普通高等教育“十一五”国家级规划教材

高等学校规划教材

编译原理和技术

丁文魁　杜淑敏　编著

计算机学科教学计划

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

普通高等教育“十一五”国家级规划教材

高等学校规划教材

编译原理和技术

丁文魁　杜淑敏　编著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

编译原理和技术是计算机专业的基础课程，本书系统介绍了与编译相关的知识。全书共分9章，第1章的编译概述讨论了编译程序的各个组成部分；第2章介绍有关形式语言的一些基本概念；第3章介绍词法分析，它是整个分析过程的一个子任务；第4章详细讨论了用于编译程序构造中的一些典型的语法分析方法；第5章引入了语法制导定义和翻译模式这两个概念，并给出了如何书写L-属性的翻译模式的方法；第6章讨论用来支持一个程序的运行时刻环境的有关存储组织的各种问题；第7章讨论的是中间代码生成；第8章介绍目标代码生成；第9章集中讨论中间代码优化。

为方便教师讲解和学生实验，本书配有光盘。

本书适合作为计算机专业编译原理课程的教材，也可作为广大工程技术人员的参考资料。

图书在版编目 (CIP) 数据

编译原理和技术 / 丁文魁，杜淑敏编著. —北京：电子工业出版社，2008.3

高等学校规划教材

ISBN 978-7-121-06060-1

Ⅰ. 编… Ⅱ. ①丁…②杜… 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 021743 号

责任编辑：许菊芳

印　　刷：

装　　订：北京市李史山胶印厂

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱　邮编　100036

开　　本：787×1092　1/16　印张：19.75　字数：506 千字

印　　次：2008 年 3 月第 1 次印刷

定　　价：31.00 元　(含光盘 1 张)

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010)88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：(010)88258888。

前　言

本书是“十一五”国家级规划教材，为方便读者使用，我们把它编写成书和附带光盘的形式出版，称为“编译原理和技术”。其中，书是基础，光盘中的演示是为读者更快更好地掌握书中陈述的原理和算法而设计的。编译技术是计算机专业的基础课程，我们希望它能成为受学生们欢迎的教材，也希望它成为从事计算机工作的广大工程技术人员欢迎的参考资料。

全书共分9章。第1章的编译概述讨论了编译程序的各个组成部分，从中读者可以了解到编译程序构造的各主要方面。第2章介绍有关形式语言的一些基本概念，这些是学习本书必要的基础知识。第3章介绍词法分析，它是整个分析过程的一个子任务。设计词法分析程序的理论基础是有限自动机。文中详细讨论了正规表达式和有限自动机的等价转换算法。第4章详细讨论了用于编译程序构造中的一些典型的语法分析方法。第5章的语法制导翻译中，为使文法的产生式和语义规则联系起来，引进了语法制导定义和翻译模式这两个概念，并给出如何书写 L-属性的翻译模式的方法。第6章讨论用来支持一个程序的运行时刻环境的有关存储组织的各种问题。之后，在第7章利用语法制导定义和翻译模式这两个概念来讨论中间代码生成。第8章在上一章的基础上介绍目标代码生成。最后，第9章集中讨论中间代码优化，比较详细地阐述了数据流分析和主要的全局优化方法。

最近，有些人提出大部分计算机专业的学生毕业后并不从事编译器的设计和开发工作，因此可以不学习这门课程。我们认为，编译程序构造是计算机科学最早获得成功的分支之一，编译技术不仅用于构造编译程序，而且可以应用它的思想和技术设计一般的软件。例如，文本编辑、读入结构化数据、新格式的快速引进以及一般的文件转换，等等。因此，编译技术是成长为一名优秀程序员所必备的背景知识。

编译原理和技术是重要的，但对于许多学生来说掌握它又显得有些困难。为帮助读者克服困难，我们为本书制作了辅助光盘，内容如下：

学习指导：把每章的知识分成三个层次：熟练掌握，掌握，了解。给读者指出学习难点，向读者建议学习流程。

演示：对分析模型、基本原理和算法用 Java 制作了大量的演示，有许多是交互式的。这些演示有助于提高学生的兴趣和求知欲，加深对课程内容的理解。

课堂：观看用 PowerPoint 制作的 PPT 幻灯片，它们是教师对学生讲解的课程内容。教师也可以用它做素材，进一步制作自己讲课用的 PPT。

实例：分析了一个小的编译程序的设计与实现，它有助于理解基本原理和算法。

习题：做习题对学习者来说是非常重要的，我们给出的答案是参考性的，且是为自学者准备的。

在本书的编写过程中，北京大学计算机系的同事和同学们提出了宝贵意见，特别是先后有许多同学参加了演示的设计和制作。在最后成稿的过程中，徐重远同学给予了很大帮助。没有他们的劳动，也就没有此教材。另外，在申请“十一五”规划教材和本教材的出版过程中，电子工业出版社的同志们付出了辛勤的劳动，在此一并表示感谢。

丁文魁　杜淑敏
北京大学计算机系
2008 年 2 月

目　录

第1章 编译概述

编译是指将一个用源语言书写的程序转换成一个等价的用目标语言书写的程序。一般而言，源语言是面向人的，而目标语言是面向计算机的。

1.1 翻译和解释

世界上存在着多种语言，人们为了通信方便，就需要建立各种语言之间的翻译。人与计算机之间的信息交流，同样存在一个翻译问题。每种计算机都有自己独特的指令系统，亦即这种机器的机器语言。人们虽然可以直接用机器语言编写程序，但很不方便，这是因为机器语言程序不易读、不易写、结构性很差。另一种方法是人们先用较为接近自然语言的高级程序设计语言（或简称高级语言）来编写程序，再借助特定的软件将它翻译成机器语言程序。读者所熟知的 FORTRAN，Pascal，LISP，C，C++等等都是高级语言。而机器语言和面向硬件的语言称为低级语言。各种汇编语言是面向硬件的语言，属于低级语言。

翻译程序是一个把源程序翻译成等价的目标程序的程序。即翻译程序是这样一种程序，它接受源程序作为输入，并产生与源程序等价的目标程序作为输出，见图 1.1。源程序是用源语言编写的，而目标程序是由目标语言书写的。例如，若要将语言 A 的一个程序翻译到语言 B 的一个程序，则称语言 A 是翻译的源语言，而语言 B 是翻译的目标语言。

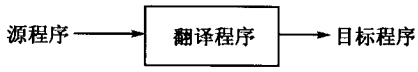

图 1.1 翻译程序

如果源语言是高级语言，而目标语言是低级语言，那么我们称这样的翻译程序为编译程序。因为程序是用来描述算法的，所以可以说，编译是指将一个用面向人的源语言表示的算法转换到一个等价的用面向硬件的目标语言表示的算法。

在编译时，把源程序翻译为目标程序；在运行时，执行这个目标程序产生计算结果。

汇编程序是这样一种程序，它把用汇编语言编写的源程序翻译为某计算机的机器语言程序。

一个源语言的解释程序是这样的程序，它以源程序作为输入，但不产生目标程序，而是按照源程序中的控制流程，或者说按照源程序中的语句的动态执行顺序，对当前该执行的语句逐条进行分析解释。进一步说，根据源语言的语义直接把当前该执行的语句转换成加工数据的操作或所需完成的功能，并直接转向相应的函数去完成这些操作和相应的功能。

如果我们对翻译程序和解释程序进行一下比较的话，就可以看到，翻译程序是以源程序的实际输入的顺序来处理程序的，并从而得到将要被执行的目标程序。而对于一个纯粹的解释程序来说，它是按照控制该源程序的逻辑流程进行工作的。源程序的每个语句一经解释就立即执行，即所谓的解释执行。于是，一个纯解释程序可能对某一源程序语句（例如循环

部分的语句）反复解释执行若干次。显然，这种方法的效率很低。因此，通常的解释程序是先将源程序比较简单地翻译成某种中间形式的程序，然后再对这种中间形式进行解释。解释程序的优点是与用户通信方便，用正文形式存储源程序也比较节省内存空间。在本书中我们不打算对解释程序进行专门的讨论，然而，感兴趣的读者不难看到，许多用于编译程序的构造技术同样也适用于解释程序。

1.2 编译程序的组成部分

大家都知道，如果我们要将一个英语句子，譬如 “I wish you success” 翻译成汉语句子的话，那就首先要对句子中出现的单词进行词法分析。构成这个句子的单词有“I”，“wish”，“you”和“success”。若从字典上查找一下即可知道：“I”是代词，“wish”是动词，“you”是代词，“success”是名词。进一步要问，这几个单词是否组成了一个合乎英语语法的句子呢？这就需要进行语法分析。因为“I”是代词，它可以做主语；“wish”是动词，可以做谓语；“you”是代词，可以做间接宾语；“success”是名词，可以做直接宾语。于是我们知道这是一个合乎英语语法的句子。为了翻译，还需要进行语义分析。这时我们注意到，“I”的含义是“我”；“wish”的含义是“希望”、“意欲”、“但愿”、“祝”等；“you”的含义是“你”、“你们”等；“success”的含义是“成功”、“成就”等。这样，就可以知道此句子的含义是“我希望你成功”。然而我们还希望把汉语句子修饰得更好一些，还需要根据上下文的关系以及汉语语法的有关规则，进行综合考虑。在这之后，我们很可能得到的最后翻译是：“祝你成功”。

与此类似，编译程序将首先根据源语言的定义来对源程序进行分析，之后进行综合并从而得到与源程序等价的目标程序。

一般说来，任何编译过程都要进行以下两个方面的工作：

- **分析** 对源程序进行结构分析和语义分析。
- **综合** 生成与源程序等价的目标程序。

观看演示

编译程序的各个组成部分。

图 1.2 显示了编译的工作流程。每个阶段的工作都是由相应的程序完成的，因此，从这个图我们也可以看出，逻辑上一个编译程序是由哪些部分组成的。下面让我们简要地介绍一下编译程序的各个组成部分。

1.2.1 分析

分析阶段的任务是根据源语言的定义，对源程序进行结构分析和语义分析，从而把源程序正文转换为某种内部表示。这种内部表示有多种形式，树形结构就是通常采用的形式之一。

结构分析程序包括词法分析程序和语法分析程序，用来分析源程序的静态结构。

词法分析程序把构成源程序的字符序列（字符串）转换为语义上关联的单词的符号序列（单词符号串）。一般说来，源程序中出现的单词可分为两类，一类是特殊的单词，如

关键字、运算符、分界符等，这些都是源语言所提供的；另一类是普通单词，如用户在源程序中定义的标识符、常数、标号等。词法分析程序所输出的单词符号常采用以下二元式表示：

（单词词类表示，单词自身的值）

例如某源程序语句(1.1)中，设 sum, first 和 count 是实型变量，经词法分析后输出的单词序列如(1.2)所示。

$$\text{sum} := \text{first} + \text{count} * 10; \tag{1.1}$$

(id, sum), (赋值号, :=), (id, first), (加号, +), (id, count), (乘号, *), (整数, 10), (分号, ;)　　(1.2)

注：id 是标识符类单词的词类表示。

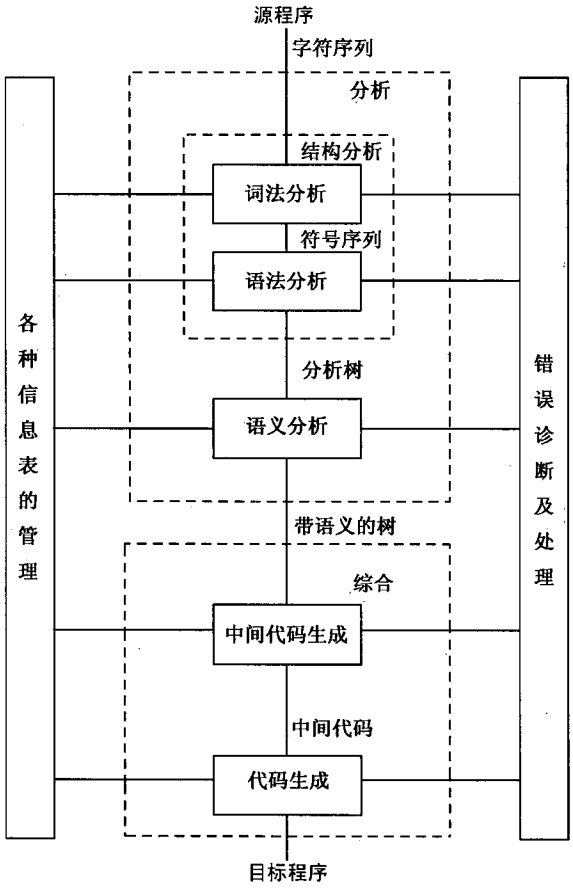

图 1.2　编译程序的工作流程

语法分析程序将词法分析程序输出的单词看做基本单位，对单词符号串进行语法分析（根据文法的产生式进行推导或归约），输出源程序的分析树，或分析树的变形。例如，语句(1.1)经语法分析构造的分析树如图1.3 所示，但请读者注意，语法分析程序可能不显式地输出它。

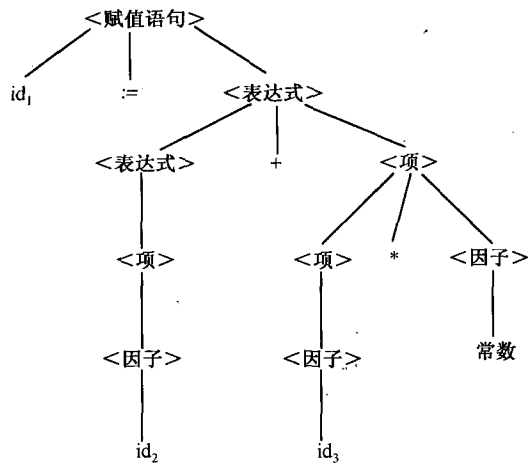

图 1.3 语句(1.1)的分析树

语义分析程序用于确定源程序的含义。在语义分析过程中，要检查源程序有无语义错误，为代码生成阶段收集类型信息。例如，语义分析的一个工作是进行类型检查，检查每个运算符是否具有语言规定允许的运算对象。在表达式 sum := first + count * 10 中，*的两个运算对象：count 是实型，10 是整型，需要在分析树上增加一个语义处理结点 inttoreal，把整型 10 转换成实型 10.0。经语义分析后，语法分析生成的分析树的某些结点将被增添有关源程序的语义信息。这些信息进一步完善了源程序的树形表示。

1.2.2 综合

综合阶段是从分析阶段得到的带语义的树形式或其他源程序的内部形式出发，并根据所制定的源语言到目标语言的对应关系，进行综合加工，从而得到与源程序等价的目标程序。此任务也可以分为两部分：中间代码生成部分和代码生成部分。

中间代码生成程序将从源程序或由源程序产生的树形式出发生成中间代码。我们可以把中间代码看成是某个抽象机的一个程序。中间代码表示应当是易于生成的，并且是易于翻译到目标程序的。中间代码表示若采用三地址代码形式，从语句(1.1)生成的中间代码如(1.3)所示。

$$\begin{aligned} &t_1 := \text{inttoreal } 10; \\ &t_2 := id_3 * t_1; \\ &t_3 := id_2 + t_2; \\ &id_1 := t_3; \end{aligned} \tag{1.3}$$

其中，t_i（$i = 1, 2, 3$）是编译程序生成的临时变量，用于存放运算的中间结果。

为减少目标程序的运行时间及较少使用存储空间，这一部分常常进行中间代码的优化。中间代码(1.3)经优化后转换成(1.4)：

$$\begin{aligned} &t_1 := id_3 * 10.0; \\ &id_1 := id_2 + t_1; \end{aligned} \tag{1.4}$$

代码生成程序将中间代码变换成特定机器上的代码，它或者是绝对指令代码，或者是可重定位的指令代码，或者是汇编指令代码。这是编译的最后阶段，它的工作与硬件系统的结构和指令密切相关。例如，假设目标计算机有两个寄存器 R1 和 R2，根据(1.4)可生成某汇编代码(1.5)：

```
MOVF    id3, R2
MULF    #10.0, R2
MOVF    id2, R1                    (1.5)
ADDF    R2, R1
MOVF    R1, id1
```

这里用#表明 10.0 为实常数。

1.2.3 表格管理

编译程序需要收集所有出现于源程序中的数据实体的有关信息。例如，编译程序应该知道一个变量的类型，一个数组的大小，一个函数的参数个数及其返回类型，等等。这些信息一般可以从源程序正文中获得，因此在概念上它们是程序正文的一部分。但是这些信息有它们特殊的存取特性。编译程序从头到尾读源程序，边读边分析。编译程序在分析声明时，必须把有用信息记录下来，否则，在分析执行部分使用这些信息时将找不到它们。也就是说，编译程序再反复地从源程序中找到它们是非常困难的。例如，在程序的声明部分，定义了一个变量，此时，编译程序要把它的名字、类型、层次等信息记录到符号表中；在程序的执行部分，使用这个变量时要查找符号表，读取它的相关信息。在编译过程中查找符号表是很频繁的，通常占用整个编译时间中的相当大一部分，因此，设计有效的符号表是相当重要的。

如图 1.2 所示，编译程序在各个阶段中都要从信息表中存取信息。因而合理的设计和使用表格是编译程序构造的一个重要问题。

1.2.4 错误处理

编译过程中任何时刻都可能发现源程序的错误，例如：

- 词法分析程序可能发现某单词的拼写错误。
- 语法分析程序可能发现语法错误，如括号不配对。
- 语义分析程序可能发现语义错误，如某运算符与其运算对象的类型不相容。
- 代码生成程序可能发现目标程序区超出了可允许的范围。
- 其他由于计算机的容量有限，编译程序的处理能力也受到限制而引起的错误，例如符号表中表项太多而超出了符号表的容量。

因此错误检查和处理是编译程序的一项很重要的任务。编译程序在某一阶段发现错误之后，就将错误报告给错误处理程序。错误处理程序产生合适的诊断信息，以帮助程序员决定错误出现的准确位置，并且对源程序进行适当的恢复，以保证程序的一致性。当然要尽量使得这样的恢复与程序原来的意图一致，并从而使得编译过程能继续进行下去。这样，就可能在一次编译中检查出尽可能多的错误，使程序员减少编译的次数。

1.3 编译程序的组织

图 1.2 给出了编译程序的各个组成部分及其相互之间的逻辑关系，那么，如何组织编译程序呢？它通常涉及如下几个方面。

首先，编译程序是按“遍”工作的。一般而言，一“遍”是指对程序正文的某种形式以正向（或反向）进行一次顺序地扫描，在扫描的过程中完成本遍的编译任务，结果或者把源程序的一种表示转换为另一种表示，或者在一种内部表示内完成某些变换。每一遍的工作从前一遍获得的工作结果开始（对于第一遍而言，是从源程序开始）。当一遍工作完成之后，它所占用的存储空间大部分被释放。下一遍进入后，即可使用这些被释放的存储空间。至于一个编译程序究竟应分成几遍，如何划分，与所面临的源语言、目标机器、编译程序的设计目标以及人员组成等诸多因素有关。遍数多一点有个好处，即整个编译程序的逻辑结构可能清晰一些，但势必增加编译时间。

其次，根据使用编译程序的环境可能对设计的编译程序有各种不同的目标要求，例如：

- 目标程序效率高
- 目标程序短
- 编译的时间短
- 编译程序小
- 具有好的错误诊断和恢复能力
- 可靠性好

遗憾的是，这些目标在一定程度上相互冲突，因此如同许多大的软件一样，必须采取折中的方法。例如，如果希望编译程序产生高效的代码，那么编译所占用的时间就要长一些。

由于这些目标不可能同时满足，因此有些机器上为同一语言配备了多种编译程序。比如一台计算机上，可以有某种语言的两个编译程序 A 和 B，A 编译用的时间短但产生的代码的运行效率低，B 则与 A 相反，即编译用的时间长但产生的代码的运行效率高。用户开发程序时，可以先用 A 进行调试，调试结束后再用 B 重新编译，产生用于今后运行的目标代码。

最后，编译的工作可以分成前端和后端。前端包括词法分析、语法分析、语义分析、中间代码生成以及中间代码优化；后端包括面向目标机器的优化和目标代码生成。显然，前端仅依赖于源语言，而后端仅依赖于目标机器。

本书讨论的编译技术，适用于常用的块结构程序设计语言，例如 FORTRAN，Pascal，C，等等。至于面向对象的语言、函数式语言以及逻辑式语言等，还需要专门的技术处理这些语言特有的语言设施。在代码生成中，仅考虑普通的计算机，而对于高性能的各种计算机结构，例如，多处理器系统，也超出了本书的教学范围。

就一般的程序设计语言，我们可以把词法分析、语法分析、语义分析、中间代码生成以及相应的表格管理和出错处理组织在一遍扫描中完成，如图 1.4 所示。

实现一个编译程序，或者用某种程序语言书写，或者用软件工具构造。早先，编译程序大多用程序语言书写，这种方法的不足之处是工作量太大，开发周期长。后来，各种自动生成工具逐渐诞生，例如 Lex（Lexical analyzer generator）和 YACC（Yet Another Compiler