编译原理



类目：计算机类
书名：编译原理
主编：卓阳灼 金碧辉 王 颖
出版社：电子科技大学出版社
开本：大 16 开
书号：978-7-5770-2143-0
使用层次：通用
出版时间：2026 年 6 月
定价：59.80 元
印刷方式：双色
是否有资源：有

策划编辑：万晓桐
责任编辑：苏博麟
封面设计：

计算机类创新融合教材
“互联网+”教育改革新理念教材

BIANYI YUANLI

编译原理

主编◎卓阳灼　金碧辉　王　颖



电子科技大学出版社
University of Electronic Science and Technology of China Press

计算机类创新融合教材
“互联网+”教育改革新理念教材

BIANYI YUANLI
编译原理

编译原理
主编◎卓阳灼　金碧辉　王　颖
电子科技大学出版社

扫描二维码
关注更多教材信息

ISBN 978-7-5770-1143-0
9 787577 011430
定价：59.80元

计算机类创新融合教材
“互联网+”教育改革新理念教材

BIANYI YUANLI

编译原理

主　编◎卓阳灼　金碧辉　王　颖
副主编◎王海鹏　温承志　韦　星
席　洁　于雪涛　王豫峰



电子科技大学出版社
University of Electronic Science and Technology of China Press
·成都·

图书在版编目(CIP)数据

编译原理 / 卓阳灼，金碧辉，王颖主编. -- 成都 ：成都电子科大出版社，2026. 6. -- ISBN 978-7-5770-2143-0

Ⅰ. TP314

中国国家版本馆 CIP 数据核字第 2025GV0055 号

编译原理

BIANYI YUANLI

卓阳灼　金碧辉　王　颖　主编

策划编辑　万晓桐
责任编辑　苏博麟
责任校对　谢忠明
责任印制　梁　硕

出版发行　电子科技大学出版社
　　成都市一环路东一段 159 号电子信息产业大厦九楼　邮编　610051
主　　页　www. uestcp. com. cn
服务电话　028-83203399
邮购电话　028-83201495

印　　刷　三河市祥达印刷包装有限公司
成品尺寸　210 mm×285 mm
印　　张　14
字　　数　356 千字
版　　次　2026 年 6 月第 1 版
印　　次　2026 年 6 月第 1 次印刷
书　　号　ISBN 978-7-5770-2143-0
定　　价　59.80 元

版权所有　侵权必究

前言

编译原理是计算机科学与技术领域的核心课程之一，主要研究程序设计语言的翻译、转换和执行机制。编译原理不仅是构建高级语言编译器的理论基础，还在程序分析、代码优化、软件安全等领域具有广泛的应用。随着编程语言的不断演进和计算机体系结构的多样化发展，编译原理在提高程序性能、增强代码可靠性以及支持新兴计算范式等方面发挥着越来越重要的作用。

本书系统地介绍了编译程序的基本原理、设计方法和实现技术，主要内容包括词法分析、语法分析、语义分析及中间代码生成、符号表与错误处理、存储组织与分配、代码优化以及目标代码生成等核心环节。本书结合理论讲解与实例分析，旨在帮助读者深入理解编译过程的各个阶段，并掌握相关工具和技术的实际应用。

本书具有以下特点。

1. 结构清晰。本书按照编译过程的自然顺序组织内容，循序渐进地介绍编译器的构造方法。

2. 理论与实践相结合。本书在阐述理论的同时，提供典型算法示例和代码片段，帮助读者将知识应用于实践。

3. 注重实用性。本书介绍了词法分析器生成工具、语法分析器生成工具等现代编译工具的使用方法。

本书适合作为计算机相关专业的教材，也可供软件工程师、系统架构师以及对编译技术感兴趣的读者参考。

由于编者水平有限，书中难免存在疏漏之处，恳请读者批评指正。

编　者

2026 年 3 月

CONTENTS

目录

第 1 章 引论 …… 1

1.1 程序设计语言的发展历程 …… 1
1.2 程序设计语言的翻译机制 …… 4
1.3 编译程序概述 …… 7
1.4 编译程序的构造方法 …… 13
1.5 编译技术的主要应用 …… 17
思考练习 …… 18

第 2 章 词法分析 …… 19

2.1 词法分析概述 …… 20
2.2 手动编写词法分析程序 …… 22
2.3 自动生成词法分析程序的原理 …… 29
思考练习 …… 46

第 3 章 语法分析 …… 47

3.1 语法分析概述 …… 47
3.2 自顶向下的语法分析 …… 48
3.3 自底向上的语法分析 …… 59
3.4 语法分析程序的自动生成 …… 78
3.5 应用案例 …… 80
思考练习 …… 82

第 4 章 语义分析及中间代码生成 83
4.1 语法制导翻译概述 83
4.2 中间语言 86
4.3 自底向上语法制导翻译 93
4.4 自顶向下语法制导翻译 120
4.5 属性文法与属性翻译 124
思考练习 128

第 5 章 符号表与错误处理 129
5.1 符号表 129
5.2 错误处理 137
思考练习 146

第 6 章 存储组织与分配 147
6.1 存储组织概述 147
6.2 静态存储分配 150
6.3 动态存储分配 152
思考练习 156

第 7 章 代码优化 157
7.1 局部优化 158
7.2 循环优化 165
7.3 全局优化概述 180
7.4 代码优化示例 190
思考练习 195

第 8 章 目标代码生成 196
8.1 目标代码生成概述 196
8.2 假想的计算机模型 197
8.3 一种简单代码生成程序 199
8.4 寄存器分配 204

8.5 由 DAG 生成目标代码 …… 208
8.6 窥孔优化 …… 212
思考练习 …… 214

参考文献 …… 215

第 1 章

引　论

学习目标

清晰掌握程序设计语言从早期到现代的发展历程，包括不同阶段的代表性语言及其特点，理解推动语言发展的主要因素。

深入理解程序设计语言的翻译机制，包括编译和解释的工作原理、区别和适用场景。

明确编译程序的定义、功能和在软件开发过程中的重要作用，了解编译程序的基本组成部分。

了解编译程序的常见构造方法，掌握编译技术在软件开发、程序优化、代码安全等领域的主要应用。

1.1 程序设计语言的发展历程

编译技术的发展历程就是程序设计语言的发展历程。

ENIAC 被公认为世界上第一台通用电子计算机，它用电子管做计算。因此，尽管它能执行复杂的操作序列(如循环、分支和子程序)，但它没有存储器，只能理解机器语言(二进制)。现在许多程序员自嘲是“码农”、干体力活的，其实，第一代程序员确实要干体力活，程序员通常要花好几周的时间才能把一个数学计算问题变成 ENIAC 能够理解和执行的一个输入操作序列；然后，程序员又要花好几天时间操作各种开关，连接各种电缆，通过穿孔卡片完成输入；最后，程序员还要通过计算机的单步执行来进行测试和验证。

1946 年，“现代计算机之父”约翰·冯·诺依曼在论文中阐述了“存储程序原理”。从此，人们把程序本身也当作数据来对待，将程序和该程序处理的数据以同样的方式进行存储。1949 年 5 月，世界上第一台“存储程序式”计算机 EDSAC 在英国剑桥大学数学实验室成功运行，为程序设计语言的发展提供了硬件支持。在 EDSAC 成功运行的同时，莫里斯·文森特·

威尔克斯还主持完成了世界上第一个汇编器，这标志着汇编语言的诞生。此后，陆续出现了许多其他汇编器。但汇编语言在计算机发展史上并不是一个非常重大的发明，因为汇编语言和机器语言很像，都是直接面向机器的，同样难以被人类读写。因此，汇编语言被称为低级语言。如果以人类进化史来类比，编译技术此时如同类人猿开始学习使用的自然工具，还没有任何理论基础。

与汇编语言不同，面向电子计算机的高级程序设计语言的出现则是计算机发展史上的里程碑。1951 年，当时还在国际商业机器公司(IBM 公司)任职的约翰·巴克斯开始针对汇编语言的缺点着手开发脱离机器的高级语言。1954 年，约翰·巴克斯正式将这种脱离机器、用于公式计算的语言命名为“FORTRAN”。FORTRAN 的编译程序使用汇编语言编写，虽然其功能简单，但它具有开创性，在当时引起了极大的反响。1957 年，第一个 FORTRAN 编译器在 IBM 704 计算机上实现，并首次成功运行了 FORTRAN 程序。FORTRAN 的出现使得当时以科学计算为主要目的的软件的生产力提高了一个数量级，奠定了高级语言的地位。

“编译器”这一术语是由“计算机软件工程第一夫人”格蕾丝·霍珀提出来的。1950 年，格蕾丝·霍珀开发了一个程序，将数学符号翻译成机器语言。她将这个程序称为“编译器”，并给这个“编译器”取了一个简单的名字“A-0”。之后的两年，格蕾丝·霍珀进一步完善了“A-0”编译器的功能，并于 1952 年在公开发表的论文中介绍了“A-2”编译器。1956 年，格蕾丝·霍珀开发出了一套完整的程序设计语言“FLOW-MATIC”。考虑到如果各地开发者将编译器稍微修改一小部分，就会出现 A 地的程序在 B 地无法执行的问题，格蕾丝·霍珀写了另一套程序“Validation”，来检查程序是否采用同样的编译方式。在此基础上，渐渐发展出一套偏重商业使用的新语言——面向商业的通用语言(Common Business Oriented Language，COBOL)。

1955 年，德国的应用数学和力学学会着手设计通用的、与计算机无关的算法语言。1957 年年末，该学会邀请国际计算机学会的专家(约翰·巴克斯位列其中)加入开发计划。1958 年 5 月，两国专家在苏黎世起草了一份名为“国际代数语言”的报告，随后，新语言被定名为 ALGOL。1958 年 12 月，ALGOL 小组发表了 ALGOL 58 报告，引起广泛的反响。在这份报告中，专家们首次使用了一种形式化的语言描述体系对 ALGOL 的语法体系结构进行描述。1960 年 1 月，第二次 ALGOL 会议在巴黎召开，来自丹麦的彼得·诺尔对约翰·巴克斯提出的语言语法描述方案进行了仔细审阅和修改，并使用这种新的方案对 ALGOL 60 的语法体系结构进行描述。后来，人们把这种语言语法描述方案称为巴克斯-诺尔范式(Backus-Naur Form，BNF)。

ALGOL 在欧洲得到了广大计算机工作者的认可，但 IBM 公司一心要推行 FORTRAN，不支持 ALGOL，导致 ALGOL 60 始终没有发展起来。尽管如此，因为采用 BNF 可以用简洁的公式把各种语法规则严格而清晰地描述出来，ALGOL 60 在程序设计语言发展史上仍是一个重要的里程碑。从此，编译器的开发有了理论支撑，编译技术蓬勃发展，为我们打造了流派众多、异彩纷呈的程序语言世界。

20 世纪 60 年代是软件发展史上极其重要的十年。一方面，1962 年美国金星探测器“水手 2 号”发射失败，引发了所谓的“软件危机”；另一方面，由于计算机硬件成本不断下降，人们对大型系统软件的需求越来越强烈。1962 年，哈佛大学的肯尼斯·艾弗森支持开发了 APL 语言，首次提出动态数据(向量)的概念。1963 年到 1964 年，IBM 公司组织了一个委员会，试

图开发一种功能齐全的大型语言，希望它兼有 FORTRAN 和 COBOL 的功能，以及类似 ALGOL 60 的完善的定义及控制结构，并将这个语言命名为“PL/1”。这是人们对开发大型通用语言的第一次尝试，提出了许多有益的新概念、新特征。然而，由于过于复杂、数据类型自动转换太灵活、可靠性差、低效，它没能普及。为了普及程序设计语言教学，美国达特茅斯学院的约翰·凯梅尼和托马斯·卡茨于 1967 年开发出了交互式、解释型语言 BASIC。BASIC 的解释程序小(仅 8 KB)，因此，BASIC 取得了众所周知的成就。同年，挪威计算机科学家奥尔·约翰·戴尔等人研发出通用模拟语言 Simula 67。它以 ALGOL 60 为基础，首次提出了类(Class)的概念，将数据和其上的操作集为一体。这是抽象数据类型及对象的鼻祖。1964 年，瑞士苏黎世理工学院的尼古拉斯·沃斯在 ALGOL 60 的基础上提出了近于结构化语言的 ALGOL W。由于简洁、完美和结构化，ALGOL W 成为软件教程中的示例语言，是后来闻名于世的 Pascal 的雏形。

1971 年，Pascal 正式问世。由于在设计之初就本着“简单、有效、可靠”的原则，在人们为摆脱“软件危机”而对结构化程序设计寄予厚望时，Pascal 很快得到了普及。在尼古拉斯·沃斯的主持下，研究者用 Pascal 编写了一个用于 Pascal 自身的编译程序，以此拉开了使用高级程序设计语言编写编译程序的大幕。1972 年，AT&T 公司贝尔实验室的丹尼斯·里奇在肯尼斯·汤普逊的系统程序设计语言 BCPL 的基础上开发了 C 语言。因为有大量环境工具支持，而且用 C 语言编写的程序短小精悍、调试速度快，所以 C 语言很快取代了 Pascal，一跃成为系统程序设计语言。除了编写各种应用程序，研究者也开始使用 C 语言编写其他语言的编译程序。

1978 年，约翰·巴克斯在图灵奖的颁奖典礼上发表了题为“程序设计能从冯·诺依曼风格中解放出来吗?”的演讲，指出了传统过程式语言的不足，激励研究者转向对多范型语言的研究，在此之后涌现了大量的非过程式语言。LISP、APL 和 ML 是典型的函数式语言，该类语言不使用赋值语句，其语法形式与数学中的函数类似；Prolog 是逻辑式语言的典型代表，这种语言以逻辑程序设计思想为理论基础，其核心是事实、规则和推理机制；Smalltalk、C++ 和 Java 等语言是面向对象程序设计语言的典型代表，这类语言的主要特征是把数据以及处理数据的操作一起作为对象进行处理。

随着计算机性能的快速提高，计算机程序也越来越复杂，程序的开发已经远比程序的运行时间紧迫。作为系统程序设计语言的补充，各种不同用途的脚本语言(如 Perl、Python、Ruby、JavaScript 等)如雨后春笋般涌现，编程语言面临的问题已经由执行效率需求与性能低下的硬件之间的矛盾转变为快速变化的市场需求与低效的开发工具之间的矛盾。适应市场需求的语言的快速开发对编译技术提出了更大的挑战。

1.2 程序设计语言的翻译机制

目前计算机只能识别机器语言编写的程序，因此，用汇编语言或高级程序设计语言(简称“高级语言”)编写的程序都必须翻译成机器语言才能被计算机识别。完成这一过程的程序称为翻译程序。

翻译程序，也称为“语言处理程序”，它可以对 A 语言编写的程序(称为源程序)进行等价分析处理。源程序经过处理后，可能被改造成等价的 B 语言程序(称为目标程序)，也可能需要与用户提供的初始数据结合在一起直接产生程序的结果。

用于编写源程序的语言称为源语言，用于编写目标程序的语言称为目标语言。每种汇编语言或高级语言都有自己的翻译程序。汇编程序、编译程序和解释程序都是翻译程序。

1. 汇编程序

如果一种翻译程序的源语言是某种汇编语言，其目标语言是某种型号的计算机的机器语言，那么这种翻译程序被称为汇编程序。汇编程序一般对源程序进行两遍扫描来完成翻译。第一遍进行存储分配，构造出第二遍扫描时用的各种表格；第二遍用机器操作码代替源程序中的符号。在两遍扫描过程中，汇编程序会对各种错误进行检查和分析，以便编程人员对错误进行修改。

图 1-1 给出了汇编程序的执行过程。源程序经过汇编程序翻译后产生目标程序，利用输入的初始数据就可以运行产生结果数据。

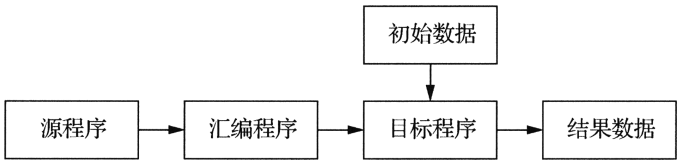


图 1-1　汇编程序的执行过程

2. 编译程序

如果源语言是高级语言，目标语言是汇编语言或低级语言，那么称这种翻译程序为编译程序(或编译器)。

图 1-2(a)和图 1-2(b)描述了编译程序的执行过程，其区别在于编译程序的目标语言不同，在图 1-2(a)中产生的是机器语言编写的目标程序，而在图 1-2(b)中产生的是汇编语言编写的目标程序。如果编译生成的目标程序采用的是汇编语言，就需要增加一个汇编程序将其转换为采用机器语言的目标程序。

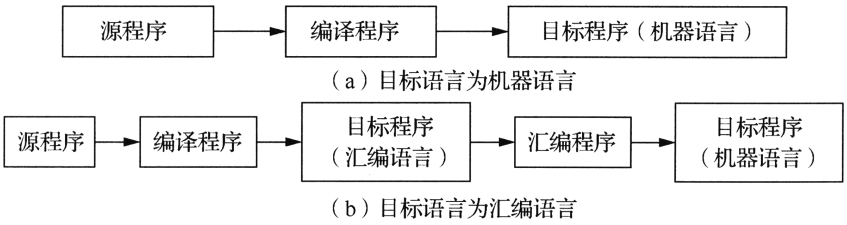


图 1-2　编译程序的执行过程

高级语言编写的源程序在机器上的执行过程可分为以下两个阶段。

(1)编译阶段：将源程序翻译成等价的目标程序。

(2)运行阶段：在机器上执行目标程序，以获得结果数据。

现代编译器只负责编译阶段的工作，运行阶段的工作由操作系统完成。将编译得到的目标程序加上运行系统(如服务子程序、动态分配程序、装配程序等)就可获得结果数据，如图 1-3 所示。

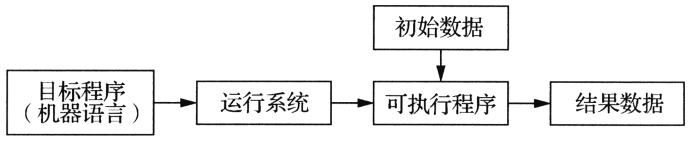


图 1-3　编译程序的运行阶段

3. 解释程序

解释程序也是一种翻译程序，它将高级语言编写的源程序作为输入，但并不产生目标程序，而是边解释边执行源程序本身。如图 1-4 所示，源程序与初始数据一起输入解释程序，解释程序读入一条源程序语句即解释执行一条语句，下一条语句的执行依赖于上一条语句的执行结果，最终输出结果数据。

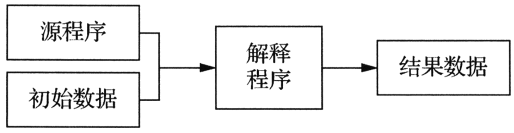


图 1-4　解释程序的执行过程

解释程序用于会话型语言，如 Python。解释程序的主要优点是易于为用户提供调试功能，对源程序的语法分析和出错处理都很及时，但是解释程序执行速度较慢，运行效率低。

编译程序和汇编程序以一种静态的方式(并不执行程序)对程序员编写的源程序加以理解，并将其转换成另一种等价的代码，这种机制称为编译机制。解释程序是另一种形式的语言处理程序，它把翻译和运行结合在一起，边翻译源程序，边执行翻译的结果，这种机制称为解释机制。如果以翻译英文原著作为类比对象，则原著相当于源程序，译著相当于目标程序，计算机的运行相当于阅读。编译程序和汇编程序对源程序的处理过程与笔译类似，即产生文本型的译著，也可以称为“离线方式”(Offline)；解释程序的工作相当于现场口译，即翻译人员一边看原著，一边翻译给在场的读者听，也可以称为“在线方式”(Online)。

【例 1.1】　典型的 C 语言的翻译程序(如 GCC)采用了一次编译的机制。GCC(GNU Com-

piler Collection，GNU 编译器套件)对源程序的处理过程如图 1-5 所示。它先对以 . c 或 . h 为扩展名的 C 语言编写的源程序进行预处理，再编译、汇编得到以 . o 为扩展名的机器代码，最后完成链接功能，生成以 . exe 为扩展名的可执行程序。

【例 1. 2】 Java 语言的翻译程序结合了编译和解释两种机制。Java 处理器对源程序的处理过程如图 1-6 所示。

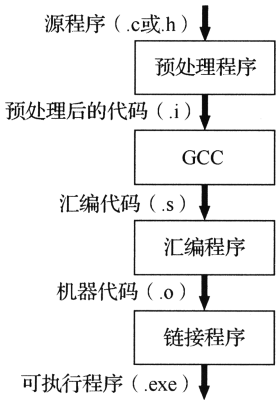


图 1-5　GCC 对源程序的处理过程

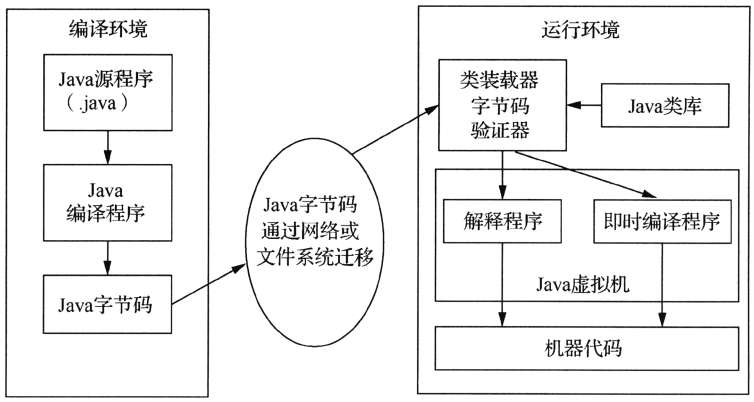


图 1-6　Java 处理器对源程序的处理过程

一个 Java 源程序首先被编译成一个被称为字节码(Bytecode)的中间表示形式。字节码的引入有效地保障了 Java 程序的可移植性和安全性。Java 的编译过程同 C/C++的编译略有不同。C/C++编译器编译生成一个对象的代码是为了在某一特定硬件平台上运行，因此，在编译过程中，编译器通过查表将所有对符号的引用转换为特定的内存偏移量，以保证程序的运行。而 Java 编译程序不对变量和方法的引用编译为数值引用，也不确定程序执行过程中的内存布局，而是将这些符号引用信息保留在字节码中，后面的工作交给 Java 虚拟机(JVM)处理。

Java 字节码最初是通过解释程序(Interpreter)来解释执行的，但由于解释执行速度比较慢，JVM 发现某个方法或代码块运行特别频繁的时候，就会认为这是“热点代码”(Hot Spot Code)。为了提高热点代码的执行效率，这些热点代码会被编译成与本地机器相关的机器代码，并进行各个层次的优化。完成这个任务的编译程序就是即时编译程序(Just In Time Compiler)。

【例 1. 3】 微软公司为其旗下的所有高级程序设计语言(如 C#、VB 等)设计了二次编译的翻译机制。C#等处理器对源程序的处理过程如图 1-7 所示。处理器首先将源程序编译成通用的微软中间语言(Microsoft Intermediate Language，MSIL)字节码，然后通过公共语言运行时(Common Language Runtime，CLR)运行环境将 MSIL 字节码编译成机器代码。

从上述对源程序的处理过程可以看出现代高级程序设计语言编译程序的发展趋势：从“一次编译”非托管到“编译+解释”或“编译+编译”的二次托管。目前很多新出现的程序设计语言都采用二次托管方式对源程序进行处理。

在三种语言处理程序中，汇编程序最容易实现，其次是解释程序，编译程序最难。因此，只要掌握了编译程序的实现方法，汇编程序和解释程序的实现就迎刃而解了。下面我们主要讨论编译程序的实现方法和原理。在后文中，我们将对“编译器”和“编译程序”这两个术语不加区分。

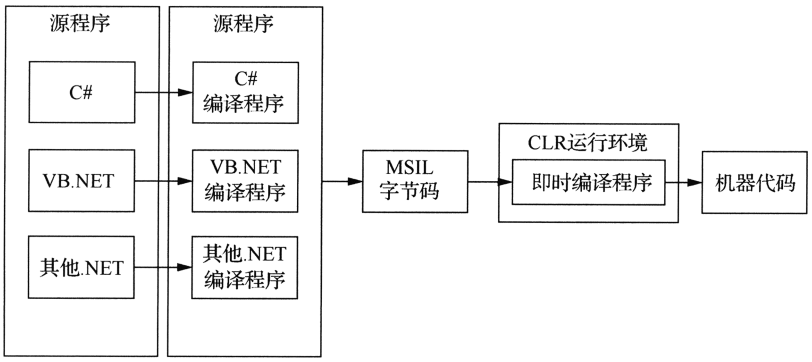


图 1-7　C#等处理器对源程序的处理过程

1.3 编译程序概述

1.3.1 编译程序的逻辑结构概述

编译程序的功能是将高级程序设计语言编写的源程序等价翻译成汇编语言或机器语言编写的目标程序。编译的过程与外文资料的翻译过程类似。例如，我们要将英文句子“I wish you happiness”翻译成中文，大致经过以下几步。

(1)阅读原文。阅读每个英文字母，这个过程可以称为“源语言读入”。

(2)分析是否构成有意义的单词。根据英文的词法规则，识别出 4 个英语单词：“I”“wish”“you”“happiness”。这个过程可以称为“词法分析”。

(3)根据英文的语法规则，对这些单词进行语法关系分析，检查这些单词是否能够组成一个符合英文语法的句子。其中：“I”是代词，可以作主语；“wish”是动词，可以作谓语；“you”是代词，可以作间接宾语；“happiness”是名词，可以作直接宾语。因此这些单词可以组成一个符合英文语法的句子。这个过程可以称为“语法分析”。

(4)对句子进行语义分析，并初步译成中文。“I”的中文意思是“我”，“wish”的中文意思是“希望”或“祝福”等，“you”的中文意思是“你”或“你们”等，“happiness”的中文意思是“高兴”或“幸福”等。因此可以将此句子初步翻译成“我希望你(们)幸福”。这个过程可以称为“语义分析及中间代码生成”。

(5)对初步译文进行润色。根据上下文的关系及中文语法的相关规则进行综合考虑，对“我希望你(们)幸福”进行必要的润色。这个过程可以称为“代码优化”。

(6)最后译文为“祝你幸福”。这个过程可以称为“目标代码生成”。

与上述英文翻译成中文的过程类似，编译过程大致可以划分为词法分析、语法分析、语义分析及中间代码生成、代码优化、目标代码生成 5 个阶段(源程序读入作为初始化过程不列

入编译过程)。

下面我们通过一个简单的 C 程序来说明编译的基本过程。

float a, b;

a = 3 * a+b;

(1)词法分析

词法分析(Lexical Analysis)的主要任务是将有独立意义的单词识别出来。进行词法分析的程序被称为词法分析器(Lexical Analyzer)或词法扫描器(Scanner)。词法分析器将源程序看成由一个个符号组成的符号流，根据语言的词法规则识别出具有独立意义的最小语法单位，即单词(Token)，如关键字、标识符、运算符等。上面的 C 程序通过词法分析可识别出如下单词。

①关键字：float。

②标识符：a、b。

③常数：3。

④运算符：=、*、+。

⑤界限符：,、;。

将单词识别出来后，为了便于编译程序处理(主要是便于构造符号表和存储管理)，需要将这些长度不统一的单词转换成长度统一、格式规范的内部编码。通常情况下，一个单词的内部编码由两部分组成：一部分是类别编码，用于区别单词种类，如关键字、标识符、运算符等；另一部分是单词的值，即单词本身的编码。单词的内部编码如图 1-8 所示。

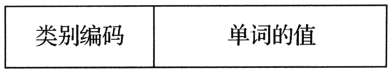


图 1-8　单词的内部编码

除了识别出单词并进行编码之外，词法分析的功能还包括剔除无用字符(空白符、回车符、注释等)。如果在单词识别过程中发现不符合词法规则的单词，词法分析器应报告发现的错误。

由于词法分析器接收的是符号流，输出的是经过内部编码的单词流，因此词法分析也被称为线性分析(Linear Analysis)。

(2)语法分析

语法分析(Syntax Analysis 或 Parsing)的主要任务是根据语法规则从单词流中识别出各种语法成分，如表达式、说明、语句、过程和函数等，并检查各种语法成分在语法结构上是否正确。通常将语法分析的结果表示为语法树，例如，语句“a = 3 * a+b;”经过分析后可以表示成如图 1-9 所示的语法树。如图 1-10 所示为这种语法树更简单的一种形式，运算符不再以叶结点的形式出现，而是作为运算对象的父结点(这种语法树被称为抽象语法树)。

进行语法分析的程序被称为语法分析器(Syntax Analyzer 或 Parser)。如果发现不符合语法规则的地方，则指出错误，以便用户修改。

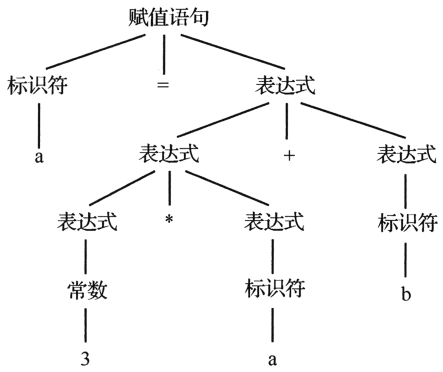


图 1-9　语句 a=3 * a+b; 的语法树

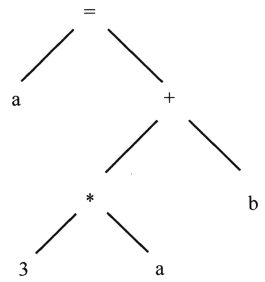


图 1-10　语句 a=3 * a+b; 的抽象语法树

(3)语义分析及中间代码生成

语义是指语言的各种语法成分的具体含义。语义分析(Semantic Analysis)的主要任务是根据语法规则分析已经识别出来的各种语法成分的含义。因此，语义分析与语法分析密切相关。通常语法分析器在分析语法成分的同时会对该语法成分进行语义分析，这种翻译的方式被称为语法制导翻译(Syntax-Directed Translation)。

在进行语义分析时，语法分析器还会进行相应的语义检查，以保证源程序在语义上的正确性。例如，要检查表达式中赋值符号两边的数据类型是否一致；对于语句“a=3 * a+b;”中的整数 3，在语义分析阶段应该将其转换为实数，再与 a 进行乘法运算。

语义分析是整个编译程序中最有实质意义的翻译任务，识别出来的各种语法成分经过语义分析被翻译成中间代码(或目标代码)。中间代码是介于源语言和目标语言之间的一种中间语言形式。生成中间代码主要是为了便于代码优化和目标代码的移植。目前常见的中间语言形式有三元式、四元式、逆波兰表示等。中间语言至少应该具有两个性质：容易产生和容易翻译成目标语言。编译程序设计者也可以自己设计中间语言，如 Java 字节码、微软中间语言等。

例如，可以将语句“a=3 * a+b;”翻译成如下四元式序列。

①(itf, 3, , t1)　　//itf 表示将 int 类型(简称整型)变量转换成浮点型变量

②(* , t1, a, t2)

③(+, t2, b, t3)

④(=, t3, , a)

(4)代码优化

代码优化(code optimization)的主要任务是对中间代码进行等价的加工处理，以得到更高质量的目标代码。空间复杂度和时间复杂度是衡量目标代码质量的两大指标，即目标代码所占用的空间要少，运行时间要短。

对上述四元式序列进行代码优化，可以将四元式①直接放在编译时处理，将整型变量 3 转换成浮点型变量 3.0，以免在运行时再进行整型变量到浮点型变量的转换。优化后的四元式序列如下。

①(* , 3.0, a, t1)

②(+, t1, b, t2)

③(=, t2, , a)

t2 在四元式③中是单目运算，因此可以用 a 代替 t2。这样再次优化后得到的四元式序列如下。

①(*, 3.0, a, t1)

②(+, t1, b, a)

(5)目标代码生成

目标代码生成(Code Generation)的主要任务是把经过优化的中间代码变成期望的目标代码(特定机器的机器语言或汇编语言)。目标程序十分依赖硬件系统，因此，如何充分利用寄存器、合理选择指令，生成尽可能短且有效的目标代码，取决于具体的计算机体系结构。

以汇编语言为例，利用寄存器 R_1 和 R_2，经过优化的中间代码可生成如下目标代码。

MOV a, R_1

MUL 3.0, R_1

MOV b, R_2

ADD R_1, R_2

MOV R_2, R_1

从概念上讲，可以把以上 5 个阶段划分成两部分：编译前端和编译后端。

与源语言相关的部分被称为编译前端，包括词法分析、语法分析、语义分析及中间代码生成 4 个阶段；与目标语言相关的部分被称为编译后端，包括代码优化和目标代码生成 2 个阶段。将编译程序划分为编译前端和编译后端，不仅有利于代码优化，而且对目标代码的生成和移植更有利。

一方面，可以给同一个编译前端配不同的编译后端，这样就能在不同的计算机上构造出同一语言的编译程序，例如 Java 处理器就采用了这种思想。对不同的计算机，Java 处理器的编译前端是相同的，产生同一种类型的中间代码——字节码，而不同的计算机上配置了能够解释执行这种中间代码的编译后端。

另一方面，也可以给不同的编译前端配同一个编译后端，这样就可以在同一台计算机上生成多种语言的编译程序。例如，源语言升级后，只需要对编译程序的编译前端进行相应的修改即可。一个典型的实例就是被广泛使用的 GCC，其编译前端是多种程序设计语言的不同分析器，支持 C、C++、Objective-C++、Java、FORTRAN、Pascal、COBOL、Ada、Go 等语言。GCC 以这些语言的源程序文件作为输入，经过词法分析、语法分析和语义分析，产生一种抽象语法树(Abstract Syntax Tree，AST)形式的中间代码；GCC 的编译后端对 AST 形式的中间代码进行分析处理，最终产生目标代码。

因此，从理论上说，如果引入中间语言，将编译程序划分为编译前端和编译后端，那么 M 个源语言和 N 个目标语言之间只需要 M+N 个编译程序；否则，需要 M×N 个编译程序。但实际上，很难找到一种通用的中间语言，适合 M 个源语言和 N 个目标语言。

尽管编译过程与英文翻译过程类似，但是编译对象毕竟不是自然语言，计算机处理器也不是人类大脑，因此编译过程中必然有一些特定的问题。例如，编译过程中涉及的各种符号

存储在什么地方，如何构造和查询各种符号表，运行时如何分配存储空间，发现不符合词法规则和语法规则的符号串时应如何处理，等等。

因此，编译的各个阶段都涉及符号表管理和错误处理。

(1)符号表管理

符号表管理(Symbol Table Management)也被称为“表格管理”，即管理编译过程中产生的各种表格。其主要功能是按照编译过程中的信息需求，生成不同用途的符号表(如常数表、名字特征表、循环层次表等)，并提供合适的方式查询、修改和维护这些表格。完成造表并对表格进行增、删、查、改的程序被称为符号表管理程序。

例如，对于“float a，b;”这条说明语句，词法分析器识别出标识符 a 和 b 后，将其插入名字特征表，但 a 和 b 的各种属性在后续阶段才能填入：标识符的数据类型要在语义分析阶段确认；标识符的地址可能要在目标代码生成阶段才能确定。

(2)错误处理

由不同用户编写的程序难免会包含各种类型的错误。编译程序不仅要能及时地发现这些错误，而且还要将错误信息(如错误类型、出错位置等)以恰当的形式及时地通知用户。为了尽可能多地发现错误，编译程序要设法将错误的影响限制在尽可能小的范围内(这种处理方式称为错误的局部化)，以保证编译程序在发现错误以后还能继续完成剩余部分的分析处理。更高级的错误处理程序还能够纠正某些错误。

在编译的不同阶段，通常会发现不同的错误。

在词法分析阶段通常会发现拼写方面的错误，例如，将半角的分号输入成全角的分号，标识符的拼写不符合要求，等等。

在语法分析阶段，通常会发现更多的语法错误，例如，嵌套的 for 循环少写了一个“}”。

在语义分析阶段，通常会发现类型不匹配、参数不匹配等问题。

综上所述，一个典型的编译程序的逻辑结构如图 1-11 所示。对于具体的编译程序而言，逻辑关系是多种多样的，有些阶段的工作可以组合和交叉进行，有的可以省略。例如，一个小型的编译程序可以在进行语义分析之后直接生成目标代码，省略中间代码生成和代码优化阶段，但是其他部分的工作基本是不能省略的。

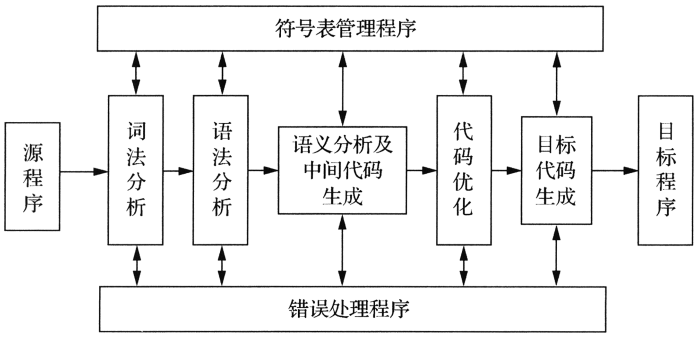


图 1-11　编译程序的逻辑结构

1.3.2 编译程序的分遍

在编译程序的工作过程中，词法分析、语法分析、语义分析及中间代码生成、代码优化、目标代码生成这 5 个阶段可以只对源程序进行一次扫描，也可以对源程序或等价源程序进行若干次扫描。对源程序或等价源程序进行相关的加工处理工作，称为遍(Pass)。

每扫描一遍源程序或等价源程序，都要进行等价变换，使其更接近目标程序。如果只对源程序进行一遍扫描便直接生成目标程序，则称编译程序是单遍的，如图 1-12 所示。而把源程序分为几遍来编译，每遍只完成编译程序中的一部分或几部分工作，则称为多遍的编译程序，如图 1-13 所示。例如，第一遍进行词法分析、语法分析，检查语法错误；第二遍生成中间语言，进行存储分配；第三遍生成可运行的目标程序。

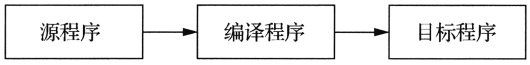


图 1-12 单遍的编译程序

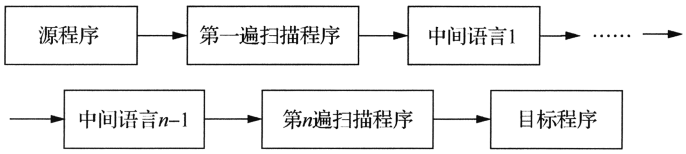


图 1-13 多遍的编译程序

决定编译程序分遍数目的因素有以下几个。

①计算机存储容量的大小。

②编译程序功能的强弱。

③源语言的繁简。

④目标程序的优化程度。

⑤设计和实现编译程序时所用工具的先进性。

⑥参加人员的数量和素质等。

一般来说，当计算机存储容量较小、源语言较烦琐、编译程序功能很强、目标程序优化程度较高、设计和实现编译程序时所用工具较先进、参加人员较多且素质较高时，适合采用多遍扫描方式。多遍扫描每遍功能相对独立、单纯，逻辑结构清晰，优化准备工作充分，并有利于多人分工合作，因此多遍的编译程序加工充分，出错处理细致，目标程序质量高；不足之处是不可避免地要进行一些重复工作，多遍之间有一定的交接工作，因此增加了编译开销。目前大多数高级程序设计语言的编译程序都是多遍的。

1.4 编译程序的构造方法

编译程序是一个非常复杂的系统软件，虽然编译理论和编译技术经过不断发展，已使编译程序的生产周期不断缩短，但目前开发一个编译程序仍需要相当长的时间，而且工作相当艰巨。因此，高效地开发高质量的编译程序一直是人们追求的目标。

1.4.1 编写编译程序的一般方法

编译程序虽然是一个非常复杂的系统软件，但本质上也是运行在计算机上的程序，所以它也是用程序设计语言编写的。根据语言类型，编写编译程序的一般方法分为下面 3 种。

(1)用机器语言编写编译程序

机器语言是早期用于编写编译程序的唯一语言，但由于机器语言难读难写，现在几乎没有人再用它编写编译程序。

(2)用汇编语言编写编译程序

汇编语言太依赖硬件环境，且程序过于冗长，因此现在已不常用。不过，通过汇编程序产生目标代码效率比较高，所以编译程序的核心部分常用它编写。

(3)用系统程序设计语言编写

无论从可读性、可靠性、可维护性，还是从编写效率来说，机器语言和汇编语言这些低级程序设计语言都不能满足编译程序的开发需求。从 20 世纪 80 年代后，几乎所有的编译程序都用高级语言编写，从而避开了许多与计算机有关的烦琐细节，大大减少了编写编译程序的工作量。

并非所有高级语言都适合用于编写编译程序。能够用于编写编译程序或其他系统软件的高级语言称为系统程序设计语言。例如，Pascal、C、C++、Java 等高级程序设计语言不仅可以用来编写其他高级程序设计语言的编译程序，还可以用于编写其自身的编译程序。ALGOL 和 FORTRAN 等语言主要用于科学计算，并不适合用于编写编译程序。

除了手动编写编译程序来对源程序进行形式化描述，还可以构建一些自动生成系统，通过输入语言的规范说明，自动生成编译程序的某些部分，如 Lex、YACC、Parser Generator、ANTLR 等。另外，研究者也在努力提供一些开发工具来帮助构造编译程序，这些工具能够在一定程度上降低编译程序的构造难度，提高构造效率，如 Make、Ant 等。

1.4.2 编译程序的开发技术

从第一个 FORTRAN 编译程序出现开始，人们就在不断探索如何利用新的方法和技术提

高编译程序的开发效率。为了方便讨论，下面首先介绍 T 型图，然后分别介绍这些开发技术。

(1)T 型图

一个编译程序通常会涉及 3 种语言：源语言、目标语言和书写语言。源语言和目标语言体现了编译程序的功能，书写语言描述了编译程序所需的运行环境。三者之间的关系可用 T 型图来表示，如图 1-14 所示。图 1-14 左上角是源语言 S，右上角是目标语言 D，底部是书写语言 W。该 T 型图代表源语言 S 的编译程序，意思是“书写语言 W 编写的源语言 S 的编译程序，其目标语言是 D”。

假设 A 机器上有一个能够直接运行产生 A 机器语言的 Pascal 的编译程序，该编译程序可用 T 型图表示，如图 1-15 所示。

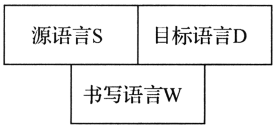


图 1-14　T 型图



图 1-15　T 型图示例 1

(2)自编译技术

自编译技术是由瑞士苏黎世理工学院的尼古拉斯·沃斯教授提出来的，他主持利用该技术成功地构造了 Pascal 的编译程序。

如果一种高级语言的编译程序也能直接用该语言写出来，那么这种语言被称为自编译语言。利用自编译语言开发自身或其他语言的编译程序的技术称为自编译技术。

假设 A 机器上已经有了如图 1-15 所示的编译器，那么我们可以在该机器上开发任何用 Pascal 书写的程序——包括 Pascal 的编译程序，如图 1-16 所示。类似地，我们也可以用 Pascal 编写 Java 的编译程序，如图 1-17 所示。

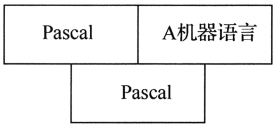


图 1-16　T 型图示例 2

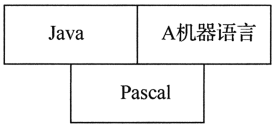


图 1-17　T 型图示例 3

用 Pascal 编写的编译程序不能直接在 A 机器上运行，因此需要利用原来的编译器进行编译处理，其处理过程和结果如图 1-18 所示。这就是利用 Pascal 语言开发自身编译程序的自编译过程。如图 1-19 所示为利用 Pascal 开发 Java 的编译程序的自编译过程。

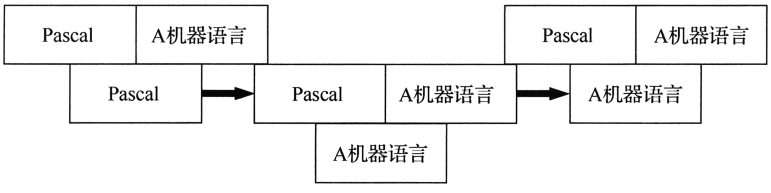


图 1-18　利用 Pascal 开发自身编译程序的自编译过程

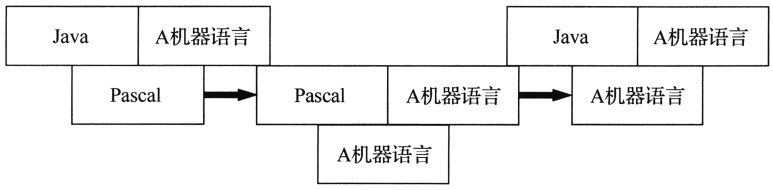


图 1-19　利用 Pascal 开发 Java 的编译程序的自编译过程

(3) 自展技术

自展技术是利用自编译技术将一个功能较弱的编译程序一级一级扩展成一个功能较强的编译程序的技术。假设我们为一种高级语言 L 在 A 机器上编写一个编译程序，则自展过程如下。

①选取 L 的一个子集 L_1(语言核心部分)，选取原则是尽量简单、刚好够书写其自身的编译程序。首先用 A 机器语言来建立 L_1 的编译程序，如图 1-20 所示。

②为了检验 L_1 是否取得适当，以及构建的 L_1 的编译程序是否正确，可用 L_1 重写一次 L_1 的编译程序，如图 1-21 所示。

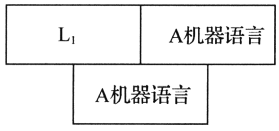


图 1-20　T 型图示例 4

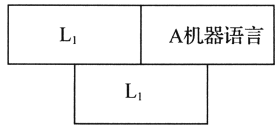


图 1-21　T 型图示例 5

③利用编译器进行自编译处理，如图 1-22 所示。

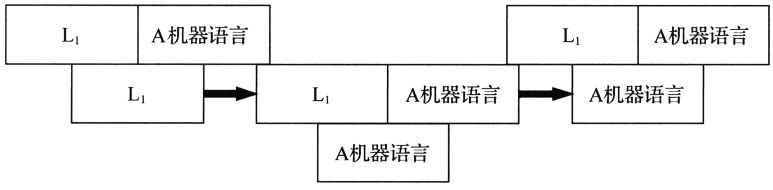


图 1-22　利用 L_1 开发 L_1 的编译程序的自编译过程

④把语言功能由 L_1 扩展到 L_2(L_2 包含 L_1)，开发 L_2 的编译程序的自编译过程如图 1-23 所示。

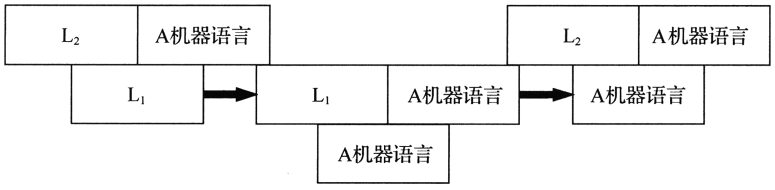


图 1-23　利用 L_1 开发 L_2 的编译程序的自编译过程

⑤把语言功能由 L_2 扩展到 L_3(L_3 包含 L_2)，开发 L_3 的编译程序的自编译过程如图 1-24 所示。

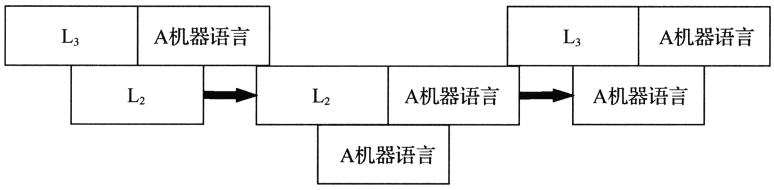


图 1-24　利用 L_2 开发 L_3 的编译程序的自编译过程

⑥类似地，对 L_4，L_5，…，L_i，…按上述过程进行多次编译，直到扩展到 L。这就是由语言核心部分 L_1 逐步扩展到 L 的自展过程。

由上述过程可以看出，使用自展技术，可以使手动编写编译程序的工作量大大减少。这种技术使得后一级编译程序建立在前一级编译程序的基础上，并逐步丰富语言功能，直至获得所需语言的编译程序。

(4)交叉编译技术

如果 A 机器上的编译程序能产生 B 机器语言的目标代码，那么这种程序被称为交叉编译程序。如图 1-25 所示为一个能在 A 机器上直接运行并产生 B 机器语言的 Pascal 的编译程序。如果 A 机器上已有 Pascal 语言的运行环境，如图 1-15 所示，则可在 A 机器上用 Pascal 编写一个编译程序，它的源语言是 Pascal，目标语言是 B 机器语言，如图 1-26 所示。将如图 1-26 所示的编译程序输入如图 1-15 所示的编译程序，即可得到如图 1-25 所示的编译程序，其处理过程如图 1-27 所示。

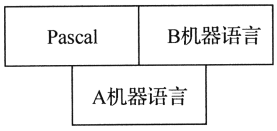


图 1-25　T 型图示例 6

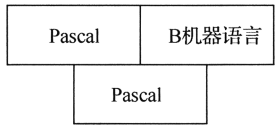


图 1-26　T 型图示例 7

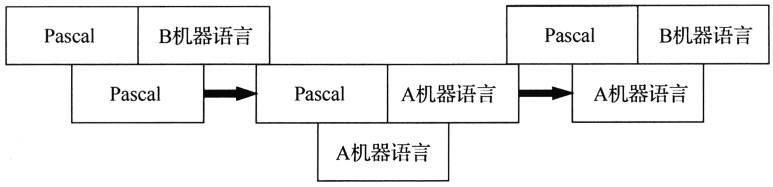


图 1-27　利用 Pascal 开发 Pascal 的编译程序的交叉编译过程

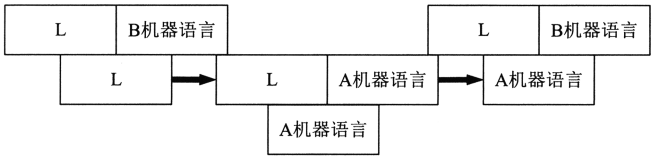
(5)移植技术

移植技术是编译程序开发中一项十分重要的技术，具体功能就是把一台计算机上的软件移植到另一台计算机上去。移植方法有多种，下面简单介绍两种典型的方法。

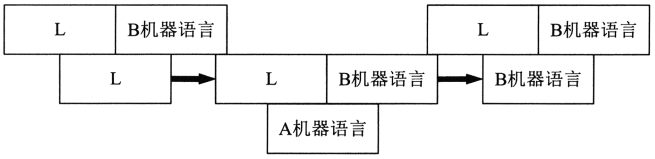
①综合几种型号的计算机硬件特性，抽象出一种通用的汇编语言。每种型号的计算机上都配有一个简单的汇编程序，用来把通用的汇编语言书写的程序翻译成机器语言程序。采用这种方法抽象出一种通用的汇编语言较为困难，因为这种通用的汇编语言既要便于书写编译程序，又要能够在各种不同型号的计算机上高效运行。

②利用交叉编译技术将由自编译语言编写的编译程序从一台计算机上移植到另一台计算

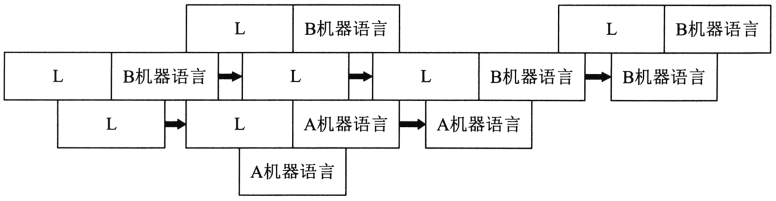
机上。假设在 A 机器上已有一个可运行的高级语言 L 的编译程序，只要我们编写一个用 L 书写的产生 B 机器语言的 L 的编译程序，按照如图 1-28 所示的方法处理，就能得到在 B 机器上能够运行的 L 的编译程序。



（a）L 的编译程序移植步骤 1



（b）L 的编译程序移植步骤 2



（c）L 的编译程序移植过程

图 1-28　L 的编译程序的移植方法

从上述讨论可以看出，利用自编译、自展、交叉编译和移植技术，使用系统程序设计语言书写编译程序，具有生产周期短、可靠性高、易修改、易扩充与维护、易移植等特点。

1.5 编译技术的主要应用

虽然并非人人都会从事构造或维护编译程序的工作，但是我们在大部分软件开发领域都会用到编译的原理和方法。下面介绍编译技术的主要应用。

(1) 文本编辑器

EditPlus、UltraEdit、Notepad++等文本编辑器可以引导用户编辑出符合语法规范的程序文档。通过对输入文档进行分析，文本编辑器能够自动高亮显示关键字、完成语法检查(如左右括号配对等)，以减少文档输入错误，提高开发效率。文本编辑器的开发利用了词法分析、语法分析、语法制导翻译等技术。

(2) 文本格式化工具

FineReader、txtFormat 等文本格式化工具可以将源程序重新排版，输出易读且清晰的程序结构，例如关键字以不同的颜色出现、注释用一种专门的字体、语句的嵌套层次结构采用缩进格式等。文本格式化工具的开发，利用了词法分析、语法分析、语法制导翻译等技术。

(3)文本检索工具

Word 等文本检索工具支持基于正规表达式的文本检索，其功能的实现利用了词法分析技术的串匹配技术。

(4)文本处理与加密工具

文本处理与加密工具首先把一个文本中满足特定特征的文本提取出来，然后对提取出来的文本进行加密处理。其本质是编写词法分析程序，对符合某种规则的单词进行提取，设计加密文法规则，并编写加密处理程序，对所提取的符号串进行加密处理。

(5)情感分析

情感分析(Sentiment Analysis)，又称为“意见抽取”(Opinion Extraction)，是对带有情感色彩的主观性文本进行分析、处理、归纳和推理的过程。情感分析是特殊的文本分类问题，既有一般文本分类的共性，也有其特殊性(如情感信息表达的隐蔽性、多义性和极性不明显等)，但其本质还是对自然语言进行词法分析、语法分析和语义分析。

(6)网页浏览器

网页浏览器本质上是 HTML 文档和 XML 文档的翻译器。上下文无关文法被用来描述 HTML 文档和 XML 文档的格式，网页浏览器获取了由 HTML 文档和 XML 文档提供的信息后，对其进行分析(使用词法分析、语法分析和语义分析等技术)，并根据分析的结果将网页内容以相应的形式显示出来。

思考练习

1. 画出编译程序的逻辑结构，简述各部分的主要功能。
2. 遍是什么？编译程序的分遍由哪些因素决定？
3. 系统程序设计语言是什么？
4. 编译程序生成技术有哪几种？分别叙述各种技术的基本思想。
5. 设 A 机器上有 C 语言的编译程序，现要用 C 语言编写 C++语言的编译程序，试用 T 型图进行描述。

第 2 章

词法分析

学习目标

深入理解词法分析在编译过程中的地位和作用，清晰阐述词法分析的基本概念和主要任务。

熟悉手动编写词法分析程序的基本流程和关键步骤，掌握词法分析中常见的字符处理和状态转换逻辑。

全面掌握自动生成词法分析程序的原理，包括有限自动机的基本概念、构造方法以及与词法分析的关联。

了解词法分析程序中词法规则的表示方法，如正则表达式的语法和语义，能够准确解读和编写简单的正则表达式。

知晓词法分析过程中错误处理的基本策略和方法，理解如何在词法分析阶段检测和报告常见的词法错误。

单词是语言中具有独立含义的最小单位。正如我们阅读一篇英文文献是从理解每个单词的意思开始的，编译程序也是首先在单词的基础上对源程序进行分析处理。词法分析是编译过程的第一阶段，其主要任务是识别单词。

通常将编译程序中完成词法分析任务的程序段称为词法分析程序、词法分析器或词法扫描器。本章主要讨论词法分析器设计和实现的有关问题，包括手工方式实现所涉及的正规文法与状态转换图问题，以及自动构造过程所涉及的正规表达式和有穷自动机问题。

2.1 词法分析概述

2.1.1 词法分析的任务

词法分析程序的主要功能是从左到右依次读入源程序的字符，根据所读入字符和该语言的词法规则识别出一个个单词。具体来说，词法分析程序的主要任务包括以下几个方面。

(1)消除无用字符。对源程序文本进行处理，消除源程序文本中的注释、空格、换行符及其他一切对语法分析和代码生成无用的信息。

(2)识别单词。扫描源程序的所有字符，按照语言的词法规则，识别出各类有独立意义的单词。

(3)对识别出来的单词进行内部编码。将长度不一、种类不同的单词用长度统一、格式规整、分类清晰的内部编码表示。

(4)建立各种表格(如名字特征表、常数表等)。

【例 2.1】 分析以下 C 语言程序。

```
void main()
{
    float a, b;
    a=3.0;
    b=5.4;
    a=a+b;
}
```

词法分析程序将剔除那些不影响程序语义的注释、无用的空格和回车等符号，识别出这些单词：void、main、(、)、{、float、a、,、b、;、a、=、3.0、;、b、=、5.4、;、a、=、a、+、b、;、}。

编译程序实现词法分析有两种方案。

(1)将词法分析放在单独的一遍扫描中完成。此时可将词法分析程序输出的单词流存放在一个中间文件上，将这个文件作为语法分析程序的输入，如图 2-1 所示。这种方案结构清晰，各遍功能单一，但由于要读写和保存中间文件，因此编译的效率较低。

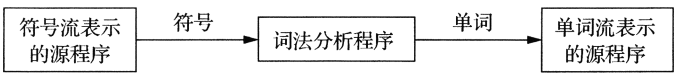


图 2-1 词法分析放在单独的一遍扫描中完成

(2)将词法分析和语法分析放在同一遍扫描中完成，如图 2-2 所示。通常将词法分析程序

设计成一个子程序，供语法分析程序随时调用。每调用一次，则从源程序字符串中读出一个具有独立意义的单词。这种结构避免了中间文件的读写，可以提高编译程序的工作效率。

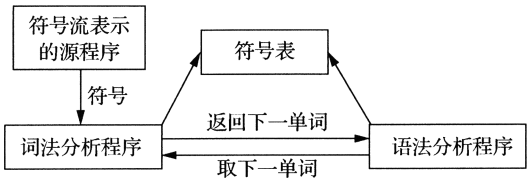


图 2-2　词法分析程序作为语法分析程序的子程序处理

词法分析程序可完成“字符流”到“单词流”的等价转换。字符流与被编译的语言密切相关(例如，C 语言的字符流由 ASCII 码组成)，“单词流”就是编译程序内部定义的数据结构。

2.1.2　单词的分类与表示

根据单词在程序设计语言中所起的作用不同，通常可以将单词分为如下几种类型。

(1)关键字(Key Word)，也称为“基本字”(Basic Word)，往往起到标识的作用。例如，Pascal 语言中标识一个程序开始的关键字是 program，标识一个复合语句开始的关键字是 begin；C 语言中标识 for 循环语句开始的关键字是 for，而 void 用于标识空值，等等。如果这些关键字在程序中只能作为某些规定的标志，而不可以有其他用途(例如，不可以做变量名，甚至不能做变量名的前缀等)，则将它们称为保留字(Reserved Word)。

(2)界限符(Separator)，也称为“分界符”，包括逗号、分号和各种括号等。

(3)运算符(Operator)，可分为算数运算符(+、-、*、/等)、逻辑运算符(not、and、or等)和关系运算符(=、<、>等)等。

(4)标识符(Identifier)，用来表示各种名字，如变量名、函数名和数组名等。

(5)常数(Constant)，分为整常数、实常数、符号串常量等，如 35、3.14 和"kid"等。

一程序设计语言的单词如何分类，本身没有特别的规定，纯属技术问题。不同的语言可以有不同的分类方式，主要以处理方便为准。

以上五种类型中的前三种(关键字、界限符和运算符)都是设计该语言之初就预先定义的，所以它们的数量是固定的，标识符和常数则由程序设计人员根据具体的需要自行定义，其数量虽然不是无穷的(基于形式语言的理论，很多程序设计语言支持产生无穷多个标识符和常数，但在实际中受机器字长的限制)，但也是巨大的。

为了便于处理已经识别的单词，编译程序通常会按照一定的方式对单词进行分类和编码。如何对单词进行分类和编码，本身也没有特别的规定，但为了编译程序处理方便，通常将单词内部编码分成两部分：类别编码和单词自身编码。这两部分被表示成二元组的形式：(类别编码，单词自身编码)。单词的类别编码通常用整数表示，可以采用以下两种编码原则。

(1)一类一种：根据单词的几大种类，为每一种类分配一个类别编码。例如，例 2.1 中，可以将关键字编码为 1，运算符编码为 2，界限符编码为 3，标识符编码为 4，常量编码为 5。

(2)一字一种：对于在设计语言之初就能确定下来具有特定含义的每个关键字、运算符和界限符，可分别指定相应的编码。例如，将 Pascal 语言中的“and”编码为 1，“for”编码为

12，“(”编码为 50。

单词自身编码也被称为单词的属性值，用来刻画单词符号的特性或特征，将同一类别的单词区别开来。关键字、运算符、界限符等专用符号在语言设计之初就确定不会发生变化，因此其属性值可以采用固定的编码；而对于标识符、常量等变化的单词，可以采用其在相应的名字特征表、常量表中的相对地址码作为其单词的属性值。

按照一类一种的编码原则，例 2.1 中的单词经过内部编码得到二元组序列(关键字及运算符、界限符用自身值代替其编码)，见表 2-1 所列。

表 2-1　例 2.1 中单词的编码结果

单词	类别编码	单词自身编码
void	1	void
main	1	main
(	3	(
)	3	)
{	3	{
float	1	float
a	4	名字特征表中的相对地址
,	3	,
b	4	名字特征表中的相对地址
;	3	;
=	2	=
3.0	5	常量表中的相对地址
5.4	5	常量表中的相对地址
+	2	+
}	3	}

2.2 手动编写词法分析程序

2.2.1 单词的描述——正规文法与状态转换图

许多程序设计语言的单词可以用乔姆斯基 3 型文法(正规文法)来描述。由形式语言和自动机理论可知，正规文法所描述的语言可以用有穷自动机来识别。为了简化问题，本节首先介绍有穷自动机的非形式化表示——状态转换图，下一节将引入有穷自动机的形式化描述，

并讨论如何利用状态转换图识别和分析单词，构造词法分析程序。

1. 状态转换图

状态转换图是设计词法分析程序的一种有效工具。状态转换图实际上是一个有限方向图：图中结点代表状态，用圆圈表示；状态之间由有向边连接，有向边上标记某个符号，其含义是某一状态下，如果当前的输入符号是有向边上标记的符号，则转换到另一状态或留在原状态。如图 2-3 所示，在 0 状态下，如果词法分析程序读入符号 a，则跳转到状态 1；如果读入的是符号 b，则跳转到状态 2。在状态 1 下，如果词法分析程序读入符号 d，则跳转到状态 3。在状态 2 下，如果词法分析程序读入符号 c，则跳转到状态 1。

一张状态转换图只能包含有限个状态，且其中至少有一个初始状态(简称“初态”)、一个终止状态(简称终态，用双圆圈表示)。在图 2-3 中，状态 0 为初始状态，状态 3 为终止状态。

一个状态转换图可以用于识别(接受)某些字符串。例如，识别 C 语言标识符的状态转换图如图 2-4 所示(这里暂时不考虑标识符的长度限制)。其中，S 为初态，Z 为终态。这个状态转换图识别(接受)标识符的过程从初态 S 开始，若编译器扫描到了一个字母或下划线，则读入该字母或下划线，并转入状态 1；在状态 1 下，若编译器又扫描到了一个字母或下划线或数字，则仍然读入它，并再次进入状态 1，重复这个过程，直到在状态 1 下发现编译器扫描到的符号不再是字母或下划线或数字时，进入状态 Z。状态 Z 是终态，它意味着到此已识别出一个 C 语言的标识符，识别过程宣告终止。终态 Z 的右上角有一个星号，这表示读进了一个不属于标识符的符号(如界限符、空格等)，应把它退还给输入串，用于识别下一个单词。

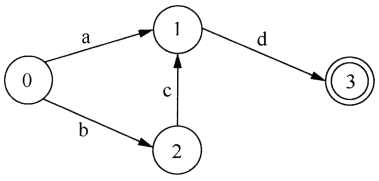


图 2-3　一个状态转换图的例子

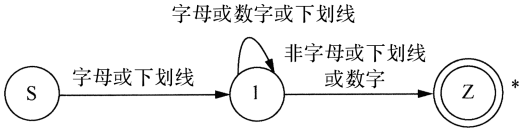


图 2-4　用以识别 C 语言标识符的状态转换图

词法规则往往可以采用正规文法来构造，而状态转换图恰恰又可用于识别单词，因此它们之间实际存在“等价”关系。下面阐释如何将正规文法转换为状态转换图。

2. 由左线性文法构造状态转换图

设文法 $G=(V_N, V, -T, P, Z)$是一个左线性文法，并假设$|V_N|=n$，则构造出的状态转换图共有 n+1 个状态，其对应的状态转换图构造步骤如下(其中，$U, B \in V_N$，$a, c \in V_T$)。

(1)将每个非终结符号设置成一个对应的状态，文法的开始符号 Z 所对应的状态为终止状态。

(2)在图中增加一个结点 S 作为初始状态，S 并非文法中的符号。

(3)对于 G 中形如 U→a 的规则，从初始状态 S 向状态 U 引一条箭弧，并标记为 a。

(4)对于 G 中形如 U→Bc 的规则，从状态 B 向状态 U 引一条箭弧，并标记为 c。

【例 2.2】 设有左线性文法 $G=(V_N, V_T, P, Z)$，$V_N=\{Z, A, B\}$，$V_T=\{0, 1\}$，其中 P：

$$Z\rightarrow A0 \mid B1 \qquad A\rightarrow Z1 \mid 1 \qquad B\rightarrow Z0 \mid 0$$

根据上述规则，该文法对应的状态转换图如图 2-5 所示。

我们利用图 2-5 很容易识别出字符串 x 是否为该文法的句子(单词)。从初始状态 S 出发，按与 x 余留部分中最左字符相匹配的原则，游历状态转换图，直到 x 读入最后一个符号为止。如果这时恰好到达状态 Z(即文法的开始符号)，则 x 是该文法所产生的句子(单词)之一，否则不是。

如图 2-6 所示为识别字符串 101001 的过程。不难看出，这种识别过程采用了自底向上的归约分析方法。编译器每一步所读入的符号与读入该符号之前的状态编号所组成的串，恰好是当前句型的句柄，该句柄所归约的结果便是下一个状态(此归约对应了文法中的某一条产生式)。该归约过程对应的语法树如图 2-7 所示(语法树中标注的序号①到⑥分别对应了图 2-6 中的每个处理步骤)。

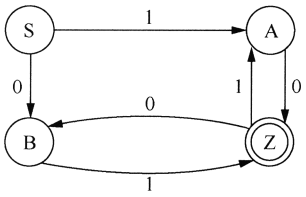


图 2-5　例 2.2 左线性文法对应的状态转换图

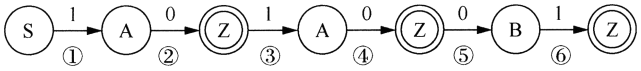


图 2-6　利用图 2-5 识别字符串 101001 的过程

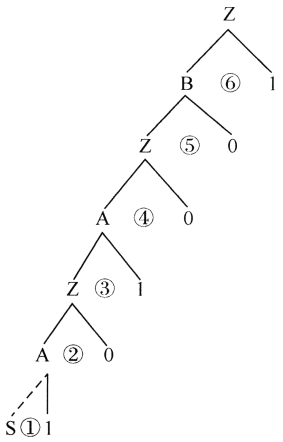


图 2-7　图 2-6 的识别过程所对应的语法树

实际上，如图 2-5 所示的状态转换图所能接受的语言是$\{01, 10\}^+$，即任意个数的“01”或“10”以任意次序组成的任意长度的非空符号串。

3. 由右线性文法构造状态转换图

设文法 $G=(V_N, V_T, P, S)$是一个右线性文法，并假设 $|V_N|=n$，则构造出的状态转换图共有 n+1 个状态，其对应的状态转换图构造步骤如下(其中，$U, B \in V_N$，$a, c \in V_T$)。

(1)将每个非终结符号设置成一个对应的状态，文法的开始符号 S 所对应的状态为初始状态。

(2)在图中增加一个结点 Z 作为终止状态，Z 并非文法中的符号。

(3)对于 G 中形如 U→a 的规则，从状态 U 向终止状态 Z 引一条箭弧，并标记为 a。

(4)对于 G 中形如 U→cB 的规则，从状态 U 向状态 B 引一条箭弧，并标记为 c。

【例 2.3】 设有右线性文法 $G[S]=(V_N, V_T, P, S)$，$V_N=\{S, A, B, C\}$，$V_T=\{0, 1\}$，其中 P：

$$S\rightarrow 1A \mid 0B \qquad A\rightarrow 0C \mid 0 \qquad B\rightarrow 1C \mid 1 \qquad C\rightarrow 0B \mid 1A$$

根据上述规则，该文法对应的状态转换图如图 2-8 所示。

如图 2-9 所示为利用图 2-8 识别字符串 101001 的过程。不难看出，这种识别过程采用了自顶向下的推导分析方法，其对应的是自顶向下构建语法树的过程，构造的语法树如图 2-10 所示。

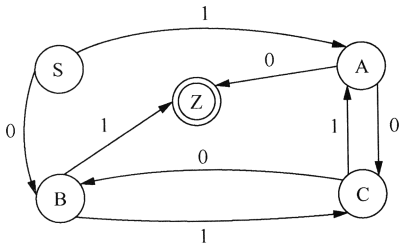


图 2-8　例 2.3 右线性文法对应的状态转换图

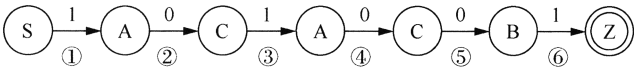


图 2-9　利用图 2-8 识别字符串 101001 的过程

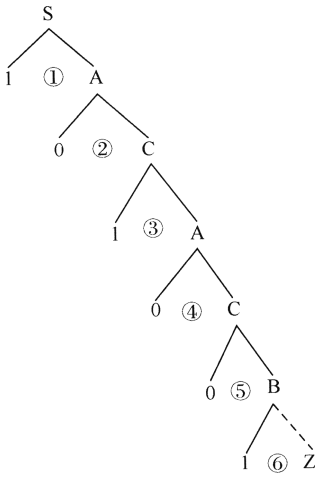


图 2-10　图 2-9 的识别过程所对应的语法树

不难看出，图 2-8 所示的状态转换图所能接受的语言也是{01，10}$^+$。也就是说，尽管例 2.2 和例 2.3 所示的文法不同，其对应的状态转换图也不同，但它们所接受的语言是完全相同的。我们可以将这种情况称为“左、右线性文法等价”“不同的状态转换图等价”“正规文法和状态转换图等价”。它们之间的等价转换方法的内涵和外延将在后续章节中详细讨论。

2.2.2 C--语言词法分析程序的设计与实现

如前所述，大多数程序设计语言的单词都可以用正规文法来生成，并可由此构造出相应的状态转换图。与此同时，利用状态转换图可以方便地判断某个字符串是不是一个合法的单词。由此可见，由状态转换图不难构造出相应程序设计语言的词法分析程序。下面我们通过一个简单例子来说明如何设计和实现一个 C++语言子集(即 C--语言)的词法分析程序。

表 2-2 给出了 C--语言的单词及编码。

表 2-2 C--语言的单词及编码

基本符号	类型	类型说明	编码(助记符)
void	1	关键字	$ void
main	1	关键字	$ main
cin	1	关键字	$ cin
cout	1	关键字	$ cout
{	2	界限符	$ LEFT
}	2	界限符	$ RIGHT
;	2	界限符	$ COLON
+	3	运算符	$ PLUS
*	3	运算符	$ MULT
标识符	4	标识符	&ID
整常数	5	常量	&INT

根据文法：

<程序> :: =void main( )<语句块>

<语句块> :: ={<语句串>}

<语句串> :: =<语句串><语句> | ε

<语句> :: =<赋值语句> | <输入语句> | <输出语句>

<赋值语句> :: =<标识符>=E;

<标识符> :: =字母 | _ | <标识符>(<字母> | _ | <数字>)

<整数> :: =<整数串><数字> | <数字>

<整数串> :: =<整数串><数字> | <非 0 数字>

<非 0 数字> :: =1 | 2 | 3 | … | 9

<数字> :: =0 | <非 0 数字>

<字母>::=A | B | C | … | Z | a | b | c | … | z

E::=T | E+T

T::=F | T*F

F::=(E) | <整数> | <标识符>

<输入语句>::=cin>><标识符>;

<输出语句>::=cout<<<计算结果>;

<界限符>::=; | { | }

<运算符>::= * | +

可以画出该文法所对应的部分状态转换图，如图 2-11 所示。

有了状态转换图，就可以写出相应的词法分析程序。下面考虑把词法分析程序作为语法分析程序的一个子程序来构造，每当语法分析程序需要一个单词时就调用这个子程序。在此不对保留字专设状态转换图，只在识别出标识符后去查保留字表，以确定是保留字还是普通标识符。

下面我们根据图 2-11 来构造由表 2-2 给出的 C--语言的单词的词法分析程序。首先我们引入一些变量和过程。

(1)字符变量 char：存放新读入的源程序字符。

(2)字符数组 token：存放构成单词的字符串。

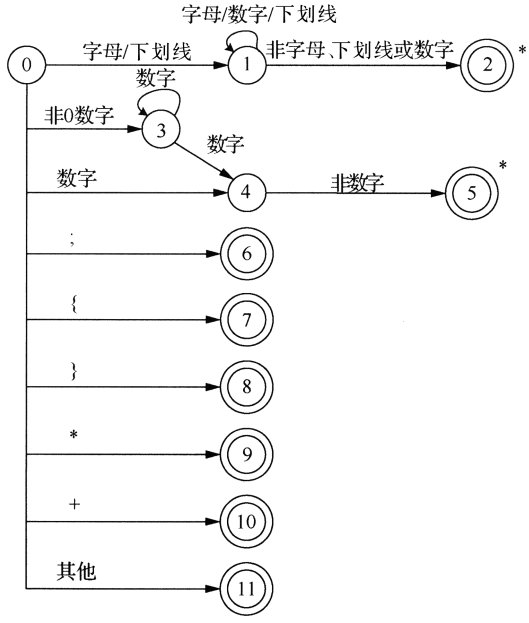


图 2-11　C--语言的单词文法所对应的部分状态转换图

(3)过程 getchrdr：读入下一个源程序字符至 char，并把指向字符的指针后移一个位置。

(4)过程 getnbc：检查 char 中是否为空白字符，若是，则反复调用 getchar 直至 char 中读入的是一个非空白字符。

(5)过程 concat：将 char 连接到 token 后面。例如，假定 token 原来的值为'STUDENT'，而 char 中存放着's'，则调用过程 concat 之后，token 的新值为'STUDENTS'。

(6)布尔函数 letter()：若 char 中字符为字母则返回 true，否则返回 false。

(7)布尔函数 digit()：若 char 中字符为数字则返回 true，否则返回 false。

(8)函数 reserve( ): 由 token 查保留字表，若 token 中的字符串为保留字则返回其类别编码，否则返回值为 0。

(9)过程 retract: 将指向字符的指针前移一个位置，char 置为空白字符。

现在，我们可以写出前面所述的 C--语言的词法分析程序:

```
Start: token: = ' ';
       getchar; gembc;
CASE char   OF
' A' ... ' Z', ' a' ... ' z', ' _ ' : BEGIN
         WHILE leRer OR digit OR " _ " DO
           BEGIN
             concat;
             getchar;
           END;
             retract;
             c = reserve;
           IF c = 0      THEN reaml( $ ID, token);
           ELSE return(c, —);
       END:
' 0' : IF! digit THEN return( $ INT, token)
   ELSE GOTO ERROR
' 1' ... ' 9' : BEGIN
         WHILE digit DO
           BEGIN
             concat;
             getchar;
           END;
             retract;
             return( $ INT, DTB);
         END;
' ; ' :   return( $ SEMI, —);
' {:   return( $ LEFT, —);
' } ' : return( $ RIGHT, —);
' + ' :   return( $ PLUS, —);
' * ' :   return( $ MULT, —);
END OF CASE;
ERROR;
GOTO start;
```

2.3 自动生成词法分析程序的原理

词法分析程序既可以通过手动编写的方式构造，也可以通过自动生成的方式构造。上一节介绍了如何利用正规文法描述单词，并借助状态转换图手动编写词法分析程序。本节将介绍词法分析程序自动生成的原理：如何使用正规表达式描述单词；如何自动生成词法分析程序。由于我们的目的是介绍词法分析程序的自动生成，因此对所述的某些结果不予证明，对证明有兴趣的读者可以参阅形式语言和自动机理论方面的书籍。

2.3.1 单词的描述——正规表达式

正规表达式是一种通过符号组成的式子来表达语言(句子集合)的方式。它简单、直观，与集合的表现形式更为相近，因此应用起来也更为方便。

每一类程序设计语言都有它自己的字符集Σ，语言中的每一个单词既可以是Σ上的单个有意义的字符(如运算符、分隔符等)，也可以是Σ上的字符按一定方式组成的有意义的字符串(如常数、保留字、标识符及关系运算符等)。如果我们把每类单词均视为一种“语言”，那么每一类单词都可用一个正规表达式来描述。正规表达式表示的“语言”叫作正规集。

在给出正规表达式的正式定义之前，我们先通过一个例子看看正规表达式大致是什么样子的，它又是如何表达某些集合的。

【例 2.4】 在字母表Σ={a，b}中，$a^*b|b^*a$就是一个正规表达式，它代表了出现若干个a后以一个b结尾，或者出现若干个b后以a结尾的一切符号串的集合。用传统的集合形式来表示就是$\{a\}^*\{b\}\cup\{b\}^*\{a\}$。我们从中可以发现正规表达式表示法和集合表示法的相同之处和不同之处。

下面给出正规表达式和正规集的形式化定义。

定义 2.1(正规表达式和正规集) ε和∅是Σ上的正规表达式，它们所表示的正规集分别为{ε}和∅。对于每一个$a\in\Sigma$，a是Σ上一个正规表达式，定义它所表示的正规集为{a}。如果e_1和e_2是Σ上的正规表达式，定义它们所表示的正规集分别为$L(e_1)$和$L(e_22)$，则：

(1)$e_1|e_2$是正规表达式，其相应正规集为$L(e_1|e_2)=L(e_1)\varnothing L(e_2)$；

(2)$e_1\cdot e_2$是正规表达式，其相应正规集为$L(e_1\cdot e_2)=L(e_1)L(e_2)$；

(3)$(e_1)^*$是正规表达式，其相应正规集为$L((e_1)^*)=(L(e_1))^*$。

有限次使用上述步骤定义的表达式才是Σ上的正规表达式。仅由这些正规表达式所表示的符号串集合才是Σ上的正规集。

正规表达式使用三个基本运算符：“·”为连接，一般可省略不写；“|”为选择；“*”为闭包。前两者为双目运算符，而闭包是单目运算符，它表示任意有限次的自重复连接。上述

运算符的优先级：“*”最优先，“·”次之，最后是“|”。在定义 2.1 中，圆括号并非正规表达式的运算符，而仅用于指示正规表达式中的子表达式。我们也可以在一个正规表达式中加一些括号来改变运算顺序。

【例 2.5】 给出字母表∑={a，b}，下列各式均是∑中的正规表达式：

a^*　　ba^*　　$a|ba^*$　　$aa|bb|ab|ba$

$a(a|b)^*$　　$(a|b)^*(aa|bb)(a|b)^*$　　$(a|b)(a|b)(a|b)(a|b)^*$

求出它们相应的正规集。

$L(a^*)=(L(a))^*=\{a\}^*=\{a, a, aa, aaa, \ldots\}$

$L(ba^*)=L(b)L(a^*)=\{b, ba, baa, \ldots\}$

$L(a|ba^*)=L(a)\cup L(ha^*)=\{a, b, ba, baa, \ldots\}$

$L(aa|bb|ab|ba)=L(aa)\cup(bb)\cup L(ab)\cup L(ha)=\{aa, bb, ab, ba\}$

$L(a(a|b)^*)=L(a)(L(a)\cup L(b))^*=\{a\}\{a, b\}^*$

$L((a|b)^*(aa|bb)(a|b)^*)=\{a, b\}^*\{aa, bb\}\{a, b\}^*$

$L((a|b)(a|b)(a|b)(a|b)^*)=L(a|b)L(a|b)L(a|b)L((a|b)^*)=\{a, b\}\{a, b\}\{a, b\}\{a, b\}^*$

【例 2.6】 用正规表达式描述“标识符”这种类型的单词。

(字母|_)|(字母|_|数字)*

【例 2.7】 ∑={a，b)　$e_1=b(ab)^*$　$e_2=(ba)^*b$，求出 e_1 和 e_2 的正规集。

$L(e_1)=L(b(ab)^*)=L(b)(L(ab))^*=\{b\}\{ab\}^*=\{b, bab, babab, \ldots\}$

$L(e2)=L((ba)^*b)=(L(ba))^*L(b)=\{ba\}^*\{b\}=\{b, bab, babab, \ldots\}$

从例 2.7 可以看出，e_1 和 e_2 的表达式虽然不相同，但它们的正规集都以 b 开头，其后跟零个或任意多个 ab，即 $L=\{b(ab)_n|n\geq 0\}$。通过例 2.7 可以论证正规表达式等价的概念。

定义 2.2(正规表达式等价)　令∑为有穷字母表，如果∑上的正规表达式 e_1 和 e_2 所表示的正规集相同，则认为两者等价，记为 $e_1=e_2$。

从正规表达式等价的定义，我们可以得到以下关系：

(1) $e_1|e_2=e_2|e_1$

(2) $e_1(e_2|e_3)=(e_1|e_2)|e_3$

(3) $e_1(e_2e_3)=(e_1e_2)e_3$

(4) $e_1(e_2|e_3)=e_1e_2|e_1e_3$

(5) $\varepsilon e_1=e_1\varepsilon=e_1$

(6) $\varnothing e_1=e_1\varnothing=\varnothing$

(7) $(e_1^*)^*=e_1^*$

(8) $(\varepsilon|e_1)^*=e_1^*$

根据定义 2.2 比较容易证明上述性质。利用这些性质可以化简正规表达式，证明正规表达式的等价关系。

2.3.2　单词的识别——有穷自动机

如前所述，使用正规文法和正规表达式可以定义语言的词法结构。在手动编写方式中，

我们将正规文法转换成状态转换图，根据状态转换图可以较为方便地编写出词法分析程序。虽然对于人类而言，状态转换图直观清楚地描述了如何分析一个符号串是不是文法所定义的句子(如果某符号串能够被状态转换图所接受，则该符号串是文法所定义的单词)，但对于计算机而言，状态转换图的描述方式是不容易理解的，也不适合用于自动生成词法分析程序。因此本节介绍状态转换图的形式化描述工具——有穷自动机，其从识别语言的角度出发，利用某种模型来判断一个符号串是否是给定语言的句子。

有穷自动机(Finite Automata，FA)也被称为有穷状态自动机或有穷状态系统，它是一种数学模型，这种模型对应的系统具有有穷数目的内部状态，系统的状态概括了对过去输入的处理情况。系统根据当前所处的状态和面临的输入就可以决定后续行为。每当系统处理完当前的输入，系统的内部状态也会发生改变。电梯控制装置就是有穷自动机的一个典型例子：用户的服务要求(即要到达的楼层)是该装置的输入信息，而电梯所处的层数及运动方向是该装置的状态。电梯控制装置并不需要记忆先前全部的服务要求，只需要记忆电梯当前所处的层数和运动方向以及还没有满足的服务要求即可。

在计算机科学中，可以找到很多利用有穷自动机原理设计的系统的应用实例，如开关电路、文本编辑程序和词法分析程序。计算机本身也可以看作一个有穷自动机，尽管其状态数目可能很大。甚至人脑也可以看作有穷自动机：根据脑科学的研究，人脑的神经元的数目可能高达 2^{35} 个。正是基于对神经元细胞模型的研究，1943 年，沃伦·麦卡洛克和沃尔特·皮茨首先提出了有穷自动机模型，如图 2-12 所示。

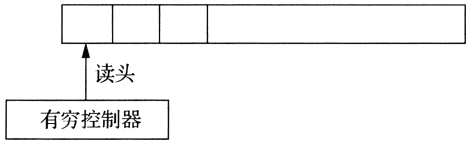


图 2-12　有穷自动机模型

模型由一条有穷长度的输入带、一个读头和一个有穷控制器组成。在这个模型中，单个的输入信息被抽象表示成一个输入符号，输入带存放所有的输入符号，每个输入符号占一个单元(方格)，输入带的长度和输入串的长度相同。在有穷控制器控制下，读头从左到右逐个扫描并读入每个符号，且根据有穷控制器的当前状态和当前输入符号转入下一状态。有穷控制器的状态数目是有限的，读头具有只读功能，不能修改输入带上的符号，不能后退，只能向前读取。和各种具体的机器一样，有穷自动机具有初始状态和终止状态。在初始状态下，读头指向输入带的最左单元，准备读入第一个符号。终止状态可以有多个，在该状态下可以接收输入串。如果在读头读入输入带上最后一个符号时，有穷自动机恰好进入了某个终止状态，则表示该输入串可以被有穷自动机识别，否则不能被识别。

可以看出，前面所述的状态转换图就是有穷自动机模型直观的图形化表示，为了实现可计算，需要给出它的形式化定义。有穷自动机分为确定的有穷自动机(Deterministic Finite Automata，DFA)和非确定的有穷自动机(Non-Deterministic Finite Automata，NFA)，下面分别给出它们的形式化定义。

1. 确定的有穷自动机

定义 2.3(确定的有穷自动机) 一个确定的有穷自动机(DFA)M 是一个五元组，即 M=(K，V_T，M，S，Z)，其中元素如下。

K 是状态有穷的非空集合，K 中每一个元素是一个状态。

V_T 是一个有穷输入字母表，V_T 中的每一个元素称为输入字符。

M 是 K×V_T 到 K 的单值映射(或函数)，即 M(q，a)=p，q，P∈K，a∈V_T。它表示当前状态为 q，输入字符为 a 时，将转到下一状态 p，p 是 q 的一个后继状态。由于映射是单值，所以称确定的有穷自动机。

S 为初始状态，是唯一初态，S∈K。

Z 是终止状态集合，Z 是 K 的子集。

【例 2.8】 设(DFA)M=({0，1，2，3)，{a，b)，M，0，{3})，其中

K={0，1，2，3}

V_T={a，b}

M：(状态转换函数)

M(0，a)=1　　M(0，b)=2

M(1，a)=3　　M(1，b)=2

M(2，a)=1　　M(2，b)=3

M(3，a)=3　　M(3，b)=3

S=0

Z={3}

利用映射 M 很容易识别定义在输入字母表 V_T 上的字符串是否被 DFA 所接受(即最后到达终止状态)。例如，对于输入字符串 abb，因为从初始状态 0 出发，有

M(0，a)=1　　M(1，b)=2　　M(2，b)=3

当输入完最后一个字符 b 时，到达了终止状态 3，所以字符串 abb 能被此 DFA 所接受(识别)。

一个 DFA 可唯一表示一张确定的状态转换图。假定一个(DFA)M 有 m 个状态和 n 个输入字符，则它的状态转换图含有 m 个状态，每个结点最多有 n 条箭弧和别的状态相连接，每条箭弧用 V_T 中的一个输入字符标记，整个图含有唯一的初态和若干个终态。

为此，我们可以画出例 2.8 的状态转换图，如图 2-13 所示。

对于 V_T 中任意字符串 x，若存在一条从初始状态到终止状态的路径，且这条路径上所有的字符连接成的字符串等于 x，则称字符串 x 可为(DFA)M 所接受(识别)。例如，对于字符串 abb，能找到一条从初始状态到终止状态的路径，如图 2-14 所示。

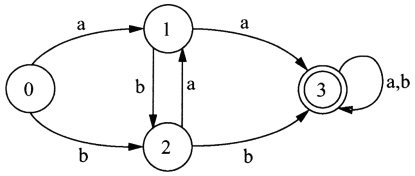


图 2-13　例 2.8 的状态转换图