普通高等教育“十五”国家级规划教材

编译原理

陈意云　张昱

高等教育出版社

普通高等教育“十五”国家级规划教材

编 译 原 理

陈意云　张　昱

高等教育出版社

内容简介

本书介绍编译器构造的一般原理和基本实现方法，主要内容包括词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成等。除了介绍命令式编程语言的编译技术外，本书还介绍面向对象语言和函数式编程语言的实现技术。本书还强调一些相关的理论知识，如形式语言和自动机理论、语法制导的定义和属性文法、类型论和类型系统等。

本书取材广泛新颖、图文并茂，注意理论联系实际。本书可作为高等学校计算机科学及相关专业的教材，也可供计算机软件工程技术人员参考使用。

图书在版编目（CIP）数据

编译原理／陈意云，张昱. —北京：高等教育出版社，2003.9
ISBN 7-04-013367-9

Ⅰ.编... Ⅱ.①陈...②张... Ⅲ.编译程序－程序设计－高等学校－教材 Ⅳ.TP314

中国版本图书馆CIP数据核字（2003）第072530号

出版发行	高等教育出版社	购书热线	010－64054588
社　　址	北京市西城区德外大街4号	免费咨询	800－810－0598
邮政编码	100011	网　　址	http://www.hep.edu.cn
总　　机	010－82028899		http://www.hep.com.cn
经　　销	新华书店北京发行所		
印　　刷	廊坊市文峰档案文化用品有限公司		
开　　本	787×1092　1/16	版　　次	2003年9月第1版
印　　张	24.75	印　　次	2003年9月第1次印刷
字　　数	500 000	定　　价	30.00元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

前　言

本书介绍编译器构造的一般原理、基本设计方法和主要实现技术，可作为高等学校计算机科学及相关专业的教材。

虽然只有少数人从事构造或维护程序设计语言编译器的工作，但是编译原理和技术对高校学生和计算机软件工程技术人员来说仍是重要的基础知识之一。本书能使读者对程序设计语言的设计和实现有深刻的理解，对和程序设计语言有关的理论有所了解，对宏观上把握程序设计语言来说，能起一个奠基的作用。本书的学习还有助于读者快速理解、定位和解决在程序调试与运行中出现的问题。

对软件工程来说，编译器是一个很好的实例(基本设计、模块划分、基于事件驱动的编程等)，本书所介绍的概念和技术能应用到一般的软件设计之中。

大多数程序员同时也是语言的设计者，虽然只是一些简单语言(如输入输出、脚本语言)的设计者，但学习本书仍有助于提高他们设计这些语言的水平。

编译技术在软件安全、程序理解和软件逆向工程等方面有着广泛的应用。

作为教材，本书有如下一些特点：

(1) 在介绍语言实现技术的同时，强调一些相关的理论知识，如形式语言和自动机理论、语法制导的定义和属性文法、类型论和类型系统等。它们是计算机专业理论知识的一个重要部分，在本书中结合应用来介绍这些知识，有助于学生较快领会和掌握。

(2) 在介绍编译器各逻辑阶段的实现时，强调形式化描述技术，并以语法制导定义作为翻译的主要描述工具。

(3) 强调对编译原理和技术的宏观理解及全局把握，而不把读者的注意力分散到一些枝节的算法上，如计算开始符号集合和后继符号集合的算法、回填技术等。出于同样的目的，本书较详细地介绍了编译系统和运行系统。

(4) 本书还介绍了面向对象语言和函数式语言的实现技术，有助于加深读者对语言实现技术的理解。书中带星号的章节，作为教学的可选部分。

(5) 作为原理性教材，本书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。

(6) 我们鼓励读者用所学的知识去分析和解决实际问题，因此本书中有很多习题是从实际碰到的问题中抽象出来的。这些习题也能激发读者学习编译原理和技术的积极性。

(7) 为了便于读者学习，本书配有习题解答(见参考文献 8)。

本书的多数章节是参考了参考文献1和参考文献7编写的，部分习题取自参考文献8。在此向有关作者表示感谢。

本书第1章到第6章以及第12章主要由陈意云编写，第7章到第11章主要由张昱编写。作者的学生陈晖准备了10.2节的初稿，李筱青准备了10.3节的初稿，吴萍和项森也为第10章的编写做了很多技术工作。

中国科学院软件研究所研究员程虎先生审阅了全书，并提出了许多宝贵的意见，在此表示衷心的感谢。

由于作者水平有限，书中难免还存在一些缺点和错误，恳请广大读者批评指正。

作 者

于中国科学技术大学

2003年5月

策划编辑　刘建元
责任编辑　刘建元
封面设计　于文燕
责任印制　韩　钢

郑重声明

目　录

第1章　编译器概述

从理论上说，构造专用计算机来直接执行某种高级语言写的程序是可能的。但是，实际上目前的计算机能执行的都是非常低级的**机器语言**。那么，一个基本的问题是：高级语言的程序最终是怎样在计算机上执行的。

能够完成从一种语言到另一种语言变换的软件称为**翻译器**，这两种语言分别叫做该翻译器的**源语言**和**目标语言**。**编译器**是一种翻译器，它的特点是目标语言比源语言低级。

本章通过描述编译器的各个组成部分来介绍编译这个课题。该课题涉及程序设计语言、机器结构、形式语言理论、类型论、算法和软件工程等方面的知识。

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。

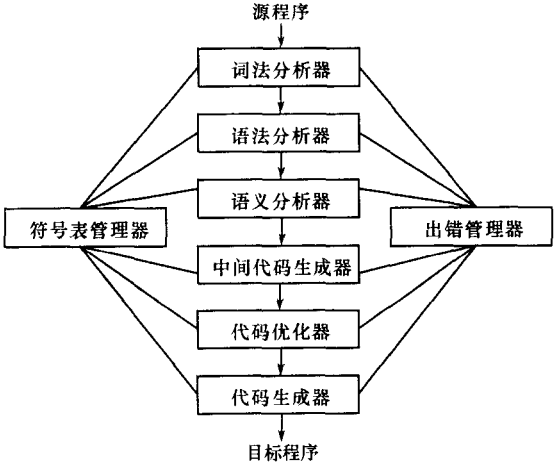

图 1.1　编译的阶段

本节以 Pascal 语言的赋值语句

```
position := initial + rate * 60
```　　(1.1)

的翻译（假定变量都是实型）为例，概要介绍编译的各个阶段。

1.1 词法分析

词法分析逐个读构成源程序的字符，把它们组成词法**记号**(*token*)流。赋值语句(1.1)的字符流在词法分析时被组成下面的词法记号流：

(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。

词法单元 position、initial 和 rate 属于同样的记号，因此需要为某些记号增加一个属性来区分属于同一记号的不同词法单元。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**，还把当前词法单元 rate 填入符号表，如果表中还没有它的话。**id** 这次出现的属性是符号表中 rate 条目的指针。

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有别于形成标识符的字符序列的。于是，语句(1.1)在词法分析后的表示是

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

多字符算符:= 和数 60 也应该有它们的内部表示，本书将在第 2 章词法分析中讨论，为直观起见，这里直接用它们在源程序中的字符序列。

编译器的词法分析也叫做**线性分析**或**扫描**。

1.2 语法分析

语法分析(syntax analysis)简称为**分析**(parsing)，它把词法记号流依照语言的语法结构按层次分组，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，因为按算术表达式的一般习惯，乘比加先完成；由于 initial + rate 后面是乘号，所以 initial + rate 不能组成一个短语。

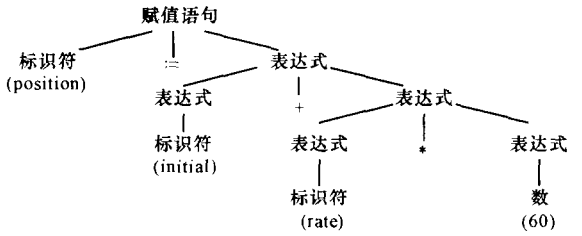

图 1.2 position := initial + rate * 60 的分析树

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) 任何一个标识符都是表达式；

(2) 任何一个数都是表达式；

(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。

规则(1)和(2)都是(非递归的)基本规则，而规则(3)是通过把算符作用于表达式来定义更复杂的表达式。这样，由规则(1)，initial 和 rate 都是表达式；由(2)，60 是表达式；由(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。

图 1.2 的分析树描绘了 position := initial + rate * 60 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，其中内部结点都是算符，内部结点的后代是它的运算对象。图 1.3(b)这样的树结构将在第 4 章 4.2 节讨论。在第 4 章的语法制导翻译中，将详细讨论编译器如何利用输入所含的层次结构来产生语法树。

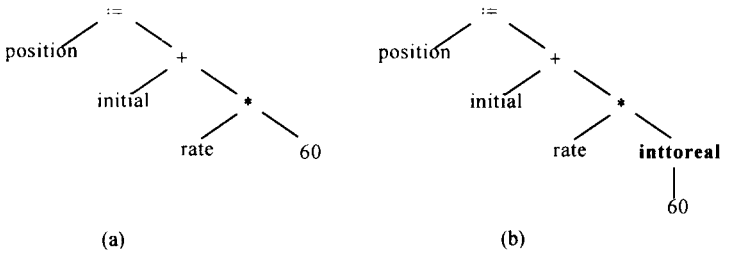

图 1.3　语义分析插入了类型转换

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。

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，看它们的类型是否适当。例如，当实数作为数组的下标时，许多语言的规定是要求编译器报告错误；当然也有些语言允许运算对象的类型隐式转换，例如二元的算术算符作用于一个整数和一个实数的时候。类型检查和语义分析在第 5 章讨论。

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，不论它们是否有相同的值。在图 1.3 中，所有的标识符都声明为实型，另外，由 60 本身可知它是整数。对图 1.3(a)进行类型检查会发现 * 作用于实型变量 rate 和整数 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。

编译器的前三个阶段对源程序分别进行不同的分析，以揭示源程序的基本数据和结构，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，下面给出几个例子。

(1) **格式打印程序**　它以源程序为输入，以程序结构清晰可见的方式输出源程序。例如：注释可以以特殊的字体或颜色出现，语句可以按它们嵌套的层次阶梯式地显示出来。显然，格式打印程序需要对源程序进行词法分析和语法分析，但不需要对源程序进行语义分析，因为一种书写格式只和语法有关。

(2) **文档抽取程序**　它抽取程序文件中的注释和函数首部等信息，形成一份程序文档。它需要对程序进行词法分析和语法分析，至少是部分的语法分析。要使得这样抽取的信息

形成一份有用的文档,注释很有讲究,否则抽取出来的文档没有什么用处。

(3) **静态检查程序** 静态检查程序读入程序,分析它,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决不会执行,或者某个变量在赋值前可能被引用。此外,使用比第五章更精致的类型检查和类型推导技术,它还能捕捉程序的安全隐患,例如通过指针使用已经释放了的存储空间。

(4) **解释器** 纯解释器在执行源程序语句时,都需分析构成该语句的字符串,以便识别和执行它指定的计算。如果给定的语句仅执行一次,纯解释的方法是所有方法中代价最小的一种,例如,它常用于交互语言的"立即命令"。如果语句重复执行,较好途径是分析源程序的字符流仅一次,用一串更适于解释的符号序列或其他的形式来代替它。因此解释器往往也做某种程度的翻译,例如,对于赋值语句,解释器可能建立像图1.3(a)那样的树,在遍历树时执行结点的操作。在树根,它发现必须完成赋值,因此调用子例程来计算右部表达式的值,然后把值存到为标识符 position 分配的存储单元。计算树根右子树的子例程发现必须计算两个表达式的和,它递归地调用自己来计算表达式 rate * 60 的值,然后再加上变量 initial 的值。

1.4 中间代码生成

语法分析和语义分析后,某些编译器产生源程序的显式中间表示,可以认为这种中间表示是一种抽象机的程序。中间表示必须具有两个性质:它易于产生并且易于翻译成目标程序。

中间表示有各种形式。在第7章,我们把中间表示看成**三地址代码**,它们像机器的汇编语言,这种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,每个三地址语句最多有三个操作数,语句(1.1)的三地址代码可以如下:

```
temp1 := inttoreal (60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```
(1.3)

这种中间形式有它的特点。首先,除了赋值算符外,每个语句至多只有一个算符,因此,在生成这些语句时,编译器必须决定运算完成的次序,语句(1.1)的乘优先于加。其次,编译器必须产生临时变量名,用以保留每个语句的计算结果。第三,某些三地址语句没有三个运算对象,例如语句(1.3)的第一个和最后一个语句。

本书在第7章叙述编译器时用的主要是中间表示。通常,除了计算表达式外,这些中间