信息科学与技术丛书

裘巍 编著

编译器设计之路

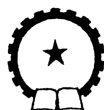
- 完整介绍一个实际编译器Neo Pascal的设计与实现
- 详细讲述LL(1)分析器、符号表系统、优化等核心话题
- 阐述系统软件的设计观点
- 探讨现代编译技术热点
- 提供编译器全部源代码

机械工业出版社
CHINA MACHINE PRESS

信息科学与技术丛书

编译器设计之路

裘　巍　编著



机 械 工 业 出 版 社

本书系统地介绍了一个实际的 Pascal 编译器 Neo Pascal 的设计与实现。结合 Neo Pascal 的源代码，详细讲述了 LL(1)语法分析器、符号表系统、中间表示、类型系统、优化技术、运行时刻的存储管理、代码生成器等编译器设计的核心话题。各章都附有少量以实践应用为主的练习题，既可作为阅读思考题，也可作为课程设计选题。

与国内其他介绍编译技术的图书相比，本书更关注的是编译器的实现细节，而不仅仅局限于理论阐述。本书可供从事编译器设计相关工作的工程人员阅读，也可作为高等院校计算机专业的编译原理课程参考书。

读者可在 http://neopascal.sourceforge.net 获得 Neo Pascal 的源代码及相关文档。

图书在版编目（CIP）数据

编译器设计之路 / 裘巍编著．—北京：机械工业出版社，2010.10

（信息科学与技术丛书）

ISBN 978-7-111-32164-4

Ⅰ．①编…　Ⅱ．①裘…　Ⅲ．①编译程序－程序设计　Ⅳ．①TP314

中国版本图书馆 CIP 数据核字（2010）第 196394 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）
策划编辑：车　忱
责任编辑：车　忱
责任印制：乔　宇
三河市宏达印刷有限公司印刷
2011 年 1 月第 1 版 • 第 1 次印刷
184mm×260mm • 28.75 印张 • 713 千字
0001－3000 册
标准书号：978-7-111-32164-4
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

电话服务
社服务中心：（010）88361066
销 售 一 部 ：（010）68326294
销 售 二 部 ：（010）88379649
读者服务部：（010）68993821

网络服务
门户网：http://www.cmpbook.com
教材网：http://www.cmpedu.com

封面无防伪标均为盗版

出 版 说 明

随着信息科学与技术的迅速发展，人类每时每刻都会面对层出不穷的新技术和新概念。毫无疑问，在节奏越来越快的工作和生活中，人们需要通过阅读和学习大量信息丰富、具备实践指导意义的图书来获取新知识和新技能，从而不断提高自身素质，紧跟信息化时代发展的步伐。

众所周知，在计算机硬件方面，高性价比的解决方案和新型技术的应用一直备受青睐；在软件技术方面，随着计算机软件的规模和复杂性与日俱增，软件技术不断地受到挑战，人们一直在为寻求更先进的软件技术而奋斗不止。目前，计算机在社会生活中日益普及，随着Internet延伸到人类世界的方方面面，掌握计算机网络技术和理论已成为大众的文化需求。由于信息科学与技术在电工、电子、通信、工业控制、智能建筑、工业产品设计与制造等专业领域中已经得到充分、广泛的应用，所以这些专业领域中的研究人员和工程技术人员越来越迫切需要汲取自身领域信息化所带来的新理念和新方法。

针对人们了解和掌握新知识、新技能的热切期待，以及由此促成的人们对语言简洁、内容充实、融合实践经验的图书迫切需要的现状，机械工业出版社适时推出了“信息科学与技术丛书”。这套丛书涉及计算机软件、硬件、网络和工程应用等内容，注重理论与实践的结合，内容实用、层次分明、语言流畅，是信息科学与技术领域专业人员不可或缺的参考书。

目前，信息科学与技术的发展可谓一日千里，机械工业出版社欢迎从事信息技术方面工作的科研人员、工程技术人员积极参与我们的工作，为推进我国的信息化建设作出贡献。

机械工业出版社

前　言

与其他自然科学相比，计算机科学的发展历史并不久远，是较新的学科体系，尚有许多未知的领域有待探索。因此，本专业学生或工程技术人员仅仅满足于学习或应用几门程序设计语言是远远不够的。一些看似抽象的课程才是提高专业人员“内功”修为的秘技，例如数据结构、操作系统、编译原理、计算机系统结构、计算机网络等。不过，经典课程的学习并不是一蹴而就的，如何学习与理解课程的精髓是值得关注的。本书将从编译器设计的角度，为读者揭示编译原理的精髓。

学习编译技术的意义

有人认为，编译技术似乎已经相当成熟了，继续深入研究是没有任何意义的。实际上，任何科学技术都是发展变化的。表面上看，编译器设计的高层问题似乎已经形成了完美的体系，但当我们深入其内核就会发现事实并非如此。现代编译器设计面临的挑战是来自目标计算机系统结构、新颖程序设计语言及本身的计算资源等多方面的。其中，任何一方面的因素都足以颠覆某些传统理论与算法。例如，在现代编译器设计中，计算资源的增加允许设计者采用更耗费时间、空间的算法，而不必过分关注其代价。对于优化算法设计者而言，这是令人兴奋的。为了追求目标代码的更优，即使设计一些时空复杂度稍高的算法也是可以接受的。

当然，从更高的层次上讲，学习编译器设计的目的还不仅仅局限于其本身的理论与技术。作为一个系统软件的设计学科，其解决问题的思路与方法更是值得读者细细品味的。这可能是一个漫长而艰辛的历程，不过，这才是经典学科的魅力所在。以品味经典为目的来学习与研究操作系统、数据库技术、计算机网络、编译技术等学科是笔者多年来的努力方向。

从 20 世纪 50 年代中期以来，编译器设计就一直是计算机科学界的一个重要研究领域。Fortran 语言之父 John Backus 认为，除非编译器生成的代码与手工编写的机器代码的性能非常接近，否则程序员就不会放弃汇编语言程序设计的思想与方法，这就是编译技术研究的源动力。从学术研究的角度讲，众所周知，图灵奖被誉为“计算机界的诺贝尔奖”，自 1966 年设立至今，54 位获奖者中就有 16 位是由于程序设计语言及编译技术的研究成果而获此殊荣的。编译技术在计算机科学领域的地位由此可见一斑。

本书的特点

国内讲述编译原理的书籍并不少见，但是基于一个实际编译器系统讨论相关实现的资料却相当稀缺。同样，国内高校的编译原理课程也存在着一个普遍的现象：讲述的内容与实际编译器设计之间存在着鸿沟。对许多专业学生而言，编译原理可能更像是一门充满数学符号与定义的计算机课程，因此很难得到学生的认可。

本书摒弃了传统教材只讲理论的不足，以笔者开发的编译器 Neo Pascal 为例，从词法分析、语法分析、语义分析、IR 生成、代码优化、目标代码生成等角度全面、系统地阐述了编译器设计与实现中的许多经典话题，包括 LL（1）语法分析器的实现、符号表的设计与实现、类型系统的实现、IR 的设计、迭代数据流分析、IR 优化、运行时刻管理、基于模式匹配的代码生成器的实现等。同时，笔者也引入了一些现代编译技术中的观点，对传统的算法

进行了一定的改进，以便达到相对更优的结果。与传统编译教材的最主要区别在于，本书并不是一本以阐述编译技术的相关理论与算法框架为主的原理性教材，而是一本依托实际编译器项目讨论其实现的书。因此，本书将不涉及与 Neo Pascal 实现无关的算法或话题。

在笔者看来，源代码是一种实现，也是设计的载体。因此，本书不是一本简单的源代码注释文本。其中，更多体现的是以提出问题并设计解决方案为核心的思想。笔者试图传达给读者的并不局限于编译器源代码的实现，而是其内核的设计原理。

笔者坚信，只有深刻理解算法分析与设计的过程，并实际参与编译器的设计，才可能体会编译技术的真谛。通过启发式的讲解，引领读者进入编译技术的殿堂正是撰写本书的目的。无论是对于在校的相关专业学生还是专业技术人员，本书都是非常适合自学使用的。对于有志于涉足编译技术领域的专业人员，本书的许多观点与方法更是值得思考。

本书涉及的 Neo Pascal 是一个开源的编译器系统，读者可以登录 http://neopascal.sourceforge.net，获得最新的项目源代码及相关文档。

本书的读者

本书预期的读者是至少掌握了 C++语言及数据结构的计算机专业人员或在校学生。对于有汇编语言程序设计、计算机系统结构、操作系统、编译原理等课程基础的读者，学习本书将事半功倍。当然，在涉及相关知识时，本书将作简单介绍。

内容概览

第 1 章　概述

本章将介绍一些关于程序设计语言、编译器的概念性话题，例如，解释与编译、编译器的架构等，这是讨论后续主题的基础。另外，本章也将简单介绍本书待实现的高级语言 Pascal。

第 2 章　词法分析

本章将介绍词法分析器的实现。与传统教材不同，本章只涉及词法分析器的设计与实现，并不讨论有限自动机的相关理论。

第 3 章　语法分析

本章将讨论形式语言描述方法及语法分析器的实现。形式语言是一种语言语法的描述形式，是语法分析器实现的基础。在语法分析器方面，本章将深入讨论 LL(1)语法分析器的原理与实现。这是一种非常经典的语法分析方法，广泛应用于许多经典编译器中。

第 4 章　符号表系统

符号表是编译器的中心数据库。不过，关于符号表的话题，传统教材却涉及甚少。虽然符号表的功能与普通数据表类似，但两者结构特点相差甚远。

第 5 章　中间表示

中间表示是一种由设计者定义与使用的内部语言。事实上，经典的中间表示形式非常之多。本章将介绍一种源自 lcc 的中间表示形式。同时，笔者将结合个人的体会，介绍一些关于中间表示的设计经验及其评价机制。另外，各种经典语句的翻译也是本章讨论的重点。

第 6 章　表达式语义

本章将涉及类型系统、表达式翻译等复杂的话题。其中，类型系统将讨论类型描述、类型兼容、类型转换、类型推断等，而表达式翻译更多关注的是数组、指针、结构、函数调用等语言机制的处理。

第 7 章　优化技术

优化技术是现代编译器设计关注的重点。鉴于优化的重要性，本书不惜耗费大量篇幅详细阐述各种基本优化算法的原理与实现。与传统教材相比，本书不局限于原理的介绍，更配以详细的源代码实现，便于读者理解，这是本书的特色之一。当然，本书并不是一本以介绍优化算法为主的高级教材，所以，并不涉及那些理论抽象、实现复杂的算法。

第 8 章　运行时刻的存储管理

本章将重点介绍栈、堆式存储分配机制的原理及其实现技术。这是一个实用性非常强的话题，故笔者将结合一些常见的语言现象进行深入剖析，包括动态数组、字符串、可变参数、调用约定等。

第 9 章　目标代码生成

本章将依托 i386 目标机实现一个完整的代码生成器。笔者采用了一种基于模式匹配实现的代码生成技术，这是一项非常高效且灵活的生成技术。同时，寄存器的分配也是本章讨论的重点。与传统教材基于模型机讨论代码生成相比，基于目标机实现代码生成器可能复杂得多，包括寄存器的分配、指令的选择等都需要考虑许多硬件结构的限制。

第 10 章　GCC 内核与现代编译技术概述

了解 GCC 内核是深入学习先进编译技术的基础，本章将以有限的篇幅介绍几个 GCC 最核心的话题，包括 GIMPLE、SSA、RTL 等。可以毫不夸张地说，它们正是 GCC 的魅力所在。其次，笔者还将介绍两个现代编译技术的研究热点，即动态编译技术与并行编译技术。并且还将提供一些资源与材料，供读者参考使用。

致谢

首先，要感谢上海交通大学张冬茉老师给予的极大鼓励与帮助，是她引领我走上了编译器设计之路。其次，要感谢我的同事潘松，在著书过程中，他给予了我最大程度的鼓励与支持。

联系方式

由于笔者水平有限，书中难免存在一些错误，殷切希望广大读者不吝批评指正。笔者的电子信箱是 qiuwei-qiuwei@163.com，QQ 号是 331341714。

目　录

出版说明
前言
第 1 章　概述 …… 1
1.1　编译技术概述 …… 1
1.1.1　程序设计语言基础 …… 1
1.1.2　程序设计语言的翻译机制 …… 4
1.1.3　编译器的基本结构 …… 5
1.2　Pascal 语言基础 …… 8
1.2.1　Pascal 语言简介 …… 8
1.2.2　Pascal 程序基本组成 …… 9
1.2.3　Pascal 的声明部分 …… 10
1.2.4　Pascal 的类型 …… 12
1.2.5　Pascal 的运算符 …… 15
1.2.6　Pascal 的语句 …… 17
1.3　开发环境与 Delphi 基础 …… 18
1.3.1　开发环境与文件列表 …… 18
1.3.2　Delphi 基础 …… 19
1.4　深入学习 …… 24
1.5　实践与思考 …… 25
1.6　大师风采——Niklaus Wirth …… 25
第 2 章　词法分析 …… 26
2.1　词法分析概述 …… 26
2.1.1　词法分析的任务 …… 26
2.1.2　单词的分类 …… 28
2.2　词法分析器的设计 …… 28
2.2.1　识别单词 …… 28
2.2.2　转换图 …… 29
2.2.3　构造词法分析器 …… 31
2.3　词法分析器的实现 …… 35
2.3.1　词法定义 …… 35
2.3.2　构造转换图与转换表 …… 36
2.3.3　相关数据结构 …… 38
2.3.4　源代码实现 …… 40
2.4　深入学习 …… 44
2.5　实践与思考 …… 45
2.6　大师风采——Dennis M. Ritchie …… 45
第 3 章　语法分析 …… 47
3.1　程序设计语言的语法描述 …… 47
3.1.1　上下文无关文法 …… 47
3.1.2　推导 …… 52
3.1.3　语法树 …… 54
3.1.4　归约简介 …… 57
3.2　语法分析概述 …… 58
3.2.1　语法分析的任务 …… 58
3.2.2　自上而下的语法分析法 …… 59
3.2.3　构造语法分析器 …… 64
3.3　语法分析器的实现 …… 71
3.3.1　文法定义 …… 71
3.3.2　语法分析表 …… 76
3.3.3　源代码实现 …… 86
3.4　深入学习 …… 90
3.5　实践与思考 …… 91
3.6　大师风采——Edsger Wybe Dijkstra …… 92
第 4 章　符号表系统 …… 93
4.1　语义分析概述 …… 93
4.1.1　程序设计语言的语义 …… 93
4.1.2　语义分析与 IR 生成的任务 …… 94
4.1.3　语法制导翻译 …… 95
4.2　符号表设计 …… 98
4.2.1　符号表概述 …… 98
4.2.2　符号表的逻辑结构 …… 99
4.2.3　符号表的实例分析 …… 109
4.3　声明部分的实现 …… 111
4.3.1　相关数据结构 …… 111
4.3.2　主程序首部声明 …… 113
4.3.3　包含文件声明部分 …… 114
4.3.4　标号声明部分 …… 118

4.3.5 常量声明部分 ……119
4.3.6 类型声明部分 ……120
4.3.7 变量声明部分 ……149
4.3.8 过程、函数声明部分 ……152
4.4 深入学习……154
4.5 实践与思考……155
4.6 大师风采——John Backus ……155
第5章 中间表示 ……156
5.1 IR 概述……156
5.1.1 IR 的作用……156
5.1.2 IR 设计及其级别……157
5.1.3 设计 IR 的重要意义……159
5.2 IR 生成……160
5.2.1 三地址代码概述……160
5.2.2 Neo Pascal三地址代码的实现……164
5.2.3 翻译机制概述 ……168
5.3 语句翻译概述……170
5.3.1 语句翻译基础 ……170
5.3.2 翻译辅助函数及其实现……173
5.4 if 语句 ……176
5.4.1 if 语句的翻译……176
5.4.2 源代码实现……177
5.5 while/repeat 语句 ……181
5.5.1 while 语句的翻译……181
5.5.2 源代码实现……181
5.5.3 repeat 语句的翻译……184
5.6 for 语句 ……184
5.6.1 for 语句的翻译 ……184
5.6.2 源代码实现……186
5.7 case 语句……192
5.7.1 case 语句的翻译……192
5.7.2 源代码实现 ……193
5.8 其他语句……199
5.8.1 break、continue 语句的翻译 ……199
5.8.2 goto 语句的翻译……201
5.8.3 asm 语句的翻译 ……204
5.9 深入学习……208
5.10 实践与思考 ……208
5.11 大师风采——Kenneth E. Iverson ……209
第6章 表达式语义 ……210
6.1 表达式概述……210
6.2 类型系统基础……211
6.2.1 类型基础 ……211
6.2.2 类型系统 ……212
6.2.3 类型转换 ……217
6.3 类型系统的实现 ……218
6.3.1 类型系统的设计……218
6.3.2 IR 的操作数 ……221
6.3.3 类型相容的实现……222
6.3.4 类型推断的实现……223
6.4 表达式翻译……226
6.4.1 表达式翻译基础……226
6.4.2 深入表达式翻译……229
6.4.3 表达式翻译的实现 ……230
6.5 操作数翻译……247
6.5.1 操作数的地址与形态 ……247
6.5.2 操作数翻译基础……248
6.5.3 简单变量操作数的翻译……252
6.5.4 记录字段操作数的翻译……262
6.5.5 数组翻译基础 ……265
6.5.6 数组元素操作数的翻译……270
6.5.7 指针运算的翻译……280
6.6 深入学习……286
6.7 实践与思考……286
6.8 大师风采——Alan Kay……287
第7章 优化技术 ……288
7.1 优化概述……288
7.1.1 什么是优化……288
7.1.2 优化级别 ……289
7.2 控制流分析……290
7.2.1 流图与基本块 ……290
7.2.2 流图的数据结构……292
7.2.3 流图的构造……293
7.2.4 优化的分类……297
7.3 数据流分析……298
7.3.1 数据流的相关概念 ……298

7.3.2 数据流分析的策略 ······ 298
7.3.3 活跃变量分析 ······ 299
7.3.4 ud 链与 du 链 ······ 301
7.3.5 更多数据流问题 ······ 302
7.4 数据流分析的实现 ······ 303
7.4.1 定值点与引用点分析的基础 ······ 303
7.4.2 定值点、引用点分析的相关数据结构 ······ 305
7.4.3 定值点、引用点分析的实现 ······ 307
7.4.4 活跃变量分析的实现 ······ 312
7.4.5 ud 链、du 链分析的实现 ······ 314
7.5 常量传播与常量折叠 ······ 321
7.5.1 常量传播基础 ······ 321
7.5.2 常量传播的实现 ······ 324
7.6 复写传播 ······ 328
7.6.1 复写传播的基础 ······ 328
7.6.2 复写传播的实现 ······ 330
7.7 代数简化 ······ 333
7.7.1 代数简化基础 ······ 333
7.7.2 代数简化的实现 ······ 334
7.8 跳转优化 ······ 339
7.8.1 跳转优化基础 ······ 339
7.8.2 条件跳转优化的实现 ······ 341
7.8.3 连续跳转优化的实现 ······ 343
7.9 冗余代码删除 ······ 345
7.9.1 冗余代码删除基础 ······ 345
7.9.2 死代码删除的实现 ······ 346
7.9.3 不可到达代码删除的实现 ······ 348
7.10 深入学习 ······ 350
7.11 实践与思考 ······ 350
7.12 大师风采——Richard Stallman ······ 351
第 8 章 运行时刻的存储管理 ······ 352
8.1 存储管理概述 ······ 352
8.1.1 存储区域 ······ 352
8.1.2 存储布局 ······ 354
8.1.3 存储分配基础 ······ 356
8.2 栈式存储分配 ······ 357
8.2.1 栈式存储分配基础 ······ 357
8.2.2 i386 栈式存储分配 ······ 360
8.2.3 深入理解栈式存储分配 ······ 365
8.3 存储分配的实现 ······ 368
8.4 存储优化 ······ 372
8.4.1 存储优化基础 ······ 372
8.4.2 存储优化的实现 ······ 374
8.5 深入学习 ······ 381
8.6 实践与思考 ······ 382
8.7 大师风采——Bjarne Stroustrup ······ 382
第 9 章 目标代码生成 ······ 383
9.1 目标代码生成概述 ······ 383
9.1.1 目标代码生成基础 ······ 383
9.1.2 指令选择 ······ 384
9.1.3 寄存器分配 ······ 385
9.2 目标机简介 ······ 386
9.2.1 目标机结构 ······ 386
9.2.2 浮点处理单元 ······ 387
9.2.3 操作数寻址方式 ······ 391
9.2.4 ptr 操作符 ······ 392
9.2.5 一个完整的汇编程序 ······ 393
9.3 构造代码生成器 ······ 393
9.3.1 自动代码生成器基础 ······ 393
9.3.2 指令模板 ······ 394
9.3.3 寄存器描述 ······ 397
9.3.4 寄存器分配 ······ 398
9.3.5 代码生成器的基本结构 ······ 402
9.4 深入学习 ······ 413
9.5 实践与思考 ······ 413
9.6 大师风采——Peter Naur ······ 413
第 10 章 GCC 内核与现代编译技术概述 ······ 414
10.1 编译技术的现状及发展 ······ 414
10.2 GCC 内核分析 ······ 415
10.2.1 GCC 的基本结构 ······ 415
10.2.2 GENERIC ······ 416
10.2.3 GIMPLE ······ 416
10.2.4 SSA ······ 426
10.2.5 RTL 概述 ······ 428

10.2.6 RTX ······ 430
10.3 动态编译技术简介 ······ 436
10.3.1 动态编译技术基础 ······ 436
10.3.2 运行时特定化 ······ 437
10.3.3 动态二进制翻译 ······ 439
10.4 并行编译技术简介 ······ 441
10.4.1 并行编译技术基础 ······ 441
10.4.2 并行计算机及其编译系统 ······ 443
10.5 深入学习 ······ 446
10.6 大师风采——Alan Perlis ······ 447
参考文献 ······ 448

第 1 章　概　　述

Reliable and transparent programs are usually not in the interest of the designer.

——*Niklaus Wirth*

1.1　编译技术概述

1.1.1　程序设计语言基础

Intel 公司的 David Kuck 院士曾经将编译器誉为“计算机科学与技术的皇后”，它是应用与系统之间的一座桥梁。在国内，由于计算机基础科学相对薄弱，人们通常更多投身于应用领域的研究，编译技术并不太受到人们的关注。因此，基于这方面的研究成果也相对较少。除了早期一些大型机的 Fortran、Algol 编译器之外，并没有真正的产品级编译器。不过，这并不意味着研究编译技术是毫无价值可言的。事实上，作为计算机科学的组成部分之一，编译技术有着极其崇高的地位，其辉煌的历史可能是其他许多学科无法比拟的。众所周知，图灵奖被誉为“计算机界的诺贝尔奖”，自 1966 年设立至今，54 位获奖者中就有 16 位是由于程序设计语言或编译技术的研究成果而获此殊荣的，见表 1-1。虽然有些大师的身影已经渐渐远去，但是他们的研究成果却为人津津乐道。

表 1-1　因程序设计语言或编译技术而获图灵奖的科学家

年份	获 奖 者	获 奖 原 因
1966	Alan J. Perlis	先进编程技术和编译架构方面的贡献
1971	John McCarthy	Lisp 语言、程序语义、程序理论、人工智能方面的贡献
1972	E. W. Dijkstra	对开发 Algol 做出了原理性贡献
1977	John Backus	在高级语言方面所做出的具有广泛和深远意义的贡献，特别是在 Fortran 语言方面
1979	Kenneth E. Iverson	在编程语言的理论和实践方面（特别是 APL）所进行的开创性的工作
1980	C. A. R. Hoare	在编程语言的定义和设计方面的贡献，如 case 语句、公理语义学等
1983	Ken Thompson, Dennis M. Ritchie	在通用操作系统理论研究，特别是 UNIX 操作系统的实现上的贡献。开发实现了 C 语言
1984	Niklaus Wirth	开发了 Euler、Algol-W、Modula 和 Pascal 一系列崭新的计算语言
1991	Robin Milner	在可计算函数逻辑（LCF）、ML 和并行理论（CCS）这三个方面突出的贡献
2001	Ole-Johan Dahl、Kristen Nygaard	面向对象编程始发于他们基础性的构想，这些构想集中体现在他们所设计的编程语言 Simula I 和 Simula 67 中
2003	Alan Kay	在面向对象语言方面的原创性思想，领导了 Smalltalk 的开发团队，以及对 PC 的基础性贡献
2005	Peter Naur	在设计 Algol 60 程序设计语言上的贡献。Algol 60 语言定义清晰，是许多现代程序设计语言的原型
2006	Frances Allen	对于优化编译器技术的理论和实践做出的开创性贡献，这些技术为现代优化编译器和自动并行执行打下了基础
2008	Barbara Liskov	在计算机程序语言设计方面的开创性工作。开发了面向对象编程语言 CLU

注：见《ACM 图灵奖:计算机发展史的缩影（1966-2006）》(第三版)，吴鹤龄，崔林，高等教育出版社。

从 20 世纪 50 年代到 80 年代末，一些程序设计语言的兴起，为编译技术的发展提供了新的契机。Ada、Fortran、Algol、Pascal、C、C++等高级语言编译器如雨后春笋一般诞生。作为编译领域的经典之作，“龙书”也是撰写于这一时期的，它的出现将原先模糊的理论体系与实现技巧完全梳理清晰了。正在人们隐约感到编译技术（尤其是前端技术）已相对成熟之际，一个崭新的时代正悄然来临。自 20 世纪 90 年代至今，计算机硬件体系的飞速发展，把编译技术的研究推向了新的高峰。

编译器是将一种程序设计语言编写的源程序等价地转换为另一种程序设计语言编写的源程序的系统软件。习惯上，将前者称为源语言，而将后者称为目标语言。读者应该对语言并不陌生，汉语、英语、C、Java 等都是语言。那么，它们的联系与区别是什么呢？

大千世界的语言一般可以分为两类，即自然语言和人工语言。自然语言就是日常生活中使用的语言，如汉语、英语。自然语言通常是在交流过程中逐渐完善的，并不是刻意定义的。人工语言则是为了特定目的、用途，而人为创造出来的语言。例如，在计算机程序设计中使用的语言、工程技术中的符号、图形语言等。实际上，读者可以将人工语言理解成为解决某一特殊问题而定义的一种语言，例如 C、Java 等就是为了解决计算机程序设计而定义的人工语言。不过，自然语言与人工语言之间并非如人们所想象的那样存在着严格的界限。虽然自然语言没有太多的人为主观因素，但是它确实也是由人类创造的。因此，读者只需了解这两个概念即可，不必过多深究。

早在 20 世纪 50 年代，计算机科学家就开始对语言处理进行了深入的研究，其中包含最重要的两个领域：一是自然语言理解与处理，二是程序设计语言及其编译技术。前者主要研究计算机如何才能正确识别与处理自然语言及其语义。而后者主要研究如何设计一种人工语言，使之有效地完成人与计算机之间的无障碍交流，计算机之所以能普及到千家万户与程序设计语言的发展是分不开的。不过，两者的研究至今仍然尚待完善，例如，至今人类还无法使用计算机自动完成英语句子到汉语句子准确无误的翻译。这主要是由于无法让计算机准确理解输入英语句子的语义，同时，也无法让计算机将预定语义使用汉语表达输出。这就需要在自然语言理解与处理领域的继续探索。

自然语言理解与处理不是本书的研究对象，笔者不多作讨论。下面，来谈谈程序设计语言及其编译技术的话题。这方面的研究主要包括：程序设计语言设计与定义、高级语言编译器设计、编译优化技术、并行编译技术、嵌入式编译技术、动态编译技术等。读者阅读完本书之后，可以根据个人兴趣选择其中某些领域作深入学习与研究。

程序设计语言是一种人工语言，而人工语言的一个重要特点就是人们可以对语言作一些严格的规定，从而不必过多关注语言表达形式的不确定性。例如，在 C 语言中，表达式 a= =10 只能用于描述“变量 a 是否等于 10”这一种语义。然而，在自然语言中，一个句子存在两种以上语义的情况并不罕见。例如，“我看见他太激动了。”这个句子表达的含义到底是什么？

可以理解为：　　　　我看见他后，我太激动了。

也可理解为：　　　　我看见他时，他非常激动。

这里“看见”的宾语选择使句子存在二义性。然而，同样的问题在程序设计语言中却很罕见。例如，在 C 语言中，对于 a+++b 这样的表达式，读者可能会疑惑运算结果到底应该

是 (a++) +b 还是 a+ (++b)。但根据 C 语言的约定，运算结果一定是(a++)+b。由此可见，在设计一门程序语言及其编译器时，避免二义性是设计者考虑的首要因素。至于为什么一定是这样的结果呢？当学习完第 2 章，读者一定会了解 C 语言这一约定的现实意义与必要性。当然，这并不意味着程序设计语言中是绝对不存在二义性的。以 C 语言为例，(++a)+(++a)的取值就将因编译器而异。

迄今为止，程序设计语言仍是人类与计算机交流的主要途径，它的应用领域也仅限于操纵与控制计算机。从第一台计算机诞生之日起，人类就始终在探索一种有效的方式与计算机进行对话交流，使之能为人类服务。虽然时隔数十年，计算机能识别与处理的语言仍然是二进制形式的机器语言描述的源程序。当然，不可否认二进制机器语言的优点非常多，但机器语言的易用性差也是不可回避的。即使是计算机专家想直接使用机器语言与计算机进行交流也是非常困难的。在 20 世纪 50 年代，计算机科学家们就已经意识到必须解决这一棘手的问题，否则计算机将无法得到普及。经过多年努力，汇编语言、C 、Pascal 等程序设计语言终于横空出世。根据语言的形式与特点，习惯上，将机器语言与汇编语言（一种比较接近机器语言的程序设计语言）称为低级语言，将其余的C 、Pascal 之类的语言称为高级语言。读者必须注意，低级语言与高级语言之分并不是说明语言本身的优劣，仅仅是说明语言的形式与机器语言的相似程度。所谓低级语言指的是与机器语言比较类似的语言，而高级语言指的是与机器语言差别较大而与自然语言比较类似的语言。由于这些非机器语言的诞生，也就出现了将非机器语言等价翻译成机器语言的需求。显然，将非机器语言翻译成机器语言的工作不能由手工完成，否则，非机器语言的产生就没有任何意义了。因此，人们试图借助于一个程序工具自动完成翻译工作。根据语言不同，翻译工具的复杂程度也不尽相同。比如，汇编语言比较接近机器语言，所以其翻译工具较易实现。而高级语言与机器语言差别较大，所以其翻译工具的实现也较为复杂。习惯上，将前者称为汇编器，而将后者称为编译器。当然，有些书上对于汇编器与编译器并没有严格区分，都将其称为编译器，反正这只是一个名词而已，读者不必深究。编译器的源语言是一种较为高级的程序设计语言，而目标语言可以是汇编语言、机器语言或者另一种高级语言。笔者必须澄清一点，人们普遍认为编译器的目标语言就是低级语言，这个观点的确没有错，但并不完整。实际上，有些编译器的目标语言可能是某一种已存在的高级语言，例如，第一个由 Bjarne Stroustrup 开发的 C++编译器的目标语言就是当时的 C 语言。假设未来的计算机 CPU 能直接处理 C 或者 Java 语言，那么，低级语言、高级语言及编译器的概念也将被重新诠释。

今天，虽然计算机的 CPU 可以达到天文级的处理速度，但是识别的指令数量却不会超过 1000 条（PC 的指令规模一般只有 300~500 条左右)。无论多么深奥、华丽的高级语言源代码最终必将被这近千条机器指令等价替换。相对于自然语言而言，程序设计语言的表达能力要小得多。不过，程序设计语言也有其自身的特性，比如，程序设计语言必须简单易学。不可能要求程序员用十年磨一剑的精神像学习英语一样去学习 C 语言。过于复杂的语言并不是经典的程序设计语言，更不能广为流传。在程序设计语言发展的历程中，这种例子并不罕见。

以上主要讨论了语言及程序设计语言的一些基本常识。程序设计语言的设计是一门技术，更是一门艺术，优秀的程序设计语言可以广为流传数十年之久。实际上，

Algol、Lisp、Fortran、Pascal、C、Smalltalk、APL、ML、Simula、Modula-1/2、PL360 等经典语言都是历经了时间的考验才传承至今的。设计程序设计语言及编译器是一项富有挑战性且能带来无限成就感的工作，希望本书能引领读者进入程序设计语言的殿堂。

1.1.2 程序设计语言的翻译机制

在日常生活中，对于两个不同体系的自然语言而言，翻译工作可能是相当复杂的。众所周知，在语法、语义等体系上，不同的自然语言之间都存在着巨大差异，因此，至今自然语言的翻译工作仍然是以手工完成为主。当然，随着对自然语言理解的深入研究，或许在不久的将来计算机准确翻译自然语言会成为可能。那么，作为一种人工语言，程序设计语言的翻译又是如何进行的呢？

程序设计语言的复杂性远不及自然语言，因此，试图让计算机完成程序设计语言的翻译的想法并非奢望。从 20 世纪 60 年代开始，许多计算机大师就着眼于高级程序设计语言及其翻译工具的研究，成就了不少优秀的程序语言，例如读者熟知的 C、Pascal 等。在没有任何理论基础的情况下，大师们进行了艰苦的探索，构造了程序设计语言翻译体系的雏型。在经典编译技术中，程序设计语言的翻译模型主要有两种：解释与编译。

首先，笔者简单介绍一下“解释”。运用解释方法进行翻译的高级语言工具被称为解释程序。解释程序是高级语言翻译程序的一种，它将某一高级语言源程序作为输入，解释一句后就提交计算机执行一句，直至源程序读取完毕。解释程序在计算机应用中并不少见，例如，早期的 BASIC 语言的翻译程序就是解释程序。另外，网络浏览器也是一种解释程序，它是将 HTML 以及 JavaScript 等脚本程序进行解释执行的。再如，UNIX/Linux 的 Shell 也是解释执行的。

下面，再来谈谈“编译”。运用编译方法进行翻译的高级语言翻译器称为编译器（或称为编译程序），这也是本书讨论的重点。虽然编译器的目标语言并不唯一，但是，汇编语言或机器语言是最为常用的目标语言。通常，编译器指的就是把用高级语言源程序翻译成等价的计算机汇编语言或机器语言的目标程序的翻译程序。编译器把高级语言源程序作为输入，而以汇编语言或机器语言的目标程序作为输出。因此，在编译完成后，编译器会生成相应的目标程序。编译器是程序设计语言翻译中应用极广的软件，例如，早期的 Turbo C、Turbo Pascal，现代的 Delphi、VC++等都是编译器。

这里，笔者必须指出，“解释”与“编译”只是两种翻译方法或者说是方案，并不能直接与某种程序设计语言挂钩。例如，有些书认为 BASIC 语言是解释语言、C 语言是编译语言。严格地说，这种说法并不正确。笔者认为程序设计语言本身并不存在解释、编译之分，C 语言一样可以运用解释的方法设计它的翻译程序，BASIC 语言同样可以运用编译方法设计它的翻译程序。

随着人们对程序设计语言翻译技术的不断深入研究，有一种新的翻译方法逐步被人们接受，就是编译与解释相结合的方法。这种翻译方法的思路是先通过编译生成一个目标程序，但这一目标程序并不是真正的机器语言程序，而是一种编译器设计人员定义的一个较低级语言描述的程序（可能近似于汇编语言）。这一程序再经过相应的解释器解释执行，如图 1-1 所示。早期的 Lisp 语言就是应用这种翻译方式实现的，它使用一种类似于语法树形式的语

言作为目标程序的描述语言。

图 1-1 “编译＋解释”翻译过程

近年来，人们热衷于编译与解释相结合的方法探索，提出了两种改进方案：即时编译、动态编译。

即时编译：先把源程序翻译成一种比较低级的内部语言描述的程序，然后，在执行装载程序时，即时地将程序编译为目标机器的本地机器语言程序，然后直接执行机器语言程序。这种实现方案被广泛应用，例如，Java 语言编译器和微软的.NET 开发平台都是应用这种方法实现的。Java 生成的目标程序是称为字节码的一种较低级语言的程序。而.NET 开发平台生成的目标程序是面向 CLR（Common Language Runtime）的程序。

动态编译：为了避免即时编译的运行代价，开始时正常解释执行，在执行中，检查执行的热点（如循环、递归调用等），发现热点后动态编译这部分代码，生成本机的机器语言程序以提高执行效率。

关于三种翻译方式的优缺点在学术界争论已久，本书不进行深入分析，只是列出几个公认的观点，见表 1-2。笔者认为，作为一位有志于深入学习与研究编译技术的读者，了解翻译方式的基本知识还是有必要的。

表 1-2　翻译方式的特点

	执 行 效 率	执行过程中的灵活性	源代码保密性
纯“解释”方式	慢	高	完全公开
纯“编译”方式	快	低	保密
“编译＋解释”方式	一般	一般	一般

1.1.3　编译器的基本结构

作为系统软件，编译器的设计与实现是非常复杂的。对于一个相对复杂的系统，通常的解决方法是将系统分解成若干较小且便于处理的小系统，分别实现后将其组织成一个完整的复杂系统，这就是“分治法”的思想。实际上，计算机科学家正是运用这种思想来设计与实现编译器、操作系统、网络通信协议等复杂的大型系统软件的。

编译器的翻译过程是非常复杂的，但就过程本身而言，与自然语言翻译却有不少相近之处。例如，把英语句子翻译为汉语句子时，通常需要经过下列几个步骤：

1）对句子中的每个英语单词进行识别。

2）对句子的语法结构进行分析。

3）分析句子的基本含义，进行初步翻译。

4）修饰译文，使之更加符合汉语的表达习惯。

5）将译文整理书写记录。

编译器的工作过程与自然语言翻译过程比较类似，亦可划分为五个阶段：词法分析、语法分析、语义分析与中间表示生成、代码优化、代码生成。

1．词法分析

词法分析的任务就是对输入的源程序进行扫描分析，识别出一个个的单词（Token），并进行归类。这里的“单词”可以理解为源程序中具有独立含义的不可分割的字符序列，与自然语言中的单词概念有一定区别。一般而言，根据程序设计语言的特点，单词可以分为五类：关键字、标识符、常量、运算符、界符。以一个 C 语言的条件语句为例：

```
if (aa && 10==0) aa=100;
```

词法分析的结果是识别出如下的单词符号：

关键字	界符	标识符	运算符	常量	运算符
if	(	aa	&&	10	==
常量	界符	标识符	运算符	常量	界符
0	)	aa	=	100	;

这里，读者只需了解词法分析的任务即可。其算法实现将在第 2 章中详述。

2．语法分析

语法分析的任务就是在词法分析的基础上，根据程序设计语言的语法规则（文法），把单词流分解成各类语法单位（语法范畴），如“语句”、“表达式”等。理论上讲，通过语法分析，编译器可以准确无误地判断输入源程序是否满足语言的语法规则。例如，语法分析可以判断如下语句是错误的。

```
if aa %% 10==9 aa=100;
for(i<0) i++;
```

不过，实际情况并非完全如此，这主要与文法定义的细化程度有直接的关系。当程序设计语言的设计人员把文法定义得比较宽泛时，也就意味着依据此语法规则，编译器不能在语法分析阶段发现所有的语法错误，只能将错误遗留给后续阶段处理。表面上看，语法分析并不像词法分析有直观的输出结果，而仅仅完成了输入源程序的语法判定工作。实际上，语法分析是编译器前面三个阶段（合称为前端）的主控模块。语法分析的设计思想与算法实现是经典编译理论重点讨论的话题，将在第 3 章中详述。

3．语义分析与中间表示生成

语义分析与中间表示生成的任务就是在语法分析的基础上，分析各语法单位的含义，并进行初步的翻译，即生成中间表示形式。有时，这两个任务是密不可分的，故通常将其合并为一个阶段讨论。语义分析主要是检查输入源程序的语义是否正确，例如，变量使用前是否定义、同一作用域内变量是否重名等。中间表示生成将根据输入源程序的语义生成语义等价中间表示形式。中间表示是一种由编译器设计人员定义、使用的形式，对于用户是完全透明的。中间表示形式的定义是值得深入研究的，因为它直接决定了编译器前、后端的设计复杂度，也决定了编译器前端与目标语言之间的耦合程度。中间表示的形式也非常多，包括四元组、三元组、语法树、DAG 图等，并不一定是读者理