普通高等教育“十一五”国家级规划教材

编译原理

（第 2 版）

陈意云　张　昱

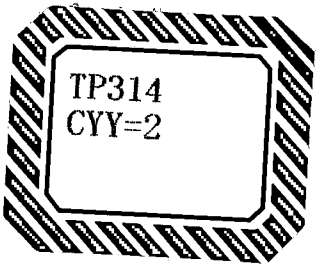
高等教育出版社
Higher Education Press

普通高等教育“十一五”国家级规划教材

编译原理

（第2版）

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，其内容包括词法分析、语法分析、语义分析、中间代码生成、目标代码生成、独立于机器的优化和依赖于机器的优化等。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。

本书内容丰富，讲解深入，并注意理论联系实际，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. —2版. —北京：高等教育出版社，2008.6

ISBN 978-7-04-023963-8

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(2008)第060077号

策划编辑 刘 艳　　**责任编辑** 萧 潇　　**封面设计** 于文燕　　**责任绘图** 郝 林
版式设计 马敬茹　　**责任校对** 姜国萍　　**责任印制** 韩 刚

出版发行 高等教育出版社
社　　址 北京市西城区德外大街4号
邮政编码 100120
总　　机 010-58581000

经　　销 蓝色畅想图书发行有限公司
印　　刷 北京外文印刷厂

开　　本 787×1092　1/16
印　　张 26.75
字　　数 600 000

购书热线 010-58581118
免费咨询 800-810-0598
网　　址 http://www.hep.edu.cn
http://www.hep.com.cn
网上订购 http://www.landraco.com
http://www.landraco.com.cn
畅想教育 http://www.widedu.com

版　　次 2003年8月第1版
2008年6月第2版
印　　次 2008年6月第1次印刷
定　　价 33.20元

物料号　23963-00

前　言

本书改版的动力主要来自下列几个方面：

（1）编译器技术的发展，特别是计算机体系结构的发展对编译器技术的推动；

（2）第1版教材在使用中得到的积极反馈和建议；

（3）作者在教学实践中的心得和体会，在相关科研活动中的进展与收获。

第2版增加了有关新理论和新技术的介绍，重写了因理论和技术发展而需要调整的部分，改写了原先有疏漏而需要改善的地方，并且继承和发扬了前一版的特色。改动主要体现在：

（1）增加了第10章“依赖于机器的优化”。简要介绍现代计算机体系结构、指令调度、基本块调度、全局调度、软件流水、并行性和数据局部性优化。

（2）重写了第9章“独立于机器的优化”。突出数据流分析的理论基础，强调在数据流分析的一般框架下解决各个具体数据流问题，使本章立足于更高层面讨论代码优化。

（3）改写了第5章“类型检查”的前两节。减少一些不必要的概念，把保留的概念区分得更清楚一些。

（4）删掉一些过时的或较少使用的内容。此外，考虑到Pascal语言目前已较少使用，因此，把第1版中采用的大部分Pascal示例代码改为C语言示例代码。

本次改版主要参考了文献[9]，在此向作者表示衷心的感谢。

特别感谢中国科学院计算技术研究所张兆庆研究员和冯晓兵研究员，他们在百忙之中仔细审阅了全稿，并对书稿提出了宝贵的意见。中国科学技术大学为本书的编写提供了充分的资金支持。付雄博士为新增的第10章准备了初稿。在此谨向所有为本次改版提供支持与帮助的人致以最诚挚的谢意。

未加星号的各章已构成编译原理知识的一个基本架构，本科阶段的教学可在此基础上适当地增删，加星号的各章可供更深入学习时使用。与本书配套的习题解答是由作者编写、高等教育出版社出版的《编译原理习题精选与解析》。由作者编写的课程实践教材《编译原理实验教程》将在2008年底由高等教育出版社出版。作者讲授编译原理课程的教学课件和历年试题可在作者的教学网页（http://staff.ustc.edu.cn/~yiyun，http://staff.ustc.edu.cn/~yuzhang）上下载。

限于时间和学识水平，书中难免存在不妥和错误之处。如果发现本书有任何错误或有任何建议，欢迎给作者发送电子邮件（yiyun@ustc.edu.cn，yuzhang@ustc.edu.cn）批评指正，作者将及时改正错误。

作　者

于中国科学技术大学

2008年3月

第1版前言

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，可作为高等院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。

作为一本教材，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

目　录

第 1 章

引　论

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，一个基本问题是：高级语言的程序是怎样变成能在计算机上执行的机器语言程序的？

能够完成从一种语言到另一种语言的变换的软件称为**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，它进行语言变换的特点是目标语言比源语言低级。

本章通过简要描述编译器的各个组成部分以及编译器技术的各种应用来介绍编译这个课题。该课题涉及编程语言、计算机体系结构、形式语言理论、类型论、算法和软件工程等方面的知识。

1.1　编译器概述

编译器的工作可以分成若干阶段，每个阶段的工作都是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。

本节以赋值语句

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

的翻译（假定变量都是实型）为例，简要介绍编译的各个阶段。

1.1.1　词法分析

词法分析阅读构成源程序的字符流，按编程语言的词法规则把它们组成词法**记号**（token）**流**。对于一个词法单元，词法分析产生的记号是

〈记号名，属性值〉

二元组。记号名是同类词法单元共用的名称，而属性值是一个词法单元有别于同类中其他词法单元的特征值。赋值语句（1.1）的字符流在词法分析时被依次组成下面这些记号。

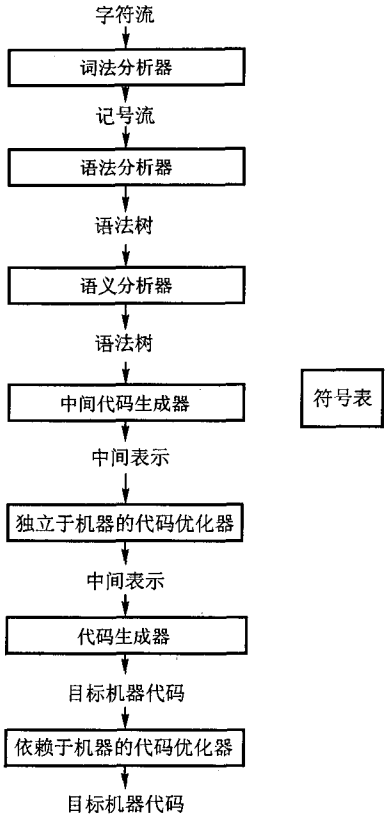


图 1.1 编译的各个阶段

(1) 标识符 position 形成的记号是〈**id**,1〉,其中 **id** 是标识符的总称,1 代表 position 在符号表中的条目,符号表的条目用来存放标识符的各种属性,如它的名字和类型。

(2) 赋值号 = 形成的记号是〈**assign**〉,因为该记号只有一个实例,因此不需要属性值来区分实例。为了直观起见,下面直接用赋值号作为记号名,写成〈 = 〉。

(3) 标识符 initial 形成的记号是〈**id**,2〉。

(4) 加号 + 形成的记号是〈 + 〉。

(5) 标识符 rate 形成的记号是〈**id**,3〉。

(6) 乘号 * 形成的记号是〈 * 〉。

(7) 数 60 形成的记号是〈60〉。从技术上讲,程序中出现的常数也要放到符号表或单独的常数表中,形成记号〈**number**,60 在表中的条目〉。这个问题的讨论留到词法分析中具体介绍。为直观起见,这里直接使用 60 在源程序中的字符序列作为记号名。

分隔记号的空格通常在词法分析时被删去。

于是,赋值语句(1.1)在词法分析后形成的记号流是

〈**id**,1〉〈 = 〉〈**id**,2〉〈 + 〉〈**id**,3〉〈 * 〉〈60〉 (1.2)

第 2 章讨论词法分析,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(syntax analysis)简称为**分析**(parsing),它按编程语言的语法规则检查词法分析输出的记号流是否符合这些规则,并依据这些规则所体现出的该语言的各种语言**构造**(construct,如函数、语句、表达式等)的层次性,用各记号的第一元建成一种树形的中间表示,这个中间表示用抽象语法的方式描绘了该记号流的语法情况。一种典型的中间表示是**语法树**。其中,内部结点表示运算,子结点代表该运算的运算对象。为记号流(1.2)建立的语法树见图 1.2(a)。

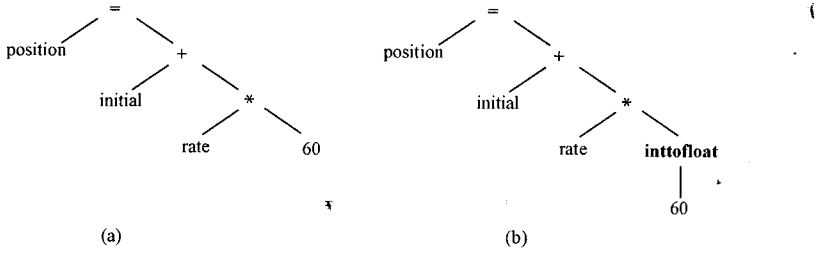


图 1.2 语义分析插入了类型转换

编译器后面各个阶段使用这种语法树进一步分析源程序和生成目标代码。第 3 章专门讨论语法分析算法,图 1.2 的语法树在 4.2 节讨论。在第 4 章的语法制导翻译中,还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。

1.1.3 语义分析

语义分析阶段使用语法树和符号表中的信息,依据语言定义来检查源程序的语义一致性,以保证程序各部分能有意义地结合在一起。它还收集类型信息,把它们保存在符号表或语法树中。

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,许多语言的定义都要求编译器报告错误。语言定义也可能允许运算对象的类型隐式转换,例如当二元算术算符作用于一个整数和一个实数时,编译器会把其中的整数转换为实数。

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 1.2 中,所有的变量都是实型。另外,由 60 本身可知它是整数。对图 1.2(a)进行类型检查会发现 * 作用于实型变量 rate 和整数 60,可以建立一个额外的算符结点 **inttofloat**

[见图 1.2(b)],它显式地把整数转变为实数。 □

类型检查和语义分析在第 5 章讨论。

1.1.4 中间代码生成

经过语法分析和语义分析后,许多编译器为源程序产生更低级的显式中间表示,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:易于产生并且易于翻译成目标程序。

第 7 章主要采用叫做**三地址代码**的中间表示,它们像一种抽象机器的汇编语言,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指令序列组成,每条三地址指令最多有三个操作数,语句(1.1)的三地址代码如下:

```
t1 = inttofloat(60)
t2 = id3 * t1                    (1.3)
t3 = id2 + t2
id1 = t3
```

这种中间形式有它的特点。首先,除了赋值算符外,每条指令至多还有一个算符,因此,在生成这些指令时,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,用以保留每条指令的计算结果。最后,某些三地址指令的运算对象不足三个,例如指令序列(1.3)的第一条和最后一条指令。

本书在第 7 章叙述编译器用的主要是中间表示。通常,除了计算表达式外,这些中间表示还要做其他事情,例如有条件控制转移、无条件控制转移和过程调用。第 4 章和第 7 章提供为编程语言典型构造产生中间代码的一些算法。

1.1.5 代码优化

独立于机器的代码优化阶段试图改进中间代码,以便产生较好的目标代码。通常,较好是指执行较快,但也可能是其他目标,如目标代码较短或目标代码执行时能耗较低。

如果中间代码生成算法比较简单,那么它将给代码优化留下很多机会。例如,一个比较自然的中间代码生成算法为语法树上的每个算符产生一条指令,因而得到(1.3)这样的中间代码。用这样的中间代码生成算法再跟上一个代码优化阶段是一种可行的办法,因为产生较优代码问题可以在代码优化阶段得以解决。例如,代码优化器会推断出,把 60 从整数转变为浮点数可以在编译时完成,从而用 60.0 代替 60 就可以把 **inttofloat** 运算删去。还有,t3 只被引用一次,就是取它的值传给 id1,因此用 id1 代替 t3,把(1.3)的最后一条指令删除也是可以的。这样,优化器可以得到如下结果:

```
t1 = id3 * 60.0
```

```
id1 = id2 + t1                                                    (1.4)
```

不同的编译器完成不同程度的优化，能完成大部分优化的编译器叫做“优化编译器”，但这时编译时间中相当可观的一部分都消耗在这种优化上。简单的优化也可以使目标程序的运行时间大大缩短，而编译速度并没有降低太多。第 9 章和第 10 章分别讨论独立于机器和依赖于机器的优化，第 8 章的代码生成也会涉及代码优化。

1.1.6 代码生成

代码生成取源程序的一种中间表示作为输入并把它映射到一种目标语言。如果目标语言是机器代码，则需要为源程序所用的变量选择寄存器或内存单元，然后把中间指令序列翻译为完成同样任务的机器指令序列。此阶段的一个关键问题是寄存器分配。

例如，使用寄存器 R1 和 R2，(1.4)的中间代码可以翻译成：

```
MOVF   id3,R2
MULF   #60.0,R2
MOVF   id2,R1                                                     (1.5)
ADDF   R2,R1
MOVF   R1,id1
```

每条指令的第一个和第二个操作数分别代表源和目的操作数，每条指令的 F 告知指令处理浮点数。(1.5)的代码把地址 id3 的内容取入寄存器 R2(暂且认为指令中 id3 代表对象 id3 的地址。变量的存储分配在第 6 章讨论)，然后把它乘上实数 60.0，#号代表 60.0 作为立即数处理。第三条指令把 id2 取入寄存器 R1，第四条指令再把寄存器 R2 的值加上去，最后一条指令把寄存器 R1 的值存入地址 id1。这样，该段代码实现了语句(1.1)的赋值。代码生成在第 8 章讨论。

1.1.7 符号表管理

编译器的一项重要工作是记录源程序中使用的变量名字，并收集每个名字的各种属性。这些属性提供该名字有关存储分配、类型和作用域等信息。如果是过程名字，还有参数的个数、每个参数的类型、参数传递方式和返回值类型等。

符号表是为每个变量名字保存一个记录的数据结构，记录的域是该名字的属性。该数据结构应该设计成允许编译器迅速地找到一个名字的记录，并在此记录中迅速地存储和读取数据。符号表将在第 7 章讨论。

语句(1.1)的编译过程总结见图 1.3。