HZ BOOKS
华章教育
PEARSON

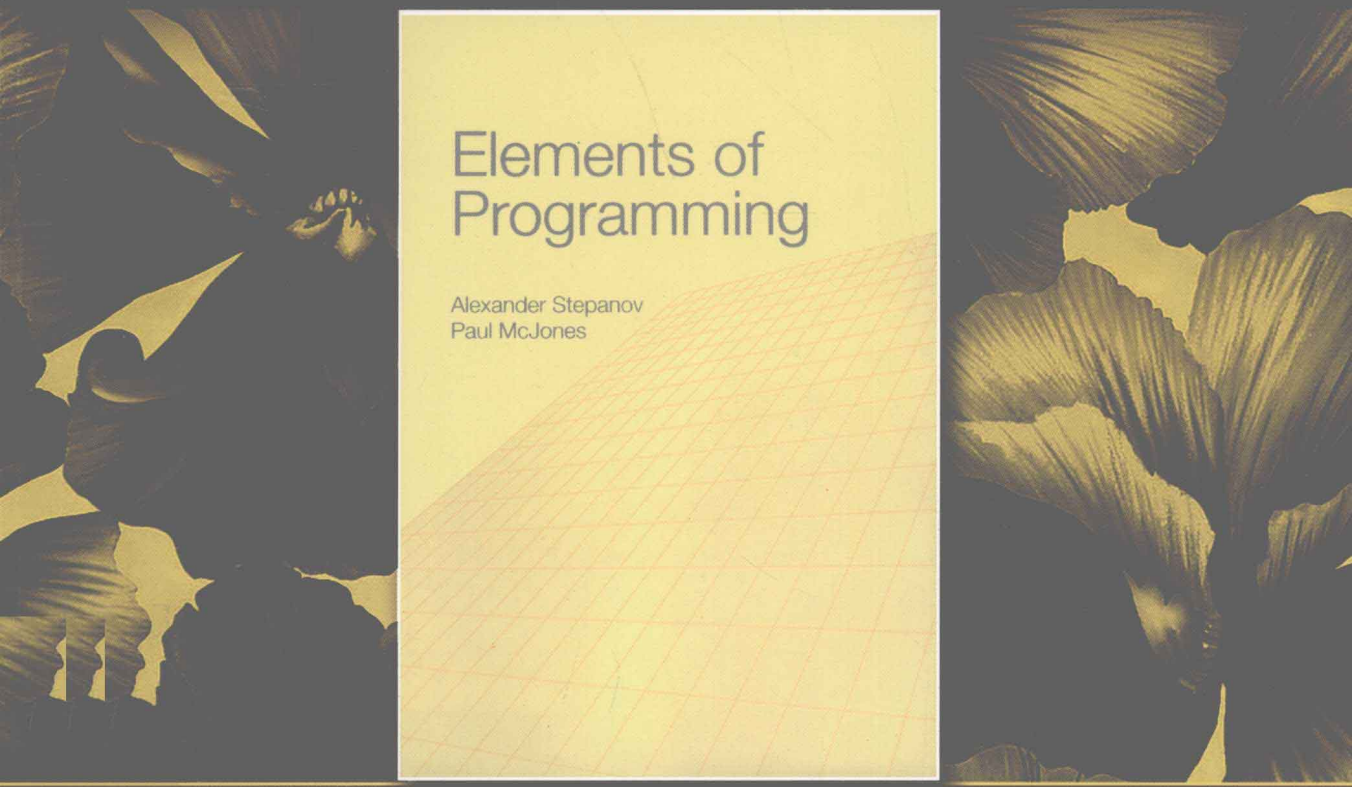
计 算 机 科 学 丛 书

追溯数学原理 探求编程原本
STL之父力作，C++之父鼎力推荐
北大数学学院裘宗燕教授倾情献译

编程原本

（美）Alexander Stepanov Paul McJones 著 裘宗燕 译

Elements of Programming



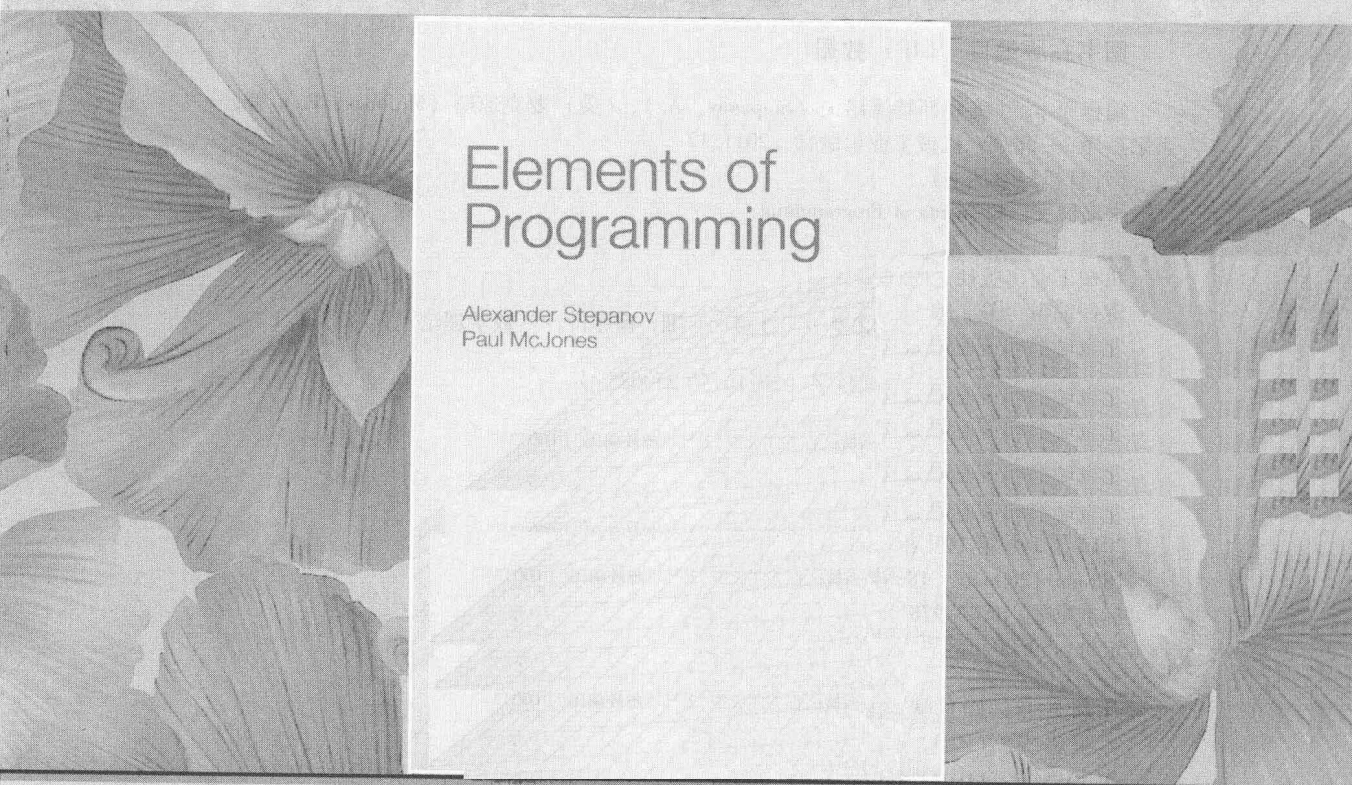
机械工业出版社
China Machine Press

计 算 机 科 学 丛 书

编程原本

（美）Alexander Stepanov　Paul McJones 著　裘宗燕 译

Elements of Programming



机械工业出版社
China Machine Press

本书将严格的数学定义、公理化和演绎方法应用于程序设计，讨论程序与保证它们能正确工作的抽象数学理论之间的联系。书中把理论的规程、基于这些写出的算法，以及描述算法性质的引理和定理一起呈现给读者，以帮助我们将复杂系统分解为一些具有特定行为的组件。

本书适合软件开发人员和需要进行程序设计的科学家及工程师阅读，也可供高等院校计算机及相关专业的师生参考。

Authorized translation from the English language edition, entitled *Elements of Programming*, 9780321635372 by Alexander Stepanov, Paul McJones, published by Pearson Education, Inc., Copyright © 2009 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD., and CHINA MACHINE PRESS Copyright © 2012.

本书封底贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

封底无防伪标均为盗版
版权所有，侵权必究
本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2010-1034

图书在版编目（CIP）数据

编程原本/（美）斯特潘诺夫（Stepanov, A.），（美）麦克琼斯（McJones, P.）著；裘宗燕译. —北京：机械工业出版社，2011.12
（计算机科学丛书）
书名原文：Elements of Programming

ISBN 978-7-111-36729-1

Ⅰ. 编…　Ⅱ. ①斯…　②麦…　③裘…　Ⅲ. 程序设计－数学理论　Ⅳ. TP311.1

中国版本图书馆 CIP 数据核字（2011）第 250685 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：关　敏
北京瑞德印刷有限公司印刷
2012 年 1 月第 1 版第 1 次印刷
185mm × 260mm · 18.5 印张
标准书号：ISBN 978-7-111-36729-1
定价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066
购书热线：（010）68326294；88379649；68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

译者序

在 Addison-Wesley 新书预告上看到本书时, 我就感觉到本书的不同凡响, 觉得应该让更多国内同行了解书中蕴涵的理念. 我立刻把信息告诉了华章的朋友. 没多久就得到了华章引进版权的消息.

市面上讨论编程的书籍浩如烟海. 说起编程, 人们头脑中浮现的多半是语言、代码、hacking、测试、排除程序错误, 以及与之相关的许多琐碎事务. 而本书作者看到的重点却不同. 在讨论编程时, 他们关注的是数学、结构、规律、规范性、抽象、推导、前后条件、验证等等. 本书作者的技术水平和成就毋庸置疑, 但为什么他们能构造出像 STL 那样的巅峰之作, 这件事却值得认真思考. 如果国内有人说程序的基础是数学, 估计会有不少人对其嗤之以鼻: "你懂得什么是程序? 写过多少行代码?" 但是, 恐怕无人敢小觑本书作者, 其见解和论据也无法忽视. 基于上述基本考虑, 本书中给出了大量精妙而且根基坚实的程序, 解决了一个个具体而重要的问题. 进一步说, 作者还揭示了这些程序的理论基础, 并从多个角度建立起它们之间的联系, 使之可能成为无数实际程序的基础构件. 作者的根本目标, 或许就是希望基于这种思维方法和开发技术, 为范围广泛的软件系统建立起坚实基础. 在这里看不到调侃和讨好读者的流行俗语或插科打诨, 只有严肃的叙述、分析和讨论. 阅读本书的过程绝不会轻松, 但我们可以相信, 在这里的付出会使人收获厚丰.

作者取 "Elements" 作为书名也很值得玩味. 我们都知道欧几里得的名著"Στοιχεία" (英译 "Elements"), 中文译为《几何原本》可能偏离了作者的本意. 欧氏应该是想为一种世界观和方法论提供一个范本, 其内涵和意义绝不限于今天所说的"几何"领域. 实际上, 《原本》希望展示的是思维和研究背后的一套基础概念和思想. 现在我们面对的是程序, 这是一个完全人造的世界, 这里的一切也有什么 "本原" 吗? 本书作者基于其经验和认识也想来尝试一下, 模

仿欧几里得探究一下位于所有编程背后的最根本的东西. 作者认为, 程序背后的本原就是数学的各种概念、技术和方法, 需要演绎、推导和证明等, 而绝不是模糊的想法和草率的编码和蛮力调试. 随着计算机被更广泛地应用于各种重要领域, 只靠朴素认识去工作将越来越显得脆弱和不可信. 从这个角度看, 本书也可以看成是作者的规劝和忠告.

我本无意翻译本书, 但经不住华章的朋友一再相邀, 只好勉强为之. 原书出自作者上课的幻灯片, 言简意赅, 涉及面广, 找到简洁又切中原义的译文常常很难, 一句译文经常需要斟酌很久, 每一遍检查都会发现许多不如意之处. 由于没有足够的时间, 经常有其他事务干扰, 这个翻译工作一拖再拖. 但无论如何, 一件事总要有个结束. 我决定现在将此工作告一段落, 将结果交给出版社和读者检验. 我希望这个中译本能对读者有所帮助, 也为其中的缺陷和不足负责.

为尽可能保持原貌, 我基于原书的 LaTeX 文件编辑译文 (感谢作者提供的原文件), 写了许多 LaTeX 宏定义, 最后用 MiKTeX 生成全书的 PDF 文件交给出版社. 这样做多花了不少时间, 但有利于保持全书 (及与原书) 的统一性, 可以减少不应出现的小错误. 当然, 这种费时费力的做法也使排版错误变成了我的个人责任. 本译本已收入原作者迄今的所有勘误, 翻译中发现的原书错误也都得到了作者确认. 为方便读者阅读, 本书实际上提供了两套索引: 原书的英文索引都予以保留, 与之对应, 我又加入了一套中文索引 (通过 LaTeX 生成), 供读者交叉参考. 原作者自创了不少术语, 没有习见译法, 我只能设法编造出相应译文, 其合理性需要时间检验. 此外, 正文中出现的术语都给出了英文对照.

有关本书的工作, 我要感谢刘海洋在使用 LaTeX 处理中文方面的若干意见, 感谢编辑发现了译稿的一些文字错误.

裘宗燕

2011年10月6日, 于北京大学理科一号楼

试读结束：需要全本请在线购买： www.ertongbook.com

前　言

本书将演绎方法应用于程序设计, 讨论程序与保证它们能正确工作的抽象数学理论之间的联系. 书中把反映这些理论的规程 (specification), 基于这些理论写出的算法, 以及描述算法性质的引理和定理一起呈现给读者. 这些算法在一种实际程序设计语言里的实现是本书的中心. 虽然规程主要是供人阅读, 但它们也应该 (或者说必须) 严格地与非形式化的、供机器使用的代码相结合, 必须在通用的同时又是抽象而且精确的.

与在其他科学和工程领域里的情况一样, 适合作为程序设计的基础的同样是演绎方法. 演绎方法能帮助我们将复杂系统分解为一些具有特定数学行为的组件, 而这种分解又是设计高效、可靠、安全和经济的软件的前提.

本书是想奉献给那些希望更深入地理解程序设计的人们, 无论他们是专职软件开发人员, 还是把程序设计看作其专业活动中一个重要组成部分的科学家或工程师.

本书编写的基本想法是让读者从头到尾完整阅读. 读者只有通过阅读代码、证明引理、完成练习, 才能真正理解书中的各方面材料. 此外我们还建议了一些项目, 其中有些是完全开放的. 本书的内容很紧凑, 认真的读者最终会看到书中各部分之间的联系, 以及我们选择这些材料的理由. 发现本书在体系结构方面的原理应该是读者的一个目标.

我们假定读者已经具有完成各种基本代数操作的能力.[1] 还假定读者熟悉逻辑和集合论的基本术语, 如普通本科生在离散数学课程中学习的内容. 附录 A 总结了书中使用的各种记法. 如果在一些特定的算法里需要某些抽象代数的概念, 书中会给出相应的定义. 我们还假定读者熟悉程序设计, 理解计算机

1. 有关基本代数知识, 推荐 Chrystal [1904].

体系结构,[2] 理解最基本的算法和数据结构.[3]

我们选用 C++, 是因为它组合了强有力的抽象设施和基础机器的正确表示.[4] 这里只用了该语言的一个小子集, 需求被写成程序里的结构化注释. 我们希望不熟悉 C++ 的读者也能阅读本书. 附录 B 描述了书中使用的 C++ 子集.[5] 在书中的任何地方, 在需要区分数学记法和 C++ 的地方, 根据所用的字体、排版和上下文就能确定用的是哪种意义 (是数学的还是 C++ 的). 虽然书中的许多概念和程序与 STL (C++ 标准模板库) 里的东西对应, 但这里的一些设计决策是与 STL 不同的. 书中还忽略了实际程序库 (如 STL) 必须考虑和处理的许多问题, 如名字空间、可见性、inline 指令等等.

第 1 章描述值、对象、类型、过程和概念. 第 2 ～ 5 章描述各种代数结构 (如半群、全序集) 上的算法. 第 6 ～ 11 章讨论抽象内存上的算法. 第 12 章讨论包含对象成员的对象. 跋给出了我们对本书中阐释的工作途径的反思.

致谢

真诚感谢 Adobe Systems 及其支持程序设计基础课程和本书的管理部门, 本书就是在这一课程中成长起来的. 特别感谢 Greg Gilley 启动了这个课程并建议我们写这本书; Dave Story 以及后来 Bill Hensler 提供了始终不渝的支持. 最后, 如果没有 Sean Parent 富于启发的管理以及对代码和正文持续的彻查, 本书也不可能完成. 本书的思想萌芽于我们与 Dave Musser 跨越了近三十年的紧密合作. 感谢 Bjarne Stroustrup 认真地发展 C++ 来支持这些想法. Dave 和 Bjarne 友善地专门到 San Jose 来并仔细审阅了早期的书稿. Sean Parent 和 Bjarne Stroustrup 为本书写了定义 C++ 子集的附录. Jon Brandt 审阅了本书的多个稿本. John Wilkinson 仔细阅读了最后的书稿, 提出了不计其数有价值的建议.

我们的编辑 Peter Gordon, 项目编辑 Elizabeth Ryan, 文本编辑 Evelyn Pyle, 审阅编辑 Matt Austern、Andrew Koenig、David Musser、Arch Robison、Jerry

2. 推荐 Patterson and Hennessy [2007].

3. 推荐 Tarjan [1983], 这是一本有关算法和数据结构的有选择的但又非常深刻的入门书.

4. 标准的参考书是 Stroustrup [2000].

5. 书中代码都能在 Microsoft Visual C++ 9 和 g++ 4 下编译运行. 所有代码, 以及几个使它们能编译直至做单元测试的简单的宏, 都可以在 www.elementsofprogramming.com 下载.

Schwarz、Jeremy Siek 和 John Wilkinson 都对本书做出了重要的贡献.

我们感谢在 Adobe 以及更早在 SGI 参与有关课程的学生们的各种建议. 我们希望自己成功地把这些课程里的材料编织为一个统一的整体. 我们衷心感谢 Dave Abrahams、Andrei Alexandrescu、Konstantine Arkoudas、John Banning、Hans Boehm、Angelo Borsotti、Jim Dehnert、John DeTreville、Boris Fomitchev、Kevlin Henney、Jussi Ketonen、Karl Malbrain、Mat Marcus、Larry Masinter、Dave Parent、Dmitry Polukhin、Jon Reid、Mark Ruzon、Geoff Scott、David Simons、Anna Stepanov、Tony Van Eerd、Walter Vannini、Tim Winkler 和 Oleg Zabluda 的意见和建议. 我们感谢 John Banning, Bob English、Steven Gratton、Max Hailperin、Eugene Kirpichov、Alexei Nekrassov、Mark Ruzon 和 Hao Song 指出了第一次印刷里的一些错误; Foster Brereton、Gabriel Dos Reis、Ryan Ernst、Abraham Sebastian、Mike Spertus、Henning Thielemann 和 Carla Villoria Burgazzi 指出了第二次印刷里的错误; Shinji Dosaka、Ryan Ernst、Steven Gratton、Volker Lukas、Qiu Zongyan、Abraham Sebastian 和 Sean Silva 指出了第三次印刷里的错误.[6]

最后, 还要衷心感谢通过其写作或者个人联系给我们教益, 使我们加深了对程序设计的理解的所有人以及相关的机构.

6. 最新的勘误信息见 www.elementsofprogramming.com.

关于作者

Alexander Stepanov 于 1967 到 1972 年间在国立莫斯科大学学习数学, 从 1972 年开始在苏联, 1977 年移民后继续在美国从事程序设计工作. 他编写过操作系统、程序设计工具、编译器和各种程序库. 他在程序设计基础方面的工作先后得到 GE、Brooklyn Polytechnic、AT&T、HP、SGI 和 Adobe 的支持. 他在 1995 年因 C++ 标准模板库的设计获 Dr. Dobb's Journal 的程序设计杰出贡献奖.

Paul McJones 于 1967 到 1971 年间在加州大学伯克利分校学习工程数学, 1967 年进入程序设计领域. 他涉足的领域包括操作系统、程序设计环境、事务处理系统, 以及企业和客户应用系统等. 他先后在加州大学、IBM、Xerox、Tandem、DEC 和 Adobe 工作. 1982 年他与合作者一起因论文 The Recovery Manager of the System R Database Manager 获得 ACM 程序设计系统和语言论文奖.

目　录

试读结束：需要全本请在线购买： www.ertongbook.com

第 1 章

基础

本章从相关思想的一个简洁分类开始介绍一些术语：值、对象、类型、过程和概念，它们表示了与计算机有关的许多不同的理念范畴. 这里还要详细讨论本书的中心观点：规范性. 对一个过程，规范性意味着它对相等的参数总返回相等的结果. 对一个类型，规范性意味着它应该有相等运算符，以及保证相等关系的拷贝构造函数和赋值. 规范性使我们能应用 等值推理 (使用等值替换) 去变换和优化程序.

1.1 理念范畴：实体，类别，类属

为了解释什么是对象、类型，以及其他基本的计算机概念，概述一下与这些概念对应的理念范畴是很有帮助的.

抽象实体 (abstract entity) 指永存的不变的事物，而*具体实体* (concrete entity) 指具体的个别的事物，其出现和存在与时间和空间有关. 一个*属性* (attribute) 是具体实体与抽象实体之间的一种对应关系，它描述了该具体实体的某种性质、度量或者品质. *标识* (identity) 是我们感知实在世界的一种基本概念，它确定一个事物在随着时间变化中的不变性. 一个具体实体的属性可以改变，但这种改变不会影响其标识. 一个具体实体的一个*快照* (snapshot) 就是在某个特定时间点上这一事物的所有属性的完整集合. 具体实体不仅包括所有物理上存在的实体，还包括法律的、经济的或者政治的实体. 蓝色和 13 是抽象实体的例子. 苏格拉底和美利坚合众国是具体实体的例子. 苏格拉底的眼睛的颜色和美国的州的个数是属性的例子.

一个*抽象类别* (abstract species) 描述一批本质上等价的抽象实体的共性.

抽象类别的例子如自然数和颜色. 一个*具体类别* (concrete species) 描述一集本质上等价的具体实体的共性. 具体类别的例子如男人和美国的州.

一个*函数* (function) 是一套规则, 它将一个或几个取自某个或某些相应类别的抽象实体 (称为其*参量*, argument), 关联到来自某个抽象类别的一个抽象实体 (称为其*结果*, result). 函数的例子如后继函数, 它将每个自然数关联于紧随其后的那个自然数; 再如将两种颜色关联于它们的混合色的函数.

一个*抽象类属* (abstract genus) 描述在某些方面类似的一些不同的抽象类别. 抽象类属的例子如数和二元运算符. 一个*具体类属* (concrete genus) 描述在某些方面类似的一些不同的具体实体. 具体类属的例子如哺乳动物和鸟.

一个实体属于某个特定的类别, 这个类别确定了该实体的构造和存在的规则. 一个实体可以属于多个类属, 每个类属描述该实体的一些特定性质.

在本章的下面部分, 我们将论证对象和值都是实体, 类型是类别, 而概念是类属.

1.2 值

如果不知道如何解释, 在计算机里能看到的也就是一些 0 和 1. 一项*数据* (datum) 就是一个 0 和 1 的有穷序列.

一个*值类型* (value type) 是一个 (抽象或具体) 类别和一集数据之间的一个对应关系. 对应于某特定实体的数据称为该实体的一个*表示* (representation); 而这一实体称为相应数据的*解释* (interpretation). 我们把一项数据和它的解释称为一个*值* (value). 值的例子如采用大尾格式的 32 位二进制补码表示的整数; 或者用连续的两个 32 位二进制序列表示的有理数, 这两个二进制序列分别被解释为用整数表示的分子和分母, 而这两个整数又用大尾格式的 32 位二进制补码表示.

一项数据相对于一个值类型是*良形式的* (well-formed), 当且仅当这一数据表示了该类型里的一个抽象实体. 例如, 每个 32 位的二进制序列解释为模二补码的整数时都是良形式的; 而 IEEE 754 浮点的一个 NaN (Not a Number, 不是数) 解释为实数就不是良形式的.

一个值类型是*真部分的*(properly partial), 如果它的值只能表示对应抽象类别里的所有抽象实体的一个真子集; 否则它就是*全的* (total). 举例说, 类型 `int`

是真部分的, 而类型 bool 是全的.

一个值类型是*唯一表示的* (uniquely represented), 当且仅当每个抽象实体至多有一个对应的值. 例如, 如果表示真值的类型使用了一个字节, 其中的 0 解释为假, 而其他情况都解释为真, 那么这个类型就不是唯一表示的. 如果一个类型把整数表示为一个符号位和一个无符号量, 它就不能为 0 提供唯一表示. 用模二补码表示整数的类型是唯一表示的.

一个值类型是*有歧义的* (ambiguous), 当且仅在当该类型里存在具有多种解释的值. 有歧义的否定是*无歧义* (unambiguous). 例如, 将长于一个世纪的纪年表示为两个十进制数的类型就是有歧义的.

一个值类型里的两个值*相等* (equality), 当且仅当它们表示的是同一个抽象实体. 说两个值是*表示相等的* (representational equality), 当且仅当它们的数据是两个相同的 0/1 序列.

引理 1.1 如果一个值类型具有唯一表示, 相等就蕴涵着表示相等.

引理 1.2 如果一个值类型无歧义, 表示相等就蕴涵着相等.

如果一个值类型是唯一表示的, 通过检查其 0/1 序列是否相同的方式就可以实现相等判断. 否则就必须以一种与该类型的解释协调的方式来实现相等判断. 如果对于一个值类型经常需要生成新值而较少检查相等, 那就可以考虑选用非唯一性的表示, 因为这样做有可能使新值的生成更快, 而使相等判断较慢. 这类的例子如, 用一对整数表示的两个有理数相等, 条件是它们可以约简到同一个最简分数. 再如用非排序序列表示的两个集合相等, 条件是经过排序并消除重复元素后, 它们的对应元素相等.

有些时候, 实现真正的*行为上的* (behavioral) 相等判断的代价过大甚至根本就不可能. 例如, 将可计算函数编码为一个类型时的情况就是这样. 在这种情况下, 我们只能接受弱得多的*表示*相等的概念, 简单判断两个值的 0/1 序列是否相等.

计算机把抽象实体上的函数*实现* (implement) 为值上的*函数* (function). 虽然这些值驻留在内存, 但一个完好实现的值上的函数并不依赖于特定的内存地址, 它实现的是一个从值到值的映射 (mapping).

在一个值类型上定义的一个函数是*规范的* (regular), 当且仅当它遵从相等性: 给它的某个参数换一个相等的值, 它一定还得出相等的结果. 大多数的数

值函数都是规范的, 非规范的数值函数的一个例子是取分母函数: 假设有理数简单地表示为一对整数, 而该函数简单返回表示分子的那个整数. 例如 $\frac{1}{2} = \frac{2}{4}$, 但 numerator($\frac{1}{2}$) $\neq$ numerator($\frac{2}{4}$). 规范函数使我们可以做*等式推理* (equational reasoning), 允许我们做等值替换.

非规范的函数依赖于其参数的具体表示, 而不仅仅是参数的解释. 在设计一个值类型的表示时, 有两项相关的工作必须处理: 实现相等判断, 确定哪些函数应该是规范的.

1.3 对象

一个*存储* (memory) 就是一集存储字 (word), 其中每个字有一个*地址* (address) 和一项*内容* (content). 地址是一种固定大小的值, 这个大小称为*地址长度*. 而内容是另一固定大小的值, 其长度称为*字长*. 通过*装载* (load) 操作可以取得一个地址的内容. 通过*保存* (store) 操作改变一个地址所关联的内容. 存储的实例如计算机主存里的一组字节, 或磁盘驱动器里的一组区块.

一个*对象* (object) 就是一个具体实体的表示, 并且是作为某个存储里一个值. 对象有*状态* (state), 其状态就是某个值类型的一个值. 对象的状态可以改变. 对于给定的与某个具体实体对应的一个对象, 其状态对应于该实体的一个快照. 一个对象拥有一集*资源* (resource), 用于保存其状态, 如一些存储字或者文件里的一些记录.

即使某对象的值是一个连续的 0/1 序列, 保存这些 0/1 的资源也可以不是连续的. 相应的解释能给出这个对象的整体. 看一个例子: 两个 `double` 可以解释为一个复数, 为此并不要求它们紧邻存放. 一个对象的资源也完全可以位于不同的存储里. 但是本书只处理位于一个具有统一地址空间的存储里的对象, 这里的每个对象有一个唯一的*起始地址* (starting address), 从它出发可以找到该对象的所有资源.

一个*对象类型* (object type) 是一种在存储中保存和修改值的模式. 与每个对象类型相对应的有一个描述该类型对象的状态的值类型. 每个对象属于某一个对象类型. 作为对象类型的例子, 请考虑按 32 位模二补码方式, 采用小尾格式和 4 字节边界对齐表示的整数.

值和对象扮演着互补的角色. 值不会改变, 与在计算机里的特定实现方式

试读结束：需要全本请在线购买： www.ertongbook.com