高等学校计算机专业规划教材

编译原理

侯文永 张冬茉 编著 陶树平 主审

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

高等学校计算机专业规划教材

编译原理

侯文永　张冬茉　编著

陶树平　主审

電子工業出版社

Publishing House of Electronics Industry

北京 · BEIJING

内 容 简 介

编译原理是计算机专业的一门重要专业课，本书旨在介绍编译程序构造的一般原理和基本方法。内容包括语言和文法、词法分析、语法分析、语法制导翻译、中间代码生成、存储管理、代码优化和目标代码生成。本书较系统地介绍了经典的、广泛应用的技术，特别注重词法分析器、语法分析器的自动生成，以及语法制导的翻译方法和以控制流分析与数据流分析为基础的代码优化，并概要介绍了属性文法和并行编译。各章之后附有习题，其中包括要求用 C 语言实现相应的分析器、翻译器、优化器、代码生成器的习题。

本书可作为高等院校计算机科学专业的教材，也可作为教师、研究生、软件技术人员的参考书。

图书在版编目(CIP)数据

编译原理/侯文永，张冬茉编著. —北京：电子工业出版社，2002.8
高等学校计算机专业规划教材
ISBN 7-5053-7950-X

Ⅰ. 编… Ⅱ. ①侯… 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(2002)第 063422 号

责任编辑：徐晓光 洪国芬
印　　刷：北京天宇星印刷厂
出版发行：电子工业出版社 http://www.phei.com.cn
　　北京市海淀区万寿路 173 信箱 邮编 100036
经　　销：各地新华书店
开　　本：787×1 092 1/16 印张：13 字数：333 千字
版　　次：2002 年 8 月第 1 版 2002 年 8 月第 1 次印刷
印　　数：6 000 册 定价：17.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。
联系电话：(010)68279077

前　言

编译原理是计算机专业设置的一门重要的专业课程。虽然只有少数人从事编译方面的工作，但是这门课在理论、技术、方法上都对学生提供了系统而有效的训练，有利于提高软件人员的素质和能力。

为了用有限的篇幅较深入地讨论编译的主要内容，本书尽可能用较简洁的方式描述这些内容。本书共9章，第1章引论，概述了编译、编译结构及编译的构造方法；第2章文法和语言，简要介绍了必要的基础知识；第3章词法分析，介绍词法分析器及其涉及的一些问题，同时描述了正规式和有限自动机以及它们之间的等价变换，从而引出了词法分析的自动生成的原理；第4章语法分析，以较大的篇幅介绍了几种典型的语法分析方法，特别是表驱动的分析方法，从而引出了语法分析器自动生成的原理；第5章语法制导翻译和中间代码生成，介绍了一些主要语句的翻译方案所涉及的问题，并描述了这些翻译方案，还简要介绍了属性文法；第6章运行时存储空间管理，介绍了几种存储分配的策略及涉及的问题，对参数传递及符号表也做了简要的介绍；第7章代码优化，在控制流分析和数据流分析的基础上，介绍了中间代码的局部优化和循环优化；第8章代码生成，在充分利用寄存器的前提下，介绍了如何生成有效的目标代码及几种目标代码优化技术；第9章概要介绍并行编译涉及的一些基础知识。

读者在学习编译原理时应努力在头脑中构造一台虚拟机，编译所涉及的各个环节都能在虚拟机中运作。完成各章后的习题对理解所涉及的内容是有益的。就某个虚拟语言（如PL/0）构造它的编译程序，对掌握编译技术是十分重要的。

陶树平教授审阅了全稿，并提出了许多宝贵意见，在此表示诚挚感谢。但由于编者限于水平，书中难免出现错误与不妥之处，敬请批评指正。

侯文永　张冬茉
上海交通大学计算机系
2002年5月20日

目　录

第1章 引 论

编译程序是什么？它与翻译程序、解释程序、汇编程序之间有什么区别？如何来设计、构造或了解一个编译程序？本章将就上述问题做出初步的讨论。

1.1 编译程序是一种特定的翻译程序

描述“计算”（computing）过程通常借助于一种程序设计语言，这种“计算”过程包含了从初始状态转变为终止状态的一系列的操作。描述算法的语言可有很多种，将一种语言写的程序转换成另一种语言写的程序，这就是翻译（translation），实现这种功能的程序便是翻译程序(translator)，显然翻译前的程序与翻译后的程序应等价。翻译前的程序为源语言程序，简称源程序(source code)，翻译后的程序为目标语言程序，简称目标程序(object code)。如将 PL/1 或 COBOL 语言写的程序转换成 C 语言程序，便属翻译之列，只是构造这样的翻译程序十分困难。

事实上，人们几乎都用面向用户的高级语言来描述他们的“计算”，除了理论上的尝试以外，几乎所有的计算机都无法直接执行高级语言写的程序，而只能执行面向机器的机器语言程序。所以必须先将高级语言程序翻译成低级语言程序，这种翻译程序便是编译程序(compiler)。显然编译程序是翻译程序中的一类。

如果把源程序记为 S，目标程序记为 T，编译程序记为 C，那么可以将编译看成一个函数，一种映射：T=C(S)。

人们常将可直接执行的机器语言的指令系统符号化，这便是汇编语言，当然汇编语言程序也必须转换成机器语言程序才能执行，这种转换程序便是汇编程序(assembly program)。汇编程序也是一种翻译程序，由于符号化的指令系统与机器指令系统之间有比较明确的对应关系，因此实现它们之间变换的汇编程序就比较容易构造，而且由于汇编语言和机器语言一样都是面向机器的，故相对于面向用户的高级语言而言，将它们都称为低级语言，编译程序就是将高级语言写的源程序转换成低级语言写的目标程序的翻译程序。

一个高级语言程序的执行通常分成两个阶段，即编译阶段和运行阶段。编译阶段将源程序变换成目标程序，运行阶段是由编译阶段生成的目标程序连同运行系统（数据空间分配子程序、标准函数程序等）接受程序的初始数据作为输入，输出“计算”结果，如图 1.1 所示。

如果编译生成的目标程序是汇编语言写的程序T′，那么在编译与运行阶段中间，还得增加一个汇编阶段，它将编译生成的汇编语言程序T′经汇编程序 A 变换成以机器语言写的目标程序 T，如图 1.2 所示。

程序除了编译执行外，还可以有另一种执行方式即解释执行。解释执行是将源程序中的语句按动态顺序，逐句逐段翻译成可执行代码，一旦具备执行条件(获得必要的初始数据等)，立即将这一段代码执行得到部分结果。即解释执行是按动态顺序步进式推进着翻译和执

行。完成这样功能的程序称为解释程序(interpreter)，可以用 I 来记它，程序的解释执行如图 1.3 所示。

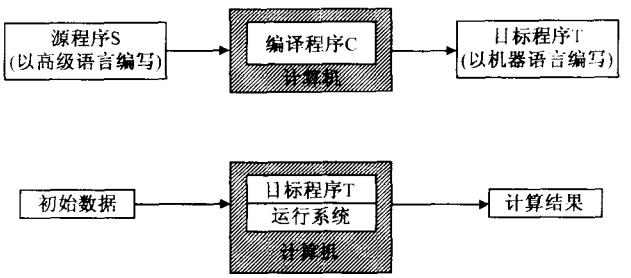


图 1.1　程序的编译执行

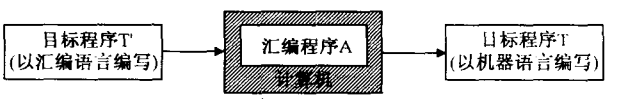


图 1.2　汇编阶段

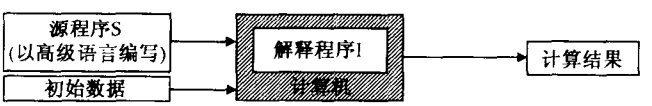


图 1.3　程序的解释执行

解释执行特别适合于交互式语言，命令语言也用于抽象机语言，但通常情况下执行效率不如编译执行高，因为编译产生整个程序的目标代码，就可以在全局范围做压缩空间缩短运行时间的优化工作；同时编译是按程序的静态顺序线性扫描源程序的，源程序中重复执行的成分只是运行阶段的重复，编译阶段并不因此重复，而解释执行则是按动态顺序重复翻译、重复执行的，速度自然会慢得多。

1.2　编译程序的结构

编译程序将源程序变换成目标程序，这是个复杂的过程，通常分成五个阶段。

1.2.1　词法分析阶段

对组成源程序的字符串进行扫描和识别，识别出一个个具有独立意义的单词(或称符号)，如基本字(if, for, begin 等)、标识符、常数、运算符、界符(括号、=、；等)，将识别出的单词用统一长度的标准形式(也可称为内部码)来表示。对以后的变换来说，这种标准形式对区分单词和单词的属性应是十分方便的。因此词法分析是将字符串变换成单词符号流。

1.2.2　语法分析阶段

在词法分析输出的单词流基础上，根据源语言的语法规则分析这种单词流是否正确地组成了各类语法单位，如短语、子句、程序段和程序等。例如 2*3.1416*R*R 是表达式，单词符号:=后应跟着一个表达式作为赋值语句的右部等。

1.2.3 语义分析、中间代码生成阶段

经过语法分析的单词流若在语法结构上是正确的，就可以在这个基础上做实质性的翻译工作，虽然可以直接翻译成可执行代码(机器语言程序)，但通常是将单词流翻译成在语义上等价的中间语言程序。中间语言是介于高级语言和低级语言之间，但并不面向具体机器，语言结构又十分接近低级语言的一种中间代码形式。中间语言相对于低级语言而言，既有一定程度的抽象，又有一定程度的对应。中间语言程序到目标语言的转换是容易的，因此语义上的等价变换主要在语义分析阶段进行。其任务是根据语言的语义规则，将语法单位逐一翻译成中间语言代码，这通常称为语法制导翻译。

1.2.4 优化阶段

前一阶段产生的中间代码是以语法单位这样的局部区间为单位而产生的，不能保证很高的效率，有必要进行等价变换，使程序占用空间少，运行时间短。常用的优化措施有删除冗余运算、删除无用赋值、合并已知量、循环优化等。有些优化措施效果很明显，例如循环中参与运算的运算量，如其值并不随着循环而发生变化，这类运算称为循环不变运算，它完全不必每次循环都计算一次，将它们提到进入循环前计算一次即可。

1.2.5 目标代码生成阶段

将中间代码转换成机器语言程序或汇编语言程序，最后完成了翻译，这一阶段的工作因为目标语言的关系而十分依赖于硬件系统。如何充分利用寄存器、合理选择指令、生成尽可能短而有效的目标代码，都与目标机的结构有关。

生成的目标代码如果是汇编语言程序，则需经由汇编程序汇编后才能执行。生成的目标代码如果是绝对指令代码，则可直接投入运行。如果是可重定位的指令代码，那么目标代码只是一个代码模块，必须由连接装配程序将输入/输出模块、标准函数等系统模块与目标代码模块连接在一起，确定数据对象和各程序点的位置(即代真)才可能形成一个绝对指令代码程序供运行。

以上五个阶段反映了编译程序的动态特征，但编译的实现还有赖于符号表管理和出错管理。

1.2.6 符号表管理

源程序中的有关数据对象的信息和程序信息是编译各阶段经常使用的，因此编译程序中还应包括一个符号表管理程序，它涉及到符号表的构造、查询和更新等各种操作，提供用户命名的标识符的各种属性、存储位置、类型、作用域等信息。对这些信息的操作是频繁的，占了编译时间很大的比例，因此符号表的结构、符号表上各种操作的算法必须精心设计。

1.2.7 出错管理程序

编译的各个阶段都可能遇到错误。出错管理程序应在发现错误后，将错误的有关信息如错误类型、错误地点向用户报告。为了尽可能多地发现错误，在发现了错误后还应继续编译，因此要设法将错误造成的影响限制在尽可能小的范围内。发现错误后能自动校正错误当然最好，只是在大部分情况下，校正的代价太大，而且效果也未必尽如人意。因此编译程序

的总框图可用图 1.4 来表示。

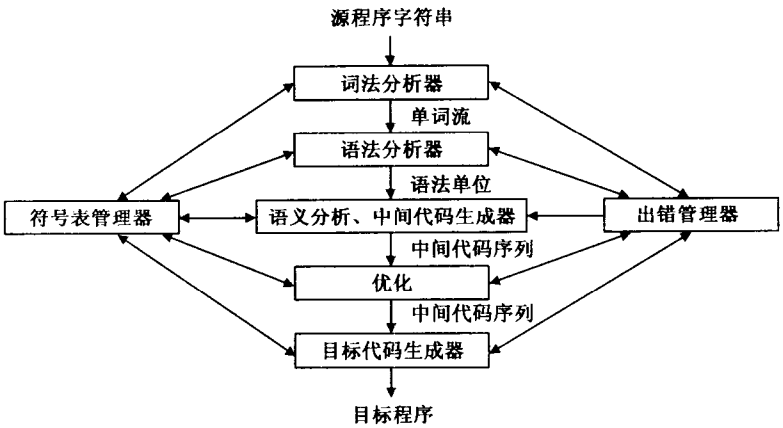


图 1.4　编译程序总框图

其中优化是可选部件，对于像学生的作业这类量不大、不会反复运行的程序来说，可以不经过优化阶段，即应该在优化所需付出的代价和优化能取得的效益之间做出选择。

1.2.8　编译阶段的前端和后端

编译的前三个阶段(词法分析、语法分析、语义分析和中间代码生成)称为编译的前端；它其实还可分成分析(词法分析、语法分析、语义分析)和综合(代码生成)两个过程。而目标代码生成则是编译的后端。中间代码的优化可归入前端。很明显编译后端是面向目标语言的，而编译前端则不是，它几乎独立于目标语言。

引进中间语言生成中间代码，不仅使编译程序的逻辑结构更合理，有利于开展中间代码级的优化，而且对编译的生成和移植更有利。将源语言 S 的程序转换成目标语言 T 的编译程序 C，如果想移植到目标语言为 T'的系统上去，那么只要重写编译 C 的后端，使之生成 T'的目标代码即可。同样如果源语言为 S'，目标语言仍为 T，则它们的间编译程序只要改写 C 的前端，使之面向 S'即可。

因此从形式上说，如果不设中间代码，不将编译分成前端 F 和后端 B，那么 m 个高级语言和 n 种目标语言之间需要 $m\times n$ 个编译程序 C。反之，只需要 m 个前端和 n 个后端即可，故工作量为 $m\times F+n\times B$。但是要选择适合 m 个高级语言和 n 个目标语言的中间语言并不容易，因为一种通用语言往往很难表示一种特定语言的微妙特性。

1.2.9　遍

在编译过程中，从头至尾地扫描一个输入文件，经过程序变换形成一个输出文件，这个过程称为编译的一遍。

除了符号表管理程序和出错管理程序外，编译的其他五个阶段都可以单独组成一遍。如果将词法分析作为一个子程序，语法分析器每当需要时调用词法分析程序获得一个单词，每当按语法规则发现形成一个短语(句柄)时，便调用相应的语义子程序做翻译工作，生成中间代码，那么将词法分析、语法分析、语义分析及中间代码生成组成一遍是十分自然的。因此编译过程一般可用两遍或三遍完成。

编译程序任务繁重，结构复杂，占用空间也很大，因此将编译过程分割成几遍来完成，不仅使编译程序结构清晰，而且每遍只完成一部分工作，对资源需求也可降低要求。

将编译分割成几遍，除了上述原因外，当然还有逻辑上的需要，也有优化上的需要，但最初的根本原因是内存的不够使用，曾有一内存仅 4 KB 的机器，将近 4 万条指令的编译程序分成 19 遍，每遍占用内存仅 2 KB 左右，这在内存价格昂贵的当时，可以说是编译分遍的一个杰作。

并不是说编译程序扫描遍数越多越好，因为从头到尾的每遍扫描使编译开销上升很多，有一些技术就是专门为了减少遍数的，如中间代码生成中的回填技术。

1.3 编译程序的生成

编译程序是一个足够复杂的程序，语言功能的完善，硬件结构的发展，环境的友好要求，都对编译程序提出了更多更高的要求。因此一个编译系统的构造并非易事，对完全想用手工方法来构造编译器来说更是如此。好在现在有各个层次的生成工具，如 lex、yacc、GAG、CGSS 等，而环境也提供开发工具来帮助产生高效的编译器，如 make 等，只要熟练运用这些工具，构成编译器的难度与强度都可大为降低。在以后章节中将把 lex、yacc 等工具的自动生成原理作为重点内容予以介绍。其实，现在要构造一个完全独立的全新的编译器的可能性很小，大部分可在现有编译器的基础上扩展，有的采用自展的方式，有的则用移植的方式。不管是用手工方法、自动生成方法、自展和移植方法或者兼而有之的综合方法，构造者对源语言的语法结构和语义，对目标语言及其面向的目标机的系统结构与操作系统功能应有准确的理解。

1.3.1 自展

一个编译程序一般涉及三种语言，因为一个编译程序本身也是一个程序(这个程序的功能是将 S 语言的程序等价变换成 T 语言的程序)，它也必须以一种语言来书写，它可以是既不同于 S，又不同于 T 的语言 I，称它为编译的实现语言。因此一个编译程序 C 的完整表示为 $C_I^{S,T}$，或者用图形来表示，因为它的形状像字母 T，故称为 T 图，如图 1.5 所示。请注意 T 图上方的 S，T 是这个程序具有的功能(将 S 语言的程序等价变换为 T 语言的程序)，T 图下方的 I 是书写这个程序所用的语言。只是除了构造编译程序外，通常使用的编译程序都是 $C_A^{S,A}$ 形式的，即 A 机器上的高级语言 S 的编译程序。

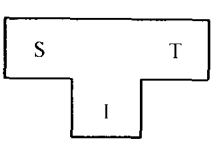
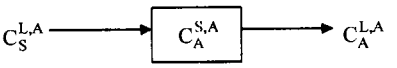


图 1.5 T 图

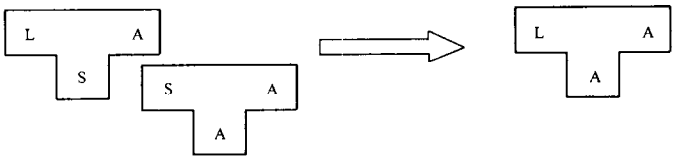
不言而喻，对一个相当规模的语言 L，想用目标语言在目标机 A 上构造一个编译程序 $C_A^{L,A}$ 是十分困难的。为此，首先可以选择 L 的一个子集 S，子集的规模只要达到具有足够的描述能力即可。相对于 L 而言，S 的编译程序 $C_A^{S,A}$ 就容易写得多，不妨认为它可以用手工完成。

第二步，可以用 S 语言来写一个 L→A 的编译程序 $C_A^{L,A}$，相对于 A 语言来说，用 S 语言写这样的编译程序也会容易得多。然后将这个 S 语言的程序 $C_S^{L,A}$ 作为 S→A 的编译程序的输入。其输出当然是 A 语言的程序，并且由于 $C_A^{S,A}$ 保证这种变换的等价性，因此输出程序将保持输入程序 $C_A^{L,A}$ 的功能，即将 L 语言程序等价变换成 A 语言程序，故输出程序是一个 A

语言书写的程序，其功能为将 L 语言程序等价变换成 A 语言的程序，即 L→A 的编译程序 $C_A^{L,A}$，如图 1.6(a)所示。用 T 图表示则如图 1.6(b)所示。



(a) 自展



(b) 自展的 T 图

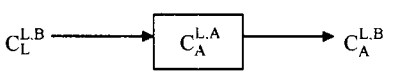
图 1.6 自展及 T 图

上述构造 $C_A^{L,A}$ 的过程便是编译的自展，或称为自编译。S 为核，如果 L 相当大，则可以设 $S \subset S_1 \subset S_2 \cdots \subset L$，通过多次自展逐步滚大。

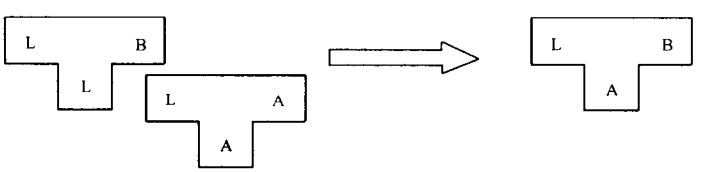
1.3.2 移植

已有 $C_A^{L,A}$，借助于 $C_A^{L,A}$ 得到 $C_B^{L,B}$，这便是移植。即将 A 机器上的 L 语言的编译移植到 B 机器上。

做法是首先用 L 语言写一个 L→B 的编译程序 $C_L^{L,B}$，将这个 L 语言程序作为 $C_A^{L,A}$ 的输入，输出为 $C_A^{L,B}$，如图 1.7(a)所示。用 T 图表示则如图 1.7(b)所示。



(a) 移植第一步

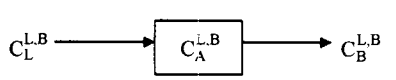


(b) 移植第一步的 T 图

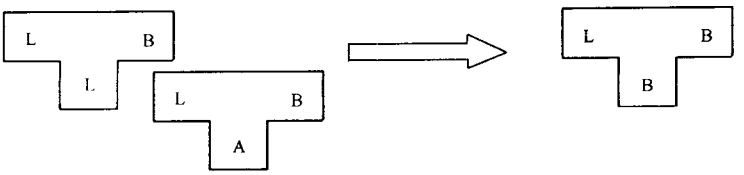
图 1.7 移植第一步及 T 图

显然 $C_A^{L,B}$ 也是一个编译程序，是用 A 语言写的将 L 语言程序编译为 B 语言的程序。这种在 A 机器上生成 B 机器的目标代码的编译称为交叉编译。

第二步，将 $C_L^{L,B}$ 作为交叉编译 $C_A^{L,B}$ 的输入，其输出 $C_B^{L,B}$ 即为所求，如图 1.8(a)所示。用 T 图表示则如图 1.8(b)所示。



(a) 移植第二步



(b) 移植第二步的 T 图

图 1.8 移植第二步及 T 图

1.3.3 对编译程序的评价

不管用什么方法生成的编译程序，对它的质量评价首先当然是正确性，即源程序到目标程序变换的等价性。在正确性的前提下，对编译程序的质量有两个层次的评价。

(1) 生成目标的质量。这主要用目标代码运行时间的长短、占用空间的大小来评定。

(2) 编译程序本身的质量。这主要用编译速度的快慢、编译占用空间的大小以及编译程序本身的结构是否良好，可读性、可维护性如何来评定。

要提高目标程序的质量，所选择的编译方法、算法当然很重要，特别是优化不容忽视，但是提高目标程序质量的种种措施都是需要代价的，它们会影响编译程序本身的质量，这两方面的要求应有所权衡。需要指出的是，片面追求目标质量而忽视编译本身的质量，以致要花很长时间才能将一个源程序编译成目标程序，这样会使用户失去信心，甚至不可容忍。另外，编译程序的可移植性与可维护性也应重视。

1.4 编译程序的学习

很明显，极少有人会去构造一个编译程序，但是编译课程中所介绍的一些原理、方法和算法并不局限于编译。有限自动机的原理、形式描述的方法、自动生成的方法、数据流和控制流分析的方法等对提高一个软件人员的素质是极为有益的。而且对一个大型的复杂软件的构造过程，编译提供了一个具体的实例，这也是对软件人员的一个重要的能力训练。何况或多或少地做些解释、翻译工作的可能是始终存在的。

编译程序是一个系统性很强的软件。在本书中将按阶段地进行讨论，但在学习这些内容时读者心中一定要有系统的概念，才不致于将所学的内容孤立化。因此建议读者找一个小型的语言，尝试用学过的方法，自行组织构造一个编译程序，这对融会贯通这些内容是有益的。