

高等学校教材

GAO DENG XUE XIAO JIAO CAI

# 编译原理

YUANLI  
BIANYI

(修订版)

主编 韩太鲁 孙忠林



中国石化大学出版社

· 高等 学 校 教 材 ·

# 编译 YUANLI BIANYI 原理

(修订版)

主 编 韩太鲁 孙忠林

副主编 董吉文 刘晓红 姚栋义

中国石油大学出版社

## 内容提要

编译原理是计算机本科专业的必修课, 课程主要内容有: 词法分析、语法分析、语义分析、中间代码生成及优化、目标代码生成等。

本书系统地介绍了编译系统的基本原理和方法, 同时介绍了本学科的一些新的内容和知识, 全书共分 8 章。作为编译系统原理和方法的应用, 本书最后给出了编译原理课程设计的题目及具体要求。

本书是山东省多所高校共同组织编写的计算机专业面向 21 世纪系列教材之一, 可作为高等学校计算机专业本科生的教材, 也可供从事计算机应用的工程技术人员或其他自学者学习参考。

## 图书在版编目(CIP)数据

编译原理/韩太鲁, 孙忠林主编. —2 版. —东营: 中国石油大学出版社, 2007.9  
ISBN 978-7-5636-2472-0

I. 编… II. ①韩… ②孙… III. 编译程序—程序设计  
IV. TP314

中国版本图书馆 CIP 数据核字 (2007) 第 135383 号

书 名: 编译原理  
主 编: 韩太鲁 孙忠林

责任编辑: 刘 静 (电话 0546-8395937)  
封面设计: 九天设计

出 版 者: 中国石油大学出版社 (山东 东营, 邮编 257061)  
网 址: <http://www.uppbook.com.cn>  
电子邮箱: cbs2006@163.com  
印 刷 者: 沂南县汇丰印刷有限公司  
发 行 者: 中国石油大学出版社 (电话 0546-8392062)  
开 本: 185×260 印张: 14 字数: 332 千字  
版 次: 2002 年 7 月第 1 版 2007 年 9 月第 2 版第 1 次印刷  
定 价: 19.80 元

版权所有, 翻印必究。举报电话: 0546-8391810

本书封面覆有中国石油大学出版社标志的激光防伪膜。

本书封面贴有中国石油大学出版社标志的激光防伪标签, 无标签者不得销售。

# 前 言

《编译原理》是计算机科学与技术专业的必修课，课程主要内容有：词法分析、语法分析、语义分析、中间代码生成及优化、目标代码生成等。

本书系统深入地介绍了编译系统的基本原理和方法，同时介绍了本学科的一些新的内容和知识，全书共分为 8 章。第 1 章介绍了算法语言的特点及发展过程，第 2 章介绍了形式语言的基本概念及简单的语法分析方法，这两章是后面几章的预备知识。第 3 章是词法分析部分，主要介绍了识别算法语言单词的词法分析器及其构造方法，同时介绍了正规式和有限自动机的基础理论。第 4 章、第 5 章在深入介绍了算法语言语法分析的基本理论及构造方法的同时给出了算法语言各类语句的文法定义，并给文法的产生式加上语法制导翻译的语义动作。为使语义翻译进一步形式化，第 5 章专门讨论了翻译文法和属性翻译文法，使语义翻译更加形式化。为了提高目标代码的质量，第 6 章介绍了代码优化的基本理论和方法。第 7 章讲述了编译阶段如何在运行时对算法语言的目标代码进行存储管理，并给出了某些约束变量的一种分配方案。第 8 章讨论了目标代码的生成方法。作为编译系统原理和方法的应用，本书最后给出了编译原理课程设计的题目及其具体要求。本书教学参考时数为 68~76 学时。

本书的第 1、3 章由董吉文编写，第 2、4 章由韩太鲁编写，第 5 章由姚栋义编写，第 6 章和课程设计部分由孙忠林编写，第 7、8 章由刘晓红编写。书中的部分 C 语言程序由赵卫东进行了编写和审定，孙忠林对全书的编辑做了大量的工作，山东科技大学信息科学与工程学院研究生张艳琦、张瑞全同学及韩红同志录入了部分书稿。山东科技大学信息科学与工程学院的吴哲辉教授对全书作了全面审阅，提出了许多宝贵意见。

本书是山东省多所高校共同组织编写的计算机专业面向 21 世纪系列教材之一，各参编学校的有关领导和专家给予了大力支持，特别是山东科技大学信息科学与工程学院郑永果教授对本书的编写做了大量的组织工作，在此一并表示感谢。

由于编者水平有限，加之成书时间比较仓促，书中难免有不妥和错误之处，敬请读者批评指正。

编 者

2002 年 6 月于泰安

# 修订版前言

2002年出版的第1版《编译原理》经过6年的教学使用，受到了各位同行的普遍肯定，也对书中存在的个别错误给出了中肯的批评和指正，编者在此表示衷心的感谢。承蒙中国石油大学出版社和山东科技大学信息科学与工程学院的大力支持，《编译原理》得以修订再版，编者感到十分荣幸。经过半年多的多方征求意见和修订，编者对第1版的内容和结构都进行了重新整理和修改。

编译技术是计算机学科中发展较早，相对成熟的计算机应用领域，各类编译器的设计开发和应用已是成熟的计算机应用技术。编译原理课程是计算机科学与技术专业的学位必修课，内容主要由词法分析、语法分析、语义分析、中间代码生成及优化、目标代码生成等部分组成。

修订后，本书结构仍然分为8章，考虑第5章的“翻译文法”和“属性翻译文法”与制导翻译基本无关，删除了此部分内容，其他章节基本保持第1版的形式。为了提高学生编译器的设计开发能力，作为编译系统原理和方法的应用环节，对本书最后给出的编译原理课程设计的题目及具体要求进行了充实。

本书的修订由孙忠林执笔，韩太鲁审定。山东科技大学信息科学与工程学院的郑永果教授、张鹏老师、吴震寰老师和王鸽老师对全书的修订参与了内容讨论，提出了许多宝贵意见；中国石油大学出版社对本书的修订做了大量的组织工作，在此一并表示衷心的感谢。

为了方便教学，本书课件已链接到中国石油大学出版社主页的教学资源板块。由于编者水平有限，虽经过仔细修订，书中难免有不妥和错误之处，敬请读者批评指正。

编 者

2007年8月于青岛

# 目 录

|                           |    |
|---------------------------|----|
| 第1章 绪 论.....              | 1  |
| 1.1 程序设计语言概述.....         | 1  |
| 1.1.1 高级程序设计语言的特点和发展..... | 2  |
| 1.1.2 高级程序设计语言的参数传递.....  | 3  |
| 1.2 编译过程概述.....           | 6  |
| 1.2.1 词法分析.....           | 7  |
| 1.2.2 语法分析和语义分析.....      | 8  |
| 1.2.3 中间代码生成和优化.....      | 9  |
| 1.2.4 目标代码.....           | 10 |
| 1.2.5 表格管理.....           | 10 |
| 1.2.6 错误处理.....           | 11 |
| 1.2.7 编译阶段的组合.....        | 13 |
| 1.3 并行编译概述.....           | 13 |
| 习 题 一.....                | 14 |
| 第2章 文法和语言.....            | 15 |
| 2.1 文法和语言的定义.....         | 15 |
| 2.1.1 概念.....             | 15 |
| 2.1.2 文法和语言的定义.....       | 16 |
| 2.1.3 文法的 Chomsky 分类..... | 19 |
| 2.1.4 文法产生式的其他表示法.....    | 20 |
| 2.2 句型的语法树和文法的二义性.....    | 20 |
| 2.2.1 语法树.....            | 20 |
| 2.2.2 文法二义性及规范归约的算法.....  | 21 |
| 2.3 文法的等价变换.....          | 23 |
| 2.3.1 文法等价的定义.....        | 23 |
| 2.3.2 文法的等价变换.....        | 23 |
| 习 题 一.....                | 27 |
| 第3章 词法分析.....             | 29 |
| 3.1 词法分析的任务.....          | 29 |
| 3.1.1 单词与扫描器的功能.....      | 29 |
| 3.1.2 扫描器分离的考虑.....       | 31 |
| 3.2 正则文法与状态转换图.....       | 32 |

|       |                       |     |
|-------|-----------------------|-----|
| 3.2.1 | 状态转换图                 | 32  |
| 3.2.2 | 正则文法的状态转换图表示          | 33  |
| 3.3   | 有限自动机                 | 34  |
| 3.3.1 | 确定有限自动机               | 34  |
| 3.3.2 | 非确定有限自动机              | 36  |
| 3.3.3 | 非确定有限自动机的确定化和最简化      | 38  |
| 3.4   | 正规式和正规集               | 43  |
| 3.4.1 | 正规式和正规集的定义            | 44  |
| 3.4.2 | 正规式与有限自动机             | 45  |
| 3.4.3 | 正规式与正规文法              | 47  |
| 3.5   | 扫描器生成                 | 49  |
| 3.5.1 | 由状态图生成扫描器             | 49  |
| 3.5.2 | 扫描器的自动生成              | 52  |
|       | 习 题 三                 | 55  |
| 第 4 章 | 语法分析                  | 57  |
| 4.1   | 自上而下分析                | 57  |
| 4.1.1 | 自上而下分析存在的问题           | 58  |
| 4.1.2 | 递归下降分析器               | 61  |
| 4.1.3 | LL(1)分析器              | 64  |
| 4.2   | 算符优先分析算法              | 68  |
| 4.2.1 | 直观算符优先分析法             | 68  |
| 4.2.2 | 算符优先分析法               | 71  |
| 4.3   | LR 分析法                | 77  |
| 4.3.1 | LR 分析过程               | 77  |
| 4.3.2 | LR(0)项目集规范族和 LR(0)分析表 | 80  |
| 4.3.3 | SLR(1)分析表             | 86  |
| 4.3.4 | LR(1)项目集规范族和 LR(1)分析表 | 88  |
| 4.3.5 | LALR(1)分析表            | 92  |
| 4.3.6 | 二义性文法的应用              | 95  |
|       | 习 题 四                 | 96  |
| 第 5 章 | 语法制导翻译与中间代码生成         | 98  |
| 5.1   | 概 述                   | 99  |
| 5.2   | 中间代码表示法               | 100 |
| 5.2.1 | 逆波兰表示法                | 100 |
| 5.2.2 | 三元式表示法                | 105 |
| 5.2.3 | 树结构表示法                | 108 |
| 5.2.4 | 四元式表示法                | 110 |
| 5.3   | 表达式及赋值语句的翻译           | 110 |
| 5.3.1 | 算术表达式与赋值语句的翻译         | 110 |

|                               |     |
|-------------------------------|-----|
| 5.3.2 布尔表达式的翻译.....           | 113 |
| 5.4 控制语句的翻译.....              | 118 |
| 5.4.1 标号与 GOTO 语句的翻译.....     | 118 |
| 5.4.2 IF 语句与 WHILE 语句的翻译..... | 119 |
| 5.4.3 FOR 语句的翻译.....          | 122 |
| 5.4.4 CASE 语句的翻译.....         | 125 |
| 5.5 数组元素的翻译.....              | 128 |
| 5.5.1 数组元素的地址计算.....          | 128 |
| 5.5.2 赋值语句中数组元素的翻译.....       | 129 |
| 5.6 过程调用语句的翻译.....            | 131 |
| 5.6.1 过程调用的四元式产生.....         | 132 |
| 5.6.2 过程调用和数组元素相混淆的处理.....    | 133 |
| 5.7 说明语句的翻译.....              | 133 |
| 5.8 类型检查.....                 | 135 |
| 5.8.1 类型系统.....               | 136 |
| 5.8.2 类型检查器的规格说明.....         | 138 |
| 习 题 五.....                    | 141 |
| 第 6 章 代码优化.....               | 145 |
| 6.1 概 述.....                  | 145 |
| 6.2 局部优化.....                 | 146 |
| 6.2.1 基本块的划分.....             | 147 |
| 6.2.2 基本块的 DAG 表示.....        | 147 |
| 6.2.3 基本块的优化处理.....           | 151 |
| 6.2.4 含有数组元素的 DAG.....        | 152 |
| 6.3 全局优化.....                 | 154 |
| 6.3.1 程序流图.....               | 154 |
| 6.3.2 到达一定值数据流方程及到达一定值链.....  | 155 |
| 6.3.3 引用一定值链的计算及应用.....       | 159 |
| 6.3.4 活跃变量数据流方程.....          | 160 |
| 6.3.5 定值—引用链.....             | 162 |
| 6.4 循环优化.....                 | 164 |
| 6.4.1 循环的定义.....              | 164 |
| 6.4.2 循环优化.....               | 165 |
| 习 题 六.....                    | 179 |
| 第 7 章 运行时存储空间管理.....          | 183 |
| 7.1 运行时存储空间组织.....            | 183 |
| 7.2 运行时存储空间分配.....            | 184 |
| 7.2.1 静态分配.....               | 184 |
| 7.2.2 栈式分配.....               | 185 |



|                          |     |
|--------------------------|-----|
| 7.2.3 堆式分配.....          | 191 |
| 7.3 FORTRAN 语言的分配方案..... | 191 |
| 7.3.1 公用语句的处理.....       | 192 |
| 7.3.2 等价语句的处理.....       | 193 |
| 7.3.3 地址分配.....          | 196 |
| 习题七.....                 | 197 |
| 第8章 目标代码生成.....          | 198 |
| 8.1 概 述.....             | 198 |
| 8.2 目标计算机模型.....         | 198 |
| 8.3 简单代码生成算法.....        | 200 |
| 8.3.1 寄存器和地址描述.....      | 200 |
| 8.3.2 寄存器分配.....         | 201 |
| 8.3.3 简单代码生成算法.....      | 202 |
| 习题八.....                 | 203 |
| 附录: 课程设计指导书.....         | 205 |
| 一. 概 述.....              | 205 |
| (一) 课程设计的目的.....         | 205 |
| (二) 课程设计的内容.....         | 205 |
| (三) 课程设计的组织.....         | 206 |
| (四) 课程设计的时间.....         | 206 |
| 二. 课程设计示例.....           | 206 |
| (一) 题目及要求.....           | 206 |
| (二) 设计过程.....            | 208 |
| 三. 课程设计题目.....           | 209 |
| 参考文献.....                | 213 |

# 第1章 绪 论

1946年第一台电子计算机的问世,是人类科学发展史上的一个里程碑,它开创了人类脑力劳动自动化的先河,丰富了人类的精神财富,增强了人类的认识能力。仅仅经过半个多世纪的时间,计算机就经历了数代更新,由初期的单纯用于科学计算发展到广泛应用于社会的各个领域。在科学技术迅猛发展的今天,计算机的发展水平和普及程度已经成为衡量一个国家科学技术发展水平和现代化程度的重要标志。

众所周知,计算机系统包括硬件和软件两大部分,而软件的功能和质量很大程度上决定着整个计算机系统的功能。软件是整个计算机系统的灵魂,是计算机和使用者之间的桥梁。软件大体上可分为系统软件和应用软件,操作系统和编译程序等属于系统软件。

要使用计算机,就得编写程序。用来编写程序的语言称为程序设计语言。程序设计语言是开发软件的工具,是人机通信的媒介。程序员利用语言提供的各种机制把自己想做的工作告诉计算机就形成了程序。在我国,先后被广泛使用的高级程序设计语言有ALGOL、BASIC、FORTRAN、COBOL、LISP、PROLOG、Pascal、C、C++等。

计算机只能接受用机器语言编写的程序,为了使高级程序设计语言所编写的程序能够被计算机所接受,必须将高级语言编写的程序翻译成具体的机器语言,这一翻译工作是由翻译程序来实现的。这有些类似于我们阅读外文资料或听外国人讲话,如果不懂外文,就需要请一位翻译来将外文翻译成中文。

给计算机装上一个编译程序,就相当于给计算机配上一个“翻译”,它可以根据高级语言的语法和语义将高级语言源程序翻译成计算机能够认识的机器语言程序,这样计算机就懂得了一种语言。当然,我们可以给计算机配上多个“翻译”,使它懂得多种语言,从而充分发挥计算机的作用。

将高级语言翻译成机器语言的翻译程序有两种:编译程序和解释程序。若源程序用高级语言所编写,经翻译加工后生成目标程序文件,然后目标程序可以在计算机上运行,这样的翻译程序就是编译程序。而解释程序的加工对象虽然也是高级语言源程序,但在翻译过程中并不生成目标程序文件,而是边翻译边执行目标代码,即解释程序同时处理源程序和源程序要加工的数据。

实际上,高级语言源程序就是按一定规则排列的符号集合,而编译程序就是把这些符号集合变成机器指令集合的转换器。编译程序又称为编译器,它是计算机中的重要系统软件,是从事计算机科学学习和工作的所有人员都应熟悉和掌握的重要知识。

## 1.1 程序设计语言概述

在计算机系统中,程序设计语言可分为机器语言、汇编语言和高级语言,人们常常

把机器语言和汇编语言统称为低级语言。

每台计算机都有自己的指令系统，包括若干条指令，每条指令让计算机执行一个特定的动作，每个指令都用二进制码来表示。不同型号的计算机，它们的指令系统一般是互不相同的。用二进制码所编制的程序可以直接在计算机上运行，这样的指令系统称为机器语言。

所谓汇编语言，就是将计算机的指令系统用符号来表示。例如，用 `ADD` 表示加法指令，用 `SUB` 表示减法指令等。它与机器语言存在着——对应的关系，即一条机器指令对应一条汇编指令，反之亦然。它不再是令人讨厌的数字代码，而是采用了容易辨认、容易记忆的符号。它的使用，将计算机工作者从浩繁的计算机二进制码编程中解放出来，提高了编程效率，同时也提高了程序的质量。

低级语言都是依赖于具体的计算机硬件，使用起来既烦琐又容易出错，程序不便于阅读和交流，实用性差，因此相继出现了很多高级语言。高级语言由表达各种不同意义的“关键词”和“表达式”按一定的语法、语义规则所组成，它与人们习惯的数学语言和工程语言较为接近，因此比较直观、自然和容易理解。使用高级语言编写的程序易读、易写，易于交流、出版和存盘，一旦发现错误也易于修改，因而使用高级语言编写程序比用机器语言或汇编语言所花费的劳动代价要低得多。

### 1.1.1 高级程序设计语言的特点和发展

高级程序设计语言随软件技术的发展而快速发展，20 世纪 60 年代初期，涌现出了大量的高级程序设计语言，例如 `FORTRAN`、`COBOL` 和 `ALGOL 60` 等。它们的出现大大降低了编程工作的劳动强度，把编程人员从烦琐的低级语言中解放出来，由它们编制的软件奠定了最初的软件产业基础。但由于早期的高级语言类型单调，程序过多依赖程序员的技术，滥用 `goto` 语句等原因使程序难读、难改、不易移植，很难编制出更大规模的程序。

二十世纪六七十年代兴起了结构化程序设计方法，希望以规范格式的软件结构消除复杂软件内部的混乱。于是结构化程序设计语言 `Pascal` 和 `C` 得到了广泛应用，一些早期的语言也纷纷改成结构化版本，例如 `FORTRAN-77` 和 `COBOL-74` 就是这个时代的产物，它们的共同特点是支持模块化程序设计思想，具有完备的数据结构、灵活通用的语句、清晰的书写格式、优美的设计风格。20 世纪 70 年代，因受益于结构化程序设计，软件规模得到很大的提高，最大软件（美国导弹预警系统）已达 385 万句。

随着硬件的不断发展，人们就想编制出规模更大的软件。庞大的软件给管理和维护带来困难，即使采用结构化程序设计在更大规模的软件开发面前也是问题百出。所以，20 世纪 70 年代中期人们借助软件工程的方法使所编软件的规模进一步扩大，到 20 世纪 80 年代，最大的软件（美国航天飞机监控系统）已达 4 000 万句。但又过了将近 10 年，证实传统的软件工程对软件危机并没有多大的缓解，大型软件投资失败，不可靠性、不可维护性、不可移植性依然大量存在，寻求新的开发和表达技术已成为燃眉之急。

20 世纪 90 年代初，在抽象数据类型和交互式环境思想的基础上，以 `Smalltalk-80`

为代表的面向对象技术的出现,给处于危机重重的软件产业带来了希望之光。面向对象以其封装性、类和实例的抽象性、继承机制提供的软件重用性、重载多态和动态束定的灵活性所支持的软件可扩充性,无疑成为当时最成功的软件技术。于是软件产业的主导语言都纷纷向面向对象技术扩充,从C语言演变而成的C++程序设计语言和Java语言是其典型代表。它除了包括高级语言的语法机制外还增加了类和实例、继承、重载运算符、虚函数、友员、内联等支持面向对象程序设计的机制,对程序的扩充、软件的重用带来极大的方便,对提高编程效率和软件质量提供了有力的保证,成为应用程序开发者和系统程序员选择使用的最佳语言。

面向对象的程序设计(OOP)立意于创建软件重用代码,具备更好的模拟现实世界环境的能力,这使它被公认为是自上而下编程的优胜者。它通过给程序中加入扩展语句,把函数“封装”进编程所必需的“对象”中。面向对象的编程语言使得复杂的工作条理清晰,编写容易。然而一些早期的具有面向对象程序设计性能的程序语言,虽然具有面向对象的特征,但不能轻松地给出可视化对象,与用户交互能力较差,程序员仍然要编写大量的代码,特别是在编制Windows接口的软件上,这个问题显得尤其突出。20世纪90年代中期以后相继推出的基于可视化和面向对象的编程语言,以其迅捷的编译速度、强大的功能和易学灵活的特点,为越来越多的程序员所喜爱,它们的典型代表有VB、VC、Delphi和Java等可视化编程环境。

编程人员不必自己建立对象,只要在提供的程序框架中加入要完成功能的代码,就可以很容易地生成具有漂亮接口和良好结构的程序。这些可视化编程环境允许在一个具有真正OOP扩展的可视化编程环境中,使用面向对象的编程语言。这种革命性的组合,使得可视化编程与面向对象的开发框架紧密地结合起来,使它们成为触手可及的促进软件重用的开发工具,从而具有强大的吸引力和生命力。

从软件产业的角度来看,当然希望得到表达能力更强、生产效率更高、易读易改的好语言,特别是计算机硬件突飞猛进的发展,使软件经常处于跟不上、赶不上、不好用的危机之中,寻求好的语言始终是软件产业的头等大事,可以肯定地说,随着计算机的发展将会有更多更好的编程语言呈现在我们面前。

### 1.1.2 高级程序设计语言的参数传递

过程(或函数)的定义和调用是程序设计语言的主要特征之一,也是模块化程序设计的主要手段和节省程序代码、扩充语言能力的主要途径。

过程(或函数)在定义之后就可以在程序的其他地方调用它。调用与被调用者之间的信息交流可以通过全局量或由参数传递来实现。下面讨论程序设计语言中参数传递的实现方式。

#### 1. 参数

首先请看下面的C++语言程序:

```
#include "stdio.h"
int max(int x,int y)
```

```

    {   int z;
        if(x>y) z=x;
        else z=y;
        return(z); }
void main()
{   int a,b,c;
    cin>>a>>b; c=max(a,b);
    cout << "max="<< c << endl; }

```

程序中定义了一个整型函数 `max`，函数中的 `x` 和 `y` 称为形式参数，简称形参。在主函数 `main` 中通过调用语句：`c = max(a,b)`；对所定义的函数 `max` 进行调用，并将函数的返回值赋给变量 `c`，其中的 `a` 和 `b` 称为实在参数，简称实参。在此，实在参数不仅仅可以是变量，也可以是常数或者是一个比较复杂的表达式，但实参的类型、个数和顺序应该与形参一致。在程序执行到调用语句时，实参与形参进行参数传递，并执行被调函数 `max`。现在的问题是如何把实在参数传递给相应的形式参数呢？实现参数传递的常用方式有传地址、传值和传名字，下面分别介绍。

## 2. 传地址

所谓传地址是指把实在参数的地址传递给相应的形式参数。过程（或函数）的每个形式参数都对应一个相应的内存空间，称为形式单元。形式单元用来存放相应的实在参数的地址。当调用一个过程（或函数）时，调用段首先把实在参数的地址传递到一个临时设置的工作单元中。当程序转入被调用段之后，被调用段再把临时设置的工作单元中的实参地址取进自己相应的形式单元中，过程（或函数）体对形式参数的任何引用或赋值都被处理成为对形式单元的间接访问，形参的值发生变化，等价于与之对应的实参发生变化。当过程（或函数）执行完毕返回时，形式单元（它们都为指针）所指的实参单元就具有了变化以后的值。

例如，下面的 Pascal 程序：

```

program example(output);
var
    x,y:integer;
procedure add(var s: integer; var r: integer);
begin
    s:=s+r;
end;
begin
    x:=0;
    y:=2;
    add(x, y);
    writeln('x=',x)
end.

```

程序中定义了过程 `add`，它包含两个形参 `s` 和 `r`，它们都是变量参数。在程序调用语句 `add(x, y)` 中，实参 `x` 与形参 `s` 对应，实参 `y` 与形参 `r` 对应，它们之间都采用传地址的方式进行参数传递。在调用语句执行过程中完成下列操作：

- ① 把实参 `x` 和 `y` 的地址分别传递到已准备好（由编译分配）的两个单元（`t1` 和 `t2`）中；
- ② `s:=t1` 且 `r:=t2`，使形参得到对应实参的地址；
- ③ 将 `s` 所指向的单元的内容（`x`）与 `r` 所指向的单元的内容（`y`）相加，完成表达式 `s+r` 的计算；
- ④ 将表达式 `s+r` 的运算结果送到 `s` 所指的内存单元（`x`）中；
- ⑤ 返回主程序。

由此可见，在程序调用过程中若形参发生变化，那么等价于对应的实参发生变化。上面程序中，实参 `x` 因形参 `s` 被赋值而由调用前的 0 变为 2。

### 3. 传值

这是一种最简单的参数传递方式。调用段把实在参数的值计算出来并放在一个能使被调用段取到的地方。被调用段开始工作时，首先把这些值抄进自己的形式单元，然后像使用局部量一样使用这些形式单元。如果实在参数不是指针变量，那么在这种情况下被调用段无法改变实参的值。

在上面关于传地址的例子中，如果把过程 `add` 的首部说明改为：

```
procedure add(s: integer; r: integer);
```

那么，形参 `s`、`r` 与它们对应的实参之间就采用传值方式进行参数传递。在这种情况下，与形参 `s`、`r` 对应的实参可以是表达式形式（传地址方式实参只能是变量），如调用语句可以为：

```
add(x, sqr(y))
```

在执行到该调用语句时，首先计算实参表达式的值，然后将表达式的值传到已经准备好的工作单元中，在过程 `add` 开始工作后，从工作单元中得到实参的值，像使用局部量一样参加过程中的运算。这样，虽然形参 `s` 的值在过程的运行过程中被改变，但并不影响实参的值。

### 4. 传名字

这是 ALGOL 60 所使用的一种参数传递方式。ALGOL 语言用替换规则解释传名字参数的意义：过程调用的作用相当于把被调用段的过程体抄到调用出现的地方，但把其中任一出现的形参都替换成相应的实参（文字替换）。如果在替换时发现过程体中的局部名和实参中的名字使用了相同的标识符（重名），则必须用不同的标识符来表示这些局部名，而且，为了表现实参的整体性，必要时在替换前先把它用括号括起来。

例如，下面的 ALGOL 过程：

```
procedure swap(M,N);  
integer M,N;  
begin  
integer L;
```

```

L:=M;
M:=N;
N:=L;

end;

```

若有过程调用语句 `swap(K(I),K(J))` 采用传名方式进行参数传递,那么过程调用的作用相当于执行了下面的语句:

```

L:= K(I);
K(I):= K(J);
K(J):=L;

```

在这种参数传递方式的实现过程中,进入被调用段之前不对实在参数预先进行计算,而是让过程体中每当使用到相应的形参时才对它计算,所以它跟传地址和传值的参数传递方式均不相同。

除以上 3 种参数传递方法之外,还有传结果等方式,这里我们不予讨论。

## 1.2 编译过程概述

编译程序完成从源程序到目标程序的翻译工作,是一个复杂的过程。从逻辑概念上来讲,一个编译程序的整个工作过程是划分成几个阶段进行的,每个阶段将源程序的一种表示形式转换成另一种表示形式,各个阶段进行的操作在逻辑上是紧密连接在一起的,图 1-1 给出了一个编译过程的各个阶段。

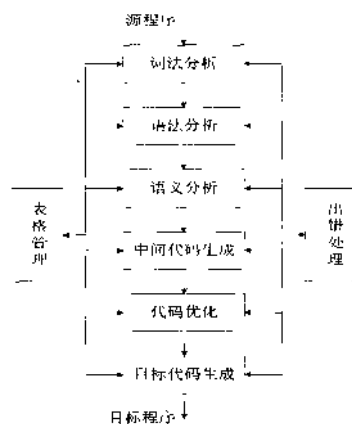


图 1-1 编译的各个阶段

图 1-1 中将编译过程划分成词法分析、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 6 个阶段,这是一种典型的逻辑划分方法。事实上,某些阶段可以组合在一起。另外还有两项重要的工作:表格管理和错误处理,它们与上述 6 个阶段都有联系。编译过程中源程序的各种信息被保留在各种不同的表格里,编译各阶段的工作都涉及构造、查找或更新有关的表格,因此需要有表格管理的工作。如果编译过程中发

现源程序有错误，编译程序应报告错误的性质和错误发生的地点，并且将错误所造成的影响限定在尽可能小的范围内，使得源程序的其余部分能继续被编译下去。有些编译程序还能自动校正错误，这些工作称之为错误处理。错误处理能力的强弱也是评价一个编译程序优劣的标准之一。下面我们介绍各阶段的任务。

### 1.2.1 词法分析

词法分析阶段是编译过程的第一个阶段。这个阶段的任务是从左到右一个字符一个字符的读入源程序，对构成源程序的字符串进行扫描和分解，从而识别出一个个单词（也称单词符号）。这里所谓的单词是指逻辑上紧密相连的一组字符，这些字符具有集体含义，比如标识符是由字母字符开头，后跟字母、数字字符的字符序列组成的一种单词。保留字（也称关键词或基本字）是一种单词，此外还有算符、常数和界符（标点符号、左右括号等）等等。例如某源程序片段如下：

```
var
    sum, first, count : real ;
begin
    sum:= first + count * 10
end.
```

词法分析阶段通过对该程序片段的扫描、分解，将会把组成这段程序的字符识别为如下单词序列：

|         |       |         |       |
|---------|-------|---------|-------|
| 1. 保留字  | var   | 2. 标识符  | sum   |
| 3. 逗号   | ,     | 4. 标识符  | first |
| 5. 逗号   | ,     | 6. 标识符  | count |
| 7. 冒号   | :     | 8. 保留字  | real  |
| 9. 分号   | ;     | 10. 保留字 | begin |
| 11. 标识符 | sum   | 12. 赋值号 | :=    |
| 13. 标识符 | first | 14. 加号  | +     |
| 15. 标识符 | count | 16. 乘号  | *     |
| 17. 整数  | 10    | 18. 保留字 | end   |
| 19. 界符  | .     |         |       |

可以看出，5个字符即“b”、“e”、“g”、“i”和“n”构成了一个分类为保留字的单词“begin”，两个字符即“:”和“=”构成了表示赋值运算的符号“:=”。这些单词间的空格在词法分析阶段都被滤掉了。

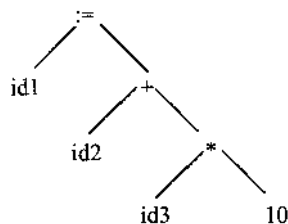
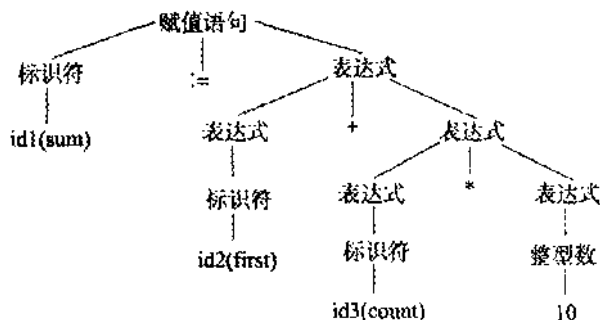
我们使用 id1、id2 和 id3 分别表示 sum、first 和 count 三个标识符的内部形式，那么经过词法分析后，上述程序片断中的赋值语句 `sum:= first + count * 10` 则表示为：

`id1:= id2 + id3 * 10` (1.1)



### 1.2.2 语法分析和语义分析

在编译程序的实现中,语法分析和语义分析常常是组合在一起的。语法分析的任务是在词法分析的基础上将单词序列分解成各类语法单位,如“程序”、“语句”、“表达式”等等。这种语法单位,一般可表示成语法树,比如(1.1)经语法分析后可以确定它是 Pascal 语言的“赋值语句”,语法分析程序将其表示成图 1-2 所示的语法树(或称分析树),它描绘了输入的语法结构。这种语法结构更常见的内部表示如图 1-3 所示。



语法分析所依据的是语言的语法规则,即描述程序结构的规则。通过语法分析确定整个输入串是否构成一个语法上正确的程序。

程序的结构通常是用递归规则表示的,例如我们可以用下面的规则来定义表达式:

- (1) 任何标识符是表达式;
- (2) 任何常数(整常数、实常数)是表达式;
- (3) 若  $e_1$  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。

类似地,语句也可以递归定义,如:

- (1)  $\langle \text{标识符} \rangle := \langle \text{表达式} \rangle$  是语句;
- (2)  $\text{while } \langle \text{表达式} \rangle \text{ do } \langle \text{语句} \rangle$  和  $\text{if } \langle \text{表达式} \rangle \text{ then } \langle \text{语句} \rangle \text{ else } \langle \text{语句} \rangle$  都是语句。

上述赋值语句  $id1 := id2 + id3 * 10$  之所以能表示成图 1-3 形式的语法树,依据的就是赋值语句和表达式的定义规则。