HZ BOOKS
华章教育

高等院校信息安全专业规划教材

# 编译与反编译技术

# Technology of Compiling and Decompiling

庞建民 陶红伟 刘晓楠 岳峰 编著

机械工业出版社
China Machine Press

高等院校信息安全专业规划教材

# 编译与反编译技术

## Technology of Compiling and Decompiling

庞建民 陶红伟 刘晓楠 岳峰 编著

机械工业出版社
China Machine Press

**图书在版编目（CIP）数据**

编译与反编译技术 / 庞建民等编著 . —北京：机械工业出版社，2016.4
（高等院校信息安全专业规划教材）

ISBN 978-7-111-53412-9

I. 编…　II. 庞…　III. 计算机网络 - 安全技术 - 高等学校 - 教材　IV. TP393.08

中国版本图书馆 CIP 数据核字（2016）第 062427 号

本书首先从正向角度介绍编译系统的一般原理和基本实现技术，主要内容有词法分析、语法分析、语义分析与处理、符号表、运行时存储组织、优化、目标代码生成和多样化编译等；然后从反向角度介绍反编译的相关原理和技术，包括反编译及其关键要素、反编译器的整体框架、反编译中的指令解码和语义描述与映射、反编译中的恢复技术、编译优化的反向处理、反编译与信息安全等。

本书可作为计算机及信息安全相关专业高年级本科生的教科书或教学参考书，也可供计算机相关专业研究生和从事编程或者软件逆向分析工作的工程技术人员参考。

出版发行：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：佘　洁 | 责任校对：董纪丽 |
| 印　　刷：北京诚信伟业印刷有限公司 | 版　　次：2016 年 4 月第 1 版第 1 次印刷 |
| 开　　本：185mm × 260mm　1/16 | 印　　张：25.25 |
| 书　　号：ISBN 978-7-111-53412-9 | 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　　投稿热线：（010）88379604

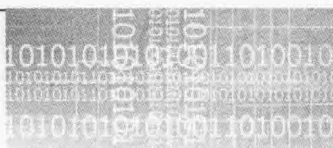
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

**版权所有 · 侵权必究**

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所　韩光 / 邹晓东

# 前言



“编译原理”是高等院校计算机科学与技术和软件工程专业的必修专业课之一，是一门理论与实践相结合的课程，对大学生科学思维的养成和解决实际问题能力的提高具有重要作用。“编译技术”是“编译原理”课程中介绍的关键技术，已经被广大计算机软件从业者所掌握和熟悉。“反编译技术”则是近几年得以迅速发展的新兴技术，许多计算机软件或信息安全从业者非常关心该项技术，但目前这方面的书籍较少，与“编译技术”结合起来讲解的更少。本书就是在这种需求以及编者在这两方面的科研实践体会的驱动下诞生的，目的是为计算机软件和信息安全从业者提供编译与反编译技术方面的知识和技能。

本书的编写得到了中国人民解放军信息工程大学和机械工业出版社的支持，在此表示诚挚的谢意。

本书由庞建民教授确定内容的选取和组织结构，由庞建民、陶红伟、刘晓楠、岳峰具体执笔，庞建民编写第1、9章，陶红伟编写第2、3、4、5、7章，刘晓楠编写第10、11、12、13章，岳峰编写第6、8、14、15章，最后由庞建民定稿。赵荣彩教授对本书的编写提出了许多宝贵的意见和建议，在此表示衷心的感谢。

本书力图反映编译与反编译及其相关领域的基础知识和发展方向，尝试用通用的语言讲述抽象的原理与技术，由于编者水平有限，书中难免有错误与欠妥之处，恳请读者批评指正。

编者

# 教学建议

| 教学内容 | 教学要点及教学要求 | 课时安排 | |
|---|---|---|---|
| | | 计算机及安全专业 | 非计算机专业 |
| 第1章 引论 | 熟悉编译器和解释器的概念，掌握二者的区别。了解编译的过程，熟悉编译器的结构、编译器的分类及其生成方式、高级语言的分类及其特点，理解编译前端和后端的概念，掌握C语言程序的编译流程，了解UNIX/Linux环境中make和makefile的概念及其应用 | 2～4 | 2～3（选讲） |
| 第2章 词法分析的理论与实践 | 了解词法分析器的功能以及输出形式，熟悉词法分析器的结构和超前搜索技术。掌握状态转换图及其实现、正规式与正规文法、NFA与DFA、正规式与有穷自动机的关系以及DFA的最小化。了解词法分析器的自动产生工具及其使用 | 6～7 | 5～6（选讲） |
| 第3章 语法分析 | 掌握上下文无关文法的相关概念，理解自上而下语法分析的概念，掌握LL(1)分析法、左递归和回溯的消除方法，学会构造预测分析程序。理解自下而上语法分析的概念，掌握LR(0)分析、SLR(1)分析、LR(1)分析和LALR(1)分析。学会使用YACC工具 | 8～10 | 6～7（选讲） |
| 第4章 语义分析与处理 | 理解语法制导定义和语法制导翻译模式的相关概念，理解语法制导翻译的基本思想。掌握属性计算的常用方法，包括基于依赖图的属性计算方法、基于树遍历的属性计算方法和基于一遍扫描的属性计算方法。掌握S-属性文法的自下而上计算、L-属性文法的自上而下翻译。理解中间语言的基本概念，掌握表达式的逆波兰表示法、DAG表示法、三地址代码。掌握说明语句的翻译、赋值语句的翻译、布尔表达式的翻译、控制语句的翻译和过程调用的处理 | 7～8 | 4～6（选讲） |
| 第5章 符号表 | 了解符号表的作用、内容、组织和实现等 | 1～2 | 1～2（选讲） |
| 第6章 运行时存储组织 | 熟悉程序运行时的存储区域划分，掌握静态存储分配、动态存储分配的思想。充分理解栈式动态存储分配中简单的栈式存储分配的实现和嵌套过程语言的栈式实现，能够分析程序运行时的栈的变化情况。了解堆式动态存储分配的两种途径：定长块管理、变长块管理。熟悉并掌握存储分配存在的安全性问题，充分理解缓冲区溢出的原理，了解相关的防范方法 | 3～4 | 2～4（选讲） |
| 第7章 优化 | 理解优化和基本块的基本概念。掌握将三地址语句序列划分为基本块的算法和以基本块为结点的控制流图构造方法。掌握常用的局部优化技术，包括删除公共子表达式、复写传播、删除无用代码、合并已知量、常数传播等。掌握基于基本块的DAG的局部优化<br>掌握如何利用程序的控制流程图来定义和查找循环，掌握常用的循环优化技术，包括循环展开、代码外提、强度削弱和删除归纳变量<br>了解进行数据流分析的几种常用方法，包括到达–定值数据流分析、活跃变量数据流分析和可用表达式数据流分析等，了解如何利用上述数据流分析结果进行全局范围内常数传播、合并已知量、删除公共子表达式和复写传播 | 6～8 | 4～6（选讲） |

试读结束：需要全本请在线购买： www.ertongbook.com

（续）

| 教学内容 | 教学要点及教学要求 | 课时安排 | |
|---|---|---|---|
| | | 计算机及安全专业 | 非计算机专业 |
| 第 8 章 目标代码生成 | 熟悉并掌握代码生成器设计中的问题，掌握线性扫描的寄存器分配方法的思想，并充分理解线性扫描寄存器分配算法。了解图着色的寄存器分配算法的思想及典型的实现过程，熟悉并掌握窥孔优化的三种典型方法，能够分析简单的代码生成过程 | 5 ～ 6 | 3 ～ 4（选讲） |
| 第 9 章 多样化编译 | 了解软件多样化的需求，特别是安全方面的需求；掌握多变体代码的特点、执行环境。理解海量软件多样性的概念及其目的，掌握多样化编译所涉及的多项技术，了解多样化编译技术的实现和应用范围 | 2 ～ 4 | 1 ～ 2（选讲） |
| 第 10 章 反编译及其关键要素 | 熟悉并掌握反编译的概念，其与编译的关系，以及反编译器的构成。熟悉反编译的基本过程，了解反编译技术的发展历程。熟悉反编译技术的局限、先决条件和评价指标，了解反编译的应用领域和研究重点 | 4 ～ 5 | 2 ～ 4（选讲） |
| 第 11 章 反编译器的整体框架 | 熟悉并掌握经典的、纯粹的反编译器的框架设计，了解经典多源反编译框架的基本构成。了解两款以反编译器为核心的二进制翻译系统的框架构造，熟悉从单一功能的反编译器到支持多源平台的反编译器，乃至利用反编译技术实现的静态二进制翻译器的设计思路、基本技术、软件系统的构造和主要功能 | 3 ～ 4 | 2 ～ 3 |
| 第 12 章 反编译中的指令解码和语义描述与映射 | 熟悉并掌握二进制 0/1 代码向汇编码转换过程中的主要知识：指令描述和解码。熟悉并掌握汇编级代码向中间表示转换过程中的基本知识：指令的语义映射 | 3 ～ 4 | 1 ～ 3（选讲） |
| 第 13 章 反编译中的恢复技术 | 熟悉数据流（或数据）恢复的过程，掌握高级控制流恢复的基本方法。掌握从低级代码中识别并还原成高级语言中的过程和函数的主要方法 | 3 ～ 4 | 2 ～ 3（选讲） |
| 第 14 章 编译优化的反向处理 | 了解常用的编译优化方法，熟悉并掌握谓词执行的概念，掌握谓词消除的方法 | 3 ～ 4 | 2 ～ 3（选讲） |
| 第 15 章 反编译与信息安全 | 了解恶意代码检测的背景，熟悉并掌握反编译技术在三种层次上的行为提取方法。了解基于推理的程序恶意性分析系统及功能模块 | 4 ～ 6 | 3 ～ 4 |
| 教学总学时建议 | | 60 ～ 80 | 40 ～ 60 |

说明：

1. 计算机或信息安全专业本科教学使用本教材时，建议课堂授课学时数为 60 ~ 80（包含习题课、课堂讨论等必要的课堂教学环节，实验另行安排学时），不同学校可以根据各自的教学要求和计划学时数酌情对教材内容进行取舍。
2. 非计算机专业的师生使用本教材时可适当降低教学要求。若授课学时数少于 60，建议主要学习第 1 章、第 2 章、第 3 章、第 4 章、第 6 章、第 7 章、第 8 章、第 10 章、第 11 章、第 12 章，第 13 章、第 15 章的内容可以适当简化，第 5 章、第 9 章、第 14 章可以不做要求。

# 目录

前言
教学建议
第1章　引论 ………… 1
1.1　编译器与解释器 ………… 1
1.2　编译过程 ………… 2
1.3　编译器结构 ………… 2
1.4　编译器的分类及生成 ………… 3
1.5　高级语言及其分类 ………… 3
1.6　编译的前端和后端 ………… 4
1.7　C语言程序的编译流程 ………… 4
1.8　UNIX/Linux环境中的make和makefile ………… 7
1.9　本章小结 ………… 12
习题 ………… 12
第2章　词法分析的理论与实践 ………… 13
2.1　词法分析器的需求分析 ………… 13
2.1.1　词法分析器的功能 ………… 13
2.1.2　分离词法分析的原因 ………… 14
2.2　词法分析器的设计 ………… 15
2.2.1　输入及其处理 ………… 15
2.2.2　单词符号的描述：正规文法和正规式 ………… 15
2.2.3　单词符号的识别：超前搜索 ………… 21
2.2.4　状态转换图及其实现 ………… 22
2.3　有穷自动机 ………… 28
2.3.1　确定的有穷自动机 ………… 28
2.3.2　非确定的有穷自动机 ………… 29
2.3.3　NFA到DFA的转化 ………… 29
2.3.4　DFA的化简 ………… 31
2.4　正规式和有穷自动机的等价性 ………… 33
2.5　词法分析器的生成器 ………… 35
2.6　本章小结 ………… 37
习题 ………… 37
第3章　语法分析 ………… 39
3.1　上下文无关文法 ………… 39
3.1.1　上下文无关文法的定义 ………… 39
3.1.2　语法树和推导 ………… 40
3.1.3　二义性 ………… 43
3.2　语法分析器的功能 ………… 45
3.3　自上而下的语法分析 ………… 45
3.3.1　LL(1)分析方法 ………… 45
3.3.2　预测分析程序 ………… 53
3.4　自下而上的语法分析 ………… 56
3.4.1　移进与归约 ………… 56
3.4.2　LR分析 ………… 57
3.4.3　LR(0)分析 ………… 60
3.4.4　SLR(1)分析 ………… 66
3.4.5　LR(1)分析 ………… 69
3.4.6　LALR(1)分析 ………… 72
3.4.7　分析方法比较 ………… 76
3.5　语法分析器的生成器 ………… 76
3.6　本章小结 ………… 78
习题 ………… 78
第4章　语义分析与处理 ………… 81
4.1　语法制导定义与语法制导翻译 ………… 82
4.2　中间语言 ………… 91
4.3　语句的翻译 ………… 95
4.3.1　说明语句的翻译 ………… 95
4.3.2　赋值语句的翻译 ………… 100
4.3.3　控制语句的翻译 ………… 106
4.3.4　过程调用语句的翻译 ………… 120

4.4 本章小结 …… 121
习题 …… 121
第 5 章 符号表 …… 124
5.1 符号表的作用 …… 124
5.2 符号表的内容 …… 125
5.3 符号表的组织 …… 127
5.4 符号表的实现 …… 129
5.5 名字的作用域 …… 132
5.6 本章小结 …… 135
习题 …… 135
第 6 章 运行时存储组织 …… 137
6.1 静态存储分配 …… 138
6.2 动态存储分配 …… 138
6.3 栈式动态存储分配 …… 140
6.3.1 简单的栈式存储分配的实现 …… 140
6.3.2 嵌套过程语言的栈式实现 …… 141
6.4 堆式动态存储分配 …… 145
6.5 存储分配与安全性 …… 146
6.5.1 缓冲区溢出原理 …… 146
6.5.2 缓冲区溢出的防范 …… 147
6.6 本章小结 …… 148
习题 …… 148
第 7 章 优化 …… 150
7.1 优化技术简介 …… 150
7.2 局部优化 …… 151
7.3 循环优化 …… 161
7.4 全局优化 …… 172
7.4.1 到达－定值数据流分析 …… 173
7.4.2 活跃变量数据流分析和定值－引用数据流分析 …… 178
7.4.3 可用表达式数据流分析 …… 182
7.4.4 复写传播数据流分析 …… 186
7.5 本章小结 …… 192
习题 …… 193
第 8 章 目标代码生成 …… 196
8.1 代码生成器设计中的问题 …… 196
8.1.1 代码生成器的输入 …… 196
8.1.2 目标程序 …… 196
8.1.3 指令选择 …… 197
8.1.4 变量存储空间分配 …… 197
8.2 寄存器分配 …… 197
8.2.1 寄存器分配描述 …… 198
8.2.2 线性扫描的寄存器分配 …… 199
8.2.3 图着色的寄存器分配 …… 201
8.3 窥孔优化 …… 202
8.3.1 规则提取 …… 202
8.3.2 扫描匹配和等价语义转换 …… 203
8.3.3 举例说明 …… 205
8.4 一个代码生成器实例 …… 205
8.4.1 待用信息和活跃信息 …… 206
8.4.2 寄存器描述和地址描述 …… 207
8.4.3 代码生成算法 …… 208
8.5 本章小结 …… 209
习题 …… 209
第 9 章 多样化编译 …… 210
9.1 软件多样化需求 …… 210
9.2 多变体执行及其环境 …… 211
9.3 海量软件多样性 …… 212
9.4 多样化编译技术 …… 213
9.5 多样化编译的应用 …… 216
9.6 本章小结 …… 217
习题 …… 217
第 10 章 反编译及其关键要素 …… 218
10.1 什么是反编译 …… 218
10.1.1 反编译概念 …… 218
10.1.2 编译与反编译 …… 219
10.1.3 反编译器 …… 219
10.2 反编译的基本过程 …… 219
10.2.1 按照反编译技术实施的顺序划分 …… 220
10.2.2 按照反编译实践中的具体操作划分 …… 224
10.2.3 按照反编译器的功能块划分 …… 227
10.3 反编译的前世今生 …… 228
10.3.1 建立——20 世纪 60 年代 …… 228
10.3.2 发展——20 世纪 70 年代 …… 229
10.3.3 瓶颈期——20 世纪 80 年代 …… 231
10.3.4 反编译的春天来了——20 世纪 90 年代 …… 232

10.3.5 持续的研究——进入21世纪 ························ 235
10.3.6 身边的反编译——我国对反编译的研究 ························ 236
10.4 反编译的局限、先决条件和评价指标 ························ 236
10.4.1 反编译技术面临的宏观问题 ························ 236
10.4.2 反编译技术面临的技术性问题 ························ 237
10.4.3 反编译的先决条件 ························ 238
10.4.4 反编译器的评价指标 ························ 238
10.5 反编译的应用领域和研究重点 ························ 239
10.5.1 应用领域 ························ 239
10.5.2 研究重点 ························ 239
10.6 本章小结 ························ 240
习题 ························ 240

**第11章 反编译器的整体框架** ························ 241
11.1 “I型”反编译器的框架 ························ 241
11.1.1 上下文环境的衔接 ························ 241
11.1.2 dcc反编译器的框架 ························ 242
11.2 经典多源反编译框架简介 ························ 243
11.2.1 UQBT ························ 243
11.2.2 Hex-Rays ························ 247
11.2.3 BAP ························ 247
11.3 具备静态反编译能力的二进制翻译器ITA ························ 248
11.3.1 ITA总体框架 ························ 248
11.3.2 二进制文件解码 ························ 249
11.3.3 语义映射 ························ 251
11.3.4 过程抽象分析 ························ 251
11.3.5 优化代码消除 ························ 251
11.3.6 C代码产生器 ························ 252
11.3.7 从ITA看静态反编译存在的普遍问题 ························ 252
11.3.8 对ITA静态反编译框架的扩展ITA-E ························ 253
11.4 具备动静结合反编译能力的二进制翻译器UTP-MBC ························ 254
11.4.1 UTP-MBC架构设计需要解决的主要问题 ························ 255
11.4.2 UTP-MBC翻译器的相关研究 ························ 256
11.4.3 一体化翻译架构设计 ························ 257
11.5 本章小结 ························ 260
习题 ························ 261

**第12章 反编译中的指令解码和语义描述与映射** ························ 262
12.1 指令描述和指令解码 ························ 262
12.1.1 相关研究 ························ 262
12.1.2 编解码描述语言SLED ························ 263
12.1.3 基于SLED的x64指令描述和解码 ························ 266
12.1.4 SLED在多源一体解码体系中的应用 ························ 270
12.2 指令的语义映射 ························ 275
12.2.1 相关研究 ························ 276
12.2.2 语义描述语言SSL ························ 276
12.2.3 中间表示 ························ 282
12.2.4 一个示例——指令原子语义描述语言ASDL ························ 284
12.3 本章小结 ························ 288
习题 ························ 288

**第13章 反编译中的恢复技术** ························ 290
13.1 数据流和数据恢复 ························ 290
13.1.1 数据流分析 ························ 290
13.1.2 数据恢复方法——以IA-64架构上的反编译为例 ························ 297
13.1.3 小结 ························ 310
13.2 高级控制流恢复 ························ 310
13.2.1 控制流恢复概述 ························ 311
13.2.2 高级控制流恢复分析 ························ 315
13.2.3 结构化算法介绍 ························ 318
13.2.4 可能的问题与解决办法 ························ 325
13.2.5 小结 ························ 325
13.3 过程恢复 ························ 325
13.3.1 相关知识简介 ························ 326
13.3.2 库函数的识别技术 ························ 328
13.3.3 用户自定义函数的过程恢复 ························ 335
13.4 本章小结 ························ 349
习题 ························ 349

第 14 章　编译优化的反向处理 …… 350
14.1　常用的编译优化方法 …… 350
14.1.1　编译优化的原则 …… 350
14.1.2　优化手段的分类 …… 350
14.2　部分编译优化的消除——谓词执行 …… 351
14.2.1　谓词执行 …… 351
14.2.2　IA-64 平台的谓词指令 …… 351
14.2.3　谓词消除 …… 353
14.3　本章小结 …… 358
习题 …… 358
第 15 章　反编译与信息安全 …… 359
15.1　基于反编译的恶意行为识别 …… 359
15.1.1　恶意代码检测背景 …… 359
15.1.2　文件格式异常信息 …… 363
15.1.3　指令序列层行为信息提取 …… 366
15.1.4　函数调用信息提取 …… 369
15.2　反编译在恶意代码检测中的应用 …… 377
15.2.1　系统架构的提出 …… 377
15.2.2　推理算法研究的基本内容 …… 378
15.2.3　恶意特征生成 …… 380
15.2.4　推理规则库的建立 …… 381
15.2.5　多重多维模糊推理算法的研究与实现 …… 385
15.3　本章小结 …… 391
习题 …… 391
参考文献 …… 392

Chapter

# 第1章 引论

人类之间的交流是通过语言进行的，但语言不是唯一的，不同的语言之间需要翻译，这就导致了翻译行业的建立。人与计算机之间也是通过语言进行交流的，但人类能理解的语言与机器能理解的语言是不同的，也需要翻译，这就导致了系列编译器的诞生。编译技术所讨论的问题，就是如何把符合人类思维方式的意愿（源程序）翻译成计算机能够理解和执行的形式（目标程序）。实现从源程序到目标程序转换的程序，称为编译程序或编译器。反编译技术所讨论的问题，就是如何把计算机能够理解和执行的形式（目标程序）翻译成便于人类理解的形式（高级语言源程序或流程图）。实现从目标程序到便于人类理解的系列文档的转换，称为反编译程序或反编译器。

编译器这个术语是由 Grace Murray Hopper 在 20 世纪 50 年代初期提出的，现代意义上最早的编译器是 20 世纪 50 年代后期的 Fortran 编译器，该编译器验证了经过编译的高级语言的生命力，也为后续高级语言和编译器的大量涌现奠定了基础。

反编译技术起源于 20 世纪 60 年代，比编译技术晚 10 年左右，但反编译技术的成熟度远不如编译技术。在半个世纪的发展过程中，出现了不少实验性的反编译器，其中以 Dcc、Boomerang 和 IDA Pro 的反编译插件 Hex_rays 最为著名。但这些反编译器都有这样或那样的缺陷。例如，Dcc 只能识别最简单的数据类型；Boomerang 无法识别复杂的数据结构，如 C++ 的类和多维数组；Hex_rays 只能产生可读性较低的 C 伪代码，且同样无法识别复杂的数据结构。因此，反编译技术还有很广阔的研究与发展空间。

本章仅对编译器和编译流程方面的知识进行概要阐述，反编译方面的概要介绍将在第 10 章给出。

## 1.1 编译器与解释器

计算机的硬件只能识别和理解由 0、1 字符串组成的机器指令序列，即目标程序或机器指令程序。在计算

试读结束：需要全本请在线购买： www.ertongbook.com

机刚刚发明的时期，人们只能向计算机输入机器指令程序来让它进行简单的计算。由于机器指令程序不易被人类理解，用它编写程序既困难又容易出错，于是就引入了代替 0、1 字符串的由助记符号表示的指令，即汇编指令，汇编指令的集合称为汇编语言，汇编指令序列称为汇编语言程序。但汇编程序实际上与机器语言程序是一一对应的，均要求程序员按照指令工作的方式来思考和解决问题，两者之间并无本质区别。因此，它们被称为面向机器的语言或低级语言。

随着计算机的发展和应用需求的增长，程序员的需求也大幅增长，但能够用机器语言或汇编语言编程的人员数量满足不了这种需求，许多科技工作者也想自己动手编写程序，因此，需要抽象度更高、功能更强的语言来作为程序设计语言，于是产生了面向各类应用的便于人类理解与运用的程序设计语言，即高级语言。尽管人类可以借助高级语言来编写程序，但计算机硬件真正能够识别和理解的语言还是由 0、1 组成的机器语言，这就需要在高级语言与机器语言之间建立桥梁，使得高级语言能够过渡到机器语言。也就是说，需要若干“翻译”，把各类高级语言翻译成机器语言。语言通常被分成三个层次：高级语言、汇编语言、机器语言。高级语言可以翻译成机器语言，也可以翻译成汇编语言，这两种翻译都称为编译。汇编语言到机器语言的翻译称为汇编。编译和汇编属于正向工程，有时还需要将机器语言翻译成汇编语言或高级语言，这通常称为反汇编或反编译，属于逆向工程。

在编译器工作方式下，源程序的翻译和翻译后程序的运行是两个相互独立的阶段。用户输入源程序，编译器对该源程序进行编译，生成目标程序，这个阶段称为编译阶段。目标程序在适当的输入下执行，最终得到运行结果的过程称为运行阶段。

解释器是另一种形式的翻译器。它把翻译和运行结合在一起进行，边翻译源程序，边执行翻译结果，这种工作方式被称为解释器工作方式。

换句话说，编译器的工作相当于翻译一本原著，原著相当于源程序，译著相当于目标程序，计算机的运行相当于阅读一本译著，这时，原著和翻译人员并不需要在场，译著是主角。解释器的工作相当于现场翻译，外宾和翻译都要在场，翻译边听外宾讲话，边翻译给听众，翻译是主角。解释器与编译器的主要区别是：运行目标程序时的控制权在解释器而不是目标程序。

## 1.2 编译过程

考虑一种场景，让一个既懂英文又懂中文的俄罗斯人将一篇英文文章翻译成中文，此人可能要经历这样几个阶段：识别英文单词、识别英文句子、理解意思、先译成俄语并进行合理修饰、译成中文。编译器对高级语言的翻译也需要经历这样几个类似阶段：先进行词法分析，识别出合法的单词；再进行语法分析，得到由单词串组成的句子；然后进行语义分析，生成中间代码；再进行中间代码级别的优化，生成优化的中间代码；最后再翻译成目标代码。可见这两种情况的各个阶段的对应非常一致。

## 1.3 编译器结构

编译过程的每个阶段工作的逻辑关系如图 1-1

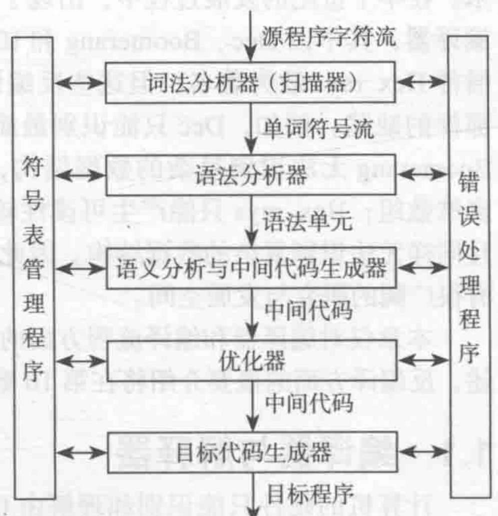


图 1-1 编译结构图

所示，图中每个阶段的工作由相关程序模块承担，其中的符号表管理程序和错误处理程序贯穿编译过程的各个阶段。这些程序模块构成了编译器的基本结构。

## 1.4 编译器的分类及生成

根据不同的用途和侧重点，编译程序可以进一步分类，换句话说，有许多不同种类的编译器变体。譬如：用于帮助程序开发和调试的编译程序称为诊断编译程序，这类编译器可对程序进行详细检查并报告错误；另一类侧重于提高目标代码效率的编译程序称为优化编译程序，这类编译器通常使用多种混合的“变换”来改善程序的性能，但这往往是以编译器的复杂性和编译时间的增加为代价的。通常，将运行编译程序的机器称为宿主机，将运行编译程序所产生的目标代码的机器称为目标机。如果一个编译程序产生不同于其宿主机指令集的机器代码，则称它为交叉编译程序（Cross Compiler）。还有一类编译器，其目标机器可以改变，而不需要重写它的与机器无关的组件，这类编译器称为可再目标编译器（Retargetable Compiler），通常，这类编译器难以生成高效的代码，因为其难以利用特殊情况和目标机器特性。目前，很多编译程序同时提供了调试、优化、交叉编译等多种功能，用户可以通过“编译选项”进行选择。

编译器本身也是一个程序，这个程序是怎么编写的呢？早期，人们是用汇编语言编写编译器。虽然用汇编语言编写出的编译器代码效率很高，但由于与高级语言编程相比，汇编语言编程难度较大，对编写编译器这种复杂的系统效率不高，因此人们改用高级语言来编写编译器。随着编译技术的逐步成熟，一些专门的编译器编写工具相继涌现，比较成熟和通用的工具有词法分析器生成器（如LEX）和语法分析器生成器（如YACC）等。还有一些工具，如用于语义分析的语法制导翻译工具，用于目标代码生成的自动代码生成器，用于优化的数据流工具等。

## 1.5 高级语言及其分类

根据应用类型的不同，涌现了多种多样的面向人类的高级语言，其中典型的有如下几类形式。

### 1. 过程式语言

过程式语言也称为强制式语言（Imperative Language）。这类语言的特点是面向语句，命令驱动。一个用过程式语言编写的程序由一系列语句组成，语句的执行会引起若干存储单元中的值发生改变。许多著名的语言，如Algol、Fortran、Pascal、C等，属于这类语言。

### 2. 函数式语言

函数式语言也称为应用式语言（Applicative Language）。这类语言的特点是用函数的方式表示其功能，而不是通过一个语句接一个语句的执行来表示具体的操作步骤。这类程序的开发过程体现为，由之前已有的函数构造更复杂的函数。Lisp、ML和Haskell属于这类语言。

### 3. 逻辑程序设计语言

逻辑程序设计语言也称为基于规则的语言。这类语言的程序执行过程是：检查由逻辑表达式表示的条件，当其为真时，则执行相应的动作。Prolog语言是这类语言的典型代表，它使用Horn子句逻辑来表述相关规则，体现程序要做什么，而不是怎么做。

### 4. 面向对象的语言

面向对象的语言已成为目前最流行的语言，这类语言的主要特征是支持封装性、继承性和多态性等特性。像近世代数中的代数系统那样，这类语言将复杂的数据和对这些数据的

操作封装在一起，构成对象；对简单对象进行扩充，在继承简单对象特性的基础上，增加新的特性，从而得到更复杂的对象；这一点与近世代数中半群、群、环、域等代数系统之间的继承等关系非常相似。通过对象的构造可以使得面向对象程序获得过程式语言的有效性，通过作用于限定范围内数据的函数的构造可以使其获得函数式语言的灵活性和可靠性。Smalltalk、C++、Java 等语言是面向对象语言的典型代表，而 OCAML、F# 则融合了函数式语言和面向对象的特性。

5. 结构化查询语言

结构化查询语言（Structured Query Language）简称 SQL，是一种数据库查询和程序设计语言，通常用于存取数据以及查询、更新和管理关系数据库系统。结构化查询语言允许用户在高层数据结构上工作，它不要求用户指定数据的存放方法，也不需要用户了解具体的数据存放方式，即使具有完全不同底层结构的不同数据库系统也可以使用相同的结构化查询语言作为数据输入与管理的接口。结构化查询语言中的语句还可以嵌套，这使其具有很大的灵活性和很强的功能。

6. 其他面向特定应用领域的语言

随着计算机应用领域的进一步拓展，涌现了多种面向特定应用领域的高级语言，较为典型的有：面向互联网应用的 HTML、XML，面向集成电路设计的 VHDL、Verilog，面向计算机辅助设计的 Matlab，面向虚拟现实的 VRML 等。这些语言推动了计算机应用的快速发展，使得计算机成为人类生活中不可或缺的重要工具。

## 1.6 编译的前端和后端

通常，编译的阶段被分成前端和后端两部分。前端是由只依赖于源语言的那些阶段或阶段的一部分组成，往往包含词法分析、语法分析、语义分析和中间代码生成等阶段，当然还包括与这些阶段同时完成的错误处理和独立于目标机器的优化。后端是指编译器中依赖于目标机器的部分，往往只与中间语言有关而独立于源语言。后端包括与目标机器相关的代码优化、代码生成和与这些阶段相伴的错误处理和符号表操作。

基于同一个前端，重写其后端就可以产生同一种源语言在另一种机器上的编译器，这已经是为不同类型机器编写编译器的常用做法。反过来，把几种不同的语言编译成同一种中间语言，使得不同的前端都使用同一个后端，进而得到一类机器上的几个编译器，却只取得了有限的成功，其原因在于不同源语言的区别较大，使得包容它们的中间语言庞大臃肿，难以实现高效率。

编译的几个阶段往往通过一遍（pass）扫描来实现，这里的“一遍扫描”通常是指读一个输入文件和写一个输出文件的过程。把几个阶段组成“一遍”，并且这些阶段的活动在该遍中交错进行是经常发生的。

## 1.7 C 语言程序的编译流程

本节以 C 语言程序的编译流程为例，介绍实际的 C 语言编译器是如何运作的。通常把整个代码的编译流程分为编译过程和链接过程。

1. 编译过程

编译过程可分为编译预处理、编译与优化、汇编等阶段。

（1）编译预处理

编译预处理即读取 C 源程序，对其中的伪指令（以 # 开头的指令）和特殊符号进行处

理。主要包括以下几个方面：

1）宏定义指令，如 # define Name TokenString、# undef 等。对于前一个伪指令，预编译所要做的是将程序中的所有 Name 用 TokenString 替换，但作为字符串常量的 Name 则不被替换。对于后一个伪指令，则将取消对某个宏的定义，使以后该串的出现不再被替换。

2）条件编译指令，如 # ifdef、# ifndef、# else、# elif、# endif 等。这些伪指令的引入使得程序员可以通过定义不同的宏来决定编译程序对哪些代码进行处理。预编译程序将根据有关的文件，将那些不必要的代码过滤掉。

3）头文件包含指令，如 # include "FileName" 或者 # include <FileName> 等。在头文件中一般用伪指令 # define 定义了大量的宏，还有对各种外部符号的声明。采用头文件的目的是使某些定义可以供多个不同的 C 源程序使用。因为在需要用到这些定义的 C 源程序中，只需加上一条 # include 语句，而不必再在此文件中将这些定义重复一遍。预编译程序将把头文件中的定义统统都加入它所产生的输出文件中，以供编译程序对之进行处理。注意，这个过程是递归进行的，也就是说，被包含的文件可能还包含其他文件。包含到 C 源程序中的头文件可以是系统提供的，这些头文件一般放在 /usr/include 目录下，在 # include 中使用它们要用尖括号（< >）。另外开发人员也可以定义自己的头文件，这些文件一般与 C 源程序放在同一目录下，此时在 # include 中要用双引号（" "）。

4）特殊符号。例如在源程序中出现的 LINE 标识将被解释为当前行号（十进制数），FILE 则被解释为当前被编译的 C 源程序的名称。预编译程序对于在源程序中出现的这些串将用合适的值进行替换。预编译程序所完成的基本上是对源程序的“替代”工作。经过此种替代，生成一个没有宏定义、没有条件编译指令、没有特殊符号的输出文件。这个文件的含义与没有经过预处理的源文件是相同的，但内容有所不同。下一步，此输出文件将作为编译程序的输入而被翻译成为机器指令序列。

5）删除注释。删除所有的注释“//…”和“/*…*/”。

6）保留所有的#pragma 编译器指令。以#pragma 开始的编译器指令必须保留，因为编译器需要使用它们。

经过预编译后的 .i 文件不包含任何宏定义，因为所有的宏已经被展开，并且包含的文件也已经被插入 .i 文件中。所以，当无法判断宏定义是否正确或头文件包含是否正确时，可以查看预编译后的文件来确定。

（2）编译与优化

经过预编译得到的输出文件中只有常量、变量的定义，以及 C 语言的关键字，如 main、if、else、for、while、{、}、+、-、*、\ 等。编译程序所要做的工作就是通过词法分析和语法分析，在确认所有的指令都符合语法规则之后，将其翻译成等价的中间代码表示或汇编代码。优化处理涉及的问题不仅同编译技术本身有关，而且同机器的硬件环境也有关。优化中的一种是对中间代码的优化。另一种优化则主要是针对目标代码的生成而进行的。对于前一种优化，主要的工作是删除公共表达式、循环优化（代码外提、强度削弱、变换循环控制条件、已知量的合并等）、复写传播，以及无用赋值的删除等。后一种类型的优化同机器的硬件结构密切相关，最主要的是考虑如何充分利用机器的各个硬件寄存器存放有关变量的值，以减少对内存的访问次数。另外，如何根据机器硬件执行指令的特点（如流水线、RISC、CISC、VLIW 等）而对指令进行一些调整使目标代码比较短，执行的效率比较高，也是优化的一个重要任务。经过优化得到的汇编代码序列必须经过汇编程序的汇编转换成相应的机器指令序列，方能被机器执行。

（3）汇编

汇编过程是把汇编语言代码翻译成目标机器指令的过程。对于待编译处理的每一个C语言源程序，都将经过这一处理过程而得到相应的目标文件。目标文件中所存放的也就是与源程序等效的机器语言代码。目标文件由段组成，通常一个目标文件中至少有两个段：①代码段。该段中所包含的主要是程序的机器指令，一般是可读和可执行的，但却不可写。②数据段。主要存放程序中要用到的各种全局变量或静态的数据，一般是可读、可写、可执行的。

UNIX环境下主要有三种类型的目标文件：①可重定位文件，其中包含适合于其他目标文件链接以创建一个可执行的或者共享的目标文件的代码和数据。②共享的目标文件，这种文件存放了适合于在两种上下文里链接的代码和数据。第一种是静态链接程序，可把它与其他可重定位文件共享的目标文件一起处理来创建另一个目标文件；第二种是动态链接程序，将它与另一个可执行文件及其他共享目标文件结合到一起，创建一个进程映像。③可执行文件，它包含了一个可以被操作系统通过创建一个进程来执行的文件。汇编程序生成的实际上是第一种类型的目标文件。对于后两种还需要其他的一些处理方能得到，这就是链接程序的工作了。

## 2. 链接过程

由汇编程序生成的目标文件并不能立即被执行，其中可能还有许多没有解决的问题。例如，某个源文件中的函数可能引用了另一个源文件中定义的某个符号（如变量或者函数调用等）；在程序中可能调用了某个库文件中的函数，等等。所有的这些问题，都需要经过链接程序的处理方能得以解决。链接程序的主要工作就是将有关的目标文件彼此相连接，亦即将在一个文件中引用的符号同该符号在另外一个文件中的定义连接起来，使得所有的这些目标文件成为一个能够被操作系统装入执行的统一整体。根据开发人员指定的与库函数的链接方式的不同，链接处理通常可分为两种：①静态链接。在该方式下，函数的代码将从其所在的静态链接库中被复制到可执行程序中。这样当该程序被执行时，这些代码将被装入该进程的虚拟地址空间中。静态链接库实际上是一个目标文件的集合，其中的每个文件含有库中的一个或者一组相关函数的代码。②动态链接。在该方式下，函数的代码被放到称作动态链接库或共享对象的某个目标文件中。链接程序此时所做的只是在最终的可执行程序中记录下共享对象的名字以及一些少量的登记信息。在该可执行文件被执行时，动态链接库的全部内容将被映射到运行时相应进程的虚地址空间。动态链接程序将根据可执行程序中记录的信息找到相应的函数代码。对于可执行文件中的函数调用，可分别采用动态链接或静态链接的方法。使用动态链接能够使最终的可执行文件比较短小，并且当共享对象被多个进程使用时能节约一些内存，因为在内存中只需要保存一份此共享对象的代码。但并不是使用动态链接就一定比使用静态链接要优越，在某些情况下动态链接可能带来一些性能上的损失。

## 3. GCC的编译链接

在Linux中使用的GCC编译器是把以上几个过程进行了捆绑，使用户只使用一次命令就完成编译工作，这确实很方便，但对于初学者了解编译过程却很不利。GCC代理的编译流程如下：①预编译，将.c文件转化成.i文件，使用的GCC命令是gcc –E（对应于预处理命令cpp）；②编译，将.c/.h文件转换成.s文件，使用的gcc命令是gcc –S（对应于编译命令cc1，实际上，现在版本的GCC使用cc1将预编译和编译两个步骤合成为一个步骤；③汇编，将.s文件转化成.o文件，使用的GCC命令是gcc –c（对应于汇编命令as）；④链接，将.o文件转化成可执行程序，使用的GCC命令是gcc（对应于链接命令ld）。

以名为hello.c的程序为例，编译流程主要经历如图1-2所示的四个过程。

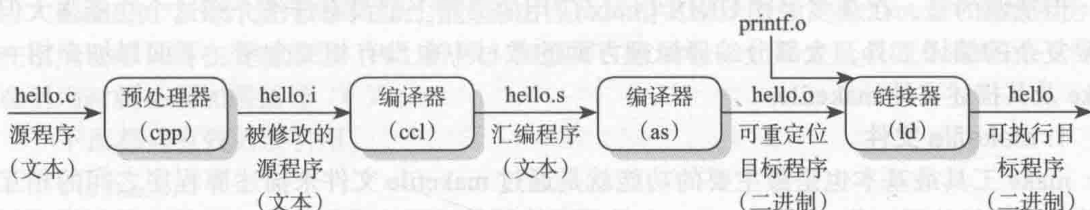



图1-2 C语言程序编译流程图

例如，hello.c为：

```
#include <stdio.h>
Int main(int argc, char *argv[])
{
    printf("hello world\n");
    return 0;
}
```

运行gcc –S hello.c可以得到hello.s文件，其内容为：

```
.file "hello.c"
.def ___main; .scl 2; .type 32; .endef
.section .rdata, "dr"
LC0:
.ascii "hello world\0"
.text
.globl _main
.def _main; .scl 2; .type 32; .endef
_main:
LFB6:
.cfi_startproc
pushl %ebp
.cfi_def_cfa_offset 8
  ...
```

所有以字符“.”开头的行都是指导汇编器和链接器的命令，其他行则是被翻译成汇编语言的代码。

C语言编译的整个过程是比较复杂的，涉及的编译器知识、硬件知识、工具链知识非常多。一般情况下，只需要知道其分成编译和链接两个阶段，编译阶段是将源程序（*.c）转换成为目标代码（一般是obj文件），链接阶段是将源程序转换成的目标代码（obj文件）与程序里面调用的库函数对应的代码链接起来形成对应的可执行文件（exe文件），其他的都需要在实践中多多体会才能有更加深入的理解。

## 1.8 UNIX/Linux环境中的make和makefile

在UNIX或Linux环境中，make是一个非常重要和经常使用的编译工具。无论是自己进行项目开发还是安装应用软件，都会经常用到make或make install。使用make工具，可以将大型的开发项目分解成多个更易于管理的模块。对于一个包括数百个源文件的应用程序，使用make和makefile工具可以简洁明快地理顺各个源文件之间复杂的相互依赖关系。针对这么多的源文件，如果每次都要键入gcc命令进行编译，那对程序员来说就是一场灾难。而make工具则可自动完成编译工作，并且可以只对程序员在上次编译后修改过的部分进行编译。因此，有效地利用make和makefile工具可以大大提高项目开发的效率。同时掌

试读结束：需要全本请在线购买： www.ertongbook.com