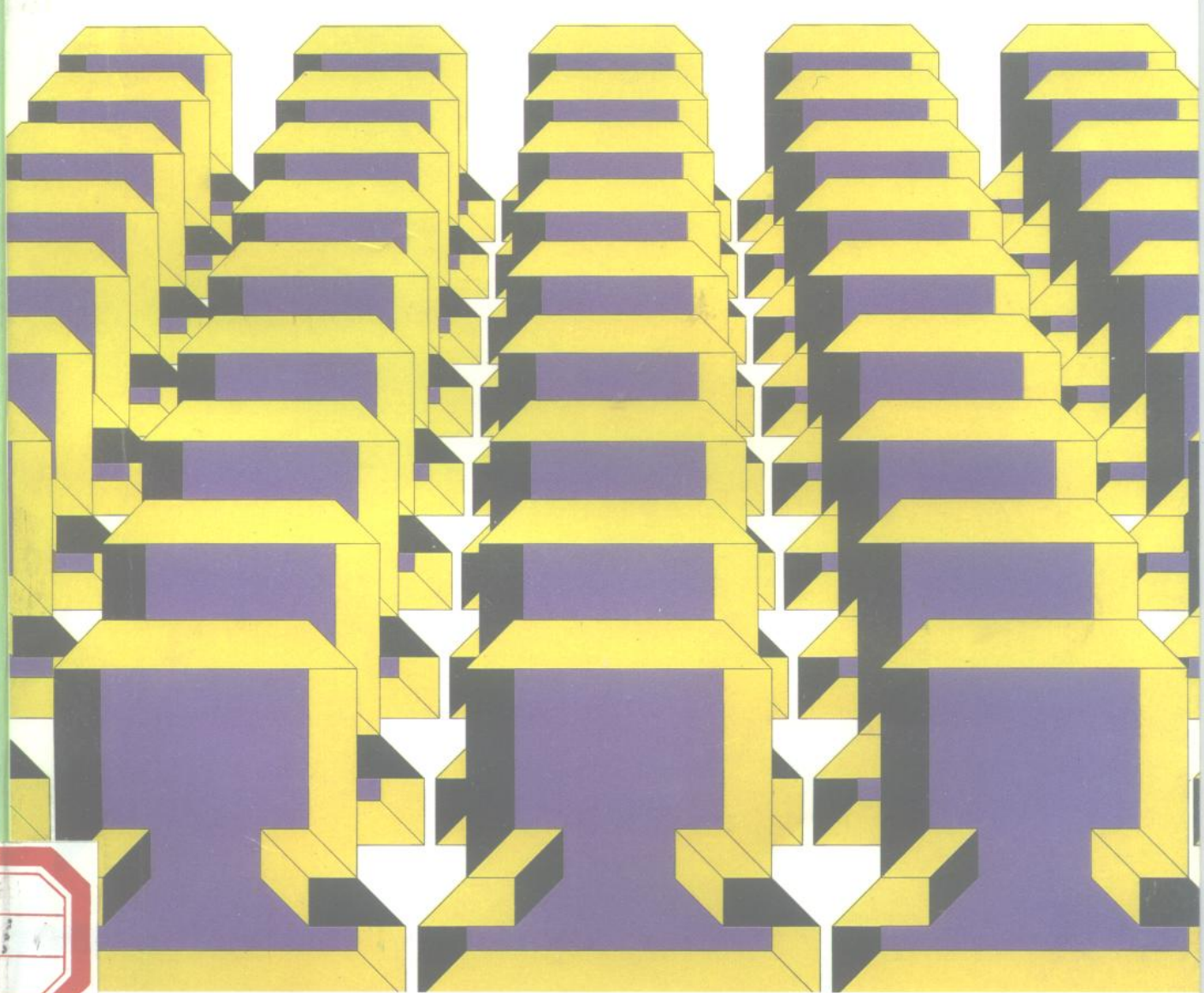
陈意云　编著

编译原理和技术

（第二版）



中国科学技术大学出版社

编译原理和技术
陈意云　编著
中国科学技

图书在版编目(CIP)数据

编译原理和技术(第二版)/陈意云　编著.—合肥：中国科学技术大学出版社，1997年12月
ISBN 7-312-00889-5

Ⅰ　编译原理和技术
Ⅱ　陈意云
Ⅲ　①计算机　②编译
Ⅳ　TP314

中国科学技术大学出版社出版发行
(安徽省合肥市金寨路96号，230026)
安徽合肥骆岗印刷总厂印刷
全国新华书店经销

开本：787×1092/16　印张：20.75　字数：518千
1989年12月第1版　1997年12月第2版
1997年12月第4次印刷
印数：13001—18000册
ISBN 7-312-00889-5/TP·175　定价：20.00元

内 容 简 介

本书介绍了编译器构造的一般原理和基本实现方法，反映了直至90年代的一些重要成果，其内容包括词法分析、语法分析、中间代码生成、代码优化和目标代码生成等.除了介绍传统程序设计语言的编译技术外，本书还介绍了面向对象语言和函数式程序设计语言的实现技术.作为原理性的教材，本书旨在介绍基本的理论和方法，而不偏向于某种源语言或目标机器.全书内容充实，图文并茂，各章节之间循序渐进，并在各章之后附有习题，供读者学习时练习和参考.

本书可作为高等院校计算机科学专业的教材，也可作为软件工程技术人员的参考书.

第二版前言

本书第一版于1989年出版之后，被国内一些大专院校和其它单位用作教材或教学参考书，受到不同程度的欢迎.为了适应程序设计语言及其编译技术的发展，也为了使本书更适合于作为一本教材，我们在第二版中对原书的内容作了部分删除、修改和补充.

我们删掉了原书中一些过时的章节，如第10章.为了避免该书的内容过多，我们也不得不删去不少内容，如语法分析中的算符优先分析算法，语法制导翻译中的属性值空间指派，类型检查中的合一算法，运行环境中的动态存储分配等.对代码生成和代码优化，我们作了较大的改动，以强调对基本概念和方法的了解，而不过多地介绍各种算法.

根据十多年的教学经验，我们在一些重要的章节中增加了一些例题和习题，以帮助读者理解概念和掌握技术.

面向对象的概念和语言在国内也逐步普及，因此我们增加第10章，专门介绍面向对象语言的实现方案.

对于命令式语言以外的语言，如函数式程序设计语言和逻辑程序设计语言，我们增加了第11章，选择函数式语言为代表来介绍非命令式语言以及它的实现技术，使读者对程序设计语言及其实现技术有更全面的了解.

在修订过程中，我们尽量保持原书风格，同时又增加一些反映编译技术发展的内容.尽管如此，但难免还有疏忽谬误之处，敬请同行指正.

陈意云

1996年12月

第一版前言

本书是根据国外的一些专著和论文编写的，它不仅包含了最经典、最广泛应用的基本编译技术，还反映了直至80年代的一些最重要的新成果，这是本书区别于其它有关这方面著作的显著特点.编者有志于编写一本适合国内大学的编译课程教材，希望本书的出版会对“编译原理和技术”的教学产生积极的影响，这也是编者的最大愿望.

虽然只有少数人要去构造或维护程序设计语言的编译器，但是读者可以把本书讨论的概念和技术应用于一般的软件设计之中.例如，建立词法分析器的串匹配技术已用于正文编辑器、信息检索系统和模式识别程序；上下文无关文法和语法制导定义已用于创建许多诸如排版、绘图系统的小语言；代码优化技术已用于程序验证器和从非结构化的程序产生结构化程序的编程之中.无疑，读者会从本书中学到许多新的软件设计方法，尤其是通过自己亲手编写一个语言(比如PL/0)的编译器，掌握一些软件设计技巧，肯定会受益非浅.

本书注重讨论在设计语言翻译时普遍遇到的问题，而不偏向于某种源语言或目标机器.第1章介绍编译器的基本结构，它是本书其余部分的基础.第2章涉及词法分析器、正规式、有限自动机和扫描器生成器，该章内容被广泛用于正文处理.第3章主要讲述了各种分析技术，这些技术包括从适于手工实现的递归下降方法到已用于分析器生成器的更为精致的LR技术.第4章介绍语法制导翻译的主要概念，本书其余各章都会应用到这些概念来描述和实现翻译.第5章提出了完成静态语义检查的主要思想，并详细讨论了类型检查和合一问题.第6章讨论了支持程序运行环境的存储组织问题.第7章从讨论中间语言开始，说明了如何把一般的程序设计语言结构翻译成中间代码.第8章涉及目标代码生成，包括简单的代码生成方法和产生表达式的优化代码的方法，同时也讨论了窥孔优化和代码生成器.第9章是代码优化的综合论述，详细探讨了数据流分析和全局优化的主要方针.最后一章讨论实现编译器的一些编程问题，软件工程和测试在构造编译器时显得尤其重要.

编者在中国科学技术大学讲述“编译原理和技术”课程时，就以本书作为教材.广大同学普遍反映效果良好，认为通过这门课程的学习，他们不仅提高了编程技巧，掌握了软件设计新技术，而且对计算机系统软件有了一个比较清楚的了解，对今后进一步的学习和研究起到了登堂入室的作用.当然，由编者水平有限，书中难免还存在一些缺点和错误，恳请广大读者批评指正.

编　者

1988年12月

目　　次

第1章 引 论

从理论上说，构造专用计算机来直接执行用某种高级语言编写的程序是可能的.但是，目前的机器能执行的是非常低级的语言，即**机器语言**.那么，一个基本的问题是：高级语言最终是怎样在计算机上执行的呢?

术语**编译**代表从面向人的源语言表示的算法到面向硬件的目标语言表示的算法的一个等价变换.本章将通过描述编译器的各个组成部分和编译器完成它的工作的环境来介绍编译这个课题，该课题涉及程序设计语言、机器结构、语言理论、算法和软件工程等方面的知识.

1.1 编译的阶段

能够完成一种语言到另一种语言变换的软件称为**翻译器**.编译器是其中的一类，它的特点是目标语言比源语言低级.编译器的工作可以分成若干阶段，每个阶段把源程序从一种表示变

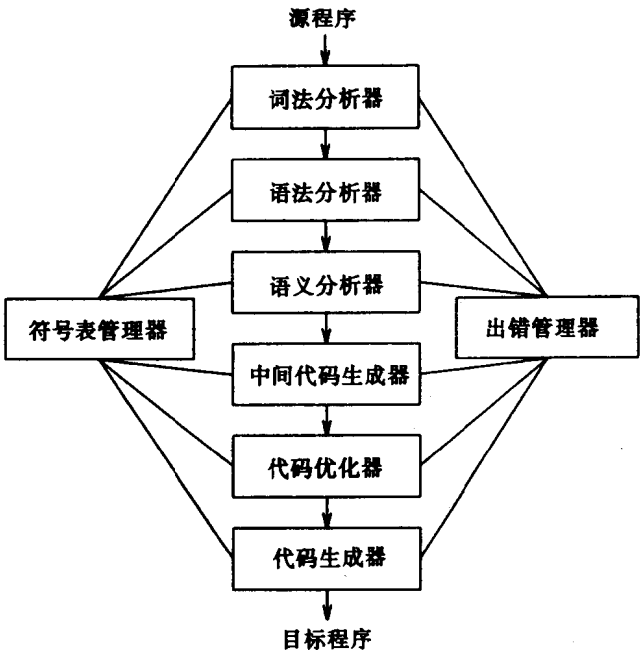


图 1.1 编译的阶段

换成另一种表示.编译过程的一种典型分解见图 1.1，图中的每个方框表示它的一个阶段.实

际上，若干阶段可以组合在一起，各阶段之间的中间表示也无需显式构造.

下面我们以 Pascal 语言的赋值语句(假定变量都是实型)

$$\text{position} := \text{initial} + \text{rate} * 60 \tag{1.1}$$

的翻译为例来概要介绍编译的各个阶段.

1.1.1 词法分析

词法分析阶段读源程序的字符，把它们组成记号流.记号流的每个记号代表逻辑上有内聚力的字符序列，比如标识符、关键字(如 if, while 等)、标点符号或多个字符的算符(如：= 等).形成记号的字符序列叫做该记号的**单词**(lexeme).

赋值语句(1.1)的字符流在词法分析时被组成下面的记号流：

(1) 标识符 position

(2) 赋值号：=

(3) 标识符 initial

(4) 加号 +

(5) 标识符 rate

(6) 乘号 *

(7) 数 60

分隔记号的空格通常在词法分析时被删去.

词法分析对某些记号还增加一个“单词值”.例如，发现 rate 这样的标识符时，词法分析器不仅产生一个记号，如 **id**，还把它的单词 rate 填入符号表，如果表中还没有它的话.**id** 的这次出现的单词值是符号表中 rate 条目的指针.

我们用 $\mathbf{id}_1$，$\mathbf{id}_2$ 和 $\mathbf{id}_3$ 分别表示 position，initial 和 rate，以强调标识符的内部表示是区别于形成标识符的字符序列的.于是，(1.1)在词法分析后的表示是

$$\mathbf{id}_1 := \mathbf{id}_2 + \mathbf{id}_3 * 60 \tag{1.2}$$

还应该为多字符算符：= 和数 60 构造记号，以反映它们的内部表示.我们把它们延迟到第 2 章“词法分析”中再讨论.

编译器的词法分析也叫做**线性分析**或**扫描**.

1.1.2 语法分析

语法分析简称为分析，它把记号流按语言的语法结构层次地分组，以形成语法短语.因此语法分析也称为**层次分析**.源程序的语法短语常用分析树表示，图 1.2 便是一例.

在表达式 initial + rate * 60 中，短语 rate * 60 是一个逻辑单位，因为按一般的算术表达式的习惯，乘比加先完成；由于表达式 initial + rate 后面是乘号，所以它不能组成一个短语.

程序的层次结构通常由递归的规则表示.例如，可以用如下规则作为表达式定义的一部分：

(1) 任何一个标识符是表达式；

(2) 任何一个数是表达式；

(3) 如果 e_1 和 e_2 都是表达式，那么

$e_1 + e_2$

$e_1 * e_2$

(e_1)

也都是表达式.

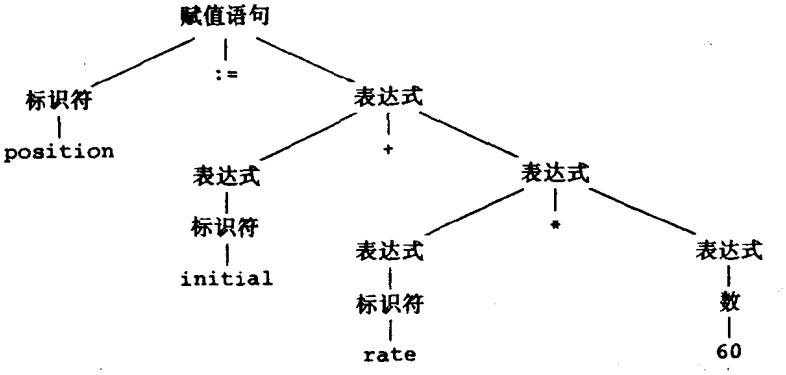


图 1.2 position := initial + rate * 60 的分析树

规则(1)和规则(2)是(非递归的)基本规则，而规则(3)是把算符作用于其它表达式来定义表达式的.于是，根据规则(1)，initial 和 rate 都是表达式；由规则(2)，60 是表达式；再由规则(3)，首先可推出 rate * 60 是表达式，然后 initial + rate * 60 也是表达式.

同样地，许多语言用类似如下的规则递归地定义语句：

(1) 如果 *identifier* 是标识符，*expression* 是表达式，那么

identifier := *expression*

是语句；

(2) 如果 *expression* 是表达式，*statement* 是语句，那么

while (*expression*) **do** *statement*

if (*expression*) **then** *statement*

也都是语句.

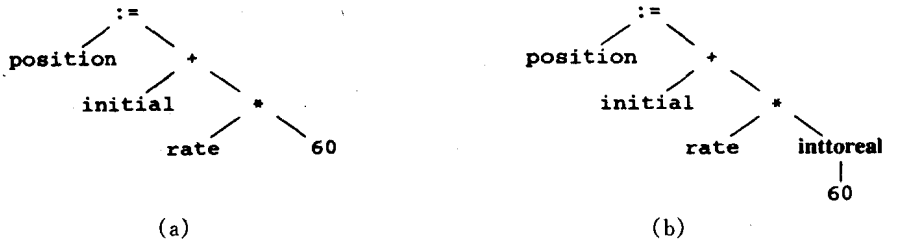
词法分析和语法分析的划分是有点任意的，通常选择可以简化整个分析任务的划分.决定这种划分的一个因素是语言的结构是否允许递归，词法结构一般不需要递归，而语法结构通常需要递归.上下文无关文法是递归规则的一种形式化表示，它可以用来指导语法分析，我们在第 3 章中将进一步研究它.

例如，递归无需用于识别标识符.标识符是由字母开头的字母和数字串.识别标识符时，我们简单地扫描输入串，直到发现一个既不是字母也不是数字的字符为止，然后把已看见的所有字母和数字组成一个标识符记号，并把这些字符记录在符号表中，再把它们从输入串中移开，下一个记号的处理便可以开始.

这种线性扫描不足以分析表达式和语句.比方说，不在输入中加上某种层次或嵌套的结构，我们就不能正确地匹配表达式的括号或语句的 begin 和 end.

图 1.2 的分析树描绘了输入的语法结构，这种语法结构更常见的内部表示由图 1.3(a)的语法树给出.语法树是分析树的浓缩表示，其中算符作为内部结点出现，它的运算对象作为它

的后代.图 1.3(a)这样的树结构将在 4.2 节中讨论.在第 4 章的语法制导翻译中,我们将详细讨论编译器如何利用输入所含的层次结构来产生输出.



(a) (b)

图 1.3 语义分析插入了整型到实型的转换

1.1.3 语义分析

语义分析阶段检查程序的语义正确性,以保证程序各部分能有意义地结合在一起,并为以后的代码生成阶段收集类型信息.它使用语法分析阶段确定的层次结构来标识表达式和语句的算符和运算对象.

语义分析的一个重要部分是类型检查,编译器检查每个算符的运算对象,看它们的类型是否合法.例如,当实数作为数组的下标时,许多语言的定义要求编译器报告错误.当然,也有些语言允许运算对象类型隐式转换,如二元的算术算符作用于一个整数和一个实数时.类型检查和语义分析将在第 5 章中讨论.

例 1.1 在机器内部,整数的二进制表示和实数的二进制表示是有区别的,即使它们有相同的值.在图 1.3 中,所有的标识符都声明为实型,而由 60 本身可知它是整数.对图 1.3(a)进行类型检查,会发现 * 作用于一个实型变量 rate 和一个整数 60,通常的办法是把整数转变为实数,可以建立一个额外的算符结点 **inttoreal**(见图 1.3(b)),它显式地把整数转变为实数.由于 **inttoreal** 的运算对象是常数,编译器也可能是用等价的实常数来代替这个整常数. □

编译器的前三个阶段对源程序分别进行不同的分析,以揭示源程序的结构和基本数据,决定它们的含义,建立源程序的中间表示.许多处理源程序的软件工具都要完成某类分析,这样的例子有:

(1) *结构编辑器*.结构编辑器取一串命令作为输入来建立源程序.结构编辑器不仅能像普通的正文编辑器那样完成正文的建立和修改,而且能分析程序正文,把恰当的层次结构加在源程序上.这样,结构编辑器能够完成一些对准备程序来说很有用的额外事情.例如,它能够检查输入是否能正确构成程序,能够自动提供关键字(如果用户键入 while,编辑器自动提供匹配的 do,并提醒用户,在这两个关键字之间必须有一个条件),能够从 begin 或左括号跳到匹配的 end 或右括号.而且,这种编辑器的输出往往和编译器分析阶段的输出类似.

(2) *格式打印器*.格式打印器分析源程序,以程序结构清晰可见的方式输出程序.例如,注解可以用特殊的字体出现,语句可以按它们嵌套的层次阶梯式地显示出来.

(3) *静态检查器*.静态检查器读入程序,分析程序,并试图不运行程序而发现一些潜在的错误.它的分析部分和第 9 章优化编译器的分析部分类似.例如,它可以检查出源程序的某些部分绝不会执行,或者某个变量在赋值前可能被引用.此外,使用第 5 章讨论的类型检查技术,