21世纪高等学校计算机专业
核心课程规划教材

编译原理及实践教程（第3版）

◎ 黄贤英 王柯柯 曹琼 魏星 编著

清华大学出版社

21世纪高等学校计算机专业
核心课程规划教材

编译原理及实践教程（第3版）

◎ 黄贤英 王树柯 曹琼 魏星 编著

清华大学出版社
北京

内 容 简 介

本书主要讲述设计和构造编译程序的一般原理、基本设计方法和主要实现技术，以高级语言程序编译的6个主要阶段——词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成为线索，阐述了各阶段的主要功能、原理、设计技术和实现方法。

本书适合作为工程实践型、应用型本科院校计算机相关专业的教材，也适合作为工程技术人员的参考书。

图书在版编目(CIP)数据

编译原理及实践教程/黄贤英等编著. —3版. —北京：清华大学出版社，2019
(21世纪高等学校计算机专业核心课程规划教材)
ISBN 978-7-302-52007-8

Ⅰ. ①编… Ⅱ. ①黄… Ⅲ. ①编译程序－程序设计－高等学校－教材 Ⅳ. ①TP314

中国版本图书馆CIP数据核字(2018)第297271号

责任编辑：付弘宇　张爱华
封面设计：刘　键
责任校对：时翠兰
责任印制：丛怀宇

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：北京嘉实印刷有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：20.25　　**字　　数**：493千字
版　　次：2008年2月第1版　2019年4月第3版　　**印　　次**：2019年4月第1次印刷
印　　数：18901～20400
定　　价：49.80元

产品编号：080239-01

前言

编译程序在计算机科学与技术的发展历史中发挥着巨大作用，是计算机系统的核心支撑软件。编译原理蕴含着计算机学科中解决问题的思路、形式化问题和解决问题的方法，对应用软件和系统软件的设计和开发有一定的启发和指导作用。构造编译程序所涉及的方法和技术在软件工程、语言转换等许多领域中有广泛的应用。

本书主要讲述设计和构造编译程序的一般原理、基本方法和主要实现技术，贯穿高级语言、系统环境、体系结构和目标代码，体现了从软件到硬件的整机概念。以高级语言程序编译的6个主要阶段——词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成为线索，阐述了各阶段的主要功能、原理、设计技术和实现方法。

为适应新工科建设的需要，本书的修订基于OBE的理念，将编译的基本理论与具体实现技术有机地结合起来，既注重理论的完整性，又将理论融于具体实例中。书中的实例具有连贯性，力求让读者建立一个完整的编译系统的模型，加深对程序设计语言的理解，掌握常用的编译技术和方法，构建一个具有一定规模的完整的编译程序，为今后从事应用软件和系统软件的开发打下一定的理论和实践基础。

本书第3版延续了前两个版本的风格和主体内容，与前两个版本衔接得比较好；同时对一些章节进行了适当的充实、删减和重新组织，力求在各主要知识点之间达到较为合理的均衡，使读者对编译程序的构造方法和实现技术能从整体上全面地掌握。第3版修改的内容主要有：

(1) 由于C语言的广泛使用，本书选用的源语言改为C语言的子集。

(2) 在第1章中增加了对高级语言的认识。在后面的章节中逐步对源语言进行分析，以便读者在了解编译方法的基础上，从高级语言的使用者过渡到高级语言的实现者和设计者。

(3) 增加了语义分析的内容及方法，使编译程序的结构更清晰。

(4) 细化了目标代码生成。目标代码选用Intel 80x86汇编代码，降低学习的难度；生成的汇编代码能直接通过常见汇编器(masm)汇编成可执行文件，直观看到运行结果，加深对整个编译过程的理解。

(5) 函数是C语言的精髓，本书增加了函数的声明、定义和调用的编译过程，并以实例展示了C语言函数的详细执行过程及内存的变化，使读者对程序的运行环境有更透彻的认识，加深对计算机系统的理解。

本书主要面向以工程实践、应用为主的本科院校，建议理论学时为32～40学时，实验学

时为16～24学时，根据需要可安排专门的课程设计。本书中加＊的章节为较难的可选内容，教师可根据具体情况选择。本书也可作为工程技术人员的参考书。

本书参考和引用了国内外大量优秀编译教材和著作中的相关内容，也参考了网络上的相关内容，在此谨向原书作(译)者深表敬意和感谢；感谢中国科学技术大学物理学院张智浩同学，他根据本书内容完整地实现了一个编译程序，验证了本书的所有算法和思想；同时感谢刘恒洋老师在本书配套的教学辅助系统的可视化方面所做的工作；感谢重庆理工大学研究生阳安志、刘野和刘广峰等对本书提出的宝贵意见和建议。

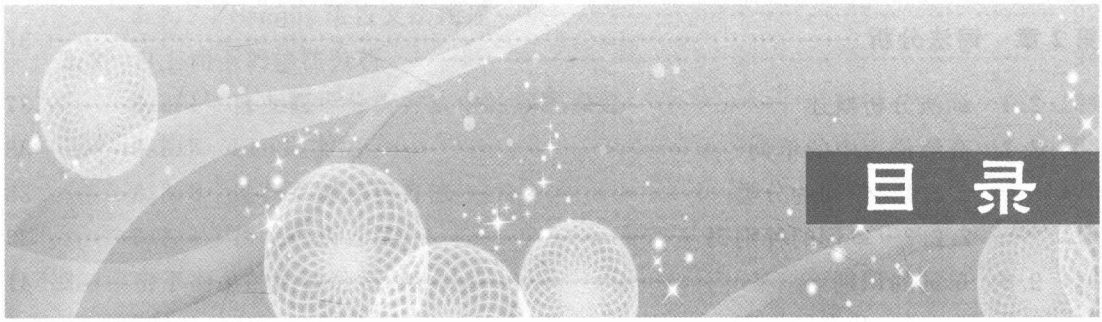
本书获得了重庆理工大学教材出版基金的资助。使用本书第1版、第2版的院校的教师和学生也为本书的改版提出了宝贵意见和建议，在此也表示衷心的感谢。

由于作者水平有限，书中难免存在疏漏之处，恳请广大读者批评指正。

本书的配套课件和源代码等资源可以从清华大学出版社网站(www.tup.com.cn)下载，如果遇到资源下载与使用的问题，请联系本书的编辑，邮箱为404905510@qq.com。

编 者

2018年12月



目 录

第 1 章　编译概述 …… 1

1.1　程序设计语言及翻译程序 …… 1
1.1.1　程序设计语言的发展 …… 1
1.1.2　翻译程序大家族 …… 2
1.1.3　高级语言的运行方式 …… 3
1.2　编译系统 …… 5
1.2.1　高级语言编译流程 …… 5
1.2.2　高级语言编译实例 …… 7
1.3　编译过程和编译程序的结构 …… 12
1.3.1　编译过程概述 …… 12
1.3.2　编译程序的结构 …… 18
1.3.3　编译阶段的组合 …… 19
1.4　编译程序的构造方法 …… 21
1.5　认识 Sample 语言 …… 22
1.5.1　高级语言的构成成分 …… 22
1.5.2　程序的结构 …… 23
1.5.3　Sample 语言规范 …… 26
1.5.4　符合 Sample 语言规范的源程序举例 …… 30
1.6　编译程序的发展及编译技术的应用 …… 31
1.6.1　编译程序的发展 …… 31
1.6.2　编译技术的应用 …… 31
1.6.3　为什么要学习编译原理及其构造技术 …… 33
1.7　本书结构 …… 34
1.8　小结 …… 35
1.9　习题 …… 35

第 2 章 词法分析 …… 37

2.1 词法分析概述 …… 37
2.2 高级语言中的单词 …… 38
2.2.1 单词的分类 …… 38
2.2.2 单词的种别码 …… 39
2.3 单词的识别 …… 41
2.3.1 状态转换图 …… 41
2.3.2 单词识别程序 …… 41
2.3.3 超前搜索技术和双界符的识别 …… 44
2.3.4 数值型常量的识别与状态转换图的合并 …… 45
2.4 词法分析器的设计 …… 47
2.5 正则表达式与有穷自动机 …… 49
2.5.1 符号和符号串 …… 50
2.5.2 集合的运算及语言的定义 …… 50
2.5.3 正则表达式 …… 52
2.5.4 有穷自动机 …… 55
2.5.5 正则表达式与有穷自动机的等价性 …… 64
2.6 词法分析器的自动生成工具 …… 67
2.6.1 Lex 概述 …… 67
2.6.2 Lex 源文件的书写 …… 68
2.6.3 Lex 的工作原理 …… 73
2.6.4 Lex 使用中的一些注意事项 …… 75
2.6.5 使用 Lex 自动生成词法分析器 …… 76
2.7 词法分析中的错误处理 …… 77
2.8 小结 …… 78
2.9 习题 …… 79

第 3 章 语法分析 …… 83

3.1 语法分析概述 …… 83
3.2 上下文无关文法 …… 84
3.2.1 文法的定义 …… 84
3.2.2 推导 …… 87
3.2.3 文法产生的语言 …… 88
3.2.4 语法树 …… 89
3.2.5 二义文法 …… 90
3.2.6 消除二义性 …… 91

*3.2.7 Sample语言文法描述 …… 92
3.3 自上而下的语法分析 …… 94
3.3.1 自上而下分析方法中的问题探究 …… 94
3.3.2 LL(1)文法 …… 98
3.3.3 递归下降分析方法 …… 103
3.3.4 预测分析方法 …… 110
3.4 自下而上的语法分析 …… 116
3.4.1 自下而上分析方法概述 …… 117
3.4.2 算符优先分析 …… 120
3.4.3 LR分析法 …… 130
3.5 语法分析器的自动生成工具YACC …… 153
3.5.1 YACC概述 …… 153
3.5.2 YACC源文件的格式 …… 154
3.5.3 YACC的翻译规则 …… 156
3.5.4 YACC的辅助程序 …… 157
3.6 语法分析中的错误处理 …… 157
3.6.1 语法分析中的错误处理的一般原则 …… 157
3.6.2 自上而下语法分析的错误处理 …… 158
3.6.3 自下而上语法分析的错误处理 …… 161
3.7 小结 …… 165
3.8 习题 …… 166

第4章 语义分析 …… 171

4.1 语义分析概述 …… 171
4.2 Sample语言的语义描述 …… 172
4.2.1 程序的语义 …… 172
4.2.2 函数的语义 …… 172
4.2.3 各种名字的声明和使用的语义 …… 173
4.2.4 各种语句的语义 …… 173
4.2.5 表达式的语义 …… 174
4.3 符号表管理技术 …… 175
4.3.1 符号表概述 …… 175
4.3.2 符号表的组织方式 …… 175
4.3.3 符号表的操作 …… 180
4.4 静态语义检查 …… 181
4.4.1 静态语义检查概述 …… 181
4.4.2 声明与定义语义检查 …… 182

4.4.3 表达式语义检查 …… 182
4.4.4 语句语义检查 …… 182
4.5 小结 …… 183
4.6 习题 …… 183

第 5 章 中间代码生成 …… 185

5.1 中间代码生成概述 …… 185
5.2 中间代码 …… 186
5.2.1 逆波兰式 …… 186
5.2.2 三地址代码 …… 187
*5.2.3 抽象语法树 …… 189
5.2.4 有向无环图表示 …… 190
5.3 属性文法和语法制导的翻译 …… 191
5.3.1 属性文法 …… 191
5.3.2 属性的计算 …… 193
5.3.3 属性的计算顺序 …… 194
5.3.4 语法制导翻译的实现方法 …… 196
5.4 常见语句的语法制导的翻译 …… 202
5.4.1 声明语句的语义处理 …… 202
5.4.2 表达式的翻译 …… 209
5.4.3 布尔表达式的翻译 …… 212
5.4.4 控制语句的翻译 …… 219
*5.4.5 函数定义及函数调用的翻译 …… 226
5.5 中间代码生成器的设计 …… 230
5.6 小结 …… 232
5.7 习题 …… 234

第 6 章 运行时存储组织 …… 236

6.1 存储组织 …… 236
6.1.1 程序执行时存储器的划分 …… 236
6.1.2 活动记录 …… 238
6.1.3 局部数据布局 …… 239
6.2 函数调用 …… 240
6.2.1 源程序中的函数 …… 240
6.2.2 函数执行时的活动 …… 241
6.2.3 名字的作用域 …… 243
6.2.4 参数的传递 …… 243

6.2.5 名字的绑定 …… 244
6.3 存储分配策略 …… 245
6.3.1 静态存储分配 …… 245
6.3.2 栈式存储分配 …… 248
6.3.3 堆式存储分配 …… 250
*6.4 垃圾回收机制 …… 253
6.4.1 可达性 …… 254
6.4.2 引用计数回收器 …… 254
6.4.3 标记-清扫回收器 …… 255
6.4.4 复制回收器 …… 256
6.5 C语言编译程序运行时存储实例 …… 257
6.5.1 内存的划分及程序执行的总体情况 …… 257
6.5.2 案例:程序运行时内存的变化 …… 258
6.6 小结 …… 262
6.7 习题 …… 262

第7章 代码优化 …… 265

7.1 代码优化概述 …… 265
7.1.1 代码优化的地位 …… 265
7.1.2 基本块的概念及流图 …… 266
7.2 局部优化 …… 269
7.2.1 删除公共子表达式 …… 270
7.2.2 复写传播 …… 270
7.2.3 删除无用代码 …… 270
7.2.4 代数恒等变换 …… 271
7.2.5 基本块的DAG表示及优化 …… 272
7.3 循环优化 …… 275
7.3.1 循环的定义 …… 276
7.3.2 代码外提 …… 277
7.3.3 强度削弱 …… 279
7.3.4 删除归纳变量 …… 279
7.4 小结 …… 280
7.5 习题 …… 280

第8章 目标代码生成 …… 284

8.1 概述 …… 284
8.2 目标机及指令系统简介 …… 286

8.2.1 80x86 体系结构 …… 286
8.2.2 80x86 中的寄存器 …… 287
8.2.3 80x86 指令系统介绍 …… 288
8.3 一个简单的代码生成器 …… 290
8.4 基本块的代码生成器 …… 292
8.4.1 引用信息和活跃信息 …… 292
8.4.2 寄存器描述和地址描述 …… 295
8.4.3 基本块的代码生成 …… 297
8.5 从 DAG 生成目标代码 …… 300
8.6 代码优化及目标代码生成器的设计 …… 302
8.6.1 目标代码生成器的结构 …… 303
8.6.2 汇编指令的选择 …… 304
8.6.3 构成完整的汇编语言程序 …… 308
8.7 小结 …… 309
8.8 习题 …… 309

参考文献 …… 311

第1章

编译概述

程序设计语言是人向计算机发送命令、描述计算过程的符号。程序设计语言有很多，计算机只能执行用机器语言编写的程序，用高级语言编写的程序不能直接在计算机上执行，要想执行它，需要通过相应的翻译程序(Translator)将其翻译为机器语言程序。编译程序就是这样一种翻译程序，它是现代计算机系统的基本组成部分，也是用户最关心的工具。编写编译程序所涉及的一些原理、技术和方法是计算机工作者所必须具备的基本知识，在计算机相关的各个领域中都有广泛的应用，学习它具有非常重要的意义。

本章首先介绍编译程序的基本概念及与之相关的一些工具，然后介绍编译的过程、编译程序的结构、编译阶段的组合以及编译程序的构造方法，了解高级语言的构成，最后简单介绍编译程序的发展及编译技术的应用。

1.1 程序设计语言及翻译程序

计算机的运行依赖于程序设计语言，因为计算机上运行的所有程序都是用某种程序设计语言编写的。只有机器语言书写的程序可以直接在计算机上执行，其他语言编写的程序必须翻译为机器语言程序才能执行。完成翻译任务的程序称为翻译程序。翻译程序的出现并不是偶然的，它的出现来源于人们编程解决实际问题的需求。本节将展示计算机上的程序设计语言及其翻译程序大家族，也将介绍高级语言的不同运行方式。

1.1.1 程序设计语言的发展

计算机是用来协助人们解决问题的工具。针对预定的任务，首先需要告诉计算机“做什么”“怎么做”，计算机才能自动处理，对给定的问题求解。为此，人们需要将有关信息告诉计算机，同时也需要计算机将计算的结果告诉人们。这样，人与计算机之间就要进行交流。正如人与人之间用语言进行交流一样，人们设计出词汇量少、语法简单、语义明确的程序设计语言(Programming Language)来实现人和计算机之间的交流。同时程序设计语言也是人与人之间的技术交流工具，在许多大型软件开发及软件维护中，程序员也需要读懂他人写的程序代码。

每当出现一种新的计算机，就随之产生一种该机器能理解并能直接执行的程序设计语言，这就是机器语言(Machine Language)。早期，人们直接用机器语言编写程序，机器语言是用二进制代码书写的、计算机能直接识别的机器指令的集合，CPU 可以直接执行，如“将2放到一个存储单元”的机器代码为“11000111 00000110 00000000 00000000 00000000 00000010”。用机器语言编程不直观，易出错，对硬件依赖性大，不同的 CPU 能执行的机器

语言不同,可移植性差;用机器语言编写程序费时、乏味,开发难度大;程序员必须受过一定的训练,熟悉计算机硬件,这在很大程度上限制了计算机的推广及应用。

随着计算机技术的发展,当需要用计算机解决的问题规模逐渐增大时,编程的工作量自然会变得非常繁重。为了提高程序的可读性和可写性,人们将机器语言符号化,以助记符的形式表示指令和地址,这就是汇编语言(Assembly Language),如"将2放到一个存储单元"的汇编代码为"MOV X, 2"。用汇编语言编写程序比用机器语言编写程序快很多,且更容易理解,但汇编语言必须翻译为机器语言才能在计算机上执行。这就需要有一个程序能够将汇编语言程序翻译为机器语言程序,这个翻译程序称为汇编程序(Assembler),又称汇编器。程序员只需要写出汇编代码,然后交给汇编器,就可以生成在计算机上可执行的机器语言程序。汇编器的出现把人们从烦琐的二进制代码中解放出来。虽然此时可以利用计算机处理一些更复杂的问题,然而汇编语言与机器语言是一一对应的,仍然依赖于机器,与人类的思维相差甚远,不易阅读和理解,程序设计的效率很低。

为了解决这些问题,1954—1957年,John Backus等人参照数学语言设计了第一个描述算法的语言(即FORTRAN语言),随后相继出现了许多语言,如ALGOL 60、C和Pascal等面向过程的语言,以及后来出现的面向问题的语言(如SQL)与面向对象的语言(如C++和Java)等。由于机器语言和汇编语言都是与机器有关的语言,通常称为低级语言(Low Level Language),将其他与机器无关的程序设计语言称为高级语言(High Level Language)。高级语言的出现缩短了人类思维和计算机语言之间的差距,编写高级语言程序类似于定义数学公式或书写自然语言,与机器无关,如"将2放到一个存储单元"用C语言书写就是"x=2",便于理解和学习。

用高级语言编程效率很高,但高级语言程序不能直接在计算机上执行,需要一个程序将高级语言(如C语言或Java、Pascal等)程序翻译为对应的低级语言(如汇编语言或机器语言)程序,这个翻译程序称为编译程序(Compiler),又称编译器。高级语言编译程序的出现使人们能够更容易地使用计算机,编程时不必考虑与机器有关的烦琐细节,使程序员独立于计算机硬件。

1.1.2 翻译程序大家族

实际上,除了需要将高级语言程序翻译为低级语言程序外,有时候为了完成工作的需要,同一种机器上的不同语言和不同种机器上的相同或不同语言书写的程序之间都可能需要进行相互翻译,这种能把一种语言(源语言,Source Language)书写的程序(源程序,Source Program)翻译为另一种语言(目标语言,Target Language)书写的程序(目标程序,Target Program)的程序统称为翻译程序。一般来说,翻译前后的程序在逻辑上是等价的。

程序设计语言很多,每种程序设计语言编写的程序在执行前,都必须翻译为机器语言程序。为了更充分地认识翻译程序,将程序设计语言按照离计算机硬件的远近分成三个层次:高级语言层、汇编语言层和机器语言层。各层次语言及其翻译程序之间的关系如图1.1所示。图1.1中方框表示不同的语言,椭圆框和平行四边形都表示翻译程序,二者的不同点在于翻译结果将在不同的计算机上运行。所有这些翻译程序构成了计算机语言翻译程序的大家族。

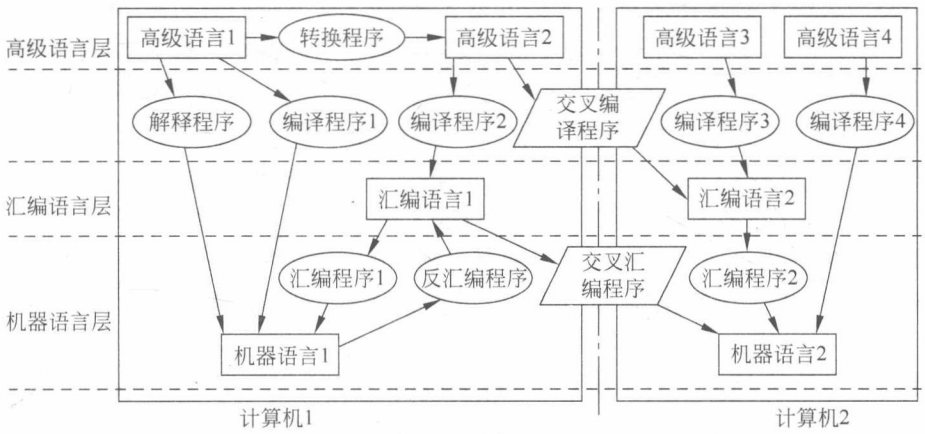


图 1.1　语言层次和翻译程序大家族

能将一种高级语言程序翻译为另一种高级语言程序的程序称为转换程序(Converter)。

从图 1.1 中可以看出,高级语言编译程序有两种翻译方式:直接翻译为机器语言程序和先翻译为汇编语言程序再翻译为机器语言程序。除了编译程序可以把高级语言程序翻译为机器语言程序外,在实际应用中,还有一种解释程序(Interpreter,又称解释器),它也能将高级语言程序翻译为机器语言程序再执行,但并不保存翻译结果。编译程序和解释程序的差别在 1.1.3 节中介绍。

汇编程序将汇编语言程序翻译为机器语言程序,反汇编程序(Disassembler)把机器语言程序逆向翻译为汇编语言程序。

假定图 1.1 中的计算机 1 和计算机 2 是指体系结构完全不同的两种计算机,如计算机 1 是基于 Intel 架构的计算机,计算机 2 是基于 MIPS 架构的计算机,两者的指令系统完全不同。交叉编译程序(Cross Compiler)能把一种计算机上的高级语言程序翻译为另一种计算机上的汇编语言或机器语言程序,交叉汇编程序(Cross Assembler)能把一种计算机上的汇编语言程序翻译为另一种计算机上的机器语言程序。如 keil 编译器能把 C 语言代码交叉编译为 51 单片机、ARM 等多个硬件平台上的机器代码。

1.1.3　高级语言的运行方式

从图 1.1 中可以看出,高级语言程序的运行有两种方式:一种称为编译方式,利用编译程序将高级语言程序翻译为机器语言程序,然后再运行这个机器语言程序;另一种方式称为解释方式,利用解释程序直接读取高级语言程序中的每个语句,翻译并直接执行。例如,Ruby 和 Perl 都是解释执行的。也就是说,通过编译运行的方式,编译和运行是两个阶段;通过解释运行的方式,翻译和运行是交叉进行的,一边读取源程序的语句,一边翻译,一边执行。图 1.2 所示是高级语言编译方式和解释方式的比较,具体如下。

(1) 编译程序是源程序的一个转换系统,解释程序是源程序的一个执行系统。也就是说,解释程序的工作结果是源程序的执行结果,而编译程序的工作结果是等价于源程序的某机器语言程序。

(2) 编译程序先把全部源程序翻译为目标程序,该目标程序可以反复执行;解释程序对源程序逐句地翻译执行,目标代码只能执行一次,若需要重新执行,则必须重新解释源程

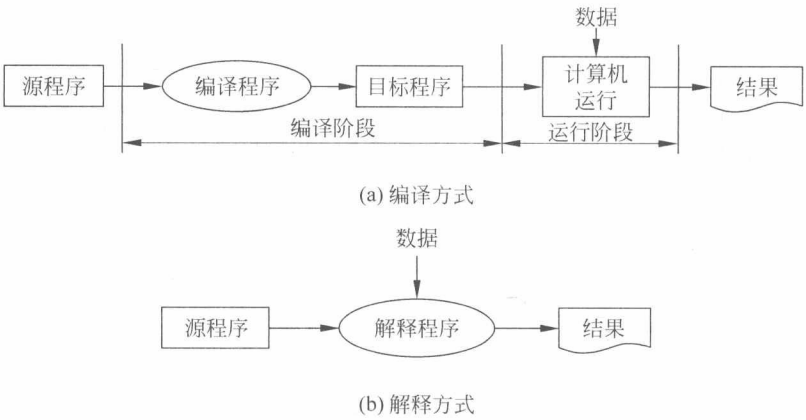


图 1.2 高级语言编译方式与解释方式执行的比较

序。编译过程类似于笔译,笔译后的结果可以反复阅读,而解释过程则类似于口译,别人说一句就译一句,翻译的结果没有保存下来。

(3) 解释程序比编译程序更加通用。解释程序一般是用高级语言编写的,能够在绝大多数类型的计算机上运行,编译程序生成的目标代码只能在特定类型的计算机上运行。现在很多脚本语言都是解释执行的。

(4) 通过编译运行,源程序和数据是在不同的时间进行处理的;通过解释运行,源程序和数据是同时处理的。

(5) 编译方式比解释方式执行快得多,因为编译方式在程序运行阶段就不需要再分析了。而解释器的错误诊断效果通常比编译器更好,因为它逐条语句地执行源程序。

其实,每种语言的执行方式可以不止一种,例如 C 语言程序一般通过编译方式来执行,也可以用解释器来解释执行;Ruby 语言程序一般通过解释方式来执行,也可以通过编译方式翻译为机器语言后再执行。也就是说,编程语言的执行方式是可以自由选择的。但是由于每种语言的特点不同,可以选择更适合它的执行方式,例如,当执行速度作为最重要的因素来考虑时就要使用编译方式,用编译方式比解释方式执行快得多,有时要快 10 倍以上;有静态类型检查、要求有更高可靠性的语言通常使用编译方式;相反,没有静态类型检查、对灵活性要求较高的语言可以采用解释方式执行。静态类型检查是指在程序执行前需要对函数的返回值、变量的类型、数组下标的范围、参数的类型等进行检查的方式;动态类型检查是指在程序执行过程中随时进行类型检查的方式。为了可移植性和效率,Java 语言采取了一种折中的执行方式:编译+解释。Java 语言自己定义了一种虚拟机代码 Bytecode,Java 程序首先通过编译方式翻译为 Bytecode,然后通过 Java 解释器解释执行 Bytecode,如图 1.3 所示。

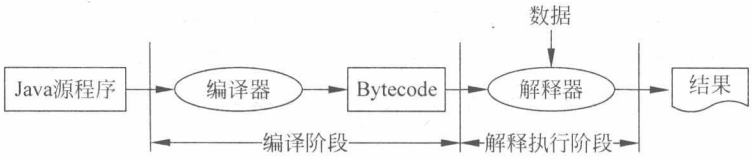


图 1.3 Java 的"编译+解释"执行方式

本书重点介绍编译程序及其相关原理、方法和技术，关于解释程序不做深入探讨。许多编译程序的构造与实现技术也同样适用于解释程序。

虽然编译程序的总体功能是实现高级语言到汇编语言(或机器语言)的翻译，但在实际应用中，根据用途和侧重点的不同，编译程序还可进一步分类为：专门用于帮助程序开发和调试的编译程序，称为诊断编译程序(Diagnostic Compiler)；着重于提高目标代码效率的编译程序，称为优化编译程序(Optimizing Compiler)。运行编译程序的计算机称为宿主机(Host)，运行编译程序所产生的目标代码的计算机称为目标机(Target)。如果不需要重写编译程序中与机器无关的部分就能改变目标机，则称该编译程序为可变目标编译程序(Retargetable Compiler)。

在图 1.1 中，编译程序实现从高级语言到机器语言的翻译有两种方案：一种是将高级语言程序直接翻译为机器语言程序；另一种是翻译为汇编语言程序，再使用已有的汇编器将汇编语言程序翻译为机器语言程序。本书后续介绍的编译程序使用第二种方案，将高级语言程序翻译为汇编语言程序，因为汇编语言程序可以很容易地通过汇编程序转换为机器语言程序。多数商业化编译程序也采用这种方式。

1.2　编 译 系 统

编译程序能够将高级语言程序翻译为汇编语言程序，进而翻译为机器语言程序。然而，如果只有编译程序，对用户来说，要执行一个高级语言程序还是不够的，这就需要有相关联的一系列程序和它一起工作，构成编译系统。高级语言编译系统是计算机系统软件最重要的组成部分之一。本节主要介绍高级语言程序编译的流程，并用实际例子介绍高级语言程序如何一步一步地转换为可执行程序。

1.2.1　高级语言编译流程

还记得自己编写第一个 C 语言程序“hello world!”的情景吗？当在开发环境中输入程序 1.1 所示的程序 hello.c 后，当时老师告诉你 C 语言代码只有经过“编译”(Compile)，才能在计算机中正确执行，让你单击一下“编译”和“运行”按钮，映入眼帘的那行字符串令你欣喜若狂！

```
//程序 1.1:hello.c
#include <stdio.h>
int main()
{
    printf("hello world! ");
    return 0;
}
```

后来你也多次重复着这一动作，计算机都能为你输出你希望的答案或者提示错误信息。然而你可能一直很疑惑，也很好奇：“编译”这个按钮背后都发生了什么？它是如何将 C 语言程序转换为计算机上的可执行程序的呢？其实这个过程很复杂，计算机在幕后要做一系列的工作，大致要经过 4 个阶段：预处理、编译、汇编和链接。它首先对 C 语言的源程序进行预处理，将其中的宏和预处理命令展开转换为标准 C 语言程序，然后进行编译，生成汇编

语言程序,再经过汇编程序汇编生成二进制目标代码,最后对目标代码进行链接,此时需要将相关的库函数和外部程序一起链接,生成可执行的机器代码,如图 1.4 所示。下面对这 4 个阶段进行简要说明。

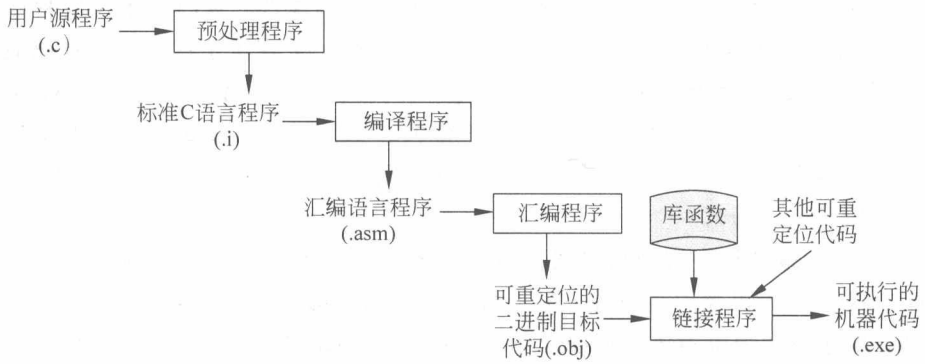


图 1.4 C 语言编译过程

1. 预处理

C 语言的预处理程序(Preprocessor)首先要对用户源程序中的宏定义和文件包含等进行处理。如在用户编写 C 语言源程序时使用了宏定义 # define max (a>b)? a:b,在预处理时需要由预处理程序将宏展开,凡是在源程序中遇到 max 的地方,全部用(a>b)? a:b 代替;如果在 C 语言源程序中使用文件包含语句 # include < stdio. h >,在预处理时,就将文件 stdio. h 中的内容插入到此处来替换此语句。有些预处理程序还能够处理语言功能的扩充,用更先进的控制结构和数据结构来增强原来语言的功能。当源程序中使用了该结构,预处理程序就把它转换为该语言编译程序能够识别的形式,加入到源程序中。有些预处理程序还可以将用户源程序的多个模块合并连接在一起。C 语言预处理的结果是标准的 C 语言程序。很多时候把预处理合并到编译中。如在 VC++ 6.0 中,通过预处理后,将. c 的源程序变为了. i 的文本文件。

2. 编译

标准的 C 语言程序由编译程序翻译为对应于某个计算机上的汇编语言程序。汇编语言是和机器语言一一对应的易于阅读的文本形式的语言。编译的结果是某种机器上汇编语言书写的程序。如在 VC++ 6.0 中,编译这一步将. i 的文本文件生成了. cod 的文本文件,这就是汇编代码。有的编译器生成以. s 或. asm 为扩展名的文件。

3. 汇编

汇编语言程序需要由汇编程序翻译为二进制的目标代码(机器语言描述的程序)。大多数编译器生成的目标代码是扩展名为. obj 的二进制文件。

当然,也有些编译程序不生成汇编代码,直接生成二进制目标代码,直接传给链接程序;还有些编译程序直接生成可执行代码。

4. 链接

目标文件本身还不能直接运行。链接程序(Linker)把目标文件转换为最终可以运行的程序。待链接的目标文件可以是多个,它们可以是分别通过编译或汇编得到的,也可以是系统提供的库文件,如程序 1.1 在链接时需要将包含 printf 的库文件的内容链接进来。链接