工业和信息化部“十四五”规划教材

编译原理简明教程

费　蓉　胡元义　主编

黑新宏　鲁晓锋　谈姝辰　副主编

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

本书系统地介绍了编译程序的设计原理及实现技术。在内容的组织上，本书强调知识的实用性，有机地结合了编译的基本理论与具体的实现技术，既注重理论的完整性，化繁为简，又将理论融于具体的实例中，化难为易，以达到准确、清楚地阐述相关概念和原理的目的。在具体内容的讲述中，思路清晰、条理分明，给出的示例丰富，实用性与连贯性强，可使读者全面、直观地认识编译的各个阶段。本书采用的算法全部由 C 语言描述，各章均附有习题，且附录中提供了习题解答。

本书既可作为计算机本科专业学生的教材，又可作为计算机软件工程人员的参考资料。

图书在版编目（CIP）数据

编译原理简明教程 / 费蓉，胡元义主编. —北京：电子工业出版社，2022.6

ISBN 978-7-121-43575-1

Ⅰ. ①编… Ⅱ. ①费… ②胡… Ⅲ. ①编译程序–高等学校–教材 Ⅳ. ①TP314

中国版本图书馆 CIP 数据核字（2022）第 090015 号

责任编辑：刘御廷
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×1092　1/16　印张：18.75　　字数：480 千字
版　　次：2022 年 6 月第 1 版
印　　次：2022 年 6 月第 1 次印刷
定　　价：69.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：lyt@phei.com.cn。

前　言

计算机语言之所以能由单一的机器语言发展到现今的数千种高级语言，就是因为有了编译技术。编译技术是计算机科学中发展得最迅速、最成熟的一个分支，它集中体现了计算机发展的成果与精华。

“编译原理”是计算机专业的一门核心课程，在计算机本科教学中占有十分重要的地位。“编译原理”课程具有很强的理论性与实践性，读者学习起来普遍感到内容抽象、不易理解。为此，本书采用了由浅入深、循序渐进的方法来介绍编译原理的基本概念和实现方法。在内容的组织上，本书有机地结合了编译的基本理论与具体的实现技术，既注重理论的完整性，化繁为简，又将理论融于具体的实例中，化难为易，以达到准确、清晰地阐述相关概念和原理的目的。各章节的理论阐述具有条理性，给出的例子也具有实用性与连贯性，可让读者全面、直观地认识编译的各个阶段，进而透彻地领悟编译原理的精髓。

本书立足于“看得懂、学得会、用得上”，以编译核心知识为纲，以实用技术为主，以丰富的示例实践为线，侧重于编译理论的具体实现。书中的算法全部采用 C 语言描述，文法也尽可能采用 C 语言文法。

本书共 10 章。第 1 章简要介绍编译的基本概念；第 2 章介绍词法分析的相关内容，主要涉及正规表达式与有限自动机；第 3 章简要地介绍文法的有关概念；第 4 章介绍自顶向下语法分析方法——递归下降分析法和 LL(1)分析法；第 5 章介绍自底向上语法分析方法——算符优先分析法和 LR 分析法；第 6 章介绍语法制导翻译与中间代码生成的有关内容，给出如何在语法分析的同时进行语义加工并产生中间代码的方法；第 7 章介绍代码优化的有关内容，主要涉及基本块优化和循环优化；第 8 章介绍目标程序运行时存储空间的组织；第 9 章讨论目标代码生成的有关内容，讲述如何由中间代码产生最终目标代码；第 10 章简要地介绍符号表的组织与错误处理的方法。

为便于读者正确地理解有关概念，各章配有一定数量的习题，且附录中给出了答案。这些习题大多选自本科生/研究生的考试题，也包括编者结合多年教学实践经验设计的典型范例，力求让读者抓住重点、突破难点，全面、深入地巩固所学知识。

由于编者水平有限，书中难免存在一些缺点和错误，恳请广大读者批评指正。

编　者

2022 年 3 月

目　录

第 1 章　绪论 …… 1
1.1　程序设计语言和编译程序 …… 1
1.2　编译程序的历史及发展 …… 3
1.3　编译过程和编译程序结构 …… 4
1.4　编译程序的开发 …… 6
1.5　构造编译程序所应具备的知识内容 …… 7
习题 1 …… 8

第 2 章　词法分析 …… 10
2.1　词法分析器的设计方法 …… 10
2.1.1　单词符号的分类与输出形式 …… 10
2.1.2　状态转换图 …… 11
2.2　一个简单的词法分析器示例 …… 13
2.2.1　C 语言子集的单词符号表示 …… 13
2.2.2　C 语言子集对应的状态转换图 …… 14
2.2.3　状态转换图的实现 …… 15
2.3　正规表达式与有限自动机简介 …… 17
2.3.1　正规表达式与正规集 …… 17
2.3.2　有限自动机 …… 18
2.4　正规表达式到有限自动机的构造 …… 21
2.4.1　由正规表达式构造等价的非确定有限自动机 …… 21
2.4.2　NFA 的确定化 …… 22
2.4.3　确定有限自动机（DFA）的化简 …… 24
2.4.4　正规表达式到有限自动机构造示例 …… 26
2.5　词法分析器的自动生成 …… 31
习题 2 …… 33

第 3 章　文法和语言 …… 36
3.1　基本概念 …… 36
3.1.1　文法和语言的定义 …… 36
3.1.2　文法产生的语言 …… 38
3.2　形式语言分类 …… 39
3.2.1　四类文法的划分 …… 39
3.2.2　四类文法的关系与区别 …… 40
3.2.3　正规表达式与上下文无关文法 …… 42

3.3 推导与语法树……43
3.3.1 推导与短语……43
3.3.2 语法树与二义性……44
习题 3……49

第 4 章 语法分析—自顶向下分析方法……51
4.1 自顶向下分析原理……51
4.1.1 自顶向下分析存在的不确定性……51
4.1.2 确定的自顶向下分析……52
4.2 递归下降分析法……56
4.2.1 算术表达式的递归下降分析器……56
4.2.2 无二义性的算术表达式递归下降分析器……58
4.3 LL(1)分析法……59
4.3.1 表驱动的 LL(1)分析器……59
4.3.2 LL(1)分析表的构造……62
习题 4……66

第 5 章 语法分析—自底向上分析方法……68
5.1 自底向上分析原理……68
5.2 算符优先分析法……70
5.2.1 算符优先文法……70
5.2.2 算符优先关系表的构造……71
5.2.3 算符优先分析算法的设计……74
5.2.4 优先函数……78
5.3 LR 分析器的工作原理……80
5.4 LR(0)分析器……86
5.4.1 LR(0)项目集规范族的构造……86
5.4.2 LR(0)分析表的构造……88
5.5 SLR(1)分析器……93
5.6 二义文法的应用……99
习题 5……103

第 6 章 语义分析和中间代码生成……107
6.1 概述……107
6.1.1 语义分析的概念……107
6.1.2 语法制导翻译方法……107
6.2 属性文法……109
6.2.1 文法的属性……109
6.2.2 属性文法……110
6.3 几种常见的中间语言……111
6.3.1 抽象语法树……111

6.3.2 逆波兰表示法 …… 112
6.3.3 三地址代码 …… 114
6.4 表达式及赋值语句的翻译 …… 116
6.4.1 简单算术表达式和赋值语句的翻译 …… 116
6.4.2 布尔表达式的翻译 …… 118
6.5 控制语句的翻译 …… 123
6.5.1 条件语句 if 的翻译 …… 123
6.5.2 循环语句的翻译 …… 125
6.5.3 三种基本控制结构的翻译 …… 127
6.5.4 多分支控制语句 case 的翻译 …… 132
6.5.5 语句标号和转移语句的翻译 …… 134
6.6 数组元素的翻译 …… 134
6.6.1 数组元素的地址计算及中间代码形式 …… 135
6.6.2 赋值语句中数组元素的翻译 …… 135
6.6.3 数组元素翻译示例 …… 136
6.7 过程或函数调用语句的翻译 …… 139
6.7.1 过程或函数调用的方法 …… 139
6.7.2 过程或函数调用语句的四元式生成 …… 140
6.8 说明语句的翻译 …… 141
6.8.1 变量说明的翻译 …… 141
6.8.2 数组说明的翻译 …… 141
6.9 递归下降语法制导翻译方法简介 …… 142
习题 6 …… 143

第 7 章 代码优化 …… 147
7.1 局部优化 …… 147
7.1.1 基本块的划分方法 …… 147
7.1.2 基本块的 DAG 方法 …… 148
7.1.3 用 DAG 进行基本块的优化处理 …… 152
7.1.4 DAG 构造算法的进一步讨论 …… 153
7.2 循环优化 …… 154
7.2.1 程序流图与循环 …… 154
7.2.2 循环的查找 …… 156
7.2.3 循环优化 …… 161
习题 7 …… 169

第 8 章 目标程序运行时存储空间的组织 …… 173
8.1 静态存储分配 …… 173
8.2 简单的栈式存储分配 …… 174
8.2.1 栈式存储分配与活动记录 …… 175

8.2.2 过程的执行 …… 176
8.3 嵌套过程语言的栈式实现 …… 179
8.3.1 嵌套层次显示表和活动记录 …… 179
8.3.2 嵌套过程的执行 …… 180
8.3.3 访问非局部名的另一种实现方法 …… 182
8.4 堆式动态存储分配 …… 185
8.4.1 堆式存储的概念 …… 185
8.4.2 堆式存储的管理方法 …… 186
习题 8 …… 188

第 9 章 目标代码生成 …… 190
9.1 简单代码生成器 …… 190
9.1.1 待用信息与活跃信息 …… 191
9.1.2 代码生成算法 …… 193
9.1.3 寄存器分配 …… 194
9.1.4 源程序到目标代码生成示例 …… 196
9.2 汇编指令到机器代码翻译概述 …… 198
习题 9 …… 204

第 10 章 符号表与错误处理 …… 206
10.1 符号表 …… 206
10.1.1 符号表的作用 …… 206
10.1.2 符号表的组织 …… 207
10.1.3 分程序结构语言符号表建立 …… 208
10.1.4 非分程序结构语言符号表建立 …… 211
10.1.5 常用符号表结构 …… 212
10.1.6 符号表内容 …… 213
10.2 错误处理 …… 214
10.2.1 语法错误校正 …… 214
10.2.2 语义错误校正 …… 220
习题 10 …… 221

附录 A 8086/8088 指令码汇总表 …… 223

附录 B 8086/8088 指令编码空间表 …… 228

附录 C 习题解答 …… 230

参考文献 …… 290

第 1 章

绪　论

计算机的诞生是科学发展史上的一个里程碑。经过半个多世纪的发展，计算机已经改变了人类生活、工作的各个方面，成为人类不可缺少的工具。计算机之所以能够如此广泛地被人们应用，应归功于高级程序设计语言。计算机语言之所以能由最初单一的机器语言发展到现今数千种高级语言，就是因为有了编译程序。没有高级语言，计算机的推广应用是难以实现的；没有编译程序，高级语言就无法使用。编译理论与技术也是计算机科学中发展得最迅速、最成熟的一个分支，它集中体现了计算机发展的成果与精华。

1.1 程序设计语言和编译程序

为了处理和解决实际问题，每种计算机都有其特定的功能，这些功能是由计算机执行一系列相应的操作来实现的。计算机所能执行的每种操作都被称为**一条指令**，计算机能够执行的全部指令集合就是该计算机的指令系统。计算机硬件的器件特性，决定了计算机本身只能直接接受由 0 和 1 编码的二进制指令和数据，这种二进制形式的指令集合被称为计算机的**机器语言**，它是计算机唯一能够直接识别并接受的语言。

用机器语言编写程序很不方便，且容易出错，编写的程序也难以调试、阅读和交流。为此，出现了用助记符代替机器语言（二进制编码）的另一种语言，即汇编语言。汇编语言建立在机器语言之上，因为它是机器语言的符号化形式，因此较机器语言直观；但是，计算机不能直接识别这种符号化语言，使用汇编语言编写的程序必须翻译成机器语言后才能被执行，这种“翻译”是通过专门的软件——汇编程序实现的。

尽管汇编语言与机器语言相比在阅读和理解上有了长足的进步，但是其依赖具体机器的特性是无法改变的，这就给程序设计增加了难度。随着计算机应用需求的不断增长，出现了更加接近人类自然语言的功能更强、抽象级别更高的面向各种应用的高级语言。高级语言已从具体机器中抽象出来，摆脱了依赖具体机器的问题。用高级语言编制的程序几乎能在不改动的情况下在不同种类的计算机上运行，且不易出错，这是汇编语言难以做到的，但是将高级语言程序翻译（编译）为最终能被直接执行的机器语言程序的难度大大增加。

由于汇编语言和机器语言一样是面向机器的，故相对于面向用户的高级语言来说，它们都被称为**低级语言**，FORTRAN、PASCAL、C、ADA、Java 这类面向应用的语言则被称为**高级语言**。因此，编译程序指的就是这样一种程序，通过它能将用高级语言编写的源程序转换成与之逻辑上等价的低级语言形式的目标程序，如图 1.1 所示。

图 1.1　编译程序的功能

一个高级语言程序的执行通常分为两个阶段，即编译阶段和运行阶段，如图 1.2 所示。编译阶段将源程序变换成目标程序；运行阶段由所生成的目标程序连同运行系统（数据空间分配子程序、标准函数程序等）接收程序的初始数据作为输入，运行后输出计算结果。

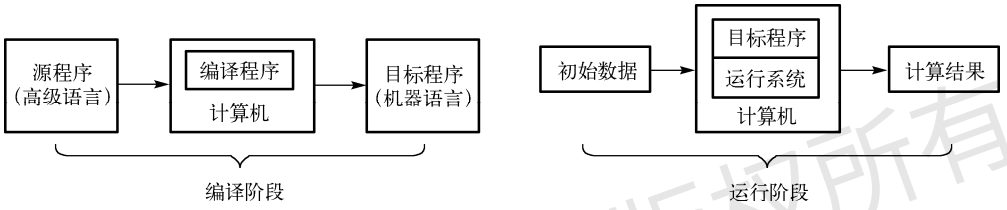

图 1.2　源程序的编译阶段和运行阶段

如果编译生成的目标程序是汇编语言形式的，那么在编译与运行阶段之间还要添加一个汇编阶段，这个阶段将编译生成的汇编语言目标程序再经汇编程序变换成机器语言目标程序，如图 1.3 所示。

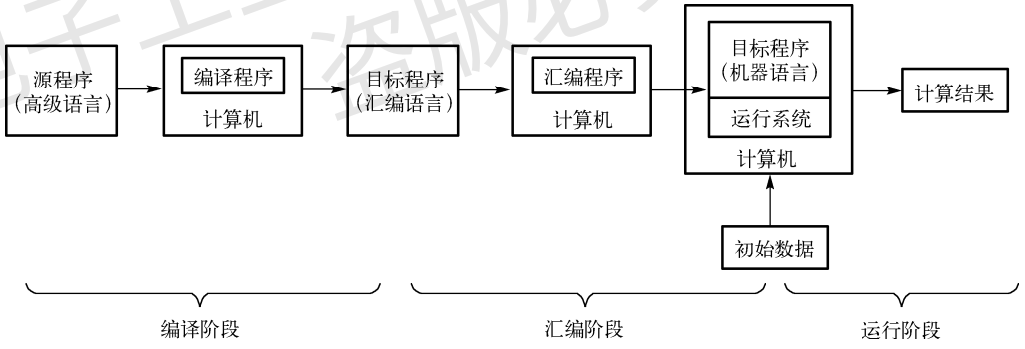

图 1.3　源程序的编译阶段、汇编阶段和运行阶段

用高级语言编写的程序也可通过解释程序执行。解释程序也是一种翻译程序，它将源程序作为输入，逐条语句地读入并解释执行，如图 1.4 所示。解释程序与编译程序的主要区别是：编译程序将源程序翻译成目标程序后，再执行目标程序；解释程序逐条读出源程序中的语句并解释执行，即在解释程序的执行过程中不产生目标程序。典型的解释型高级语言是 BASIC 语言。

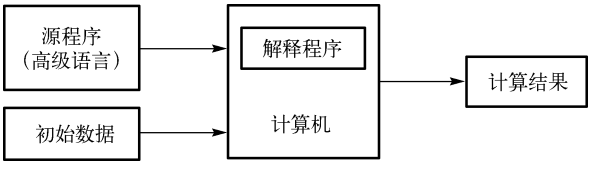

图 1.4　解释程序的解释执行过程示意

1.2 编译程序的历史及发展

二十世纪四十年代，由于冯·诺依曼在程序存储计算机（stored-program computer）方面的开创性工作，计算机可以执行编写的一串代码或程序，这些程序最初都是用机器语言（Machine Language）编写的。机器语言就是计算机能够执行的全部指令集合的二进制形式。例如，

```
C7 06 0000 0002
```

表示在 IBM PC 上使用 Intel 8x86 处理器将数字 2 移至地址 0000（十六进制）的指令。用机器语言编写程序很不方便，且容易出错，因此这种代码形式很快就被汇编语言（Assembly Language）取代。在汇编语言中，指令和存储地址都以符号形式给出。例如，汇编语言指令

```
MOV X, 2
```

与前面的机器指令等价（假设符号存储地址 X 是 0000）。汇编程序（Assembler）将汇编语言的符号代码和存储地址翻译成与机器语言相对应的二进制代码。汇编语言大大提高了编程的速度和准确性，至今人们仍在使用它，当有存储容量小和速度快的要求时，尤其如此。但是，汇编语言依赖于具体机器的特性是无法改变的，这就给编程和程序调试增加了难度。显然，编程技术发展的下一个重要步骤是，用更简洁的数学定义或自然语言来描述和编写程序，它应与任何机器无关，而且可以通过一个翻译程序将其翻译为可在计算机上直接执行的二进制代码。例如，前面的汇编语言代码“MOV X, 2”可以写成简洁的与机器无关的形式“$X=2$”。

1954—1957 年，IBM 公司 John Backus 带领的一个研究小组开发了 FORTRAN 语言及其编译器。由于编译程序理论与技术的研究刚刚起步，该小组为 FORTRAN 语言的开发付出了巨大的辛劳。与此同时，波兰语言学家 Noam Chomsky 开始了他的自然语言结构研究。Noam Chomsky 根据文法（Grammar，产生语言的规则）的难易程度及识别它们所需的算法，对语言进行了分类，定义了 0 型、1 型、2 型和 3 型四类文法及相应的形式语言，且分别与相应的识别系统相联系。2 型文法（Context-free Grammar，上下文无关文法）被证明是程序设计语言中最有用的文法，它代表着目前程序设计语言结构的标准。Noam Chomsky 的研究结果最终使得编译器结构异常简单，甚至还具有自动化功能。有限自动机（Finite Automata）和正规表达式（Regular Expression）与上下文无关文法紧密相关，它们与 Noam Chomsky 的 3 型文法相对应，并且引出了表示程序设计语言的单词符号形式。接着，又产生了生成有效目标代码的方法——最初的编译器，它们被沿用至今。随着语法分析研究的深入，重点转移到编译程序的自动生成研究。开发的这种程序最初被称为编译程序的**编译器**，但是，因为它们仅仅能够自动完成编译器的部分工作，所以更确切地被称为**分析程序生成器**（Parser Generator）；在这些程序中，最著名的是 Steve Johnson 于 1975 年为 UNIX 系统编写的语法分析器自动生成工具 YACC（Yet Another Compiler-Compiler）。类似地，有限自动机的研究产生了被称为 LEX 的另一种词法分析器自动生成工具。

二十世纪七八十年代，大量研究都关注于编译器其他部分的自动生成，其中包括代码生成。这些努力并未取得多大的成功，因为这部分工作过于复杂，人们对其本质也不甚了解。

现今编译器的发展包括更为复杂的算法应用程序，用于简化或推断程序中的信息，且这又与具有此类功能的更复杂的程序设计语言发展结合在一起。其中，典型的算法有用于函数语言编译 Hindley-Milner 类型检查的统一算法。目前，编译器越来越成为基于窗口的交互开发环境（Interactive Development Environment，IDE）的一部分，该开发环境包括编辑器、链接程序、调试程序和项目管理程序。尽管近年来对 IDE 进行了大量研究，但是基本的编译器设计在 40 多年间并无多大的改变。

现代编译技术已经转向到并行编译研究，但是本书只介绍经典的编译理论和技术。

1.3 编译过程和编译程序结构

编译程序的工作过程是指从输入源程序开始到输出目标程序为止的整个过程。这个过程非常复杂，一般来说可以划分为五个阶段：词法分析阶段、语法分析阶段、语义分析和中间代码生成阶段、优化阶段、目标代码生成阶段。

1．词法分析阶段

词法分析阶段的任务是输入源程序，对构成源程序的字符串进行扫描和分解，识别出一个个单词符号，如基本字（if、for、begin 等）、标识符、常数、运算符和界符（如“(”“)”“=”“;”）等，用统一长度的标准形式（也称**内部码**）来表示所识别的单词，以便于后续语法工作的进行。因此，词法分析工作是将源程序中的字符串变换成单词符号流的过程，词法分析所遵循的是语言的构词规则。

2．语法分析阶段

语法分析阶段的任务是在词法分析的基础上，根据语言的语法规则（文法规则），将单词符号流分解成各类语法单位（语法范畴），如“短语”“子句”“句子（语句）”“程序段”“程序”。通过语法分析，可以确定整个输入串是否构成一个语法上正确的“程序”。语法分析所遵循的是语言的语法规则，语法规则通常用上下文无关文法描述。

3．语义分析和中间代码生成阶段

语义分析和中间代码生成阶段的任务是，对各类不同语法范畴按语言的语义进行初步翻译，具体包含两方面的工作：一是对每种语法范畴进行静态语义检查，如变量是否定义、类型是否正确等；二是在语义检查正确的情况下进行中间代码的翻译。注意，中间代码是介于高级语言的语句和低级语言的指令之间的一种独立于具体硬件的记号系统（即与机器无关），它既有一定程度的抽象，又与低级语言的指令十分接近，因此将其转换为目标代码比较容易。将语法范畴翻译成中间代码所遵循的是语言的语义规则，常见的中间代码有四元式、三元式、间接三元式和逆波兰记号等。

4．优化阶段

优化阶段的任务主要是，对前一阶段产生的中间代码进行等价变换或改造（另一种优化针对目标机器，即对目标代码进行优化），以期获得更高效（节省时间和空间）的目标

代码。常用的优化措施有删除冗余运算、删除无用赋值、合并已知量、循环优化等。例如，其值并不随循环而发生变化的运算可在进入循环前计算一次，而不必每次循环都进行计算。优化所遵循的原则是程序的等价变换规则。

5．目标代码生成阶段

目标代码生成阶段的任务是，将中间代码（或经优化处理后）变换成特定机器上的机器语言程序或汇编语言程序，实现最终的翻译工作。最后阶段的工作因为目标语言的关系而十分依赖硬件系统，即如何充分利用机器现有的寄存器，合理地选择指令，生成尽可能短且有效的目标代码，这些都与目标机器的硬件结构有关。

上述编译过程的五个阶段是编译程序工作时的动态特征，编译程序的结构可以按照这五个阶段的任务分模块进行设计。编译程序的结构示意如图 1.5 所示。

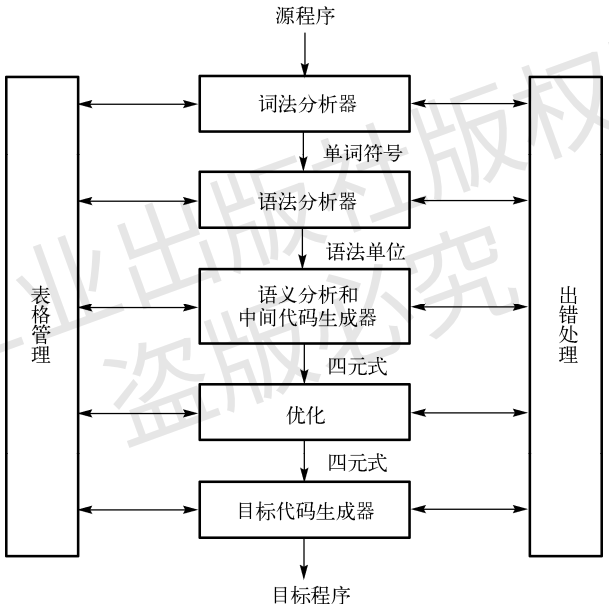

图 1.5 编译程序的结构示意

编译过程中源程序的各种信息保留在不同的表格中，编译各阶段的工作都涉及构造、查找或更新有关的表格，编译过程的绝大部分时间都用在制表、查表和更新表格的事务上，因此编译程序中还应包括一个表格管理程序。

出错处理与编译的各个阶段都有联系，与前三个阶段的联系尤为密切。出错处理程序应在发现错误后，将错误的有关信息如错误类型、出错地点等报告给用户。此外，为了尽可能多地发现错误，应在发现错误后还能继续编译下去，以便发现更多的错误。

一个编译过程可分为一遍、两遍或多遍完成，每遍都完成所规定的任务。例如，第一遍只完成词法分析的任务，第二遍完成语法分析和语义加工工作并生成中间代码，第三遍实现代码优化和目标代码生成。当然，也可以一遍完成整个编译工作。至于一个编译过程究竟应分为几遍完成以及如何划分，与源程序语言的结构、目标机器的特征有关。分多遍完成编译过程可以使整个编译程序的逻辑结构更清晰，但是遍数多势必增加读/写中间文件的次数，消耗过多的时间。

1.4 编译程序的开发

由于计算机语言功能的完善、硬件结构的发展、环境界面的友好等都对编译程序提出了更多、更高的要求，因此构造一个编译系统并非易事。虽然编译理论和编译技术的不断发展已使得编译程序的生产周期不断缩短，但是要研制完成一个编译程序仍需要相当长的时间，工作也相当艰巨。因此，高效、高质量地生成一个编译程序，一直是计算机系统设计人员追求的目标。

编译程序的任务是将源程序翻译成某台计算机上的目标程序。因此，开发人员首先要熟悉这种源程序语言，准确无误地理解源程序语言的语法和语义。此外，开发人员需要确定编译程序的开发方案及方法，这是编译开发过程中非常关键的一步，其作用是使编译程序具有易读性和易改性，便于将来对编译程序的功能进行更新和扩充。选择合适的语言编写编译程序也非常重要，语言选择不当会使开发的编译程序可靠性较差，难以维护且质量无法保证。目前，大部分编译程序都是用 PASCAL、C 和 ADA 这类高级语言编写的，不仅降低了开发工作量，也缩短了开发周期。最后，开发人员对目标机器要有深入的研究，这样才能充分利用目标机器的硬件资源和特点，产生质量较高的目标程序。

编译程序的开发常用自编译、交叉编译、自展和移植等技术实现。

1．自编译

用某种高级语言书写自己的编译程序被称为**自编译**。例如，如果 A 机器上已有可以运行的 PASCAL 语言，那么可以用 PASCAL 语言编写一个功能更强的 PASCAL 编译程序，然后借助原有的 PASCAL 编译程序对新编写的 PASCAL 编译程序进行编译，编译后即得到一个能在 A 机器上运行的功能更强的 PASCAL 编译程序。

2．交叉编译

交叉编译是指用 A 机器上的编译程序产生可在 B 机器上运行的目标代码。例如，如果 A 机器上已有可以运行的 C 语言，那么可用 A 机器中的 C 语言书写一个编译程序，它的源程序是 C 语言程序，产生的目标程序则是基于 B 机器的，即能够在 B 机器上执行的低级语言程序。

以上两种方法都假定已有一个系统程序设计语言可以使用，如果没有可使用的系统程序设计语言，那么可以采用自展或移植的办法来开发编译程序。

3．自展

自展方法如下：首先确定一种非常简单的核心语言 L_0，用机器语言或汇编语言书写出它的编译程序 T_0；然后，将语言 L_0 扩展为 L_1，此时有 $L_0 \subset L_1$，并用 L_0 编写 L_1 的编译程序 T_1（即自编译）；再后，将语言 L_1 扩展为 L_2，此时有 $L_1 \subset L_2$，并用 L_1 编写 L_2 的编译程序 T_2……以此类推，直至完成所要求的编译程序。

4．移植

移植是指 A 机器上的某种高级语言的编译程序稍加改动后能够在 B 机器上运行。一

个程序若能较容易地从 A 机器上搬到 B 机器上运行，则称该程序是**可移植的**。移植具有一定的局限性。

用系统程序设计语言来书写编译程序虽然缩短了开发周期，提高了编译程序的质量，但是实现的自动化程度不高。实现编译程序的最高境界是，能够有一个自动生成编译程序的软件工具，只要将源程序的定义及机器语言的描述输入该软件，就能自动生成该语言的编译程序，如图 1.6 所示。

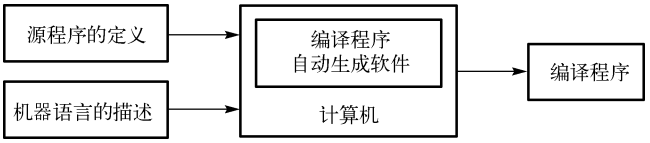

图 1.6 编译程序自动生成示意图

为了实现编译程序的自动生成，计算机科学家和软件工作者进行了大量研究工作，随着形式语言学研究的发展及编译程序自动生成工作的进展，出现了一些编译程序中某部分的自动生成系统，如 UNIX 操作系统下的软件工具 LEX 和 YACC 等。

1.5 构造编译程序所应具备的知识内容

要在某台机器上为某种语言构造一个编译程序，必须掌握如下三方面的内容。

（1）对被编译的源语言（如 C、PASCAL 等），要深刻理解其结构（语法）和含义。例如，下面的 for 循环语句：

```
for(i = 1; i <= 10 + i; i++ )
    x = x + 1;
```

就存在对循环终值的理解问题。一种理解是，以第一次进入 for 循环的 i 值计算出循环终值，该循环终值在循环中不再改变，即循环终值为 11；另一种理解是，循环终值表达式 10 + i 中的 i 值随循环不断改变，此时 for 语句将出现死循环。例如，TURBO PASCAL 就是按第一种语义进行翻译的，而 Turbo C 和 VC++ 6.0 是按第二种语义翻译的。此外，如果出了循环后还要引用 i 值，那么这个 i 值究竟是循环终值还是循环终值加 1？因此，对语义的不同理解可以得到不同的编译程序。

C 语言中的 for 循环语句还可以写成下面的形式：

```
for(i = 0, j = 5; i < 3, j > 0; i++, j--)
    printf("%d,%d\n", i, j );
```

逗号表达式“$i < 3, j > 0$”作为循环终值表达式，C 语言规定该表达式的值取逗号表达式中最右一个表达式的值，因此当 i 值为 3 时循环不终止，只有当 j 值为 0 时才结束循环。不了解 C 语言的逗号表达式的功能，就会得到错误的运行结果。

此外，对于下面的 C 语言函数：

```
#include<stdio.h>
int f(int x, int y)
{
```

```
    return x + y;
}
void main()
{
    int i = 3;
    printf("%d\n",f(i, ++i));
}
```

C 编译程序对函数参数传递的处理是由右向左进行的，因此先传递的是第二个参数++*i*，即 *i* 值先自增，由 3 变为 4，也就是说，这时函数 *f* 的两个实参的值都为 4，最终程序的运行结果是 8 而不是 7；不了解 C 语言的函数传递方式，就很容易得到错误的结果。

即使是同一条 C 语言语句，不同版本的编译系统翻译的结果也不一样。例如，

```
i = 3;
k =(++i) +(++i) +(++i);
```

对 VC++ 6.0 来说，执行语句“*k* =(++*i*) +(++*i*) +(++*i*);”的过程是，首先执行前两个“++*i*”，即对 *i* 进行两次自增，*i* 值变为 5，然后相加得到 10；接下来将这个 10 与第三个“++*i*”相加，即对 *i* 进行自增，其值由 5 变为 6，最后 10 加 6 得到 16，所以 *k* 值最终为 16。对 Turbo C 来说，执行语句“*k* =(++*i*) +(++*i*) +(++*i*);”的过程是，首先执行三次“++*i*”，即先对 *i* 进行三次自增，*i* 值变为 6，然后将自增后的三个 *i* 值相加，结果为 18，所以 *k* 值最终为 18。也就是说，不同的 C 编译程序给出了不同的解释。

（2）必须对目标机器的硬件和指令系统有深刻的了解。例如，产生两个数相加的指令在 IBM PC 的 8086/8088 汇编中假定用下面两种指令实现：

```
ADD   AX, 06  或  ADD   BX, 06
```

粗略看来，这两条加法指令除了寄存器不同，没有本质上的差别，其实不然。由于 AX 是累加器，因此从机器指令的代码长度来说（见附录 A），第一条指令比第二条指令节省一个字节。此外，从 PC 硬件结构来看，AX 本身就是累加器，且相加的结果也在累加器中，这就节省了传送的时间；第二条指令首先要将 BX 中的值送到累加器中，相加后又要从累加器中取出结果，再送回寄存器 BX。显然，第二条指令要比第一条指令费时，因此，只要可能，就应尽量生成像第一条指令这样的目标代码。

（3）必须熟练掌握编译方法，编译方法掌握得如何将直接影响到编译程序的成败，好的编译方法可能得到事半功倍的效果。

由于编译程序是一个极其复杂的系统，因此在讨论中只好将它分解，一部分一部分地研究。在学习编译程序的过程中，应注意前后联系，切忌用静止、孤立的观点看待问题；作为一门技术课程，学习时还必须注意理论联系实际，多练习、多实践。

习 题 1

1.1 完成下列选择题。

（1）下面叙述中，正确的是_____。

A．编译程序是将高级语言程序翻译成等价的机器语言程序的程序

B．机器语言因其使用过于困难，所以现在计算机根本不使用机器语言

C．汇编语言是计算机唯一能够直接识别并接受的语言

D．高级语言接近人们的自然语言，但是其依赖具体机器的特性是无法改变的

（2）将编译程序分成若干“遍”是为了_____。

A．提高编译程序的执行效率

B．使编译程序的结构更加清晰

C．利用有限的机器内存，并提高机器的执行效率

D．利用有限的机器内存，但降低了机器的执行效率

（3）构造编译程序应掌握_____。

A．源程序　　B．目标语言　　C．编译方法　　D．A～C 项

（4）编译程序绝大多数时间花在_____上。

A．出错处理　　B．词法分析　　C．目标代码生成　　D．表格管理

（5）编译程序是对_____。

A．汇编程序的翻译　　B．高级语言程序的解释执行

C．机器语言的执行　　D．高级语言的翻译

1.2 计算机执行用高级语言编写的程序有哪些途径？它们之间的主要区别是什么？

1.3 画出编译程序的总框图。如果你是某个编译程序的总设计师，设计编译程序时应当考虑哪些问题？