高 等 学 校 教 材

编译原理

何炎祥

高等教育出版社

内容提要

本书主要介绍设计和构造编译程序的基本原理和方法。内容包括形式语言理论和自动机理论、常用的词法分析方法、各种经典的语法分析方法、语法制导翻译方法、存储器的组织与管理方法、符号表的组织与造查表方法、代码优化和代码生成方法、并行编译程序及编译自动化技术等。

本书注重理论与实践、原理与方法的互通，基本概念阐述清晰，讲授深入浅出，循序渐进，系统性强。各章之后还附有难度不一的习题供复习、思考和探索之用。本书既可作为高等院校计算机专业的教材，也可供相关专业师生和科技工作者及软件研发人员学习和参考。

图书在版编目（CIP）数据

编译原理/何炎祥. —北京：高等教育出版社，
2004.8
ISBN 7-04-015391-2

Ⅰ.编… Ⅱ.何… 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（2004）第 067267 号

策划编辑 刘建元 **责任编辑** 武林晓 **市场策划** 陈 振
封面设计 于文燕 **责任印制** 陈伟光

出版发行	高等教育出版社	**购书热线**	010-64054588
社 址	北京市西城区德外大街 4 号	**免费咨询**	800-810-0598
邮政编码	100011	**网 址**	http://www.hep.edu.cn
总 机	010-82028899		http://www.hep.com.cn
经 销	新华书店北京发行所		
印 刷	北京市白帆印务有限公司		
开 本	787×1092 1/16	**版 次**	2004 年 8 月第 1 版
印 张	19.25	**印 次**	2004 年 8 月第 1 次印刷
字 数	390 000	**定 价**	25.00 元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

前　言

编译程序(compiler)是计算机的重要系统软件,也是高级程序设计语言的支撑基础。本书主要介绍设计和构造编译程序的基本原理和方法。

本书共分12章。第1章讲述编译程序的功能、结构、工作过程、组织方式、编译程序与高级语言的关系以及编译自动化方面的基本知识。第2章介绍形式语言理论,仅仅给出了便于理解、有助于研究各种分析方法和设计构造编译程序的形式语言理论,并着重介绍了上下文无关文法。

有穷自动机是描述词法的有效工具,也是进行词法分析的主要理论基础。第3章专门讨论有穷自动机,讨论它与正规文法、正规表达式之间的对应关系以及它的确定化和最小化方面的知识,略去了像Turing机及可计算性理论方面的内容。第4章讨论词法分析的功能和词法分析程序的设计方法。

上下文无关文法可用于描述现今大多数高级程序设计语言的语法,也是语法分析的主要理论支柱。为此,在接下来的几章里,主要讨论了与上下文无关文法相关的各类语法分析方法。

第5章介绍自上而下分析方法,包括LL(k)文法、LL(1)分析方法和应用十分广泛的递归下降分析方法。第6章讨论自下而上分析方法的一般原理和优先分析方法,包括简单优先分析技术和算符优先分析方法。第7章专门讨论自下而上的LR(k)分析方法,包括LR(0)、SLR(1)、规范LR(1)以及LALR分析表的构造算法。

第8章介绍语法制导翻译方法,主要讨论了语法制导翻译方法的基本原理、属性翻译文法以及它们在中间代码生成中的应用。

第9章讨论运行时的存储组织与管理,其中考虑了一些重要的语言特征,如过程调用、参数传递、数组和记录的存取方式以及多种存储分配技术。

第10章讨论符号表的组织和存取符号表的各种方法。第11章介绍常用的优化方法。第12章简单讨论代码生成的原理。

我们认为某些形式语言理论和自动机理论对设计构造编译程序是极其有用的,但现有的不少形式语言理论及自动机理论与设计和构造编译程序的关系不大。本书试图在沟通设计和构造编译程序的理论与实践、原理与方法等方面做一点尝试。

"编译原理"这门课程是计算机专业的主干课和必修课,也是计算机专业高年级课程中较难学习的一门课程,其先导课程是汇编语言程序设计、计算机组成原理、数据结构、高级语言程序设计和离散数学等。本课程的参考学时数为72,使用者可根据具体情况对教材内容

进行取舍，例如，工科院校的学生可略过第7章、第8章，并可精减第2章的内容，从而使授课学时数减至54。

本书可作为高等院校计算机专业的本科教材，也可供教师、研究生及有关科技工作者学习和参考。

本书成书过程中，得到了武汉大学教务处、武汉大学计算机学院和高等教育出版社的鼎力协助。此外，书中还引用了一些专家学者的研究成果，在此一并表示感谢，并敬请读者不吝赐教。

何炎祥

2004年5月于武昌珞珈山

目　录

第1章 引 论

编译程序是高级语言的支撑基础，是计算机系统中重要的系统软件之一。编译原理是计算机科学技术中发展最为迅速的一个分支，现已基本形成了一套比较系统、完整的理论和方法。本章主要介绍为什么需要编译程序以及编译程序的功能、体系结构、工作过程、组织方式、编译程序与高级程序设计语言的关系，以及编译自动化和并行编译程序等方面的基本知识。

1.1 翻译程序

1.1.1 程序设计语言

众所周知，自然语言是人类传递信息、交流思想和情感的工具，程序设计语言则是人与计算机联系的工具。人正是通过程序设计语言指挥计算机按照人的意志进行运算和操作、显示信息和输出运算结果的。

最早的计算机程序设计语言是机器语言（指令系统）。机器语言中的指令都是用二进制代码直接表示的，指令难记、难认，因而机器语言程序难写、难读、难修改。一个机器语言程序的编写时间往往是它运行时间的几十倍到几百倍。随着计算机科学技术的发展，出现了符号语言和汇编语言等程序设计语言，它们虽然比机器语言前进了一步，但仍属于计算机低级语言，用来编写程序还是很不方便。程序编写的效率低下阻碍了计算机科学技术的发展和推广应用，因此，程序设计语言自身的自动化势在必行。

1954 年，FORTRAN I 语言的问世标志着计算机高级程序设计语言的诞生。随后，计算机高级程序设计语言如雨后春笋，层出不穷。至今，全世界已经出现的计算机高级程序设计语言大概有几千种，但典型的常用语言不过十几种。计算机高级程序设计语言仍在不断发展，目前流行的面向对象程序设计语言是对传统的面向过程程序设计语言的一种挑战。

计算机高级程序设计语言独立于机器，比较接近自然语言，因而容易学习和掌握，且程序编写效率高，编写出的程序易读、易理解、易修改、易移植。

1.1.2 翻译程序

用高级程序设计语言（简称高级语言）编写程序方便且效率高，但计算机不能直接执行

用高级语言编写的程序，只能直接执行机器语言程序。因此，用高级语言编写的程序必须由一个翻译程序翻译成机器语言程序。

翻译有两种方式，一种是编译方式，另一种是解释方式。

翻译程序接收一个源程序（用某种源语言书写），并将它转换成一种目标程序（用目标语言书写）。若源程序用某种高级程序语言编写，而目标程序是用汇编语言或机器语言编写，如图 1.1 所示，那么这个翻译程序就被称为编译程序（compiler）。当然，在编译方式中，如果目标程序是汇编语言程序，那么还需由另一个称为汇编程序的翻译程序将它进一步翻译成机器语言程序。

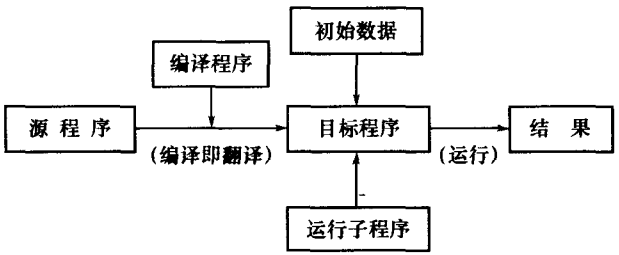

图 1.1　编译方式

一个高级语言的编译程序就是这个高级语言的翻译程序。每个计算机程序都表示了某种行为，都可以完成某些特定任务，人们对于编译过程的主要要求就在于源程序与目标程序的行为应该是完全相同的。也就是说，不管由一个高级语言程序（如一个 Pascal 程序）转换得来的机器语言程序的细节如何，这个高级语言程序对于相同的输入都应该产生相同的结果，而且执行机器语言程序的结果与执行原来高级语言程序的结果也应是完全一致的。这样，编译程序的存在就可以使高级语言独立于机器。程序员在使用高级语言编写程序时不需要考虑那些直接与机器有关的繁琐细节，而可以把这些工作交给编译程序去处理。

源程序是编译程序加工的对象，一个源程序往往可能被分解成几个模块文件。在编译系统中有一个预处理程序，负责将源程序的模块文件合并成一个文件，并且完成宏展开等任务。众所周知，人们不可能通过替换几个单词就能简单地把一种语言翻译成另一种语言，而必须首先分析源程序，弄清楚它的基本含义和结构，这个过程称之为“分析”（parsing）。然后，在这些结构上进行一次或多次转换，最终构成相应的目标程序。

从图 1.1 可知，编译方式中源程序的编译和目标程序的运行是分成两个阶段完成的。经编译所得的目标程序，计算机暂时还不能直接执行，必须由连接装配程序将目标程序和编译程序以及运行子程序连接成一个可执行程序，这个可执行程序才可直接被计算机执行。

由于编译方式具有上述这些特征，因此，很多高级语言，如 FORTRAN、ALGOL、Pascal、COBOL、C 和 C ++ 语言等等，均采用这种编译方式。

和编译方式不同，解释方式并不先产生目标程序然后再执行之，而是对源程序边翻译边

执行,如图 1.2 所示。按解释方式进行翻译的翻译程序称为解释程序。解释方式的主要优点是便于对源程序进行调试和修改,但其加工处理过程的速度较慢。BASIC 是一种交互式语言,它的程序是一种行结构,因而对 BASIC 程序宜采用解释方式。

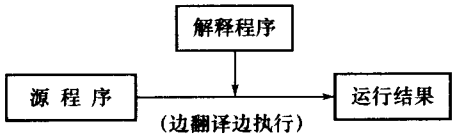

图 1.2 解释方式

1.2 为什么需要编译程序

所有的计算机程序归根到底都被转化成一串长长的二进制数序列,然后由计算机执行这个序列以完成某个任务。如果人们能够迅速且无差错地将某个任务的需求描述转换成二进制数序列,那么就不需要任何程序语言和编译程序了。遗憾的是,人们做不到这一点,而计算机做这种转换工作则十分合适,既快速又准确。也就是说,计算机(编译程序、解释程序)可以辅助人们进行程序设计。

人们在进行程序设计时难免出现各种错误,而计算机上的一些系统程序能够自动发现在程序中存在的错误,甚至能够在一开始就防止错误的发生。那些最有效的防错设施应该被合并到程序语言的设计中,一个高效的程序设计语言应该能够使程序员专心于编写一个简明、清晰、结构良好的算法描述,而不用顾及其他细节的考虑。不过需要指出的是,用高级语言编写的程序同一个熟练程序员用汇编语言编写的程序相比,执行效率会损失不少。因为高级语言给程序员强加了很多限制,比如控制结构的格式、带限制的数据格式等等,而这些在汇编语言中是不存在的。从另一个侧面来说,这种效率的损失也说明一种高级语言不可能对所有的目标机都很适合。比如 Pascal 和 PL/1 适合于采用栈结构的机器,而对于一个像 IBM370 这样的多寄存器机型,其编译程序就需要采取一些优化技术。

一个大型程序语言的编译程序的设计和实现与程序语言的复杂性及目标机的某些特性密切相关,编写一个编译程序少则需要 3 个人月,多则需要 30 人年。

编译程序总是针对相应的程序语言而设计的(比如:针对 Pascal 语言的编译程序 TURBO Pascal,针对 C 语言的 TURBO C)。程序语言、与之对应的编译程序以及其他一些组件共同构成了一个完整的设计环境,使得设计、构造大型、可靠的软件系统所需要的时间大为缩短。编译程序具有如下 5 个特点:

1. 模块性

为了更有效地编写和方便地调试程序,对于一些大型的系统,往往采用将其划分为较小

的程序块的方法来设计，这种较小的程序块也就是所谓的“模块”。模块的划分对于软件设计过程的意义在于：

（1）整个设计任务可以适当地分配给设计组的成员，可有效缩短整个任务的开发时间；

（2）承担设计任务的每个组员不必详细了解整个系统，只需注重他自己所负责的部分即可；

（3）编译时间可以按任务划分的比例而减少。

一个程序模块可以看做是一组变量说明及其相关过程的集合。其中一些属于内部变量和过程，具有隐藏性，不需要被外界所理解，它们只在模块内部起作用，不会被外部的操作所影响；而另一些则是可供外部使用的过程和变量，它们是输出型的，需要使用它们的用户可以对其定义和特点进行了解。正是由于外部变量和过程的存在，使得一个模块能够通过从其他模块引入其变量和过程的方式来援引这个模块。比较理想的编译程序应该具有单独编译某个模块的能力，同时，它不应因为源程序的一、两处简单修改就对整个源程序重新编译。

经过分划的系统最终要拼装成一个整体。在模块完成以后，首先由编译程序加工，将模块转换成对应的目标模块。所有模块都转换成目标模块后，再由连接装配程序（linking-loader）将目标模块连接在一起，从而产生一个完整的、可执行的目标程序。

2. 静态解释和动态解释

作为编译程序的设计者，应该了解到源语句序列（静态解释）和运行时的程序（动态解释）间的区别。

从理论上说，编译程序应该能发现并报告源程序的所有错误（既包括静态的词法、语法错误，也包括动态运行时可能出现的错误）。但由于编译程序总是工作在按静态意义解释的源语句序列上，它并不能真正“看”到源语句的动态执行情况，而推断一个程序的动态行为是很困难的。有些困难是由动态产生出的一些控制标志造成的，例如，WHILE 循环中通过一个 Boolean 条件的值来判断是否该终止循环，但这个条件可能依赖于某些变量，而这些变量又可能受 WHILE 循环中的某些语句的影响。所以，编译程序实际上只可能发现并报告在静态可计算性制约下的那些错误。

3. 机器无关性

现代计算机技术的飞速发展使得各种不同的机器结构层出不穷，这些结构一般是非标准化的。多样化的结构虽然有可能促进高性能、高性价比计算机的开发，但同时也给软件系统的移植带来了难题。

对于采用高级语言编写的软件系统，有可能采用重新编译的方法将其移植到另一台采用不同结构的计算机，而这样做的开销通常比重新用低级语言编写该软件要小的多，难度也小的多。这种可移植的特性被称为“可携带”，目前在可携带语言的研究方面已经取得明显进展。如果一个软件系统是可携带的，那么只需要极小的开销就可以使其在多个计算机系统上运行。

4. 语言标准化

程序的可携带性应该由语言本身在其设计阶段实现，而不应是编译程序的实现技巧。语言的定义应该标准化，应该与任何特定的计算机系统的特性无关，同时也不受任务编译程序实现者一时的灵感所支配。

程序语言的标准化可通过出版物、公认的标准文件或政府法规来实现。例如，Pascal 最初是由 Niklaus Wirth 为教学目的而设计的，在由 Jensen 和 Wirth 合写的“Pascal User Manual and Report”中存在不少的遗漏，这些遗漏后来被其他实现者用不同方式进行了修补，从而产生了几十种互不兼容的 Pascal 实现版本。现在使用比较广泛的是 Borland 公司的 Turbo Pascal。

程序设计语言的标准必须不涉及到任何特定的计算机特性才有实用价值。计算机的特性可以通过数据形式及其相关操作的某种组合延伸到程序中去。

5. 程序语言特征

编译程序编写者的任务是对其所接收的语言提供一种准确的实现，同时在可能的情况下也提供一个适当的程序设计开发环境。为此，需要先对程序语言具有的某些主要特征进行归纳，而其编译程序应该支持这些特征：

(1) 支持程序模块分划；
(2) 支持对所有语句位置和数据的符号访问；
(3) 提供结构化的控制和数据语句；
(4) 实施类型说明和变量使用之间的协调；
(5) 提供自动的类型转换；
(6) 在不同的计算机和操作系统间应该是可携带的。

1.3 编译程序的工作过程

编译程序的主要功能是将源程序翻译成等价的目标程序，这个翻译过程(也称编译过程)十分复杂，所以它一般首先分析源程序，然后综合成目标程序。编译程序在分析阶段检查源程序的正确性后，再将其分解为若干基本成分。这其中的工作也包括建立一些表格，改造源程序为中间语言程序，如图 1.3 所示。

符号表是源程序中所有标识符及其属性的集合。属性包括种类(如变量、数组、结构、函数、过程等)、类型(如整型、实型、布尔型、字符型等)以及目标程序所需的其他信息。常数表记载了常数的机内表示形式及分配给它的地址。中间语言程序是介于源程序和目标程序之间的一种中间形式的程序(中间代码)，编译程序首先将源程序转换成中间代码，再将中间代码转换成目标程序，中间代码的表示形式取决于编译程序下一步将如何使用和加工它。常用中间代码的形式包括逆波兰表示、三元式、四元式和树形表示等。一般来说，对于多趟

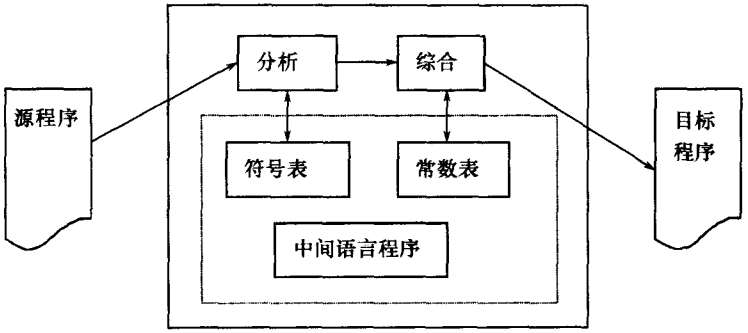

图 1.3　编译程序的工作过程

扫描的编译程序，越到后来的阶段所产生的中间代码在形式上越接近于目标程序。不过，中间语言程序对于编译程序来说并不是必需的。

1.3.1　分析

分析主要包括词法分析、语法分析以及语义分析三个部分。

1. 词法分析

词法分析是编译过程的基础，其任务是扫描源程序，根据语言的词法规则，分解和识别出每个单词，构造符号表、常数表以及将源程序转换成中间语言程序。单词是语言中最小的语义单位，如语言中的关键字(保留字或基本字)、标识符、常数、运算符和界限符等。

例如，有 Pascal 程序段如下：

```
sum := 0;
n := 1;
while n <= 100 do
  begin
    sum := sum + 1.0/n;
    n := n + 1
end;
```

在该程序段中，包含有关键字 while、do、begin、end，标识符 sum、n，常数 0、1、1.0、100，运算符 + 、/、<= ，界限符:=、;等，它们都是单词。

2. 语法分析

语法分析是在词法分析的基础上进行的，其主要任务是根据语言的语法规则检查源程序是否合乎语法。如不合乎语法，则进行相应的出错处理，如显示出错性质、出错部位等；若合乎语法，则分析上下文，以便将单词符号组合成各类语法单位，如单词组合成语句，语句组合成程序等。如上例中的程序段可分析出表达式 n <= 100、赋值语句 sum := 0，等等。

3. 语义分析

语义分析进一步检查合法程序结构的语义正确性,其目的是保证标识符和常数的正确使用,把必要的信息收集、保存到符号表或中间代码程序中,并进行相应的处理。例如,对于赋值语句

x := e;

需要检查表达式 e 与变量 x 的类型是否一致;如果不一致,编译程序将报告出错。

1.3.2 综合

综合部分根据符号表、常数表和中间语言程序生成目标代码,主要工作包括存贮分配、中间代码优化和目标代码生成。

1. 存贮分配

主要任务是为程序和数据分配运行时的存贮单元,这一工作并不是集中完成的,而是在词法分析、目标代码生成和运行阶段分别完成的,因此常常不把它作为一个单独的步骤列出。

2. 中间代码优化

中间代码优化通过调整和改变中间代码中某些操作的次序,以最终产生更加高效的目标代码(目标程序)。和中间代码生成一样,代码优化也不是编译的必经阶段。

3. 目标代码生成

这是编译程序的最后阶段。如果编译程序采用了中间代码,那么,目标代码生成阶段的任务则是将中间代码或优化之后的中间代码转换为等价的目标代码,即机器指令或汇编指令,这里需要利用到符号表、常数表中的信息。值得注意的是,目标代码依赖于具体计算机的硬件系统结构和指令系统。

1.4 编译程序的结构

上节已经说到,编译程序分为分析和综合两个部分,进一步可划分为词法分析、语法分析、语义分析(中间代码生成)、代码优化、存贮分配(分散在编译的某几个阶段执行)和目标代码生成程序。此外,编译程序还包括表格处理程序和出错处理程序。编译程序的典型结构如图 1.4 所示。

在编译过程中,源程序的各种信息需要保留在各种表格之中,在编译的各个阶段都需要查找或更新有关的表格。这个任务是由编译程序中的表格处理程序来完成的。出错处理程序负责发现源程序中可能出现的错误,并把错误报告给用户,指出错误的性质和发生错误的位置。为了最大限度地发现源程序中的错误,出错处理程序将错误所造成的影响限制在尽可能小的范围之内,使得编译程序能继续编译源程序的余下部分,以便编译一次能检查出

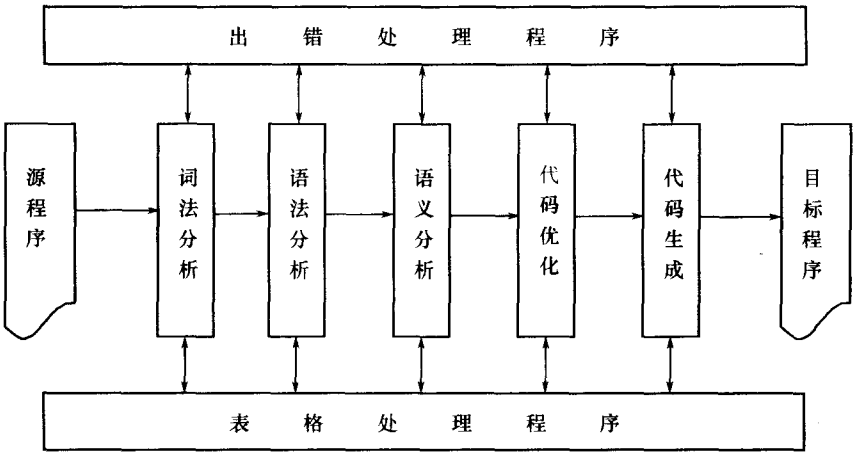

图 1.4　编译程序结构

尽可能多的错误。

这里划分出的几个步骤只是表示一个逻辑联系。实际上，编译过程往往分为前端和后端。前端包括词法分析、语法分析、语义分析、中间代码生成和中间代码优化，主要依赖于源程序；后端包括目标代码生成，依赖于计算机硬件系统和机器指令系统。这种组织方式便于编译程序的移植，若要将编译程序移植到不同类型的机器，只需修改编译程序的后端即可。

编译过程还可采用“分遍”的形式，即编译过程可以由一遍或多遍编译程序来完成。

对于源程序或中间代码程序，从头至尾扫描一次并完成所规定的工作称为一遍。在一遍中，可以完成一个或相连几个逻辑步骤的工作。例如，可以把词法分析作为第一遍，语法分析和语义分析作为第二遍，代码优化和存储分配作为第三遍，代码生成作为第四遍，从而构成一个四遍扫描的编译程序。也可以把每个逻辑步骤作为一遍或几遍完成，如可将“代码优化”分为“优化准备”和“实际代码优化”两遍进行，这种分法适合于存储空间小，要求高质量目标程序的场合。

一个编译程序是否需要分遍，如何分遍，通常根据下面的因素而定：宿主机的存储容量的大小；编译程序功能的强弱；源语言的繁简；目标程序优化的程度；设计和实现编译程序时使用工具的先进程度以及参加人员的多少和素质，等等。一般情况下，当源语言较繁、编译程序功能很强、目标程序优化程度较高且宿主机存储容量较小时，采用多遍扫描方式。分遍的好处是：多遍的功能独立、单纯；相互联系简单；逻辑结构清晰；优化准备工作充分并有利于多人分工合作。不足之处是不可避免地要做些重复性工作，且多遍之间有一定的交接工作，因而增加了编译程序的长度和编译时间。

一遍的编译程序是一种极端情形。在这种情形下，整个编译程序同时驻留在内存中，编译程序的各部门之间采用“调用转接”方式连接在一起。当语法分析需要新符号时，它就调

用“词法分析程序”；当识别出某一语法结构时，它就调用“语义分析程序”。语义分析程序对识别出的结构进行语义检查并调用“存储分配”、“优化”和“代码生成”完成相应的工作，如图 1.5 所示。

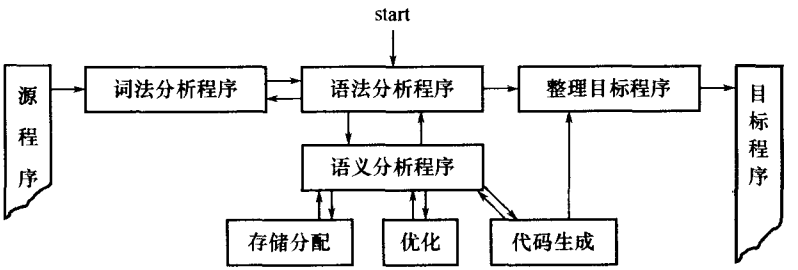

图 1.5 一遍编译程序

图 1.5 中把语法分析程序作为主程序。

在内存较小的计算机系统中实现功能很强的编译程序时，最好采用多遍方式，但对于某些仅具有慢速存取的中间存储器的小型计算机系统，配制多遍编译程序是比较耗时的。此外也应注意，并不是每种高级语言都可以用一遍编译程序实现的。

对多遍编译程序而言，每遍的输出结果是下一遍的输入对象，例如，设 L_{i-1} 是遍 $Comp_i$ 的编译对象，L_i 是其加工的结果，则有：

$$Comp_1(L_0)\rightarrow Comp_2(L_1)\rightarrow\cdots\rightarrow Comp_n(L_{n-1})=L_n$$

其中，L_0 是源程序，L_n 是目标程序，L_1、L_2、…、L_{n-1} 是中间代码程序。显然，中间代码程序被加工后不需要继续保留，因此在加工过程中，后一中间代码程序 L_i 可以覆盖前一中间代码程序 L_{i-1} 中已加工过的部分，如图 1.6 所示。

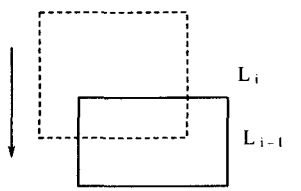

图 1.6 中间代码的覆盖

1.5 编译程序的组织方式

如果宿主机的内存空间有限，则需要外存辅助，在这种情况下，编译程序可以按照下面的方式组织。

首先在内存中开辟三个覆盖区（共享区）：一个存放最大的 $Comp_i$；一个存放输入对象（扫描对象）；一个存放加工结果（部分处理结果）。

然后在外存开辟编译程序的暂存区和输入对象/加工结果的存放区，并且由“编译总控程序”负责控制和调度编译程序的各遍，如图 1.7 所示。

当然，也可以由 OS 直接调入 $Comp_1$ 并启动它运行，$Comp_1$ 工作完毕后，自动调入 $Comp_2$ 并启动它运行，依次下来。同时也可以采用作业说明书的方式组织和控制编译程序的运行。