编译原理

蒋立源　主编

西北工业大学出版社

第一章 绪 论

程序设计语言是用来编写程序的工具。程序设计语言可分为两大类。第一类称为低级语言，包括机器语言、汇编语言以及其它面向机器的程序设计语言。这类语言对计算机的依赖性强、直观性差、编写程序的工作量大，只有对相应计算机的结构比较熟悉，且经过一定训练的程序人员才能较好地使用此类语言。第二类称为高级语言，有几百种之多，但除了一些专用语言之外，得到广泛运用的只有其中少数几种，如 BASIC、FORTRAN、ALGOL、PASCAL、COBOL、C 等。高级语言不论在对解题算法的描述能力上，还是在编写和调试程序的效率上，都远比低级语言优越。

然而，计算机硬件只懂自己的指令系统，即它只能直接执行用相应机器语言编写的代码程序，而不能直接执行用高级语言或汇编语言编写的程序。因此，要在计算机上实现除机器语言之外的任一程序设计语言，就首先应使此种语言为计算机所“理解”。解决这一问题的方法有两种：“一种是对程序进行翻译，另一种是对程序进行解释。

所谓翻译，是指在计算机中放置一个能为计算机直接执行的翻译程序，它以某一种程序设计语言（源语言）所编写的程序（源程序）作为翻译或加工的对象，当计算机执行翻译程序时，就将它翻译为与之等价的另一种语言（目标语言）的程序（目标程序）。“源”和“目标”这两个术语总是相对于一类特定的翻译程序和翻译过程而言的。如果一个翻译程序的源语言是某种高级语言，其目标语言是相应于某一计算机的汇编语言或机器语言，则称这种翻译程序为编译程序。汇编程序也是一种翻译程序，它的源语言和目标语言分别是相应的汇编语言和机器语言。

由此可见，欲按编译方式在计算机上执行用高级语言编写的程序，一般需经过两个阶段：第一阶段称为编译阶段，其任务是由编译程序将源程序编译为目标程序，若目标程序不是机器代码，而是汇编语言程序，则尚须汇编程序再行汇编为机器代码程序；第二阶段称为运行阶段，其任务是在目标计算机上执行编译阶段所得到的目标程序。在执行目标程序时，一般还应有一些子程序配合进行工作，例如：常见的数据格式转换子程序、标准函数计算子程序、浮点解释子程序、数组动态存贮分配子程序、下标变量地址计算子程序等等都属此类。这些子程序组成一个子程序库，称为运行系统。运行子程序库中的各个子程序，大都按模块化的结构来编制。显然，库中的子程序愈丰富，各子程序的功能愈强，编译程序本身就愈简明紧凑。

编译程序与运行系统合称为编译系统。

源程序的编译（或汇编）和目标程序的执行不一定在同一个计算机上完成。当源程序由另外的计算机进行编译（或汇编）时，我们将此种编译（或汇编）称为交叉编译（或汇编）。

图 1-1 粗略地显示了按编译方式执行一个高级语言程序的主要步骤。

用高级语言编写的程序也可以通过解释程序来执行。解释程序也以源程序作为它的输入，它与编译的主要区别是在解释程序的执行过程中不产生目标程序，而是解释执行源程序本身。这种边翻译边执行的方式工作效率很低，但由于解释程序的结构比编译程序简单，且占用内存较少，在执行过程中也易于在源程序一级对程序进行修改，因此一些规模较小的语言，如 BASIC，也常采用此种方式。然而就目前的情况来看，纯粹的解释程序并不多见，通常的做法是把

编译和解释作某种程度的结合。例如，有的先将源程序翻译为某种易于进行解释执行的内部中间语言形式，然后再对此中间语言程序进行解释执行；有的甚至在进行上述翻译时，还对一部分出现比较频繁的结构（如算术表达式等）产生目标代码。在采取上述这些措施后，解释程序执行效率不高的缺陷将有可能得到部分弥补。

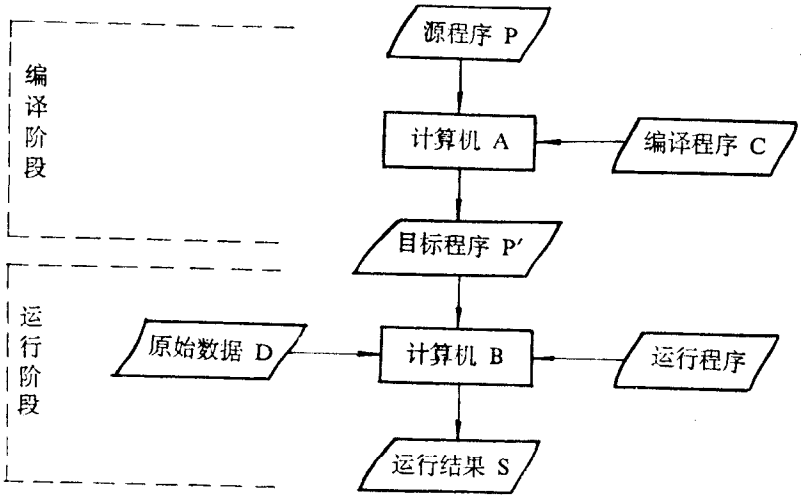

图 1-1　计算机执行高级语言程序的步骤

编译程序已成为现今任何计算机系统的最重要系统程序之一。本课程的目的，在于向读者介绍设计和构造编译程序的基本原理和基本方法，其中许多方法也同样适用于构造解释程序或汇编程序。事实上，任何一个熟悉编译程序构造的人，是不难旁通解释程序或汇编程序的工作原理和实现方法的。因此，限于篇幅，对于有关构造解释程序和汇编程序方面的问题，本书将不再涉及。

1.1　编译过程概述

编译程序的主要功能是把用高级语言编写的源程序翻译为等价的目标程序。既然编译过程是一种语言的翻译过程，因此我们就可将编译程序的工作过程与通常外语资料的翻译过程进行类比，这对于更直观地了解一个编译程序一般应由哪些部分组成，以及各个组成部分应如何进行工作等是有助益的。

抽象地看，任何一本外文资料都是由字母、标点符号（包括空格和其它符号）并按相应语法规则所组成的字符串。因此，任何欲进行外文翻译的人，都应具备如下能力：①能认识外语的字母及标点符号；②能识别出文中的各个单词；③会查字典；④懂得此种外语的语法；⑤具有目标语言的修辞能力。至于如何进行翻译，概括地讲无非是做两方面的工作；一是进行分析，二是进行综合。所谓分析，就是从第一行的第一个字母开始，依次阅读原文中的各个符号，逐个识别出原文中的各个单词，然后根据语法规则进行语法分析，即分析原文中如何由单词组成短语和句子，以及句子的种类特点等。此外，在识别单词和进行语法分析的过程中，还要不时地查阅字

典，做语法正确性的检查，进行相应的语义分析，并做一些必要的信息簿记工作等等。所谓综合，就是根据上述分析所得到的信息，拟定译稿，进行修辞加工，最后写出译文。

类似地，编译程序在其工作过程中，也需做两方面的工作，即先分析源程序，然后再综合为目标程序。为了便于理解编译程序在此两方面应包括的工作环节，现将源程序的编译和外文资料的翻译两过程的主要工作列表对比如图 1－2 所示。

	翻译外文书刊	编译源程序
分析	阅读原文 识别单词 分析句子	输入并扫视源程序 词法分析 语法分析
综合	修辞加工 写出译文	修饰优化 目标代码生成

图 1－2

尽管编译过程与外文书刊翻译的工作过程比较类似，但由于编译程序所翻译的毕竟不是自然语言，因此，就必然有其自身特有的一些工作。比如中间代码的产生，编译过程中信息表的构造与查询，以及运行时的存贮空间的分配，对语法和语义错误进行必要的处理等杂务工作。诸如此类的工作还有很多，兹不一一列举。

总之，编译程序是计算机的一个十分复杂的系统程序。为便于构造或分析一个编译程序，宜将整个编译程序分解为若干个组成部分，每一部分都用一段相对独立的程序去完成整个编译过程的一部分功能。就一个典型的编译程序而论，一般都含有下面八个部分：

1. 词法分析程序（也称为扫描器）；
2. 语法分析程序（有时简称为分析器）；
3. 语义分析程序；
4. 中间代码生成程序；
5. 代码优化程序；
6. 目标代码生成程序；
7. 错误检查和处理程序；
8. 各种信息表格的管理程序。

在下一节中，我们将简要说明上述各个部分的功能，并指出如何将这八个部分组成一个完整的编译程序。

1.2 编译程序的逻辑结构

在上一节中，我们已概括地介绍了编译程序的工作过程，并指出了一个典型的编译程序一般所包含的八个组成部分。图 1－3 表示了这八个部分间的控制流程和信息流程（分别用实线和虚线表示）。

下面，我们用一个微型 PASCAL 语言（PASCAL/M）所编写的程序为例，分别介绍这八个部分的功能，并分别给出每一部分对此程序进行加工处理可能得到的结果。我们假定此语言只有如下四种语句：

①PROGRAM 语句；

②说明语句；

③BEGIN－END 语句；

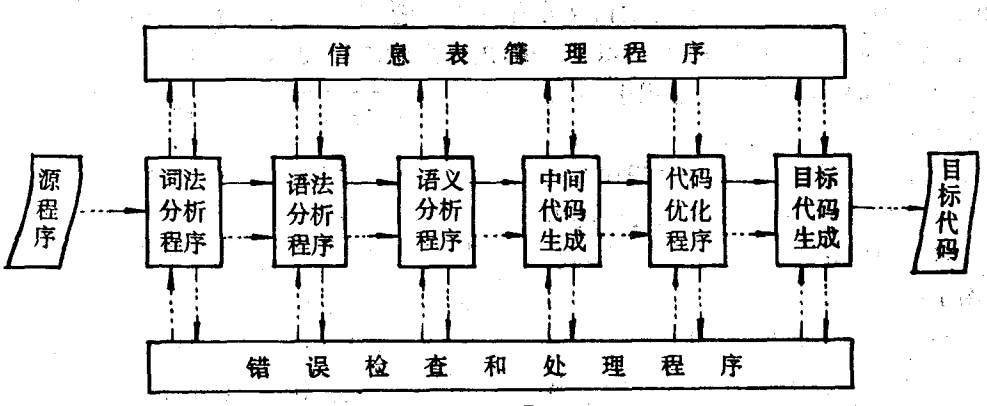

图 1-3 编译程序的逻辑结构

④赋值语句。

每一 PASCAL/M 程序都以一个 PROGRAM 语句开头，此语句中的标识符用来给程序命名；PROGRAM 语句之后是说明语句，用来指明程序中所出现的各个变量的数据类型(假定 PASCAL/M 中只有整型变量)；在一系列说明语句之后，再跟以一个 BEGIN - END 语句，在保留字 BEGIN 和 END 之间，应有一个或多个赋值语句，图 1-4 所示的 PASCAL/M 程序，对于我们后面的讨论来说，是一个恰当的例子。

```
PROGRAM source;
  {this little souroe program is used to
    illustrate compiling procedure.}
VAR x,y,z:integer;
    a:integer;
BEGIN
 {this program has only four statements.}
  x:=23+5;
  z:=x DIV -3;
  y:=z+18*3;
  a:=x+(y-2) DIV 4;
END.
```

图 1-4 一个 PASCAL 源程序 SOURCE

1.2.1 词法分析程序

作为编译程序的输入，源程序仅仅是一个长长的字符串，扫描器将把这种形式的源程序转换为便于编译程序其余部分进行处理的内部格式。扫描器的工作任务如下：

①识别出源程序中的各个基本语法单位(也称为单词或语法符号)；

②删除无用的空白字符、回车字符以及其它与输入介质相关的非实质性字符；

③删除注释；

④进行词法检查，报告所发现的错误。

现考虑图 1-4 所示的程序 source。扫描器依次查看缓冲区中源程序的各个字符，根据当

前正查看之字符的种类，并参考扫描过程中前面所得到的信息，就能准确地判断当前正扫描的字符在源程序中所处的地位。概括而言，不外下述五种情况之一：

①它是正处理的注释中的一个字符；

②它是一个无用的空白字符；

③它是下一个单词的首字符；

④它是正识别的单词中的一个字符；

⑤它是一个不合词法规则的字符，或是一个不属于本语言字符集中的一个字符。

显然，如果扫描器根据上述不同的情况作不同的处置，并产生预定形式的输出，那么，它就能圆满地完成上面所提到的四项任务。

图 1－5 给出了扫描器对程序 suorce 进行处理后的一种可采用的输出形式。其中，程序里的各个单词已被一一识别出来，并用一个特定的标志符号"＃"将相邻的两个单词加以分隔，程序中的非实质性符号已被全部删除。

```
# PROGRAM # source #；# VAR # x # ，# y # ，# z # ：# integer # ；# a# ：
# integer # ；# BEGIN # x # ：= # 23 # ＋ # 5 # ；# z # ：= # x # DIV# － # 3 # ；
# y # ：= # z # ＋ # 18 # ＊ # 3 # ；# a # ：= # x # ＋ #（# y
# － # 2 #）# DIV # 4 # ；# END # ．#
```

图 1－5　SOURCE 经扫描器处理后的一种输出

这里我们只是假定把源程序表示为某种意义上的规整形式。为提供给语法分析程序处理，则应设计更为合适的单词内部格式。一种经常使用的方法是用形如(Class，Value)的序偶(即二元式)来作为一个单词的内部表示。其中，Class 为一整数码，用来指示该单词的类别；Value 则是单词的值。

对于程序 source 而言，如果我们假定可将相应语言的单词符号分为四类：

1. 保留字；　2. 专用符号；　3. 标识符；　4. 整数。

而且，用数码 1，2，3，4 分别表示这四类单词的种类码，用单词内部编码表示相应单词的值，则 source 经扫描器处理后，所输出的用内部编码格式表示的单词符号串如下(为便于阅读，我们用单词符号本身的名字加单引号来表示它们的内部编码)：

```
(1,'PROGRAM')(3,'source')(2,';')(1,'VAR')(3,'x')(2,',')(3,'y')(2,',')(3,'z')
(2,':')(1,'integer')(2,';')(3,'a')(2,':')(1,'integer')(2,';')(1,'BEGIN')
(3,'x')(2,':=')(4,'23')(2,'+')(4,'5')(2,';')(3,'z')(2,':=')(3,'x')(1,'DIV')
(2,'-')(4,'3')(2,';')(3,'y')(2,':=')(3,'z')(2,'+')(4,'18')(2,'*')(4,'3')
(2,';')(3,'a')(2,':=')(3,'x')(2,'+')(2,'(')(3,'y')(2,'-')(4,'2')(2,')')
(1,'DIV')(4,'4')(2,';')(1,'END')(2,'.')
```

1.2.2　语法分析程序

语法分析程序以词法分析程序所输出的用内部编码格式表示的单词序列为输入，其任务是分析源程序的结构，判别它是否为相应程序设计语言中的一个合法程序。为了完成这种分析，一般的途径是由语法分析程序试着为其构造一棵完整的语法树。如果这种尝试成功，就表明该输入串在结构上的确是一个合乎语法的程序，否则，源程序中就必然存在语法错误。通常，这种语法树实质上是一个有标记的有序树形结构，它的叶子上的标记是程序中的各个单

词，而其内部结点（包括根结点）上的标记，则是程序设计语言的有关语法构造名，即语法范畴。

对源程序的语法分析工作是在相应程序设计语言的语法规则指导下进行的，语法规则描述了该语言的各种语法成分的组成结构。通常我们可以用所谓前后文无关文法（CFG）或与之等价的 Backus－Naur 范式（BNF）将一个程序设计语言的语法规则确切地描述出来，而且整个语法分析过程能够按照此种描述机械地进行。至于前后文无关文法的定义及其有关问题，我们将在第二章中详细介绍。

应当指出，上面所谓为源程序建立一棵语法树，仅仅是就概念上而不是就逻辑上而言的。在实际进行语法分析的过程中，并不一定真正为源程序建立一个树形的数据结构，也不一定需要对整个源程序产生一棵语法树，而是参照建立语法树的思路一步一步地进行语法分析。总之，语法分析是一个相当复杂的过程，在第四章里，我们将用相当大的篇幅来讨论有关语法分析方面的问题。

1.2.3 语义分析程序

上节我们对语法分析方面的一些问题作了初步的介绍。然而我们知道，对任何一种程序设计语言来说，它都具有两方面的特征，即语法特征和语义特征。前者用来定义语言各语法成分的形式或结构，后者则用来规定各语法成分的含义和功能，即规定它们的属性或在执行时应进行的运算或操作。因此，在编译过程中也需要对源程序进行语义分析。例如，对源程序 source，经语法分析后，虽然明确了它的组成结构，但并不知道其中所出现的一些量的属性和意义，更不知道各语法结构具有何种功能。但经过语义分析之后，就会得知各语法成分的含义和用途，以及应进行的运算和操作。

在进行语义分析的过程中，还应进行相应的语义检查，以保证源程序在语义上的正确性。通常，所需进行检查的项目是十分繁杂的。例如，对常用的一些语言来说，需要检查：在说明语句中是否有矛盾的类型说明；在表达式中，对某些运算符而言，是否有类型不匹配的运算对象；在过程调用中，实在参数与形式参数是否在个数、次序、种属等方面按相应语言的规定进行对应；诸如此类等等。由此可见，在语义分析过程中，语义分析程序也需要进行频繁的造表和查表工作。

由于程序语言的语义至今还没有找到一种公认的方法来系统地描述它们。在多数情况下，人们不得不采用一种半机械化的方法来解决语义分析方面的问题。当前比较流行的是一种所谓“语法制导翻译”的方法。这种方法把编译程序的语法分析和语义分析有机地组织起来，穿插地进行。

1.2.4 中间代码生成

为了处理上的方便，特别是为了便于代码的优化处理，通常在语义分析后不直接产生机器语言或汇编语言形式的目标代码，而是生成一种介于源语言和目标语言之间的中间语言代码。目前常见的中间代码形式有逆波兰表示、三元式、四元式及树形结构等等。如对于源程序 source 中的第四条赋值语句，其对应的逆波兰表示可以写成：

' a'' x'' y'' 2'' -'' 4'' div'' +'' :='

又如对于源程序 source，若采用四元式作为中间代码，则可表示为：

```
(prologue,' source')
(add,' 23',' 5', T)
(store, T,,' x')
(div,' x',' -3', T)
(store, T,,' z')
(mult,' 18',' 3', T)
(add,' z', T, T)
(store, T,,' y')
(sub,' y',' z', T)
(div, T,' 4', T)
(add,' x', T, T)
(store,' x', T, T)
(epilogue)
```

(请注意：对说明性语句，以后不产生目标代码)

其中 T 是由编译程序所指派的临时变量名。

中间代码的产生是与语义分析紧密相连的。但由于迄今对于程序语言的语义描述还没有一个公认的形式化系统，因此，对编译程序中间代码生成部分的设计，在一定程序上仍凭借经验来完成。对于采用语法制导翻译的编译程序，通常的做法是将产生中间代码的工作交给语义过程来完成。即每当一个语义过程被调用而对相应的语法结构进行语义分析时，它就根据此语法结构的语义，并结合在分析时所获得的语义信息，产生相应的中间代码，再把后者放到中间代码的序列中去。本书将采用此种处理方法。关于中间代码生成的有关问题，我们将在第五章中详细进行讨论。

1.2.5 代码优化程序

为了得到质量较高的目标代码，常常在中间代码生成和目标代码生成两个阶段之间，插入一个代码优化的处理阶段。这里所说的目标程序的质量，通常有两个衡量的标准：一个是目标程序所占用存贮空间的大小，即所谓空间指标；另一个是目标程序运行时所需的时间，即所谓时间指标。

一般说来，一个不进行优化处理的编译程序所产生的目标程序，其质量往往是比较低的。代码优化所涉及的范围很广。如果从与具体计算机的关系上看，可分为与机器无关的优化和与机器相关的优化。如果从与源程序的关系看，又可分为局部优化和全局优化。应当指出，对于一个进行优化处理工作的编译程序而言，虽然它在工作时可得到质量较高的目标程序，然而却以增加编译程序本身的时空复杂度和可靠性作为代价。另外，也有这样一些优化项目，它们在时间效率和空间效率上是相互矛盾的。故在设计一个编译程序时，究竟应考虑哪些优化项目以及各种优化项目进行到何种程序，应权衡利弊，根据具体情况而定。

例如，对于程序 source，经过优化后的中间代码如下：

```
(prologue,' source')
(store,28 ,' x')
(div,' x',' -3', T)
(store, T,,' z')
(add,' z',' 54', T)
(store, T,,' y')
(sub,' y',' 2', T)
(div, T,' e', T)
(add,' x', T, T)
(store,' x', T, T)
(epilogue)
```

其中，第二和第五个四元式就是由编译程序分别计算常数子表达式 23+5 及 18*3 之值，并

对原四元式进行相应替换后的形式。

1.2.6 目标代码生成程序

目标代码生成程序接受语义分析（或优化处理）之后所产生的中间代码，并结合在前面各阶段对源程序进行分析和加工所得到的有关信息，将中间代码翻译为机器语言或汇编语言形式的目标程序。如果对源程序的编译不设置中间代码生成阶段，则在语法和语义分析之后将直接产生目标程序。

由于目标程序总是按某一具体计算机的机器语言或汇编语言来产生的，因此，在设计目标代码生成程序时，首先需要确定源语言的各种语法成分（或中间语言各种结构）的目标结构，即确定源语言或中间语言的每一结构所对应的机器指令组成汇编语句组，我们称之为框架。框架在结构上比较固定，但其中也包含有某些待定部分，需要在生成具体的目标代码时，根据各语法成分在源程序中的确切形式和有关的参数加以确定。

在确定了各种语法结构的目标结构之后，接着就需要针对不同的情况，制定从中间代码到目标代码的翻译策略或算法，然后据此编写出目标代码生成程序。在制定此种策略或算法时，总的要求是所生成的目标代码有较高的执行效率。为此，就要做到：①使所生成的目标代码尽可能短；②充分发挥计算机可用资源的效率。例如，尽量使用执行速度快的指令，充分利用计算机的寄存器，以节省访问内存所用的时间等等。总之，编写目标代码生成程序这一部分工作对具体计算机的依赖性很强，工作内容也比较杂乱琐碎，在设计时需要仔细考虑，认真对待。

通常，目标代码可采用如下三种形式之一：

1. 具有绝对地址的机器指令代码。它们在内存中有固定的位置，编译程序在生成这种形式的目标代码之后，即可立即投入运行。

2. 汇编语言形式的目标程序。此种形式的目标程序需要经汇编程序汇编，以产生相应的机器代码。

3. 模块结构的机器指令，此种形式的目标程序一般具有浮动地址，需经连接程序将它们和另外一些运行子程序连接装配之后才能投入运行。

例如，对于 source，所生成的 8086 型汇编语言形式的目标代码如下：

```
Procedure/Function: SOURCE
        PUSH    BP
        MOV     BP, SP                  DEC   AX
        SUB     SP, 0004H               DEC   AX
        LCALL   INIFQQ                  SAR   AX, 1
L8:                                     JNS   $+3
        MOV     X, 001CH                ADC   AX, 0000H
L9:                                     SAR   AX, 1
        MOV     CX, FFFDH               JNX   $+3
        MOV     AX, X                   ADC   AX, 0000H
        CWD                             ADD   AX, X
        IDIV    CX                      MOV   A, AX
```

```
        MOV     Z, AX           L12:
L10:                            I3:
        MOV     XA, Z                   MOV  SP, BP
        ADD     AX, 0036H               POP  BP
        MOV     Y, AX                   LRET
L11:
        MOV     AX, Y
```

1.2.7 错误检查和处理程序

程序人员在编写程序时，错误是难免的。一个仅能处理绝对正确的源程序的编译程序并无实用价值。一个较完善的编译程序应具有广泛的程序查错能力，并能准确地报告错误的种类及错误出现的位置。同时，翻译程序还应具有一定的“校错”能力。

除报错外，编译程序还生成一些另外的注释性信息，这些信息将有助于程序人员调整其程序及编写程序说明书。例如，常见的两种辅助手段是根据请求打印“对照图”和卸出各变量之值。对源程序 source 而言，其对照图可以有如下的格式：

NAME	TYPE	REFERENCED IN LINE NUMBER
souroe	entry	1
x	integer	4, 8, 9, 11
y	integer	4, 10, 11
z	integer	4, 9, 10
a	integer	5, 11

由于在编译系统的各个部分都可能有程序诊断的问题，而且所涉及的内容或项目是如此的广泛，因此，目前还没有一种统一的方法能够系统地解决整个程序诊断方面的问题。通常的做法是，在编译系统的各个部分，视编译工作的进程和需要，分别于其中插入一些程序段落，来进行有关程序诊断方面的工作。这些程序段落的总体就组成了该编译系统的错误检查和处理程序，或诊断程序。

1.2.8 信息表管理程序

在编译过程中，需要经常收集、记录或查询源程序中所出现的各种量的有关属性（信息）。为此，编译程序需要建立或持有一批不同用途的表格（如常数表、各种名字表、循环层次表等等）。另外，视采用编译方法的不同，在编译过程中，还将保持一些专用的表格，如 LL 分析表，LR 分析表，状态矩阵等等。这些表格配合相应的编译算法进行工作，且在设计编译程序时即已造出。

一般而言，在编译过程的各个阶段，都必须进行频繁的造表和查表工作，而且这些工作将占去相当大的一部分编译时间。因此，合理地组织编译程序中的各种表格，并恰当地选用相应的造表和查表算法，实为提高编译程序工作效率的有效途径之一。

一般信息表的登记项由关键字和与之相关联的信息组成，其形式如图 1-6 所示。

名　字	信　息
⋮	⋮

图 1-6　信息表的结构

在编译程序中，造表和查表的工作系由一组专用的程序来完成，它们被分别安插在编译程序的有关部分，这一组程序就组成了相应编译程序的表格管理程序，在第六章中，我们将针对一些常见程序设计语言（如 FORTRAN、PASCAL）的特点，分别介绍符号表的组织以及相应的造表查表算法。

1.3　编译程序的组织

前面我们根据一个典型的编译程序所应具有的功能，将编译程序划分为八个组成部分，简要地介绍了各个部分应完成的基本工作，并指出了这八个部分间的相互关系，如图 1-3 所示。然而，需要注意的是，上面所说的各部分间的关系，是指它们之间的逻辑关系，而不一定是执行时间上的先后关系。事实上，可按不同的执行流程来组织上述各部分的工作，这在很大程度上依赖于编译过程中对源程序扫描的遍数，以及如何划分各遍扫描所进行的工作。此处所说的“遍”，是指对源程序或对其内部表示从头到尾扫视一遍，并进行有关的加工处理工作。例如，对于要求经一遍扫描就能完成从源程序到目标代码翻译的编译程序，我们可以语法分析程序为中心来组织它的工作流程，如图 1-7 所示。显然，由于整个编译程序只对源程序进行一次扫描，故不必产生中间代码。

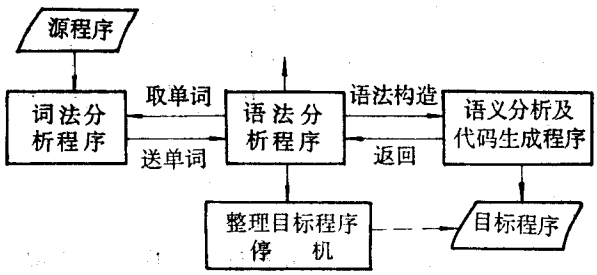

图 1-7　编译程序的工作流程

对于某些程序语言，例如 PASCAL，用一遍扫描的编译程序去实现比较困难，宜于采用多遍扫描的编译程序结构。具体的做法是将整个编译程序划分为若干个相继执行的模块，每一模块都对它前一模块的输出扫描一遍，并在扫描过程中完成前述八个部分的一个或几个部分，然后将工作的结果送入二级存贮器供下一模块加工。显然，第一模块所扫描的是字符串形式的源程序，最后一个模块所输出的是目标代码，每一中间模块输出的是与源程序等价的

内部表示或中间代码。

在设计一个编译程序时，如何确定遍数，如何组织各遍中的工作，主要取决于源语言的具体情况及编译程序运行的具体环境，如语言的结构，计算机各种软硬件的配置，以及对编译程序本身运行效率的要求等等。一般而言，多遍扫描源程序具有如下优点：

1. 由于采用了模块结构，各遍扫描的功能相对独立，整个编译程序的结构比较清晰；

2. 由于对源程序及其内部表示进行多次扫视和加工，有利于进行比较细致和充分的代码优化处理；

3. 由于编译程序可按模块逐次调入内存，有利于采用复盖技术，以减少编译程序所占用的内存空间。

由于分遍问题对具体语言及编译程序的运行环境有很强的依赖性，因此，在本书中，我们将不再详细讨论这一问题，仅在个别有关的地方稍微提及。

上面，我们简要地介绍了编译程序的功能、结构、工作流程以及有关的一些问题。由于编译程序进行翻译的对象是用高级语言所写的源程序，因此，我们将首先在下一章中讨论语言的描述或语言的语法定义问题，然后再按图 1－3 所示的结构，对一个编译程序进行剖析，并在后续的各章中分别介绍编译程序各个部分的构造原理和实现方法。

习　题

1－1. 何谓源程序、目标程序、翻译程序、编译程序和解释程序？它们之间可能有何种关系？

1－2　一个典型的编译系统通常由哪些部分组成？各部分的主要功能是什么？

1－3　选择一种你所熟悉的程序设计语言，试列出此语言中的全部关键字，并通过上机使用该语言以判明这些关键字是否为保留字。

1－4　选取一种你所熟悉的语言，试对它进行分析，以找出此语言中的括号，关键字 END 以及逗号有多少种不同的用途。

1－5　试用你常用的一种高级语言编写一短小的程序。上机进行编译和运行，记录下操作步骤和输出信息，如果可能，请卸出中间代码和目标代码。

第二章　前后文无关文法和语言

随着高级语言的出现和使用，必然会面临编译程序的设计和编译理论的研究。编译过程实际上是个十分复杂的信息加工过程，其加工的对象是用某种高级语言所编写的程序。因此，为了使编译工作有效进地进行，我们首先遇到的问题是如何确切地描述或定义一种程序设计语言，其次是如何识别或分析这种语言。自然，也还有其它方面的一些问题。

在本世纪50年代，N. Chomsky首先对语言的描述问题进行了探讨。在对某些自然语言进行研究的基础上，他提出了一种用来描述语言的数学系统，并以此定义了四类性质不同的文语和语言。人们把用一组数学符号和规则来描述语言的方式称为形式描述，而把所用的数学符号和规则称为形式语言。这里所说的“形式”，是指仅考虑数学符号间的推演，而不涉及符号的具体含义。

自1956年Chomsky建立文法的数学模型之后，对形式语言的理论和应用的研究，对与形式语言有密切关系的自动机理论的研究，都非常活跃且富有成果。其原因不仅是它们有效大的理论意义。更主要的还在于它们有较大的实用价值。尽管Chomsky所建立的文法的数学模型尚不足以描述自然语言，用它们来描述程序语言也并不完备（因为语言的语义部分至今尚未找到一种令人满意的形式化描述系统）。然而，所取得的研究成果不仅对编译理论，而且对诸如信息工程、人工智能以及数理语言学等领域均产生了深远的影响。目前，形式语言与自动机理论已成为计算机科学中的一个重要分枝，可以说：“在不了解语言及自动机理论的技术和结果的情况下，就不能对计算机科学进行严肃的研究”[6]。

本章将初步介绍形式语言中的某些基本概念和知识，重点是与编译技术密切相关的一些术语和概念，诸如文法、语言、句子、句型、短语、句柄以及句型的分析等等，以便为后续有关章节的学习打好基础。

2.1　文法及语言的表示

为了确切地描述一种语言，人们自然首先要问：什么是语言？为了回答这一问题，我们不妨从自然语言谈起。

随着人类社会的进化，人与人之间，这群人与另一群人之间的联系与交往日趋频繁，自然需要一种媒介或工具来传递和交流信息。在此情况下，作为这种工具的语言也就应运而生。据法国科学院的统计，目前在世界各地使用的语言竟达2 796种之多！因此，Webster曾把语言定义为：“为相当大地区的公众所懂得并使用的‘话’，以及组成这些‘话’的方法的统一体。”显然，这样的定义，对于确切描述程序语言，对于建立语言的数学理论的目的而言是不精确的，甚至是无用的。所以后来又有人将语言定义为：“某一字母表上符号串（句子）的集合。”此定义仍需精确化。这是因为：第一，尚须为所定义语言中的句子提供一种结构性的描述；第二，最好再提供一种手段，以便能准确地确定什么是该语言中的正确句子，而什么不是。

因此，如果我们有办法刻画一个语言的句子，也就定义了该语言。对于目前正使用的各种自然语言来说，上述目的还远远没有达到，因为从现今已有的各种阐述语法的书籍来看，不论它们的篇幅有多大，也不管它们的内容多么详尽，却都没有提供这样一种方法，利用它就能精确地描述该语言全部句子的结构。然而，对于绝大多数程序设计语言来说，此一问题已得到了很好的解决。例如，在1960年，P. Naur和J. Backus首先用BNF对ALGOL语言进行了描述。用它来描述一种程序语言时，尽管语义的描述还不得不借助自然语言，但它在程序设计语言的语法描述上，则具有足够的能力。以后，除非特别声明，所谓定义了某种语言，均指对其语法进行了定义，而置其语义于不顾。

一般而言，可视不同情况，采用如下三种方法来表示或定义一种语言。

1. 当一个语言仅含有有限个句子时，可采用枚举法来表示此种语言，即把该语言中的全部句子一一列举出来即可。例如，若一个语言L只含有如下的两个句子，则可将它表示为：

$$L=\{\text{I am a teacher, You are students}\}$$

然而，绝大多数重要的语言都是无穷的（即可以有无穷多个句子，请注意，这里所说的‘句子’，与程序语言中的‘语句’并非同一概念），对于它们，上述枚举法显然已经失效。因此，我们就面临着如何为无穷的语言寻找一种有限表示的问题。事实上，从“产生一种语言”和“识别一种语言”这两种不同的观点出发，我们可分别采用如下两种方案来进行无穷语言的有限表示。

2. 制定有限条规则，用来产生所需要描述的语言之中的全部句子（句子的数目可以有限，也可以无限），这些规则也就组成了稍后我们所要定义的文法。

3. 建立一种装置（更确切地说，是构造一种算法或过程），此装置以某一字母表上的所有符号串作为输入，并检验或识别这些符号串，当一个符号串是此字母表上某给定语言中的句子时，就接受它，反之，则拒绝接受。我们将此类装置称为自动机。以后我们将会看到，自动机也就是语言识别器或识别程序的抽象描述。

2.2 文法和语言的定义

在上一节中，曾把语言定义为某个字母表上符号串（句子）的集合。本节，我们将使这一定义精确化。为此，我们先引入某些今后常用的概念。

2.2.1 基本概念和术语

1. 字母表　字母表是由若干元素所组成的有限非空集合，其中，每一元素称之为符号，故有时又将字母表称为符号集。

符号是一个抽象的实体，仅在某些特定的使用场合，才分别赋予它们以具体的含义。因为符号是一个最基本的概念，所以无须再给以形式定义。

通常，在一个字母表中，可用阿拉伯数字、大写及小写英文字母、各种算术运算符、常用的标点符号以及使用上认为方便的其它符号来表示它的元素。例如，{a, b, c, +, ·}就是含有5个元素的一个字母表。

2. 符号串　用字母表中的符号所组成的任何有限序列称之为符号串（有时也称为符号行或字）。例如，设A={a, b, c}是一个字母表，则a, b, c, aa, ab, ac, ba, bb, bc, ca,

cc，cb，aaa 等等都是 A 上的符号串。一个字母表上全部符号串所组成的集合显然为一无限集。

我们把符号串中所含符号的个数称为该符号串的长度。例如，符号串 abc 的长度为 3，记为

$$|abc|=3$$

特别，我们把不包含任何符号的符号串称为空符号串，记为 ε。显然，$|\varepsilon|=0$。

如无特别说明，今后我们常用 a，b，c，…，及 S，T，U，…，X，Y，Z 等表示符号；用 t，u，…，x，y，z 等表示符号串，用 A，B，C，…等给符号串集合命名。此外，为了讨论问题方便，有时也给上述记号加下标。

3. 符号串的前缀、后缀及子串　设 x 是一个符号串，我们把从 x 的尾部删去若干个（包括 0 个）符号之后所余下的部分称为 x 的前缀。仿此，也可定义一个符号串的后缀。例如，若 x＝abc，则 ε，a，ab 及 abc 都是 x 的前缀；而 ε，c，bc 及 abc 都是 x 的后缀。若 x 的前缀（后缀）不是 x 本身，则称为 x 的真前缀（真后缀）。

从一个符号串中删去它的一个前缀和一个后缀之后所余下的部分称为此符号串的子串。例如，若 x＝abcd，则 ε，a，b，c，d，ab，bc，abc，bcd 及 abcd 都是 x 的子串。可见，x 的任何前缀和后缀都是 x 的子串，但其子串不一定是 x 的前缀或后缀。特别，ε 和 x 本身既是 x 的前缀与后缀，也是它的子串。

4. 符号串的连接和方幂　设 x 和 y 是两个符号串，如果我们将符号串 y 直接拼接在 x 之后，则称此种操作为符号串 x 和 y 的连接，记为 xy。例如，若 x＝NPU，y＝1 108，则 xy＝NPU1 108。而 yx＝1 108NPU。可见，xy 一般不等于 yx。

显然，空符号串 ε 与任何符号串 x 的连接还是 x 本身，即 $\varepsilon x=x\varepsilon=x$，因此，可将 ε 视为连接操作的单位元素。

我们不难将上述定义推广到任意多个符号串相连的情况。

一个符号串 x 与其自身的 n－1 次连接称为此符号串的 n 次方幂，记作 x^n，即

$$x^1=x,\quad x^2=xx,\quad x^3=x^2x=xx^2=xxx,\quad \cdots$$

$$x^n=x^{n-1}x=xx^{n-1}=\underbrace{xxx\cdots x}_{n\text{个}}$$

特别，我们定义 $x^0=\varepsilon$。

5. 符号串集合的和与积　设 A，B 为两个符号串的集合，则将集合 A 和 B 的和与积分别记作 A＋B（或 A∪B）及 AB，且定义为：

$$A+B=\{\omega\mid\omega\in A\text{ 或 }\omega\in B\}$$

$$AB=\{xy\mid x\in A\text{ 且 }y\in B\}$$

即集合 A＋B 中含有且仅含有 A 和 B 中的所有符号串，集合 AB 则由形如 xy 的所有符号串组成，其中 x∈A，且 y∈B。例如，若 A＝｛a，b，c｝，B＝｛00，11｝，则

$$A+B=\{a,\ b,\ c,\ 00,\ 11\}$$

$$AB=\{a00,\ a11,\ b00,\ b11,\ c00,\ c11\}$$

特别，若用∅表示空集（请注意，∅，ε 与｛ε｝三者间的区别），则显然有：

$$\varnothing+A=A+\varnothing=A$$

$$\varnothing A=A\varnothing=\varnothing$$

$$\{\varepsilon\}\ A=A\ \{\varepsilon\}\ =A$$

6. 符号串集合的方幂与闭包␣根据符号串集合的积运算，我们可定义符号串集合 A 的方幂运算如下：

$$A^0=\{\varepsilon\},\ A^1=A,\quad A^2=AA,\ \cdots,\quad A^n=A^{n-1}A=AA^{n-1}\qquad (n>0)$$

再根据符号串集合的和运算，我们又可分别定义符号串集合 A 的正闭包 A^+ 及自反传递闭包 A^* 如下：

$$A^+=A^1\cup A^2\cup\cdots\cup A^n\cdots=\bigcup_{i=1}^{\infty}A^i$$

$$A^*=A^0\cup A^1\cup\cdots A^n\cup\cdots=\bigcup_{i=0}^{\infty}A^i=\{\varepsilon\}\cup A^+$$

容易证明：符号串 $x\in A^+$，当且仅当存在某个 n，有 $x\in A^n$；符号串 $y\in A^*$，当且仅当：或者 $y=\varepsilon$，或者 $y\in A^+$。类似地，我们也可定义字母表上的和、积、方幂及闭包等运算。事实上，这只需把字母表中的每一元素均视为长度为 1 的符号串即可。

例如，若 $A=\{a, b, c\}$，则

$A^1=\{a, b, c\}$

$A^2=\{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$

$A^3=\{aaa, aab, aac, aba, \cdots, ccc\}$

……

$A^+=\{a, b, c, aa, ab, ac, ba, bb, \cdots\}$

$A^*=\{\varepsilon, a, b, c, aa, ab, ac, ba, \cdots\}$

可见，字母表 A 的正闭包 A^+ 就是 A 上所有符号串所组成的集合，而 A^* 仅比 A^+ 多含一个空符号串 ε。

2.2.2 文法和语言的形式定义

现在，我们从“产生语言”的角度出发，给出文法和语言的形式定义。所谓产生语言，如前所述，是制定出有限个规则，借助于它们，就能产生出此语言的全部句子。

为便于深刻理解定义文法和语言时所采用的方式，我们不妨以一个由某些英语句子所组成的语言为例来进行讨论。假定所要定义的语言中的每一个句子都具有“主—谓—宾”这样极为简单的结构，根据通常的英语语法知识，我们可首先将“句子”作为此语言的第一个语法实体，并用如下的语法规则加以描述：

① <句子>::=<主语短语><动词短语>

其中：每个用一对角括号“<”和“>”括起来的部分是所要定义语言中的一个语法实体（或称为语法单位、语法结构、语法范畴等等）；符号“::=”为一整体记号，其含义是：“…定义为…”，所以，规则①的含义就是：“语法范畴<句子>被定义为<主语短语>后跟一个<动词短语>”。然而，语法范畴<主语短语>及<动词短语>尚需进一步定义，根据英语语法，可再添加下面的语法规则：

② <主语短语>::=the<名词>

③ <动词短语>::=<动词><宾语短语>

仿此，我们可写出其余的语法规则如下：

④ <宾语短语>::=<冠词><名词>

⑤ <名词>::=monkey

⑥ <名词>::=banana

⑦ <动词>::=ate

⑧ <动词>::=has

⑨ <冠词>::=the

⑩ <冠词>::=a

下面，我们来说明：如何应用上述规则去产生或推出相应的语言，即推导出语言的全部句子。所谓推导出语言的句子，是指从语言最大的一个语法范畴（句子）开始，反复用语法规则中“::=”右边的符号串去替换它的左部符号，直到所得的符号串中不再包括需要替换的语法范畴为止。每使用某个规则替换一次，就说进行了一步直接推导，并用符号“⇒”表示这种替换操作。例如，对于句子 the monkey ate a banana，我们首先用规则①进行第一步推导，就得到了

<主语短语><动语短语>

记为 <句子>⇒<主语短语><动语短语>

所得的符号串中含有两个语法范畴，可对其中的任一个进行替换，例如使用规则③对<动词短语>进行替换，我们又得到

<句子>⇒<主语短语><动词短语>

⇒<主语短语><动语><宾语短语>

重复上述过程，我们便得到如下的推导序列：

推导步序	所用的规则	所得的符号串
1	①	<句子>⇒<主语短语><动词短语>
2	③	⇒<主语短语><动词><宾语短语>
3	②	⇒the<名词><动词><宾语短语>
4	④	⇒the<名词><动词><冠词><名词>
5	⑤	⇒the monkey<动词><冠词><名词>
6	⑦	⇒the monkey ate<冠词><名词>
7	⑩	⇒the monkey ate a<名词>
8	⑥	⇒the monkey ate a banana

上面的句子，是从<句子>出发，通过 8 步直接推导推出的，故将上述序列称为具有长度为 8 的推导。如不关心推导的中间过程，我们常把从一个语法范畴到一个符号串的推导用记号“$\overset{+}{\Rightarrow}$”表之。例如，对于上面的直接推导序列。我们把经过 5 步的推导记为

<句子>$\overset{+}{\Rightarrow}$the monkey<动词><冠词><名词>

而将经过 8 步的推导记为

<句子>$\overset{+}{\Rightarrow}$the monkey ate a banana

等等。

从上面所给的 10 条规则可以看出，一个语言中的同一语法范畴，有时会对应着若干条左