



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。

版权所有 侵权必究

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

高等教育出版社依法对本书享有专有出版权。任何未经许可的复制、销售行为均违反《中华人民共和国著作权法》，其行为人将承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。为了维护市场秩序，保护读者的合法权益，避免读者误用盗版书造成不良后果，我社将配合行政执法部门和司法机关对违法犯罪的单位和个人给予严厉打击。社会各界人士如发现上述侵权行为，希望及时举报，本社将奖励举报有功人员。

反盗版举报电话：(010) 58581897/58581698/58581879/58581877

传 真：(010) 82086060

E - mail：dd@hep.com.cn 或 chenrong@hep.com.cn

通信地址：北京市西城区德外大街4号

高等教育出版社法律事务部

邮 编：100011

购书请拨打电话：(010)64014089 64054601 64054588

目 录

第 1 章 编译器概述	(1)	3.1 上下文无关文法	(39)
1.1 词法分析	(2)	3.1.1 上下文无关文法的定义	(39)
1.2 语法分析	(2)	3.1.2 推导	(41)
1.3 语义分析	(4)	3.1.3 分析树	(43)
1.4 中间代码生成	(5)	3.1.4 二义性	(43)
1.5 代码优化	(6)	3.2 语言和文法	(44)
1.6 代码生成	(6)	3.2.1 正规式和上下文无关 文法的比较	(45)
1.7 符号表管理	(7)	3.2.2 分离词法分析器的理由	(45)
1.8 错误诊断和报告	(7)	3.2.3 验证文法产生的语言	(46)
1.9 阶段的分组	(9)	3.2.4 适当的表达式文法	(46)
习题 1	(9)	3.2.5 消除二义性	(47)
第 2 章 词法分析	(10)	3.2.6 消除左递归	(49)
2.1 词法记号及属性	(10)	3.2.7 提左因子	(50)
2.1.1 词法记号、模式、词法单元	(11)	3.2.8 非上下文无关的语言结构	(51)
2.1.2 词法记号的属性	(12)	3.2.9 形式语言鸟瞰	(52)
2.1.3 词法错误	(13)	3.3 自上而下分析	(53)
2.2 词法记号的描述与识别	(13)	3.3.1 自上而下分析的一般方法	(54)
2.2.1 串和语言	(13)	3.3.2 LL(1)文法	(55)
2.2.2 正规式	(15)	3.3.3 递归下降的预测分析	(56)
2.2.3 正规定义	(16)	3.3.4 非递归的预测分析	(58)
2.2.4 状态转换图	(17)	3.3.5 构造预测分析表	(60)
2.3 有限自动机	(20)	3.3.6 预测分析的错误恢复	(62)
2.3.1 不确定的有限自动机	(21)	3.4 自下而上分析	(65)
2.3.2 确定的有限自动机	(22)	3.4.1 归约	(65)
2.3.3 NFA 到 DFA 的变换	(23)	3.4.2 句柄	(66)
2.3.4 DFA 的化简	(27)	3.4.3 用栈实现移进一归约分析	(67)
2.4 从正规式到有限自动机	(29)	3.4.4 移进一归约分析的冲突	(69)
2.5 词法分析器的生成器	(32)	3.5 LR 分析器	(70)
习题 2	(36)	3.5.1 LR 分析算法	(71)
第 3 章 语法分析	(39)		

3.5.2 LR 文法和 LR 分析	4.4.3 模拟继承属性的计算	(132)
方法的特点	4.5 递归计算	(135)
(74)	4.5.1 自左向右遍历	(136)
3.5.3 构造 SLR 分析表	4.5.2 其他遍历方法	(137)
(75)	4.5.3 多次遍历	(138)
3.5.4 构造规范的 LR 分析表	习题 4	(140)
(83)	第 5 章 类型检查	(143)
3.5.5 构造 LALR 分析表	5.1 类型在程序设计语言中的作用	(144)
(87)	5.1.1 引言	(144)
3.5.6 非 LR 的上下文无关结构	5.1.2 执行错误和安全语言	(145)
(90)	5.1.3 类型化语言的优点	(147)
3.6 二义文法的应用	5.2 描述类型系统的语言	(148)
(92)	5.2.1 定型断言	(149)
3.6.1 使用文法以外的信息来解决	5.2.2 定型规则	(150)
分析动作的冲突	5.2.3 类型检查和类型推断	(151)
(92)	5.3 简单类型检查器的说明	(151)
3.6.2 特殊情况产生式引起的	5.3.1 一个简单的语言	(152)
二义性	5.3.2 类型系统	(152)
(94)	5.3.3 类型检查	(154)
3.6.3 LR 分析的错误恢复	5.3.4 类型转换	(156)
(95)	* 5.4 多态函数	(157)
3.7 分析器的生成器	5.4.1 为什么要使用多态函数	(157)
(97)	5.4.2 类型变量	(158)
3.7.1 分析器的生成器 Yacc	5.4.3 一个含多态函数的语言	(160)
(97)	5.4.4 代换、实例和合一	(161)
3.7.2 用 Yacc 处理二义文法	5.4.5 多态函数的类型检查	(162)
(100)	5.5 类型表达式的等价	(167)
3.7.3 Yacc 的错误恢复	5.5.1 类型表达式的结构等价	(168)
(103)	5.5.2 类型表达式的名字等价	(169)
习题 3	5.5.3 记录类型	(170)
(105)	5.5.4 类型表示中的环	(171)
第 4 章 语法制导的翻译	5.6 函数和算符的重载	(172)
(111)	5.6.1 子表达式的可能类型集合	(172)
4.1 语法制导的定义	5.6.2 缩小可能类型的集合	(174)
(111)	习题 5	(175)
4.1.1 语法制导定义的形式		
(111)		
4.1.2 综合属性		
(113)		
4.1.3 继承属性		
(113)		
4.1.4 属性依赖图		
(114)		
4.1.5 属性计算次序		
(115)		
4.2 S 属性定义的自下而上计算		
(116)		
4.2.1 语法树		
(117)		
4.2.2 构造语法树的语法制导定义		
(117)		
4.2.3 S 属性的自下而上计算		
(119)		
4.3 L 属性定义的自上而下计算		
(121)		
4.3.1 L 属性定义		
(122)		
4.3.2 翻译方案		
(122)		
4.3.3 预测翻译器的设计		
(126)		
4.3.4 用综合属性代替继承属性		
(128)		
4.4 L 属性的自下而上计算		
(129)		
4.4.1 删除翻译方案中嵌入的动作		
(129)		
4.4.2 分析栈上的继承属性		
(130)		

第 6 章 运行时存储空间的组织和管理	(181)	7.3.4 数组元素地址计算的 翻译方案	(226)
6.1 局部存储分配策略	(181)	7.3.5 类型转换	(230)
6.1.1 过程	(182)	7.4 布尔表达式和控制流语句	(231)
6.1.2 名字的作用域和绑定	(182)	7.4.1 布尔表达式的翻译	(232)
6.1.3 活动记录	(183)	7.4.2 控制流语句的翻译	(233)
6.1.4 局部数据的安排	(184)	7.4.3 布尔表达式的控制流翻译	(235)
6.1.5 程序块	(185)	7.4.4 开关语句的翻译	(237)
6.2 全局存储分配策略	(187)	7.4.5 过程调用的翻译	(240)
6.2.1 运行时内存的划分	(187)	习题 7	(241)
6.2.2 静态分配	(188)	第 8 章 代秒生成	(245)
6.2.3 栈式分配	(190)	8.1 代码生成器设计中的问题	(245)
6.2.4 堆式分配	(196)	8.1.1 目标程序	(245)
6.3 非局部名字的访问	(197)	8.1.2 指令选择	(246)
6.3.1 无过程嵌套的静态作用域	(198)	8.1.3 寄存器分配	(247)
6.3.2 有过程嵌套的静态作用域	(198)	8.1.4 计算次序选择	(247)
6.3.3 动态作用域	(202)	8.2 目标机器	(248)
6.4 参数传递	(203)	8.2.1 目标机器的指令系统	(248)
6.4.1 值调用	(203)	8.2.2 指令的代价	(249)
6.4.2 引用调用	(204)	8.3 基本块和流图	(251)
6.4.3 复写 - 恢复调用	(204)	8.3.1 基本块	(251)
6.4.4 换名调用	(205)	8.3.2 基本块的变换	(253)
习题 6	(206)	8.3.3 流图	(254)
第 7 章 中间代码生成	(215)	8.3.4 下次引用信息	(255)
7.1 中间语言	(215)	8.4 一个简单的代码生成器	(256)
7.1.1 后缀表示	(215)	8.4.1 寄存器描述和地址描述	(256)
7.1.2 图形表示	(216)	8.4.2 代码生成算法	(257)
7.1.3 三地址代码	(217)	8.4.3 寄存器选择函数	(258)
7.2 声明语句	(219)	8.4.4 为变址和指针语句产生代码	(259)
7.2.1 过程中的声明	(219)	8.4.5 条件语句	(260)
7.2.2 作用域信息的保存	(219)	习题 8	(261)
7.2.3 记录的域名	(222)	* 第 9 章 代码优化	(269)
7.3 赋值语句	(223)	9.1 优化的主要种类	(269)
7.3.1 符号表中的名字	(223)	9.1.1 代码改进变换的标准	(269)
7.3.2 临时名字的重新使用	(224)	9.1.2 公共子表达式删除	(272)
7.3.3 数组元素的地址计算	(225)	9.1.3 复写传播	(273)

9.1.4 死代码删除	(274)	* 10.3 无用单元收集	(324)
9.1.5 代码外提	(275)	10.3.1 标记和清扫	(325)
9.1.6 强度削弱和归纳变量删除	(275)	10.3.2 引用计数	(326)
9.1.7 优化编译器的组织	(276)	10.3.3 拷贝收集	(327)
9.2 流图中的循环	(278)	10.3.4 分代收集	(328)
9.2.1 必经结点	(278)	10.3.5 渐增式收集	(330)
9.2.2 自然循环	(279)	10.3.6 编译器与收集器之间 的相互影响	(330)
9.2.3 前置结点	(280)	习题 10	(334)
9.2.4 可归约流图	(280)	* 第 11 章 面向对象语言的编译	(337)
9.3 全局数据流分析介绍	(281)	11.1 面向对象语言的概念	(337)
9.3.1 点和路径	(282)	11.1.1 对象和对象类	(337)
9.3.2 到达-定值	(283)	11.1.2 继承	(338)
9.3.3 可用表达式	(286)	11.1.3 信息封装	(341)
9.3.4 活跃变量分析	(289)	11.2 方法的编译	(341)
9.4 代码改进变换	(290)	11.3 继承的编译方案	(344)
9.4.1 公共子表达式删除	(291)	11.3.1 单一继承的编译方案	(345)
9.4.2 复写传播	(292)	11.3.2 重复继承的编译方案	(347)
9.4.3 寻找循环不变计算	(294)	习题 11	(352)
9.4.4 代码外提	(294)	* 第 12 章 函数式语言的编译	(355)
9.4.5 归纳变量删除	(297)	12.1 函数式程序设计语言简介	(355)
习题 9	(300)	12.1.1 语言构造	(355)
第 10 章 编译系统和运行系统	(306)	12.1.2 参数传递机制	(357)
10.1 C 语言的编译系统	(306)	12.1.3 变量的自由出现和约束 出现	(358)
10.1.1 预处理器	(307)	12.2 函数式语言的编译简介	(360)
10.1.2 汇编器	(308)	12.2.1 几个受启发的例子	(360)
10.1.3 连接器	(310)	12.2.2 编译函数	(362)
10.1.4 目标文件的格式	(311)	12.2.3 环境与约束	(363)
10.1.5 符号解析	(313)	12.3 抽象机的系统结构	(364)
10.1.6 静态库	(314)	12.3.1 抽象机的栈	(365)
10.1.7 可执行目标文件及装入	(316)	12.3.2 抽象机的堆	(366)
10.1.8 动态连接	(317)	12.3.3 名字的寻址	(366)
10.1.9 处理目标文件的一些工具	(319)	12.3.4 约束的建立	(368)
10.2 Java 语言的运行系统	(319)	12.4 指令集和编译	(369)
10.2.1 Java 虚拟机语言简介	(320)	12.4.1 表达式	(369)
10.2.2 Java 虚拟机	(321)		
10.2.3 即时编译器	(322)		

12.4.2 变量的引用性出现	(371)	12.4.6 letrec 表达式和局部变量	(378)
12.4.3 函数定义	(372)	习题 12	(380)
12.4.4 函数应用	(373)	参考文献	(382)
12.4.5 构造和计算闭包	(376)		

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。

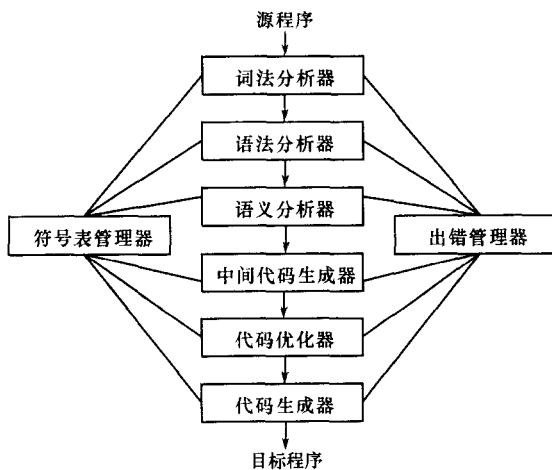


图 1.1 编译的阶段

本节以 Pascal 语言的赋值语句

$$\text{position} := \text{initial} + \text{rate} * 60 \quad (1.1)$$

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 id_1 、 id_2 和 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

$$id_1 := id_2 + id_3 * 60 \quad (1.2)$$

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。

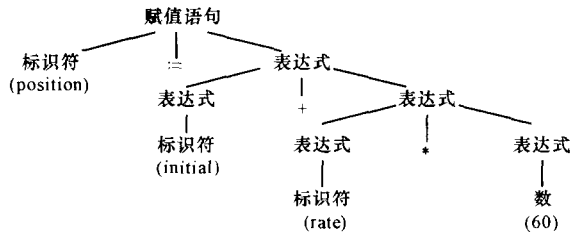


图 1.2 position := initial + rate * 60 的分析树

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

$$\mathbf{while} (expression) \mathbf{do} \mathbf{statement}$$

$$\mathbf{if} (expression) \mathbf{then} \mathbf{statement}$$

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。

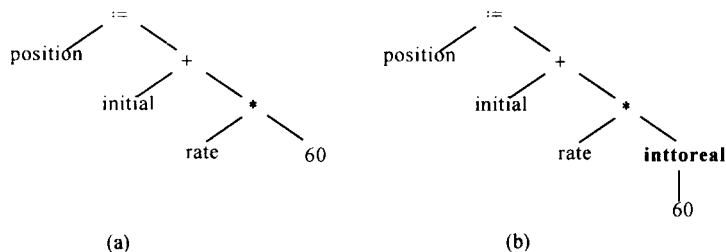


图 1.3 语义分析插入了类型转换

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