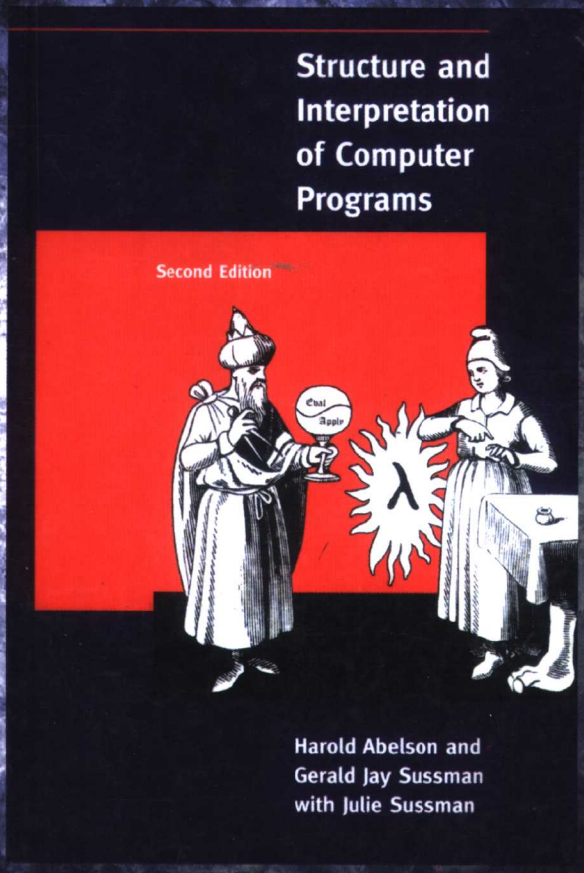
计 算 机 科 学 丛 书

原书第2版

计算机程序的构造和解释

(美) Harold Abelson Gerald Jay Sussman Julie Sussman 著
麻省理工学院

裘宗燕 译
北京大学



Structure and Interpretation of Computer Programs

Second Edition

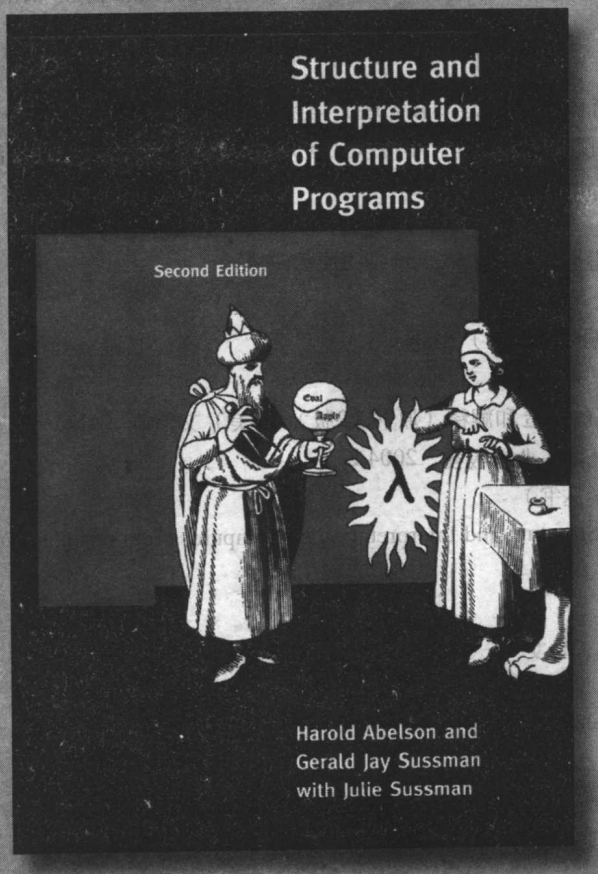
机械工业出版社 China Machine Press
中信出版社 CITIC PUBLISHING HOUSE

计 算 机 科 学 丛 书

原书第2版

计算机程序的构造和解释

(美) Harold Abelson Gerald Jay Sussman Julie Sussman 著 裘宗燕 译
麻省理工学院 北京大学



Structure and Interpretation of Computer Programs

Second Edition

机械工业出版社 China Machine Press
中信出版社 CITIC PUBLISHING HOUSE

本书从1980年开始就是美国麻省理工学院计算机科学专业的入门课程教材之一，从理论上讲解计算机程序的创建、执行和研究。主要内容包括：构造过程抽象，构造数据抽象，模块化、对象和状态，元语言抽象，寄存器机器里的计算等。本书描述生动有趣，分析清晰透彻，是计算机专业学生入门必读教材，也是计算机专业人士不可或缺的参考读物。

Harold Abelson, et al: Structure and Interpretation of Computer Programs, Second Edition (ISBN 0-262-01553-0).

本书版权登记号：图字：01-2002-0604

图书在版编目（CIP）数据

计算机程序的构造和解释（原书第2版）/（美）艾伯森（Abelson, H.）等著；裘宗燕译. －北京：机械工业出版社，2004.2

（计算机科学丛书）

书名原文：Structure and Interpretation of Computer Programs, Second Edition

ISBN 7-111-13510-5

Ⅰ.计… Ⅱ.①艾… ②裘… Ⅲ.程序设计 Ⅳ.TP311.1

中国版本图书馆CIP数据核字（2003）第112670号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：吴 怡

北京瑞德印刷有限公司印刷·新华书店北京发行所发行

2004年2月第1版第1次印刷

787mm×1092mm 1/16·30.75印张

印数：0 001 - 4 000册

定价：45.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业

的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：(010) 68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周立柱	周克定	周傲英	孟小峰	岳丽华
范　明	郑国梁	施伯乐	钟玉琢	唐世渭
袁崇义	高传善	梅　宏	程　旭	程时端
谢希仁	裘宗燕	戴　葵		

秘　书　组

武卫东	温莉芳	刘　江	杨海玲

带着崇敬和赞美，将本书献给活在计算机里的神灵。

"我认为，在计算机科学中保持计算中的趣味性是特别重要的事情。这一学科在起步时饱含着趣味性。当然，那些付钱的客户们时常觉得受了骗。一段时间之后，我们开始严肃地看待他们的抱怨。我们开始感觉到，自己真的像是要负起成功地、无差错地、完美地使用这些机器的责任。我不认为我们可以做到这些。我认为我们的责任是去拓展这一领域，将其发展到新的方向，并在自己的家里保持趣味性。我希望计算机科学的领域绝不要丧失其趣味意识。最重要的是，我希望我们不要变成传道士，不要认为你是宪鲁圣经的人，世界上这种人已经太多了。你所知道的有关计算的东西，其他人也都能学到。绝不要认为似乎成功计算的钥匙就掌握在你的手里。你所掌握的，也是我认为并希望的，也就是智慧：那种看到这一机器比你第一次站在它面前时能做得更多的能力，这样你才能将它向前推进。"

Alan J. Perlis（1922年4月1日－1990年1月7日）

序

教育者、将军、减肥专家、心理学家和父母做规划（program），而军人、学生和另一些社会阶层则被人规划（are programmed）。解决大规模问题需要经过一系列规划，其中的大部分东西只有在工作进程中才能做出来，这些规划中充满着与手头问题的特殊性相关的情况。如果想要把做规划这件事情本身作为一种智力活动来欣赏，你就必须转到计算机的程序设计（programming），你需要读或者写计算机程序——而且要大量地做。有关这些程序具体是关于什么的、服务于哪类应用等等的情况常常并不重要，重要的是它们的性能如何，在用于构造更大的程序时能否与其他程序平滑衔接。程序员们必须同时追求具体部分的完美和汇合的适宜性。在这部书里使用“程序设计”一词时，所关注的是程序的创建、执行和研究，这些程序是用一种Lisp方言书写的，为了在数字计算机上执行。采用Lisp并没有对我们可以编程的范围施以任何约束或者限制，而只不过确定了程序描述的记法形式。

本书中要讨论的各种问题都牵涉到三类需要关注的对象：人的大脑、计算机程序的集合以及计算机本身。每一个计算机程序都是现实中的或者精神中的某个过程的一个模型，通过人的头脑孵化出来。这些过程出现在人们的经验或者思维之中，数量上数不胜数，详情琐碎繁杂，任何时候人们都只能部分地理解它们。我们很少能通过自己的程序将这种过程模拟到永远令人满意的程度。正因为如此，即使我们写出的程序是一集经过仔细雕琢的离散符号，是交织在一起的一组函数，它们也需要不断地演化：当我们对于模型的认识更深入、更扩大、更广泛时，就需要去修改程序，直至这一模型最终到达了一种亚稳定状态。而在这时，程序中就又会出现另一个需要我们去为之奋斗的模型。计算机程序设计领域之令人兴奋的源泉，就在于它所引起连绵不绝的发现，在我们的头脑之中，在由程序所表达的计算机制之中，以及在由此所导致的认识爆炸之中。如果说艺术解释了我们的梦想，那么计算机就是以程序的名义执行着它们。

就其本身的所有能力而言，计算机是一位一丝不苟的工匠：它的程序必须正确，我们希望说的所有东西，都必须表述得准确到每一点细节。就像在其他所有使用符号的活动中一样，我们需要通过论证使自己相信程序的真。可以为Lisp本身赋予一个语义（可以说是另一个模型），假如说，一个程序的功能可以在（例如）谓词演算里描述，那么就可以用逻辑方法做出一个可接受的正确性论证。不幸的是，随着程序变得更大更复杂（实际上它们几乎总是如此），这种描述本身的适宜性、一致性和正确性也都变得非常值得怀疑了。因此，很少能够看到有关大程序正确性的完全形式化的论证。因为大的程序是从小东西成长起来的，开发出一个标准化的程序结构的武器库，并保证其中每种结构的正确性——我们称它们为惯用法，再学会如何利用一些已经证明很有价值的组织技术，将这些结构组合成更大的结构，这些都是至关重要的。本书中将详尽地讨论这些技术。理解这些技术，对于参与这种被称为程序设计的具有创造性的事业是最最本质的。特别值得提出的是，发现并掌握强有力的组织技术，将提升我们构造大型的重要程序的能力。反过来说，因为写大程序非常耗时费力，这也推动着我们去发明新方法，减轻由于大程序的功能和细节而引起的沉重负担。

与程序不同，计算机必须遵守物理定律。如果它们要快速执行——几个纳秒做一次状态转换——那么就必须在很短的距离内传导电子（至多1.5英尺）。必须消除由于大量元件而产生的热量集中。人们已经开发出了一些巧妙的工程艺术，用于在功能多样性与元件密度之间求得一种平衡。在任何情况下，硬件都是在比我们编程时所需要关心的层次更低的层次上操作的。将我们的Lisp程序变换到“机器”程序的过程本身也是抽象模型，是通过程序设计做出来的。研究和构造它们，能使人更加深刻地理解与任何模型的程序设计有关的程序组织问题。当然，计算机本身也可以这样模拟。请想一想：最小的物理开关元件在量子力学里建模，而量子力学又由一组微分方程描述，微分方程的细节行为可以由数值去近似，这种数值又由计算机程序所描述，计算机程序的组成……

区分出上述三类需要关注的对象，并不仅仅是为了策略上的便利。即使有人说它不过是人头脑里的东西，这种逻辑区分也引起了这些关注焦点之间符号流动的加速，它们在人们经验中的丰富性、活力和潜力，只能由现实生活中的不断演化去超越。我们至多只能说，这些关注焦点之间的关系是基本稳定的。计算机永远都不够大也不够快。硬件技术的每一次突破都带来了更大规模的程序设计事业，新的组织原理，以及更加丰富的抽象模型。每个读者都应该反复地问自己“到哪里才是头儿，到哪里才是头儿？”——但是不要问得过于频繁，以免忽略了程序设计的乐趣，使自己陷入一种喜忧参半的呆滞状态中。

在我们写出的程序里，有些程序执行了某个精确的数学函数（但是绝不够精确），例如排序，或者找出一系列数中的最大元，确定素数性，或者找出平方根。我们将这种程序称为算法，关于它们的最佳行为已经有了许多认识，特别是关于两个重要的参数：执行的时间和对数据存储的需求。程序员应该追求好的算法和惯用法。即使某些程序难以精确地描述，程序员也有责任去估计它们的性能，并要继续设法去改进之。

Lisp是一个幸存者，已经使用了四分之一个世纪。在现存的活语言里，只有Fortran比它的寿命更长些。这两种语言都支持着一些重要领域中的程序设计需要，Fortran用于科学与工程计算，Lisp用于人工智能。这两个领域现在仍然很重要，它们的程序员都如此倾心于这两种语言，因此，Lisp和Fortran都还可能继续生存至少四分之一个世纪。

Lisp一直在改变着。这本教科书中所用的Scheme方言就是从原来的Lisp里演化出来的，并在若干重要方面与之相异，包括变量约束的静态作用域，以及允许函数产生出函数作为值。在语义结构上，Scheme更接近于Algol 60而不是早期的Lisp。Algol 60已经不可能再变为活的语言了，但它还活在Scheme和Pascal的基因里。很难找到这样的两种语言，它们能如此清晰地代表着围绕这两种语言而聚集起来的两种差异巨大的文化。Pascal是为了建造金字塔——壮丽辉煌、令人震憾，是由各就其位的沉重巨石筑起的静态结构。而Lisp则是为了构造有机体——同样的壮丽辉煌并令人震憾，由各就其位但却永不静止的无数简单的有机体片段构成的动态结构。在两种语言里都采用了同样的组织原则，除了其中特别重要的一点不同之外：托付给Lisp程序员个人可用的自由支配权，要远远超过在Pascal社团里可找到的东西。Lisp程序大大抬高了函数库的地位，使其可用性超越了催生它们的那些具体应用。作为Lisp的内在数据结构，表对于这种可用性的提升起着最重要的作用。表的简单结构和自然可用性反应到函数里，就使它们具有了一种奇异的普适性。而在Pascal里，数据结构的过度声明导致函数的专用性，阻碍并惩罚临时性的合作。采用100个函数在一种数据结构上操作，远远优于用10个函数在10个数据结构上操作。作为这些情况的必然后果，金字塔矗立在那里千年不变，而有机体则必须

演化，否则就会死亡。

为了看清楚这种差异，请将本书中给出的材料和练习与任何第一门Pascal课程的教科书中的材料做一个比较。请不要费力地去想象，说这不过是一本在MIT采用的教科书，其特异性仅仅是因为它出自那个地方。准确地说，任何一本严肃的关于Lisp程序设计的书都应该如此，无论其学生是谁，在什么地方使用。

请注意，这是一本有关程序设计的教科书，它不像大部分关于Lisp的书，因为那些书多半是为人们在人工智能领域工作做准备。当然，无论如何，在研究工作规模不断增长的过程中，软件工程和人工智能所关心的重要程序设计工作正趋于相互结合。这也解释了为什么在人工智能领域之外的人们对Lisp的兴趣在不断增加。

正如由其目标可以预见到的，人工智能的研究产生出许多重要的程序设计问题。在其他程序设计文化中，问题的洪水孵化出一种又一种新的语言。确实，在任何非常大的程序设计工作中，一条有用的组织原则就是通过发明新语言，去控制和隔离作业模块之间的信息流动。这些语言趋向于变得越来越不基本，逐渐逼近系统的边界，逼近我们作为人最经常与之交互的地方。作为这一情况的结果，在这种系统里包含着大量重复的复杂的语言处理功能。Lisp有着如此简单的语法和语义，程序的语法分析可以看作一种很简单的工作。这样，语法分析技术对于Lisp程序几乎就没有价值，语言处理器的构造对于大型Lisp系统的成长和变化不会成为阻碍。最后，正是这种语法和语义的极端简单性，产生出了所有Lisp程序员的负担和自由。任何规模的Lisp程序，除了那种寥寥几行的程序外，都饱含着考虑周到的各种功能。发明并调整，调整恰当后再去发明！让我们举起杯，祝福那些将他们的思想镶嵌在重重括号之间的Lisp程序员。

Alan J. Perlis

纽黑文，康涅狄格

第2版前言

软件很可能确实与其他任何东西都不同，它的本意就是被抛弃：这一观点的全部就是总将它看作一个肥皂泡吗？

——Alan J. Perlis

自1980年以来，本书的材料就一直在MIT作为计算机科学学科入门课程的基础。在本书第1版出版之前，我们已经用这一材料教了4年课，而到这个第2版出版，时间又过去了12年。我们非常高兴地看到这一工作被广泛接受，并被结合到其他一些教材中。我们已经看到自己的学生掌握了本书中的思想和程序，并将它们构筑到新的计算机系统或者语言的核心里。这就在文字上实现了一个古犹太教法典的双关语，我们的学生已经变成了我们的创造者。我们非常幸运能有如此有能力的学生和如此有建树的创造者。

在准备这一新版本的过程中，我们结合进了成百条澄清性建议，它们来自我们自己的教学经验，也来自MIT和其他地方的同行们的评述。我们重新设计了本书里主要的程序设计系统中的大部分，包括通用型算术系统、解释器、寄存器机器模拟器和编译器，也重写了所有的程序实例，以保证任何符合IEEE的Scheme标准（IEEE 1990）的Scheme实现都能运行这些代码。

这一版本中强调了几个新问题，其中最重要的是有关在不同的途径中，计算模型里对于时间的处理所起的中心作用：带有状态的对象、并发程序设计、函数式程序设计、惰性求值和非确定性程序设计。这里为并发和非确定性新增加了几节，我们也设法将这一论题集成到整本书里，贯穿始终。

本书第1版基本上是按照我们在MIT一学期课程的教学大纲撰写的。由于有了第2版中增加的这些新材料，在一个学期里覆盖所有内容已经不可能了，所以教师需要从中做一些选择。在我们自己的教学里，有时会跳过有关逻辑程序设计的一节（4.4节）；让学生使用寄存器机器模拟器，但并不去讨论它的实现（5.2节）；对于编译器则只给出一个粗略的概述（5.5节）。即使如此，这还是一个内容非常多的课程。一些教师可能希望只覆盖前面的三章或者四章，而将其他内容留给后续课程。

万维网站点www-mitpress.mit.edu/sicp为本书的使用者提供支持。其中包含了取自本书的程序，示例程序作业、辅助材料和Lisp的Scheme方言的可下载实现。

第1版前言

一台计算机就像是一把小提琴。你可以想象一个新手试了一个音符并丢掉了它。后来他说，听起来真难听。我们已经从大众和我们的大部分计算机科学家那里反复听到这种说法。他们说，计算机程序对个别具体用途而言确实是好东西，但它们太缺乏弹性。一把小提琴或者一台打字机也同样缺乏弹性，那是你学会了如何去使用它们之前。

——Marvin Minsky，“为什么说程序设计很容易成为一种媒介，用于表述理解浮浅、草率而就的思想”

本书是麻省理工学院（MIT）计算机科学的入门教材。在MIT主修电子工程或者计算机科学的所有学生都必须学这门课，作为“公共核心课程计划”的四分之一。这里还包含两个关于电路和线性系统的科目，还有一个关于数字系统设计的科目。我们从1978年开始涉足这些科目的开发，从1980年秋季以后，我们就一直按照现在这种形式教授这个课程，每年600到700个学生。大部分学生此前没有或者很少有计算方面的正式训练，虽然许多人玩过计算机，也有少数人有许多程序设计或者硬件设计的经验。

我们所设计的这门计算机科学导引课程反映了两方面的主要考虑。首先，我们希望建立起一种看法：一个计算机语言并不仅仅是让计算机去执行操作的一种方式，更重要的，它是一种表述有关方法学的思想的新颖的形式化媒介。因此，程序必须写得能够供人们阅读，偶尔地去供计算机执行。其次，我们相信，在这一层次的课程里，最基本的材料并不是特定程序设计语言的语法，不是有效计算某种功能的巧妙算法，也不是算法的数学分析或者计算的本质基础，而是一些能够用于控制大型软件系统的智力复杂性的技术。

我们的目标是，使完成了这一科目的学生能对程序设计的风格要素和审美观有一种很好的感觉。他们应该掌握了控制大型系统中的复杂性的主要技术。他们应该能够去读50页长的程序，只要该程序是以一种值得模仿的形式写出来的。他们应该知道在什么时候哪些东西不需要去读，哪些东西不需要去理解。他们应该很有把握地去修改一个程序，同时又能保持原来作者的精神和风格。

这些技能并不仅仅适用于计算机程序设计。我们所教授和提炼出来的这些技术，对于所有的工程设计都是通用的。我们在适当的时候隐藏起一些细节，通过创建抽象去控制复杂性。我们通过建立约定的界面，以便能以一种“混合与匹配”的方式组合起一些标准的、已经很好理解的片段，去控制复杂性。我们通过建立一些新的语言去描述各种设计，每种语言强调设计中的一个特定方面并降低其他方面的重要性，以控制复杂性。

设计这门课程的基础是我们的一种信念，“计算机科学”并不是一种科学，而且其重要性也与计算机本身并无太大关系。计算机革命是有关我们如何去思考的方式，以及我们如何去表达自己的思考的一个革命。在这个变化里最基本的东西，就是出现了这样一种或许最好是

称为过程性认识论的现象——这就是如何从一种命令式的观点去研究知识的结构，这一观点是与经典数学领域中所采用的更具说明性的观点完全不同的。数学为精确处理“是什么”提供了一种框架，而计算则为精确处理“怎样做”的概念提供了一种框架。

在教授这里的材料时，我们采用的是Lisp语言的一种方言。我们绝没有形式化地教授这一语言，因为完全不必那样做。我们只是使用它，学生可以在几天之内就学会它。这也是类Lisp语言的重要优点：它们只有不多几种构造复合表达式的方式，几乎没有语法结构。所有的形式化性质都可以在一个小时里讲完，就像下象棋的规则似的。在很短时间之后，我们就可以不再去管语言的语法细节（因为这里根本就没有），而进入真正的问题——弄清楚我们需要去计算什么，怎样将问题分解为一组可以控制的部分，如何对这样的部分开展工作。Lisp的另一优势在于，与我们所知的任何其他语言相比，它可以支持（但并不是强制性的）更多的能用于以模块化的方式分解程序的大规模策略。我们可以做过程性抽象和数据抽象，可以通过高阶函数抓住公共的使用模式，可以用赋值和数据操作去模拟局部状态，可以利用流和延时求值连接起一个程序里的各个部分，可以很容易地实现嵌入性语言。所有这些都融合在一个交互式的环境里，带有对递增式程序设计、构造、测试和排除错误的绝佳支持功能。我们要感谢一代又一代的Lisp大师，从John McCarthy开始，是他们铸造起了这样一个具有空前威力的如此优美的好工具。

作为我们所用的Lisp方言，Scheme试图将Lisp和Algol的威力和优雅集成到一起。我们从Lisp那里取来了元语言的威力，它来自简单的语法形式，程序与数据对象的统一表示，以及带有废料收集的堆分配数据。我们从Algol那里取来了词法作用域和块结构，这是当年参加Algol委员会的那些程序设计语言先驱者们的礼物。我们想特别提出John Reynolds和 Peter Landin，为了他们对丘奇的lambda演算与程序设计语言的结构之间关系的真知灼见。我们也认识到应该感谢那些数学家们，他们在计算机出现之前，就已经在这一领域中探索了许多年。这些先驱者包括丘奇（Alonzo Church）、罗塞尔（Barkley Rosser）、克里尼（Stephen Kleene）和库里（Haskell Curry）。

致　谢

我们希望感谢许多在这本书和这一教学计划的开发中帮助过我们的人们。

可以把这门课看做是课程“6.231”的后继者。“6.231”是20世纪60年代后期由Jack Wozencraft和Arthur Evans, Jr. 在MIT教授的有关程序设计语言学和lambda演算的一门美妙课程。

我们由Robert Fano那里受惠良多。是他组织了MIT电子工程和计算机科学的教学计划，强调工程设计的原理。他领着我们开始了这一事业，并为此写出了第一批问题注记。本书就是从那里演化出来的。

我们试图去教授的大部分程序设计风格和艺术都是与Guy Lewis Steele Jr.一起开发的，他与Gerald Jay Sussman在Scheme语言的初始开发阶段合作工作。此外，David Turner、Peter Henderson、Dan Friedman、David Wise和Will Clinger也教给我们许多函数式程序设计社团所掌握的技术，它们出现在本书的许多地方。

Joel Moses教我们如何考虑大型系统的构造。他在Macsyma符号计算系统上的经验中得到的真知灼见是，应该避免控制中的复杂性，将精力集中到数据的组织上，以反映所模拟世界里的真实结构。

这里的许多有关程序设计及其在我们的智力活动中位置的认识是Marvin Minsky和Seymour Papert提出的。从他们那里我们理解了，计算是一种探索各种思想的表达方式的手段，如果不这样做，这些思想将会因为太复杂而无法精确地处理。他们更强调说，学生编写和修改程序的能力可以成为一种威力强大的工具，使这种探索变成一种自然的活动。

我们也完全同意Alan Perlis的看法，程序设计有着许多乐趣，我们应该认真地支持程序设计的趣味性。这种趣味性部分来源于观看大师们的工作。我们非常幸运曾经在Bill Gosper和Richard Greenblatt手下学习程序设计。

很难列出所有曾对这一教学计划的开发做出过贡献的人们。我们衷心感谢在过去15年里与我们一起工作过，并在此科目上付出时间和心血的所有教师、答疑老师和辅导员们，特别是Bill Siebert、Albert Meyer、Joe Stoy、Randy Davis、Louis Braida、Eric Grimson、Rod Brooks、Lynn Stein和Peter Szolovits。我们想特别向Franklyn Turbak（现在在Wellesley）在教学上的特殊贡献表示谢意，他在本科生指导方面的工作为我们的努力设定了一个标准。我们还要感谢Jerry Saltzer和Jim Miller帮助我们克服并发性中的难点，Peter Szolovits和David McAllester对于第4章里非确定性求值讨论的贡献。

许多人在他们自己的大学里讲授本书时付出了极大努力，其中与我们密切合作的有Technion的Jacob Katzenelson、Irvine加州大学的Hardy Mayer、牛津大学的Joe Stoy、普度大学的Elisha Sacks以及挪威科技大学的Jan Komorowski。我们特别为那些在其他大学改制这一课程，并由此获得重要教学奖的同行们感到骄傲，包括耶鲁大学的Kenneth Yip、加州大学伯克利分校的Brian Harvey和康乃尔大学的Dan Huttenlocher。

Al Moyé安排我们到惠普公司为工程师教授这一课程，并为这些课程制作了录像带。我们感谢有那些才干的教师——特别是Jim Miller、Bill Siebert和Mike Eisenberg——他们设计了结

合这些录像带的继续教育课程，并在全世界的许多大学和企业讲授。

其他国家的许多教育工作者也在翻译本书的第1版方面做了许多工作。Michel Briand、Pierre Chamard和André Pic做出了法文版，Susanne Daniels-Herold做了德文版，Fumio Motoyoshi做了日文版。

要列举出所有为我们用于教学的Scheme系统做出过贡献的人是非常困难的。除了Guy Steele之外，主要的专家还包括Chris Hanson、Joe Bowbeer、Jim Miller、Guillermo Rozas和Stephen Adams。在这项工作中付出许多时间的还有Richard Stallman、Alan Bawden、Kent Pitman、Jon Taft、Neil Mayle、John Lamping、Gwyn Osnos、Tracy Larrabee、George Carrette、Soma Chaudhuri、Bill Chiarchiaro、Steven Kirsch、Leigh Klotz、Wayne Noss、Todd Cass、Patrick O'Donnell、Kevin Theobald、Daniel Weise、Kenneth Sinclair、Anthony Courtemanche、Henry M. Wu、Andrew Berlin和Ruth Shyu。

除了MIT实现之外，我们还应该感谢那些在IEEE Scheme标准方面工作的人们，包括William Clinger和Jonathan Rees，他们编写了R^4RS；以及Chris Haynes、David Bartley、Chris Hanson和Jim Miller，他们撰写了IEEE标准。

Dan Friedman多年以来一直是Scheme社团的领袖。这一社团的工作范围已经从语言设计问题，扩展到围绕着重要的教育创新问题，例如基于Schemer's Inc. 的EdScheme的高中教学计划，以及由Mike Eisenberg和由Brian Harvey和Matthew Wright撰写的绝妙著作。

我们还要感谢那些为本教材的成书做出贡献的人们，特别是MIT出版社的Terry Ehling、Larry Cohen和Paul Bethge。Ella Mazel为本书找到了最美妙的封面图画。对于第2版，我们要特别感谢Bernard和Ella Mazel对本书设计的帮助，以及David Jones作为TEX专家的非凡能力。我们还要感谢下面的读者，他们对于这个新书稿提出了深刻的意见：Jacob Katzenelson、Hardy Mayer、Jim Miller，特别是Brian Harvey，他对于本书所做的也就像Julie对于Harvey的著作*Simply Scheme*所做的那样。

最后我们还想对资助组织表示感谢，它们多年来一直支持这项工作的进行。包括来自惠普公司的支持（由Ira Goldstein和Joel Birnbaum促成），还有来自DARPA的支持（得到了Bob Kahn的帮助）。

目　录