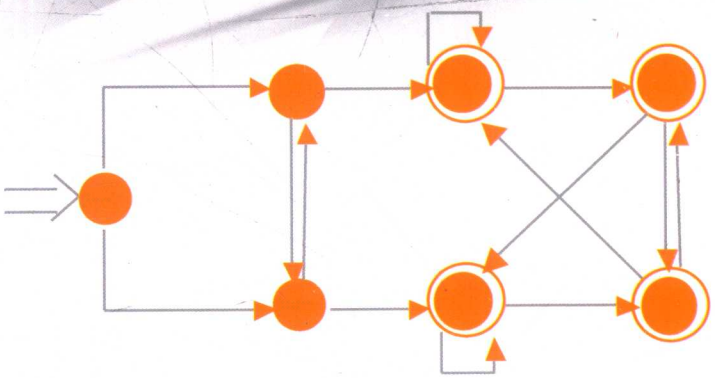
普通高等教育创新型人才培养规划教材



编译方法及应用

BIANYI FANGFA JI YINGYONG

许 清 刘香芹 编著

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

普通高等教育创新型人才培养规划教材

编译方法及应用

许　清　刘香芹　编著

北京航空航天大学出版社

内 容 简 介

本书全面地讨论了编译器设计方面的主要问题，包括词法分析、语法分析、语法制导翻译、目标代码生成等分析技术。作者长期从事编译方法课程的教学工作，将多年的教学体会与认识融入教材中，选择学生熟悉的C语言作为编译对象语言融入课程中，不仅包含编译原理的基本理论，还列举了一些实例，特别是将编译理论与实际应用相结合，使学生可以体会到编译的理论和技术在软件设计中的应用。

本书有较强的实用性，可作为应用型本科计算机科学与技术专业、普通本科计算机及相关专业编译原理课程的教材，也可以供相关专业的研究生、计算机软件技术人员等作参考。

图书在版编目(CIP)数据

编译方法及应用 / 许清，刘香芹编著. -- 北京 :
北京航空航天大学出版社，2016.8

ISBN 978-7-5124-2143-1

Ⅰ. ①编… Ⅱ. ①许… ②刘… Ⅲ. ①编译程序－程序设计 Ⅳ. ①TP314

中国版本图书馆CIP数据核字(2016)第120781号

编译方法及应用

许 清 刘香芹 编著

责任编辑 刘晓明

*

北京航空航天大学出版社出版发行

北京市海淀区学院路37号(邮编100191) http://www.buaapress.com.cn

发行部电话：(010)82317024 传真：(010)82328026

读者信箱：goodtextbook@126.com 邮购电话：(010)82316936

涿州市新华印刷有限公司印装 各地书店经销

*

开本：710×1 000 1/16 印张：16.75 字数：357千字

2017年1月第1版 2017年1月第1次印刷 印数：2 000册

ISBN 978-7-5124-2143-1 定价：39.00元

前　言

编译程序是计算机系统软件的重要组成部分。编译程序的理论与技术在计算机科学中是比较成熟的学科，其内容属于计算机语言处理的范畴。“编译原理”是计算机科学与技术专业主要的基础课程之一，主要介绍高级语言编译程序构造的一般原理和基本实现方法。这些原理和方法除了用于构造编译程序之外，也广泛应用于一般软件的设计与实现中。例如：建立词法分析器的串匹配技术已用于文本编辑器、信息检索系统和模式识别器；上下文无关文法和语法制导翻译等概念已用于诸如排版、绘图系统这样的小语言程序；代码优化技术已用于程序验证器和从非结构化程序产生结构化程序的程序验证器之中；编译器是软件工程的很好的实例，包括它的基本结构设计、模块划分、表驱动的编程方法等；在软件安全、程序理解和软件逆向工程等方面都有着广泛的应用。

按照国家对软件人才的要求，本专业人员应具备4种基本的专业能力，即

- 计算思维能力；
- 算法的设计与分析能力；
- 程序设计和实现能力；
- 计算机软硬件系统的认知、分析、设计与应用能力。

课程通过对构造模型问题的形式化描述，可以训练学生的逻辑思维能力和抽象思维能力，使学生了解和初步掌握“问题、形式化描述、自动化（计算机化）”这一最典型的计算机问题求解思路，达到能理解、建立和应用形式模型的目的。

课程通过对编译程序整体结构的认识，使学生可以了解到大型软件的组织结构和方法，这种“抽象的思维逻辑”，不仅可以应用于软件的开发与设计中，也可以应用于日常的生活、工作中。

本教材有如下特点：

① 针对以计算机应用为目的的普通院校本科生。书中大部分实例是以C语言作为研究对象，虽然C语言可能不是最好的编译系统案例，但是，由于本科学生比较熟悉C语言的结构及语法，选用C语言作为实例，

可以使学生没有语言的陌生感，更加容易接受和理解。

② 教师自研配套课件。自从1998年教育部提倡多媒体教学以来，我们编译原理的任课教师着手研制开发编译原理辅助教学软件，从脚本的撰写，到课件的制作，完全是由任课教师自己完成的。

③ 配有习题、习题解答，以及上机辅导。我们已经在校内出版了《编译原理习题解答》，应用了两届，效果很好，学生期末考试成绩得到很大的提升。

④ 教材中融入了关于编译技术和方法的应用，使学生体会到了枯燥的编译理论的应用价值。

首先感谢黄正文老师，在他的带领下，我们前期完成了《编译原理》辅助教学软件的设计及制作工作，该软件融入了黄正文老师30多年来在编译教学中积累的经验，应用于授课中取得了良好的效果。由于有几十年教学工作的积累，加之有前期工作的基础，我们萌发了将这个课件转换成教材的想法。在学校的支持下，经过我们编写小组成员的不懈努力，终于完成了这项工作。

本书的绝大部分工作由编译原理任课教师许清、刘香芹完成，特别是刘香芹老师在文稿的撰写中付出了大量的时间与精力，内容涉及C语言及C++语言程序设计部分，李胜宇、张荣博老师做了很多指导和书写工作；郑志勇老师做了文稿的校对和审核工作；李佳佳老师在助课过程中找出了校内版文稿中出现的错误。本书由许清老师整理、统稿。

徐蕾教授、黄正文教授为全书提出了许多宝贵的意见，在此表示衷心的感谢！本书的部分章节参考了参考文献[1]至参考文献[16]中的内容，对这些作者也表示感谢！

作为一名教师，能将教学经验和体会分享给学生，感到非常的欣慰！

由于作者水平有限，对于书中存在的一些缺点和错误，恳请广大读者批评指正。

作　者

于沈阳航空航天大学

2016年6月

目　录

第 1 章　编译程序概述

在计算机系统中，计算机使用的语言可以分为三个层次：高级语言层、汇编语言层和机器语言层。在计算机上执行一个高级语言程序一般要分两步：第一步，用一个编译程序把高级语言程序翻译成机器语言程序；第二步，运行所得的机器语言程序，求得计算结果。

要把高级语言程序变换为可以在计算机上执行的形式，完成这个变换任务的语言翻译系统称之为编译程序。编译程序是计算机系统软件最重要的组成部分之一。多数计算机系统都配有不止一种高级语言的编译程序，而有些高级语言甚至配置了多种不同性能的编译程序。

1.1　高级语言概述

高级语言是抽象程度比较高的计算机语言，需要经过编译程序将其编译成特定机器上的目标代码才能执行，一条高级语言的语句往往需要若干条机器指令来完成。

高级语言是不依赖于机器的计算机语言，是由编译程序为不同机器生成不同的目标代码（或机器指令）来实现的。那么，要将高级语言编译到什么程度呢？这又跟编译的技术有关了，应该既可以编译成直接可执行的目标代码，又可以编译成一种中间表示，然后拿到不同的机器和系统上去执行。这种情况通常又需要支撑环境，比如解释器或虚拟机的支持，将 Java 程序编译成 bytecode，再由不同平台上的虚拟机执行就是很好的例子。所以，说高级语言不依赖于机器，是指在不同的机器或平台上高级语言的程序本身不变，而由通过编译器编译得到的目标代码去适应不同的机器。从这个意义上来说，通过交叉汇编，一些汇编程序也可以获得不同机器之间的可移植性，但这种途径获得的移植性远远不如高级语言来得方便和实用。

高级语言源程序一般要经过如下的处理才能执行。执行过程如图 1.1 所示。

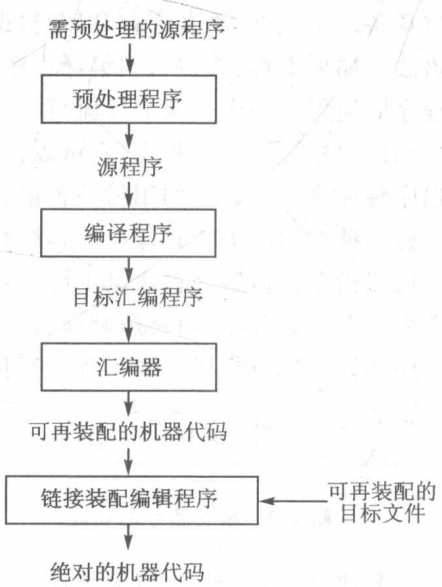


图 1.1　对高级语言的处理过程

1.2 编译程序

1.2.1 编译程序与解释程序

简单地说，**编译程序**就是一个语言翻译程序。翻译程序的功能是把一种源语言程序翻译成逻辑等价的另一种目标语言程序。用源语言编写的有待翻译的程序，称为源程序。源语言可以是汇编语言，也可以是高级程序设计语言（比如C++语言等）；目标语言是汇编语言或机器语言之类的“低级语言”。

编译程序是专门以高级程序设计语言的源程序作为翻译对象进行翻译处理的。所以，高级语言编写的源程序要上机执行，通常首先要经过编译程序加工成为低级语言表示的目标程序，若目标程序是以汇编语言表述的，则还要经过一次汇编程序的加工。如果把编译程序看成一个“黑盒子”，则它所执行的转换工作可以用图1.2来说明。

图1.2 编译程序的功能

编译程序的基本功能是把高级语言源程序翻译成等价的目标程序，除此之外，它还应具备语法检查、语义检查，以及错误处理等功能。语法检查是检查源程序是否合乎语法。如果不符合语法，则编译程序要指出语法错误的部位、性质和有关信息。编译程序应使用户程序一次编译就可尽量多地查出错误。

编译程序就是一个程序，它可以读入用某种高级语言（源语言）编写的程序，并把该程序翻译成一个等价的用另一种语言（目标语言）编写的程序。

另一种常用的语言翻译程序是**解释程序**，解释程序同样是将高级语言源程序翻译成机器指令。但解释程序又不同于编译程序，它不是将高级语言的源程序翻译之后产生一个目标程序，而是边翻译边执行，即输入一句，翻译一句，执行一句，直至整个程序翻译并执行完毕。解释程序所执行的操作如图1.3所示。

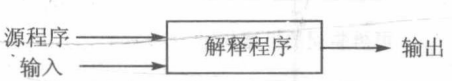


图1.3 解释程序的功能

编译程序和解释程序之间的主要区别如下：

编译程序先把全部源程序翻译成目标程序，然后再执行，而且目标程序可以反复执行。解释程序以源程序作为输入，但不生成整个的目标程序，而是边解释边执行源程序本身；对源程序中要重复执行的语句（例如循环体中的语句）需要重复

地解释执行，因此从速度上看，较之编译方式要多花费执行时间，效率较低。编译方式下，源程序已经生成目标程序，可以反复执行，因此比解释方式快。但在解释方式下，有利于程序的交互式调试，解释程序给出的错误诊断信息通常比编译程序好。编译过程类似笔译，笔译的结果可以反复阅读。解释程序类似即席翻译（口译），别人说一句，它就翻译一句。

很多语言处理系统组合了编译和解释两个程序，源程序先被翻译成一种中间表示形式的程序，再接收输入数据并对源程序进行解释而生成输出结果，具体转换工作如图 1.4 所示。例如 Java 语言是混合语言处理系统，它包括编译和解释程序，但还是称为 Java 编译系统。PL/0 编译系统也是这样的混合语言处理系统。

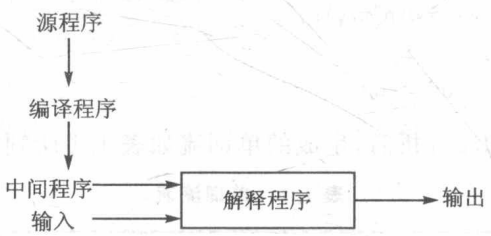


图 1.4　混合语言处理系统

1.2.2　编译程序的工作过程

编译程序完成从源程序到目标程序的翻译工作，是一个复杂的整体的过程。从概念上来讲，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 1.5 所示为一个编译过程的各个阶段，这是一种典型的划分方法。将编译过程划分成词法分析、语法分析、语义分析与中间代码生成、中间代码优化和目标代码生成等阶段，通过源程序在不同阶段所被转换成的表现形式来理解编译各个阶段完成的工作任务。

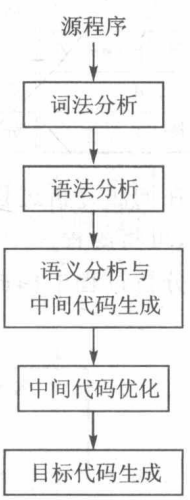


图 1.5　编译的各个阶段

第一阶段，词法分析。这个阶段的主要功能是审查源程序是否有词法错误，为代码生成阶段收集信息，比如，审查标识符的拼写错误。词法分析是自左向右一个一个字符地读入源程序，对构成源程序的字符流进行组合，构成单词，因此这个阶段也称为扫描源程序。这里所说的单词是指逻辑上紧密相连的一组字符，字符间的间隔用空格表示。常见的单词种类有：保留字（关键字或基本字），如 main、void int、float、char、if、else、for、while、return 等；标识符、运算符，如＋、－、×、÷等；分界符，如标点符号和左右括号等；常数等。

词法分析输出的单词又叫 TOKEN 字，单词符号一般由两部分组成，即单词种别码和单词自身值，单词的机内表示如图 1.6 所示。

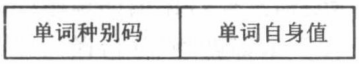


图 1.6　单词的机内表示

单词有一词一码，如保留字，单词自身值可以省略，用码值代表单词；单词还有多词一码，如标识符、常数等。一般情况下，标识符的自身值可以表示成其 ASC II 码，常数的自身值用其二进制表示。

例如，某源程序(C 语言)片段如下：

```
main()/* a example */
{ int x=13,y=5,temp;
    printf("x= %d,y= %d\n"x,y);
    temp=x+y*10;
    printf("temp= %d\n",temp);}
```

上例源程序经过词法分析后，生成的单词流如表 1.1 所列。

表 1.1　单词流表

单词流	单词流	单词流	单词流	单词流
保留字　main	分界符　(	分界符　)	分界符　{	保留字　int
标识符　x	分界符　=	常数　13	分界符　,	标识符　y
分界符　=	常数　5	分界符　,	标识符　temp	分界符　;
保留字　printf	分界符　(	分界符　"	标识符　x	分界符　=
算符　%d	分界符　,	标识符　y	分界符　=	算符　%d
算符　\n	分界符　"	标识符　x	分界符　,	标识符　y
分界符　)	分界符　;	标识符　temp	分界符　=	标识符　x
算符　+	标识符　y	算符　*	常数　10	分界符　;
保留字　printf	分界符　(	分界符　"	标识符　temp	分界符　=
算符　%d	算符　\n	分界符　"	分界符　,	标识符　temp
分界符　)	分界符　;	分界符　}		

在词法分析过程中，将滤掉源程序中的注释、多余的空格等，且检查词法错误。此时，可将源程序中的名字及其属性构造一个部分符号表，以备后阶段查用。上例在词法分析过程中构造的部分符号表如表 1.2 所列。

表 1.2　部分符号表

名字栏	信息栏		
main	保留字		
x	简单变量	整型	存储单元地址
y	简单变量	整型	存储单元地址
temp	简单变量	整型	存储单元地址

第二阶段，语法分析。这个阶段的主要功能是在词法分析的基础上，根据语言的语法规则对源程序的单词流进行分析，把单词组合成各类语法单位，识别出像“表达式”、“语句”、“程序段”、“程序”等这样的语法成分。通过语法分析，确定整个输入串是否构成语法上正确的“程序”。例如：sum＝first＋count * 10 代表一个“赋值语句”，而其中的 first＋count * 10 代表一个“算术表达式”。因而，语法分析的任务就是识别 first＋count * 10 为算术表达式，同时，识别上述整个符号串属于赋值语句这个范畴。

这里的语法规则通常用上下文无关文法描述。

例如，对赋值语句有如下语法规则：

＜赋值语句＞→＜变量＞＝＜表达式＞

＜变量＞→＜标识符＞

＜表达式＞→＜表达式＞ ＋ ＜表达式＞|＜表达式＞ － ＜表达式＞|＜变量＞| ＜数＞

设有源程序片段：

```
main()
{
    float sum,first,count;
    scanf(" %d, %d",&first,&count);
    sum = first + count * 10;
    printf("sum = %d",sum);
    }
```

对其中的赋值语句分析结果如图 1.7 所示。

这是一棵正确的语法树，符合赋值语句文法，即能由该语法规则生成该赋值语句的语法树。

词法分析和语法分析本质上都是对源程序的结构进行分析。但是词法分析的任务通过对源程序进行一种线性扫描即可完成，比如识别标识符，因为标识符的结构是以字母打头，后跟字母和数字构成的符号串，只要顺序扫描输入流，当遇到非字母或非数字的字符时，表示识别一个完整的标识符结束，将前面发现的所有字母和数字组合在一起即可构成单词标识符。但这种线性扫描不能用于识别递归定义的语法单元，比如不能用此办法去匹配表达式中的括号。总的来说，语法分析是一种层次结构分析，主要分为自上而下分析和自下而上分析两种，后续章节将要介绍描述程序结构的工具——上下文无关文法。

第三阶段，语义分析与中间代码的生成。这一阶段的主要功能是审查源程序有无语义错误，为代码生成阶段收集信息。对语法分析所识别出的各类语法单位，分析其含义，进行初步翻译并产生中间代码。这一阶段的工作是，首先对每种语法单位进

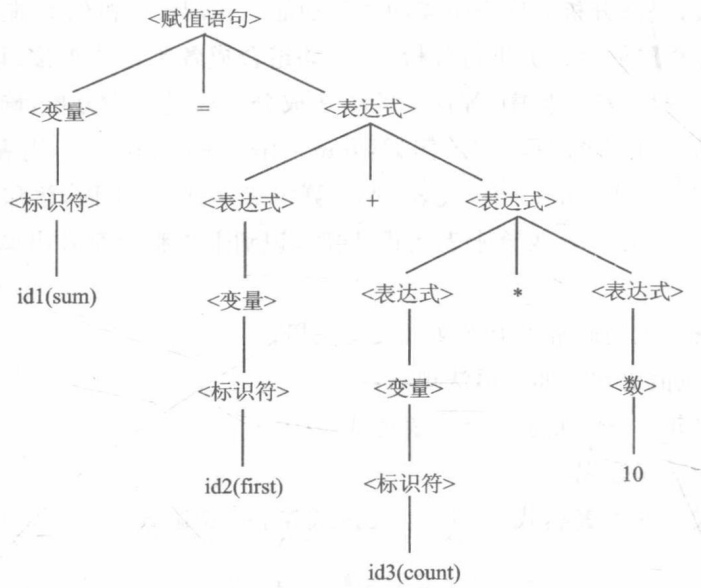


图 1.7　sum＝first＋count * 10 语法分析树

行静态语义检查，如类型检查，审查每个算符是否具有语言规范允许的运算对象，当不符合语言规范时，编译程序应报告错误。如有的编译程序要对实数用作数组下标的情况报告错误；又如某些语言规定运算对象类型可以被强制，那么当两目运算符施于一个整型运算对象和一个实型运算对象时，编译程序应将整型对象转换成实型对象而不认为是源程序错误，假如上例中，语句 sum＝first＋count * 10，“ * ”的两个运算对象分别是 count 和 10，count 是实型，10 是整型，则语义分析阶段进行类型审查之后，在语法分析所得到的分析树上增加一个语义处理结点，表示整型变成实型的一目运算 inttoreal，其插入语义处理结点的树如图 1.8 所示。再如，编译程序会对实数变量是否定义、是否重复定义，以及形参和实参的一致性做检查等。

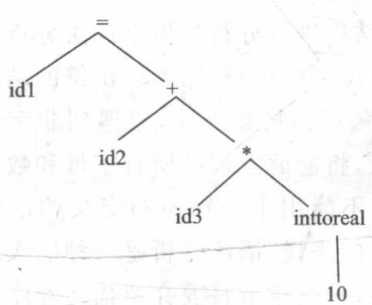


图 1.8　插入语义处理结点的树

如果语义正确，则接下来进行另一方面的工作，即进行中间代码的翻译或生成目标代码等。这一阶段所依循的是语言的语义规则。通常使用属性文法描述语义规则，从这里开始进行“翻译”，并将其翻译成中间代码的形式。中间代码是一种独立于机器硬件系统的符号系统，它的含义明确、便于处理。常见中间代码的形式有：三元式、四元式、逆波兰式和树形表示等。

例如，采用“三地址指令”的“四元式”形式作为中间代码，形式如下所示：

算符	左操作数	右操作数	运算结果

算符：算术运算的加减乘除、逻辑运算的∧和∨等。

意义是：对“左、右操作数”进行某种运算（由算符指明），把运算所得的值作为“结果”保留在顺序生成的临时变量里（用临时变量下标区别临时变量的生成顺序）。在采用四元式作为中间代码的情形下，中间代码产生的任务就是按照语言的语义规则把各类语法范畴翻译成四元式序列。

例如，sum＝first＋count＊10，赋值语句生成的四元式序列为

(1) (inttoreal,10,_,T_1)

(2) (＊,id3,T_1,T_2)

(3) (＋,id2,T_2,T_3)

(4) (＝ ,T_3,_,id1)

上述四元式整齐、标准，由此生成目标代码较容易。其中，id1、id2 和 id3 是如图 1.8所示的标识符；T_1、T_2和 T_3是编译期间随着四元式的生成顺序而引入的临时变量；第一个四元式意味着把整型常数 10 转换为实型数，结果存入临时变量 T_1 中，即整型向实型转换；第二个四元式意味着 count 的值乘以 T_1 结果存入临时变量 T_2 中；第三个四元式意味着 first 的值和 T_2 的值相加，结果存入临时变量 T_3 中；第四个四元式意味着临时变量 T_3 的值存入变量 id1(sum)中。

第四阶段，代码优化。代码优化的目的是提高目标程序的时、空质量。原则上讲，优化可以在编译的各个阶段进行，但最主要的一类优化是在目标代码生成之前、对语法分析后的中间代码生成时进行的。这类优化不依赖于具体的计算机（与计算机硬件无关）。另一类重要的优化是在目标代码生成时进行的，它在很大程度上依赖于对具体计算机合理地分配寄存器，而利用系统资源则是要考虑的主要问题。

这一阶段的主要功能是对前一类中间代码进行优化，对前段产生的中间代码进行加工变换，以期在最后阶段能产生出更为高效（节省时间和空间）的目标代码。优化的主要方法有：公共子表达式的提取、循环优化、删除无用代码等。有时，为了便于“并行运算”，还可以对代码进行并行化处理。优化所依循的原则是程序运行结果的等价原则、时空有效原则、代价合算原则。

例如，前例生成的四元式序列：

(1) (inttoreal,10,_,T_1)

(2) (＊, id3,T_1,T_2)

(3) (＋, id2,T_2,T_3)

(4) (＝,T_3,_,id1)

经优化后生成的四元式为

(1) (＊,id3,10.0,T_1)

(2) (＋, id2,T_1,id1)

第五阶段，目标代码生成。这个阶段的主要功能是将中间代码或者经过优化处

理之后的代码转换为低级语言代码。这个阶段的工作依赖于硬件系统结构和机器指令的含义。这个阶段的工作非常复杂，转换过程主要涉及机器硬件系统功能部件的运用、机器指令的选择、各种数据类型变量的存储空间分配，以及寄存器和后援寄存器的调度等。

目标代码的形式可以是绝对机器指令代码、可重定位的机器指令代码或汇编指令代码。如果目标代码是绝对指令代码，则这种目标代码可立即执行；如果目标代码是可重定位的机器指令代码，则需要先做地址重定位，然后再执行；如果目标代码是汇编指令代码，则需要汇编器汇编之后才能运行。必须指出的是，现在多数实用编译程序所产生的目标代码都是一种可重定位的指令代码。这种目标代码在运行前必须借助于一个链接装配程序，把各个目标模块（包括系统提供的库模块）链接在一起，确定程序变量（或常数）在主存中的位置，装入内存中指定的起始地址，使之成为一个可以运行的绝对指令代码程序，如图 1.1 所示。

例如，上例经优化后的四元式序列：

(* ,id3,10.0,T_1)

(+,id2,T_1, id1)

生成的目标代码为

```
(1) MOV    id3,     R2
(2) MUL    #10.0,   R2
(3) MOV    id2,     R1
(4) ADD    R1,      R2
(5) MOV    R1,      id1
```

图 1.5 所示的编译程序五个阶段的划分是一种典型的划分方法。事实上，并非所有编译程序都分成这五个阶段，例如，某些阶段可能组合在一起，此时，这些阶段间源程序的内部表示形式就没有必要构造出来；有些编译程序对优化没什么要求，则优化阶段可以省去；在某些阶段下，为了加快编译速度，中间代码产生阶段也可以省去；有些最简单的编译程序是在语法分析的同时产生目标代码（省略了中间代码生成阶段和代码优化阶段）。但是，多数实用的编译程序的工作过程大致都有上面所说的五个阶段。

1.3 编译程序的结构

1.3.1 编译程序结构简介

上述五个阶段是编译程序工作时的动态特征。编译程序的结构可以按照这五个阶段的任务分模块进行设计。具体编译程序逻辑结构总体框架如图 1.9 所示。

词法分析器，又称**扫描器**。输入源程序（一般用高级语言编写的程序称为源程