\equiv \xrightarrow{K} Q, P \triangleleft \{X\}_{K^{-1}} \vdash P \triangleleft X$$

$$\frac{P \ believes \ \overset{K}{\longmapsto} Q, P \ sees \ \{X\}_{K^{-1}}}{P \ sees \ X}$$

5. 消息新鲜性规则(freshness rule)

消息新鲜性规则只有 1 条,它的两种表示方法如下:

$$R_{11} \quad P \mid\equiv \#(X) \vdash P \mid\equiv \#(X, Y)$$

$$\frac{P \ believes \ fresh(X)}{P \ believes \ fresh(X, Y)}$$

R_{11} 的含义是,如果一个公式的一部分是新鲜的,则该公式全部是新鲜的。

为简明起见,以下只列举出推理规则的一种表示方法,并省略关于这些规则的说明。

6. 信念规则(belief rules)

信念规则共有 4 条,分别如下:

$$R_{12} \quad P \mid\equiv X, P \mid\equiv Y \vdash P \mid\equiv (X, Y)$$

$$R_{13} \quad P \mid\equiv (X, Y) \vdash P \mid\equiv X$$

$$R_{14} \quad P \mid\equiv Q \mid\equiv (X, Y) \vdash P \mid\equiv Q \mid\equiv X$$

$$R_{15} \quad P \mid\equiv Q \mid\sim (X, Y) \vdash P \mid\equiv Q \mid\sim X$$

7. 密钥与秘密规则(key and secret rules)

密钥与秘密规则共有 4 条，分别如下：

$$R_{16}\quad P \mid\equiv R \overset{K}{\longleftrightarrow} R' \vdash P \mid\equiv R' \overset{K}{\longleftrightarrow} R$$

$$R_{17}\quad P \mid\equiv Q \mid\equiv R \overset{K}{\longleftrightarrow} R' \vdash P \mid\equiv Q \mid\equiv R' \overset{K}{\longleftrightarrow} R$$

$$R_{18}\quad P \mid\equiv R \overset{X}{\rightleftharpoons} R' \vdash P \mid\equiv R' \overset{X}{\rightleftharpoons} R$$

$$R_{19}\quad P \mid\equiv Q \mid\equiv R \overset{X}{\rightleftharpoons} R' \vdash P \mid\equiv Q \mid\equiv R' \overset{X}{\rightleftharpoons} R$$

BAN 逻辑的推理步骤如下：

(1) 用逻辑语言对系统的初始状态进行描述，建立初始假设集合。

(2) 建立理想化协议模型，将协议的实际消息转换成 BAN 逻辑所能识别的公式。

(3) 对协议进行解释，将形如 $P \rightarrow Q : X$ 的消息转换成形如 $Q \lhd X$ 的逻辑语言。

解释过程中遵循以下规则：

① 若命题 X 在消息 $P \rightarrow Q : Y$ 前成立，则在其后 X 和 $Q \lhd Y$ 都成立；

② 若根据推理规则可以由命题 X 推导出命题 Y，则命题 X 成立时，命题 Y 也成立。

(4) 应用推理规则对协议进行形式化分析，推导出分析结果。

以上步骤可能会重复进行，例如，通过分析增加新的初设、改进理想化协议等。

讨论：

（1） BAN 逻辑的应用举例 （2）BAN 逻辑的不足 （3）BAN 逻辑的改进

BAN 逻辑改进的方向有：

1）确立一个可靠的语义，用以验证初始假设的正确性和确保推理的可靠性。

2）减少理想化步骤的模糊度，进而消除理想化步骤。

3）建立计算机化的自动分析过程，将各类攻击模型化并进行分析。

第七章 递归程序设计及其正确性证明

7.1 递归与递归程序的计算原则

递归算法实际上是一种分而治之的方法，它把复杂问题分解为简单问题来求解。对于某些复杂问题(例如 hanio 塔问题)，递归算法是一种自然且合乎逻辑的解决问题的方式，但是递归算法的执行效率通常比较差。因此，在求解某些问题时，常采用递归算法来分析问题，用非递归算法来求解问题；另外，有些程序设计语言不支持递归，这就需要把递归算法转换为非递归算法。

一种简化的函数递归程序模型的一般形式如下：

```
F(x1,x2,…,xn) ≡ if p1 then E1
                else if p2 then E2
                ……
                else if pm then Em
                else Em+1
```

其中，pi(i=1,2,…,m)是测试谓词，Ei(i=1,2,…,m+1)是表达式。其中 F 可以在 Ei 和 pi 中出现。

递归程序的正确性证明思路：（1）证明对于“最简单”的数据，程序运行正确；（2）假设对于“较简单”的数据，程序运行正确的“归纳假设”，在此基础上证明对于“较复杂”的数据， 程序亦运行正确。注：其中的“数据”可以是一般的数值，也可以是由数值、原子或表等组成的某种数据结构。

递归程序的计算规则：（1）递归程序可以采用不同的规则来计算。（2）理论上已经证明，如果同一个递归程序的不同计算规则如果都终止，那么结果一定也相同。（3）不同的计算过程虽然结果相同，但是时间复杂度和空间复杂度往往不同；实际工作的编译器往往使用“最左最内”规则。其他计算规则包括：“最左”规则、“平行最内”规则、“平行最外”规则等。（4）采用不同的计算规则来计算的递归程序。对相同的子变元，计算过程可能终止，也可能不终止。

结论：同一个递归程序的不同计算规则如果都终止，那么结果一定也相同。

7.2 递归程序设计的正确性证明

结构归纳法：对程序的递归结构进行归纳，这也是结构归纳法和一般的数学归纳法的区别所在。

（1）证明对于“最简单”的数据程序运行正确；（2）假设对于“较简单”的数据，程序运行正确（归纳假设），在此基础之上，证明对于较复杂的数据，程序也运行正确。

课堂练习：结构归纳法实例证明部分正确性。

良序归纳法：良序归纳法是一种较强形式的结构归纳法，适合更为复杂的递归程序证明。 设（W，-<）是一个良序集，P(x)是一个命题，为了证明对于所有的 x∈W，P(x), 只要 （1） 证明 P(x0)为真，x0 是 W 中“最小”元素；（2）归纳假设：假设对于所有的 x’-<x, P(x’)为真，在此基础上证明 P(x)为真。

课堂练习：良序归纳实例证明部分正确性。

课堂讨论：**递归需要很多资源，效率比较低：时间效率和空间效率。**

递归是一种比迭代循环更强的循环结构，可以证明每个迭代程序原则上总可以转换成与等价的递归程序，但是，反之不然，即并不是每个递归程序都可以转换成与它等价的迭代程序。但就效率而言，递归程序的实现往往要比迭代程序耗费更多的时间与存储空间。

7.3 程序变换技术

程序变换的基本思想：程序变换的基本思想是把程序设计工作分为两个阶段进行，即程序生成阶段和程序改进阶段。在程序生成阶段，先用一种清晰和抽象的语言来描述算法，并在此基础上设计一个面向问题的，易于理解的、正确的函数型递归程序，这时暂不考虑程序的运行效率。在程序改进阶段，将通过一系列保持正确性的变换规则，对程序的数据结构和算法进行求精，最终将该程序转换为一个具体的面向过程且运行效率高的程序文本。

变换规则：5 种基本规则：定义规则、取样规则、展开和封叠规则、用定律规则、抽象规则。

派生变换规则是一组基本变换规则序列的缩写，或可有基本变换规则和归纳法加以证明的变化规则。

7.4 尾递归程序的转换模式

如果一个函数中所有递归形式的调用都出现在函数的末尾，我们称这个递归函数是尾递归的。当递归调用是整个函数体中最后执行的语句且它的返回值不属于表达式的一部分时，这个递归调用就是尾递归。尾递归函数的特点是在回归过程中不用做任何操作，这个特性很重要，因为大多数现代的编译器会利用这种特点自动生成优化的代码。尾递归适用于运算对当前递归路径有依赖的问题，传统递归适用于运算对更深层递归有依赖的问题。递归算法转换为非递归算法有两种方法，一种是直接求值，不需要回溯；另一种是不能直接求值，需要回溯。前者使用一些变量保存中间结果，称为直接转换法，通常用来消除尾递归和单向递归，将递归结构用循环结构来替代；后者使用栈保存中间结果，称为间接转换法，一般需根据递归函数在执行过程中栈的变化得到， 在数据结构中有较多实例，如二叉树遍历算法的非递归实现、图的深度优先遍历算法的非递归实现等等。使用非递归方式实现递归问题的算法程序,不仅可以节省存储空间,而且可以极大地提高算法程序的执行效率。

课堂讨论：为何要进行尾部调用优化？尾部递归与迭代有何区别？

第八章 软件形式规格说明和验证技术

8.1 形式化方法概述

8.2 规格说明技术

8.3 规格验证技术

8.4 形式化语言（以 Z 语言为例）

8.5 研究与应用展望

3 能力培养实践

4 教学案例

5 教学方法探讨

6 教学资源开发和建设

7 问题与建议