普通高等教育计算机专业“新形态·立体化”规划教材

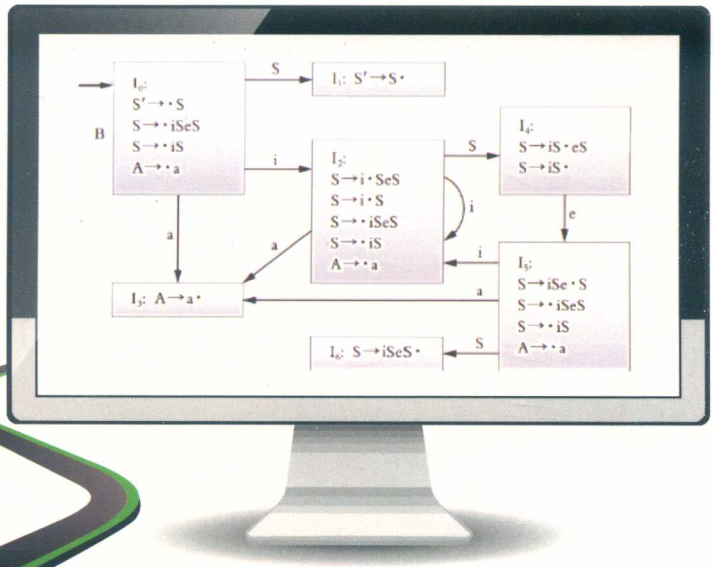
“十二五”普通高等教育本科国家级规划教材

编译原理

（第3版）

主　编　李劲华　陈　宇　丁洁玉

主　审　何炎祥



北京邮电大学出版社
www.buptpress.com

普通高等教育计算机专业"新形态"立体化"规划教材

"十二五"普通高等教育本科国家级规划教材

编译原理

（第3版）

主　编　李劲华　陈　宇　丁洁玉

主　审　何炎祥

广益教育"九斗"
APP 操作说明

北京邮电大学出版社
·北京·

内 容 简 介

本书介绍计算机编译系统及其构造的基本原理和技术，主要内容包括词法分析、语法分析、语法制导的语义分析及中间代码生成、目标代码生成、代码优化以及符号表和运行时环境。概述编译构造中的基础理论——如形式语言、有限自动机和属性文法。从编译构造的技术角度，描述编译程序的各类算法，以及编译程序的自动构造工具——词法分析生成器 Lex 和语法分析生成器 YACC。

本书对编译构造的基础知识阐述清晰，保持编译构造的逻辑性和系统性，便于阅读，可作为普通高等院校计算机类及相关专业的本科教材，也可供教师、研究生及有关专业人员学习和参考。

图书在版编目（CIP）数据

编译原理 / 李劲华，陈宇，丁洁玉主编 . —3 版 . —北京：北京邮电大学出版社，2019.6

ISBN 978-7-5635-5674-8

Ⅰ . ①编… Ⅱ . ①李… ②陈… ③丁… Ⅲ . ①编译程序—程序设计 Ⅳ . ① TP314

中国版本图书馆 CIP 数据核字（2019）第 002486 号

书　　名　编译原理（第 3 版）
主　　编　李劲华　陈　宇　丁洁玉
责任编辑　向　蕾
出版发行　北京邮电大学出版社
社　　址　北京市海淀区西土城路 10 号（100876）
电话传真　010-82333010　62282185（发行部）　010-82333009　62283578（传真）
网　　址　www.buptpress3.com
电子信箱　ctrd@buptpress.com
经　　销　各地新华书店
印　　刷　北京时捷印刷有限公司
开　　本　787 mm × 1 092 mm　1/16
印　　张　20.5
字　　数　510 千字
版　　次　2019 年 6 月第 3 版　2019 年 6 月第 1 次印刷

ISBN 978-7-5635-5674-8　　定价：39.00 元

preface 前 言

在计算机向微型化、巨型化、多样化发展的今天，各种类型的程序设计语言相互融合，新的程序设计语言不断涌现，但编译程序在计算机的基础地位没有改变，编译程序依旧不可或缺。编译构造一直属于ACM/IEEE（计算机协会/电气电子工程师学会）计算课程（ACM/IEEE Computing Curriculum）的核心知识域，编译构造的原理和技术是研究计算机、理解和设计程序语言的关键。纵览计算机历史会发现很多被誉为程序设计大师的人，例如，创建了Pascal语言及其系列的图灵奖获得者Niklaus Wirth（尼古拉斯·沃斯）教授，第一个写出微型机上运行Basic语言的Bill Gates（比尔·盖茨），成为Sun公司副总裁的Java缔造者James Gosling（詹姆斯·戈斯林）博士，C++之父Bjarne Stroustrup（本贾尼·斯特劳斯特卢普）教授，世界上最优秀的程序员、Delphi架构师和C#缔造者Anders Hejlsberg（安德斯·海尔斯伯格）等，无一不是编译领域的泰斗。编译领域堪称计算机程序设计英才的沃土。

编译构造的研究不仅推动了计算机的研究和发展，还在软件建模语言、硬件描述语言、领域特定语言等的翻译、软件开发以及软件安全、软件工程和软件逆向/再向工程中应用广泛。例如，正则表达式处理文本和符号串便捷高效，已经成为现代编程语言（Java、C#、Python、R、Scala等）的标准库成员；编译器的组成——扫描器、分析器，甚至整个编译器也都出现在语言的系统库中，允许应用程序员使用。

我国的IT（互联网技术）已经取得了巨大进步，然而研究编译的基础理论、构造实用的编译系统，对我们仍然是一个巨大的挑战。本书介绍计算机程序语言编译构造的基本原理、设计方法和关键技术，可以作为普通高等学校计算机科学及相关专业的教材和参考书。

同其他教材相比，本书有以下几个特点。

（1）根据编译程序的构成和编译过程的顺序，逐步介绍编译的基本原理、设计方法和构造技术，把读者的思路和精力保持在编译程序的构造上，强调对编译程序的宏观理解和全局把握。对理论基础采用“及时、按需、足够”的原则，减少读者阅读和理解的难度。

（2）在内容上增加了现代流行程序设计语言（如C、Python、Java）的特性、处理和例子。加入了作者的研究，例如，在自底向上的LR分析中，采用了图的深度优先策略，以增量的方式构造出识别活前缀的有限自动机。

（3）详细解释并描述了编译的主要算法，以便读者阅读理解和上机编程实现。

（4）为便于理解教材中的理论知识，为本书及每章提供了微课导学的二维码。同时也以二维码的形式提供了课后拓展阅读材料，主要是Java和Python语言为主的现代编译构造技术，主要来源于书籍和官方网站。我们翻译、编写、编辑了这些材料，风格和组织与教材有很大差异，但希望给读者一些帮助。

本书按照编译的过程来组织，力图保持知识的逻辑性和连贯性。第 1 章概述了编译的基本概念、编译程序的组成、编译的过程以及编译系统。以后各章按照需要和逻辑关系展开编译构造的知识。

第 2 章首先介绍词法分析的设计和词法分析程序的手工构造，然后讲述有限状态自动机的理论以及它在词法扫描器自动生成的应用。

第 3 章介绍计算机编程语言的形式化方法，包括上下文无关文法的基本概念和文法的分类。为第 4 章和第 5 章的语法分析提供理论基础。

第 4 章介绍自顶向下的语法分析方法，包括表驱动的 LL(1) 分析和递归下降分析。

第 5 章讨论自底向上的算符优先分析方法、各种类型的 LR 分析方法及语法分析的自动生成。

第 6 章介绍编译程序符号表的组织与管理，以便于理解语义分析和代码生成。

第 7 章讨论编译构造所需要的运行时环境，包括运行时的内存组织、分配和管理。

第 8 章概述语义分析，重点描述属性文法以及语法制导的语义分析技术，为第 9 章提供基础。

第 9 章首先介绍中间代码的知识，重点讲述语法制导的中间代码翻译。

第 10 章介绍目标代码生成的原理和技术。

第 11 章集中介绍代码优化的基本技术，主要包括中间代码的局部优化和目标代码的优化方法。

本书每章都附有各种类型的练习题，便于读者理解基本概念和原理，掌握编译的基本算法和实现技术。

使用本书要求读者学习和掌握高级程序设计语言，如 C、Java 或 Python，最好具备离散数学、数据结构、计算机组成和汇编语言的基本知识。

编译知识理论抽象，算法丰富。建议使用本书时，配合上机实验，在加深对编译构造理解的同时，掌握编译构造的基本技术。编译中有很多算法，如 NFA 到 DFA 的转换、求首符集、计算 LR 项集等，都可以作为上机实践题。

本书结合作者李劲华多年的科研工作和教学实践编著而成，主要参考了文献 [1] 和 [2] 以及国内外许多学者的研究成果，此外，何炎祥教授担任本书的主审，对本书提供了许多宝贵的意见和建议；陈宇、丁洁玉参与了本书大纲的讨论，参与撰写了第 3 章和第 4 章的内容；王奕、王海文、许杰、姜卫、程妍、梁开健等对本书提出了许多宝贵的意见，在此一并表示衷心的感谢。

由于编者学术有限，书中难免存在疏误和不足，恳请广大读者予以批评、指正和商讨。

主编电子邮件地址：lijh@qdu.edu.cn

编　者

2019 年 1 月

课程导读

contents 目 录

第1章 概　论

本章导学

1.1 为什么学习编译

编译程序构造的原理和技术一直属于 ACM/IEEE 计算课程的核心知识域，是计算机科学必备的专业基础知识。另外，编译程序的构造是计算机科学中一个非常成功的分支，也是最早获得成功的分支之一，它所建立的理论、技术和方法值得我们深入研究和学习。

首先，编译构造正确地建立了研究的问题领域和研究方式：分析输入内容、构造一个语义表示并合成输出。对于不同的源语言，可以用一个单一的语义表示，产生出不同的目标语言，运行在不同的环境中。另外，编译构造可以划分成便于控制和管理不同的阶段，每个阶段的工作结果正好对应编译程序的子系统或模块。编译程序的这种分析—合成模式以及解释程序的解释模式已经成为软件开发领域最成功的设计模式和软件架构，在软件开发中获得了广泛的应用。

其次，针对编译程序构造的某些部分已经开发了标准的形式化技术，依据它们研制的编译程序生成工具，极大地减轻了编译程序的构造工作，使得编译程序成为计算机系统最可靠的基础软件之一。这些形式化技术包括有限自动机理论、上下文无关文法、正规表达式、属性文法、机器代码描述、数据流分析方程式等。编译程序自动生成技术具有的高效性、正确性、灵活性、易更改性和易扩展性等优点，也扩大到普通的软件开发领域，例如，数据库程序的自动产生器以及从图形化的 UML 自动生成高级语言的程序。

再次，编译程序包含许多普遍使用的数据结构和算法，例如散列法（哈希算法）、栈机制、堆机制、垃圾收集、集合算法、表驱动算法、图算法等。尽管其中的每一种都可以独立地学习，但是，在诸如编译程序这样一个有意义的环境中学习将更有教育意义。

然后，编译程序的许多构造技术已经得到了广泛的应用。许多应用程序非常接近于编译程序，已经采用了编译构造的部分技术，例如读入格式化的数据、文件转换问题等。如果数据有清晰的结构，就可以为它设计文法，使用分析程序生成器自动地产生一个分析程序。从编译程序构造技术受益的文件转换系统的范例包括：著名的 TeX/LaTeX 文本格式化程序（应用广泛的、共享的文字处理软件的文件格式），该程序把 TeX 文本转换成机器独立的 dvi 格式；PostScript（一种普遍应用的、与机器和软件无关的文件格式，主要用于文件的传输和打印）解释程序，它将 PostScript 文本转换为打印机的指令。这种技术还有利于迅速引进新

的文件格式，例如，能够快速地创建 HTML、XML 和 UML 等文件的读入和分析程序。

最后，学习编译原理和技术还有助于我们理解程序设计语言，编写优秀的软件。我们不妨浏览一下计算机的历史，就会发现很多被称为程序设计大师的人都是编译领域的泰斗：创建了 Pascal 语言及其系列的图灵奖获得者 Niklaus Wirth 教授，写出第一个微型机上运行的 Basic 语言的 Bill Gates，成为 Sun 公司副总裁的 Java 缔造者 James Gosling 博士，Delphi 的架构师与 C# 缔造者的 Anders Hejlsberg，C++ 之父 Bjarne Stroustrup 教授等等。

世界上已经出现了近千种计算机程序设计语言，其中主要使用的不过数十种，如 Java、C、C++、C#、SQL、JavaScript 等，而且一些语言已经有三四十年的历史。然而，计算机更加广泛的应用，对计算机程序设计语言提出了个性化的不同的需求，之前广泛使用的程序设计语言也还在不断扩展和完善。一些新型的程序设计语言也不断涌现，如 Python、R、Scala。随着软件成为驱动创新、数字经济的核心技术，计算机程序设计语言的创新和发展也不会停止。

1.2 什么叫编译程序

编译程序是计算机系统经典、核心的系统软件。自从 20 世纪 50 年代出现了以 Fortran 为代表的计算机高级编程语言以后，在抽象层次和使用方便等方面都超过机器代码和汇编程序的高级编程语言层出不穷，已经超过数千种。而且，随着计算机系统更加广泛的应用，研制和开发高级编程语言的工作远远没有停止的趋势。按照现在的计算机体系结构和组成原理以及软件开发的理论和实践，高级程序语言仍将是开发计算机应用系统的关键技术，与之不可分离的是高级程序语言的编译程序。

用高级编程语言，如 Fortran、Pascal、Ada、Smalltalk、C、C++、C# 和 Java，编写程序方便而且效率高，但是，计算机需要把用高级编程语言编写的程序翻译成机器语言的代码或汇编程序才能运行。**翻译程序**或**翻译器**是把一种语言（**源语言**）转换成等价的另外一种语言（**目标语言**）的程序。如果源语言是高级编程语言，目标语言是机器代码和汇编语言这样的低级语言，这类翻译程序就叫做**编译程序**或**解释程序**。如果目标语言是汇编语言，则还需要汇编成机器代码之后才可以运行。由于汇编语言和机器代码十分接近，像其他编译教材一样，本书通常不区分目标程序是机器代码还是汇编程序。

源程序的运行实质上包含两个过程：首先是把**源程序**用编译程序翻译成机器可以执行的**目标程序**或**目标代码**，然后才能接受输入数据运行，如图 1.1 显示的就是这种**编译执行方式**。

运行高级语言程序的另外一种方式是**解释执行**，它需要的翻译程序不是编译程序，而是**解释程序**。解释程序不产生源程序的目标代码，而是对源程序逐条语句的分析，根据每个语句的含义执行产生结果，如图 1.2 所示。Visual Basic.NET、R、SQL 和多数脚本语言是按照解释方式运行的。解释方式的主要优点是便于对源程序进行调试和修改，但是其加工过程降低了程序的运行效率。

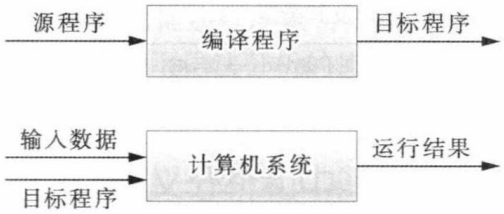


图 1.1　高级语言程序的编译执行方式

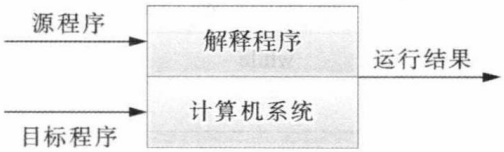


图 1.2　高级语言程序的解释执行方式

理论上，每一种语言都可以根据需要采用编译执行方式或解释执行方式，实践中每种语言都主要采用一种执行方式。根据执行方式，高级编程语言可以划分为三类：编译型、解释型、混合型。C、C++、Fortran 等属于典型的编译型语言，JavaScript、SQL、R 等属于典型的解释型语言，而 Java、C#、Python、Scala 等则属于混合型高级编程语言。例如，对于 Java 程序，首先要用一个编译器把 Java 源程序编译成 Java 类（一种中间语言），然后由解释器把它翻译成目标机器的代码再执行。

由于编译程序和解释程序的原理基本一样，本书主要讲解编译程序的基本原理和技术，它们也可以适用于解释程序的构造与实现。

混合式语言及 Java 的运行

1.3　编译过程概述

把计算机高级编程语言翻译成计算机可以执行的代码的工作包括一系列的活动和任务，是一个复杂的完整过程。编译过程可以和把英文翻译成中文的过程相比较。在翻译一篇英文的时候，我们首先要确定英语的单词、标点符号、分隔符等，把句子分解成单词以后去理解单词的基本含义；然后，分析英语句子的词组和语法结构，理解原句的含义，根据已知的中英文句型进行初步的翻译；接下来对译文进行修饰和润色，最后写出译文。

计算机程序的编译过程与此类似，一般划分为五个阶段：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。

1.3.1　词法分析

词法分析的任务是逐步地扫描和分解构成源程序的字符串，识别出一个一个的**单词符号**或**符号**。词法分析的工作主要包括识别出程序中的单词符号，在编译程序符号表中查找并登记单词符号及其信息，如单词符号的类型、内部表示、数值等。计算机高级语言的单词符号通常包括标识符、关键字或基本字、标点符号、常数、运算符、分隔符等类型。例如，对于 C 语言的 while 语句

```
while(i<100)sum+=i++;
```　　(1.1)

词法分析的结果识别出下列单词如表 1.1 所示。

这些单词是 C 语言的基本单词符号，是构成 C 语言程序的基本组成成分，是人们理解和编写 C 语言程序的基本要素。在词法分析的过程中通常都把分隔符类型中的空格符号删除掉。

表 1.1　语句 while(i<100) sum+=i++; 的单词及其类型

符　　号	类　　型
while	关键字
(	分隔符
i	标识符
<	运算符
100	整常数
)	分隔符
sum	标识符
+=	复合赋值符
i	标识符
++	运算符
;	分隔符

编译程序的词法分析也叫**词法扫描**或**线性扫描**。本书的第 3 章重点学习词法分析的基本原理、技术和工具，如有限状态机、正规表达式及其之间的等价转换以及词法分析生成器 Lex 等。

1.3.2　语法分析

语法分析的任务是在词法分析基础上，根据语言的语法规则把单词符号串分解成各类**语法单元**（**语法范畴**、**语法短语**），例如“表达式”、“子句”、“语句”、“块程序”、“函数”和“程序”等。语法分析是把线形序列的单词符号，根据语言的语法规则，按照层次分解，结果通常表示成**语法分析树**。

例如 C 语言的 while 语句格式为

```
while(表达式)循环体句;
```

其中的表达式包含赋值表达式、逗号表达式、复合赋值表达式等。上面的例子（1.1）经过语法分析之后可以表示成图 1.3 的分析树。

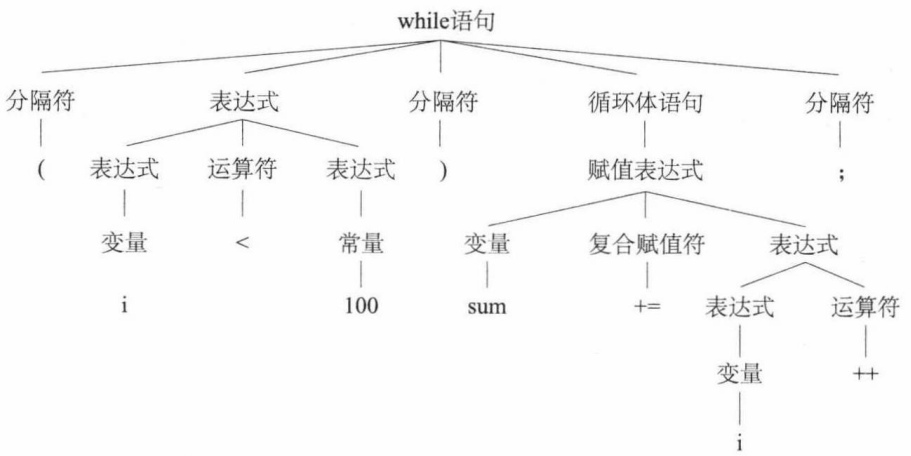


图 1.3　语句 while(i<100) sum+=i++; 的语法分析树

第3章讨论描述程序语言规则的形式语言基础，之后的第4章和第5章将分别详细讨论典型的自顶向下语法分析的递归下降分析LL(1)技术，自底向上的语法分析方法——算符优先分析和LR分析以及语法分析工具YACC。

1.3.3 语义分析和中间代码生成

语义分析的任务是检查程序语义的正确性，解释程序结构的含义。语义分析包括检查变量是否有定义，变量在使用前是否具有值，数值是否溢出等，其中的一个重要部分是进行类型的检查和转换。编译程序检查每个运算符的运算对象，看它们的类型是否合适，如果不合适，分析的结果应当是报告错误，同时给出类型转换的建议。例如，一个整数类型的变量和一个实数类型的变量相加，通常的编程语言要求编译发出警告信息，并且建议把整数类型转换成实数类型以后再进行运算，结果也是实数类型。

语义分析完成之后，编译程序通常就依据语言的语义规则，利用**语法制导技术**把源程序翻译成某种**中间代码**。所谓“中间代码”是一种定义明确、便于处理、独立于计算机硬件的记号系统，可以认为是一种抽象的程序。引入中间代码的目的主要是使编程语言及其程序不依赖于特殊的计算机类型，方便程序的移植和运行，同时可以利用词法分析、语法分析和语义分析对程序进行各种处理，应用在各类集成化软件开发环境中。

对中间代码的基本要求是既要便于把源程序翻译成中间代码，又要易于把它翻译成各种计算机的指令或汇编语言。中间语言有多种形式，其中一类是**三地址代码**，很像机器的汇编语言，这种抽象机器的每个存储单元的作用类似于寄存器。三地址代码由三地址语句序列组成，每条三地址语句最多包含三个操作数。源程序的语句（1.1）在语义分析和中间代码翻译阶段可以变换成如下的三地址代码：

```
L_begin:    if i<100 goto L_body                    (1.2)
            goto L_end
L_body:     t1:=sum+i
            sum:=t1
            t2:=i+1
            i:=t2
            goto L_begin
L_end:
```

为了执行语义分析、生成中间代码，编译程序需要管理程序中的各种符号和程序结构的信息，为程序分配和管理运行时需要的存储空间，第6章、第7章分别学习符号表、运行时存储空间的组织和管理。对高级编程语言含义的形式化表示要依据语言的语法结构，我们将在第8章学习一种为语法增添语义的方式——**属性文法**，结合语法制导技术进行语义分析。在第9章介绍典型的中间代码语言，讨论高级程序语言基本结构的语法制导的翻译方法。

1.3.4 代码优化

代码优化的主要任务是对前一阶段产生的中间代码进行等价变换，以便产生执行速度

更快、占用空间更小的目标代码。由于中间代码生成主要是按照语义规则把源程序自动地翻译成中间代码，通常不考虑生成代码的执行效率，因此给代码优化留下了很多机会。例如，在语句（1.1）的翻译代码（1.2）中就可以去掉临时变量 t_1 和 t_2，变换为

```
Lbegin:     if i<100 goto Lbody                    (1.3)
            goto Lend
Lbody:      sum:=sum+i
            i:=i+1
            goto Lbegin
Lend:
```

同样，在下一阶段产生目标代码以后，也可以对目标代码进行优化，改善最终形成的目标程序的效率。无论是中间代码的优化，还是目标代码的优化都要占用编译程序的时间和空间资源，编译的优化需要更多的均衡。本书的第 11 章将集中讨论代码优化的各种策略和算法，重点介绍中间代码和目标代码的基本优化技术。

1.3.5 目标代码生成

编译的最后一个阶段是**目标代码生成**，其主要任务是把（经过优化处理的）中间代码翻译成特定的机器指令或汇编程序。这个阶段的工作依赖于计算机的硬件结构和指令系统，主要涉及机器指令的选择、各种类型变量存储空间的分配以及寄存器的分配和调度等。

例如，使用寄存器 R_0、R_1 和 R_2 可以把中间代码（1.3）翻译成下列优化过的目标代码：

```
            MOV     #100,R0         //把常数 100 存入寄存器 R0
            MOV     i,R1            //把变量 i 的值存入寄存器 R1
            MOV     sum,R2          //把变量 m 的值存入寄存器 R2
Lbegin:     CMP     R1,R0           //比较 R1 和 R0 的值，结果存入状态寄存器 CT
            J>=     Lend            //状态寄存器 CT=1 或 2，即 R1³R0，程序转入单元 Lend
            ADD     R1,R2           //把寄存器 R1 加 R2，结果送入 R2/
            INC     R1              //寄存器 R1 的值加 1
            J       Lbegin          //无条件转移到地址 Lbegin
Lend:
```

目标代码的形式可以是绝对地址指令代码，或可重定位的指令代码，也可以是汇编指令序列。如果是绝对地址的指令代码，则这种目标程序可以立即执行。如果目标代码是汇编代码，则需要汇编器汇编之后才能运行。目前多数编译程序都是产生一种可重定位的指令代码，这种指令代码在运行前需要连接装配程序，把各个目标模块（包括系统提供的库模块）连接起来，确定程序中的变量或常数在主存中的位置，装入内存的起始地址，才能形成一个可以运行的绝对地址代码程序。

本书的第 10 章讨论生成目标代码时需要考虑的因素，并介绍一个简单的代码生成算法。

Python 编译器

1.4 编译程序的构成

上述的编译过程反映了编译工作的动态特征，可以按照这些过程的各个阶段来设计编译程序。如图 1.4 所示，展示的编译程序体系结构已经成为经典的软件设计模式，可以作为编译程序的设计参考模型。典型的编译程序包括五个基本功能模块和两个辅助模块。

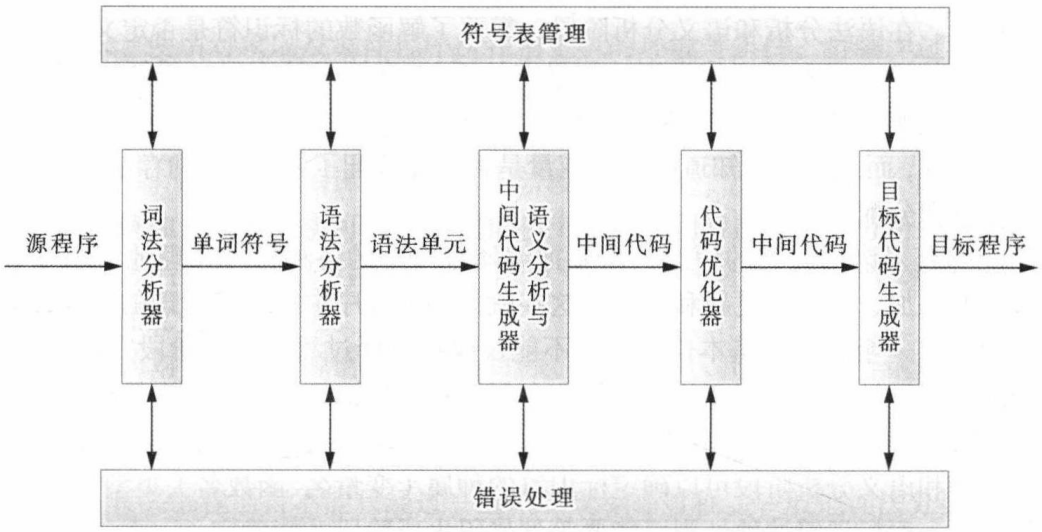


图 1.4 编译程序结构图

1.4.1 基本功能模块

词法分析器，又叫**扫描器**，对输入的源程序执行词法分析工作，输出单词符号序列。

语法分析器，又叫**分析器**，对单词符号序列进行语法分析，识别出各类语法单元，判断输入的符号串是否构成语法正确的“程序”。

语义分析与中间代码生成器，对语法正确的各类程序单元解释语义，为每个变量、函数等组织并分配存储空间，检查类型是否一致等，并把它们翻译成一定形式的中间代码。

代码优化器，执行对中间代码的优化处理。这项工作通常把中间代码表示成等价的**程序流图**、**抽象语法树**等形式，根据程序的控制流和数据流等信息进行代码删除、代码换位等改造，提高代码的执行效率。

目标代码生成器，根据特定的机器把中间代码翻译成目标代码，并进行优化处理。

有的编译程序把词法分析器作为一个独立的子程序，在语法分析的时候调用。有的编译程序用语法分析器构造并输出表示语法结构的语法树，然后依据语法树进行语义分析和中间代码生成。还有的编译程序在语法分析完成之后并不构造语法树，而是在语法分析的时候调用相应的语义子程序，同时完成语义检查和中间代码生成。在这种编译程序中，扫描器、分析器和中间代码生成器这三者并不是如图 1.4 所示的顺序关系，而是以分析器为核心，不断调用扫描器和语义子程序。

为了完成这五个基本任务，编译程序还需要组织和管理程序中的每个变量、常数、函数等信息。如果源程序出现了错误，需要编译程序能够准确识别和正确处理。这些工作分别在**符号表管理**和**错误处理**的模块中完成，这些辅助模块在编译过程中和编译的基本功能模块保持通讯。

1.4.2 符号表的组织与管理

在编译过程的各个阶段中，经常需要知道程序中各个符号和程序结构的信息。例如，在词法分析阶段，如何判断一个标识符是否是编程语言的关键字，等号“=”是不是该语言定义的运算符。在语法分析和语义分析阶段，需要了解函数的标识符是否定义过以及参数的个数、类型、传递方式和返回值类型（如果有的话）。在语义检查时，需要知道运算数的类型，以便执行类型检查和类型转换。在代码生成阶段，需要知道程序中的变量类型，以便合理地分配内存，而且还必须知道同名的变量是否已经分配了内存、在内存中的地址是什么、是否需要再重新分配内存。

为了能够获得诸如此类的信息，需要把编译程序中的各种符号合理地组织和管理，方便符号信息的添加、查询、更新和删除。这些就是编译程序中**符号表管理**的主要功能。

一般而言，一个符号的基本信息通常不能仅仅在编译过程的一个阶段完成。这就表明，符号表是随着编译的过程动态变化的。例如，每当扫描器识别出一个标识符的时候，就要查询它是否在符号表中，如果不在，就把它填入语法符号表，这时还不能确定该标识符的其他属性。在语法和语义分析阶段可以确定标识符的种属（变量名、函数名）、类型（整型、实型）、作用域等信息，而标识符的地址则可能要等到代码生成阶段才能确定。

1.4.3 错误诊断和报告

人们编写的程序难免会有错误和疏漏。软件工程研究的一个重要定律就是在软件的系统分析、设计、编码实现以及运行维护的整个生命周期中，及时地发现和消除错误。错误发现得越晚，修改这些错误的成本就越高。而且，在运行阶段错误的处理代价与运行前相比成指数增长。因此，我们希望一个编译程序不仅能把书写正确的源程序进行翻译，而且，也能有效地识别、诊断、分析和报告程序中的各种错误。这些就是编译程序的**错误处理**模块的基本功能。

程序的错误各种各样，编译程序可以识别的错误主要是语法错误和语义错误这两类。语法错误又可以分成词法错误和句法错误。语法错误指的是源程序中不符合语法规则的错误，通常可以在词法分析和语法分析阶段检测出来。例如，词法分析可以发现“非法字符”拼写错误之类的错误；在语法分析阶段可以发现诸如“括号不匹配”、“关键字拼错”（例如，把 C 语言的 do…while 循环语句中的一个单词写成 whle）等。语义错误指的是源程序中违背语言语义规则的错误。语义错误通常可以在语义分析阶段检查出来，但是，也有的语义错误只能在程序运行时发现。语义错误通常包括变量的作用域错误、类型不一致、说明错误等。

本书没有集中讨论错误的诊断和处理，在第 2、4、5、8 章等有关章节中分别介绍错误处理。

1.5 其他与编译有关的概念和技术

1.5.1 遍的概念

上面描述的编译过程把编译程序看作是一个变换系统：每一阶段所产生的中间结果都作为下一阶段的输入，由下一阶段进行处理，并产生相应的输出。源程序是这个变换系统的第一个输入，目标代码就是最后的输出。在编译程序具体实现时，往往根据不同的高级编程语言、设计要求、使用对象以及编译程序所在宿主机的环境等条件，将编译过程组织为若干**遍**（**趟**）。一个编译程序最终经过几遍完成，就称为几遍编译。

遍就是对源程序及其中间结果从头到尾扫描一次，进行有关的加工处理，产生新的中间结果或目标程序。既可以把编译过程的若干阶段合为一遍，又可以把一个阶段的编译工作分解成若干遍。例如，词法分析可以一遍扫描整个源程序同时进行词法分析；代码优化可以分别在语义分析之后及目标代码生成之后用两遍完成。当一遍中包含若干编译阶段的时候，各个阶段的工作相互交叉和渗透。

编译程序遍的划分依赖于源语言、设计目标、设计人员、硬件条件等诸多因素，没有统一的规则。遍数多，有助于设计清晰的编译程序的结构和逻辑，但是会增加编译器的输入/输出，影响编译时间。遍数少的编译器则设计逻辑复杂、占用较大内存，但是编译的速度快。需要说明的是，有些编程语言不能只经过一遍就实现编译过程。一般把词法分析、语法分析、语义分析及中间代码生成安排作为一遍，把目标代码生成及其优化作为一遍处理。

1.5.2 编译的前端和后端

通常，编译都划分成**前端**和**后端**。**编译前端**只依赖于源程序，独立于目标计算机。编译前端的工作包括词法分析、语法分析、语义分析、中间代码生成及其优化，文法错误的处理和符号表的组织也在编译前端完成。**编译后端**的工作主要是目标代码的生成和优化，独立于源程序，完全依赖于目标机器和中间代码。把编译程序分成前端和后端已经成为目前编译程序的设计实践，其显著优点是，可以优化配置不同的编译程序组合，实现编译的重用，保持语言与机器的独立性。

可以在编译前端将几种源语言程序编译成相同的中间语言，然后配上一个相同的编译后端，这样就可以为同一台计算机构造出不同的编译程序。另外，还可以为一个编译前端配上不同的编译后端，以生成不同的目标代码，就可以实现一次编程，运行在不同的目标计算机。为了实现编译程序可以改变目标机，通常需要良好定义的中间语言。例如，较早的中间语言是为 Pascal 设计的 P 代码，为庞大、复杂的 Ada 语言设计的中间语言是一种称为 Diana 的树形结构。又如，在 Java 语言环境中设计了中间代码 Bytecode，任何机器只要安装了执行 Bytecode 的 Java 解释器，就可以执行 Java 程序，从而实现 Java 语言的平台无关性。

1.5.3 编译程序的分类

如同存在着形形色色的程序设计语言一样，也存在着各种各样的编译程序。根据不同