第 4 章　词法分析

编译器分为多个组件。从大的层面上分，分成编译器的前端和编译器的后端两个部分。这里的前后端和 Web 开发不同，编译前端指的是词法分析、语法分析和中间代码生成，即转化为中间代码及之前的内容都称为前端；编译后端指的是和目标机相关的部分，包括中间代码优化和目标代码生成与优化。

前端内容的核心是理解、分析输入的程序，转化成机器内部可以表示、执行的代码。其中第一步，称之为词法分析。词法分析，就是将输入的源程序转化为单词序列。在一个程序里，包括在人类语言里，单词是处理的不可分割的最小单元，即后续的处理以单词为单位进行。在后面语法分析和语义分析中，处理对象均为单词序列。

词法分析器又称扫描器，具体包括两个任务：

① 识别单词——从用户的源程序中把单词分离出来；

② 翻译单词——把单词转换成机内表示，便于后续处理。

词法分析流程如图4.1所示。

这一章的内容包括词法分析的基本概念、词法分析程序的设计与实现、算术常数处理机的设计与实现。本章对应的思维导图如图4.2所示。

4.1　词法分析的基本概念

4.1.1　单词的分类与识别

先简单讲一下概念，什么是单词。在自然语言，比如在中文里，中国是个单词，太阳是个单词。那在计算机高级程序语言中，什么是单词呢，这个的定义不是特别直接，从我们感性的认识出发，回顾学过的高级程序语言 C，C++ 等，if 是个单词，while 是个单词。可以看出无论是自然语言还是程序语言，**单词是承载语义信息的最小语法单位**。

单词按语法功能可分为标识符，常数、关键字和界符。常数包括算数常数，逻辑常数以及字符串常数等，

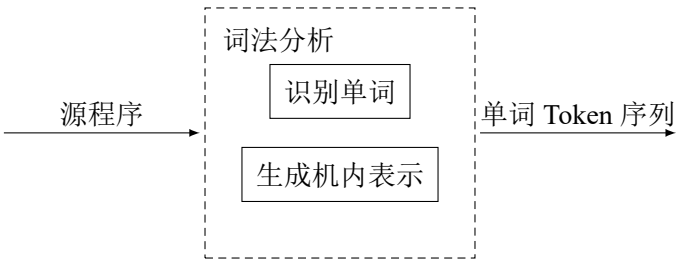


图 4.1: 词法分析流程图

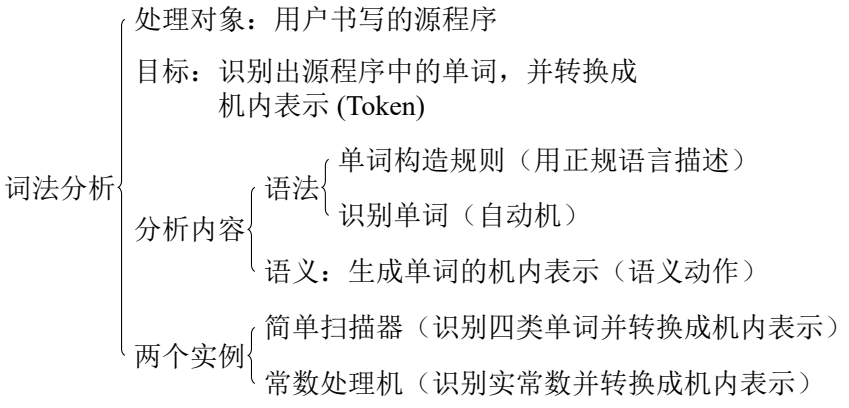


图 4.2: 词法分析思维导图

关键字指系统内部定义的，具有固定意义的，如在 C 语言中的 if、else、while 这些。界符则包括标点、算符，其中算符包括单字符算符、双字符算符。具体地，我们按单词的语法功能，将单词分为：

① 标识符——用户给一些变量起的名字；

② 常量——以自身形态面对用户和系统；

③ 关键字——系统内部定义，具有固定的意义，通常用来区分语法单元；

④ 界符——

(a) 单字符界符——+ − * / , ; : = < …

(b) 双字符界符——:= <= <> >= == /* */ …

(c) 其他

识别单词的基础是分析单词的构词规则。从单词的构词规则上看，如果字母开头，大概率是关键字或标识符，如果是数字开头，很有可能是数值型常量，如果以‘（单引号）或“（双引号）开头，则可能是字符型常量或字符串常量，如果是其他一些没有涵盖到的，如 + − */ 这类的，那它可能是界符或者其他形式的定义。不难发现，我们可以根据上述这些构词规则，区分出不同类型的单词。综上，**单词的识别主要包括以下四种情况**：（构词规则）

① 字母开头——关键字或标识符；

② 数字开头——数值型常量；

③ ‘或“开头——字符型常量或字符串常量；

④ 其它——多数以自身形态识别之，如 +，:=，…。

进一步，如何区分关键字和标识符？在早期，有的编译程序，强迫用户在输入关键字时，必须用某种特殊符号将其括起来；如：begin→#begin#，或书写时用黑体字，这种方式不友好。现在大多数编译程序使用“保留字”，即标识符不能用关键字。系统预先构建关键字表，拼好的字符串，先查关键字表，查到了，视为关键字，否则，视为标识符。

4.1.2 单词的机内表示

在计算机中如何表示单词，怎么从编译器的角度看这些单词呢？计算机中通常使用一些结构化、非常简单、易于存储和读取的方式。单词在机器中的表示并不复杂，有以下一些基本的要求。

第一，要求长短统一。比如：对于变量名 A 和 ABC，都是标识符，在编译器内对于这两个字符串，是怎么保存定位呢？假设不考虑结束字符，A 长度 1 个字符，ABC 长度 3 个字符，显然，不同的变量名，需要的

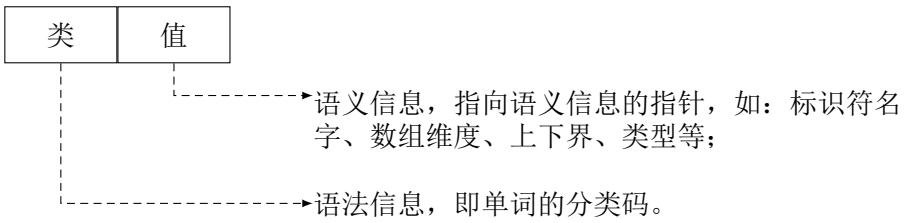


图 4.3: 单词 Token 表示法

存储长度不同。从计算机存储，数据结构组织的角度来看，变长的存储方式过于繁琐，一般来说，我们希望存储的形式是长度统一的，可以用一个固定的结构存储。

第二，要求语法、语义信息分开。这在当前阶段可能不太好理解，简单说来，语法和语义在一个单词的表示中是不一样的。在自然语言中，比如说，“打车”的“打”，“打”从自然语言角度看，语法上表示是一个动作，一个动词，而语义上则要根据“打”的具体使用来定，像“打车”的语义是招手拦车的一个事，如果是“打气”，指的是加油打气，如果是“打饭”，也不是“打车”里的意思，是指去食堂盛饭。在计算机中，每个单词也存在着语法和语义的功能。

由此可以看出，程序设计语言的单词不但类别不同，而且长短不一，机内表示可以使：

① 长短统一；

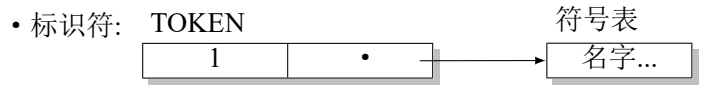
② 语法、语义信息分开。

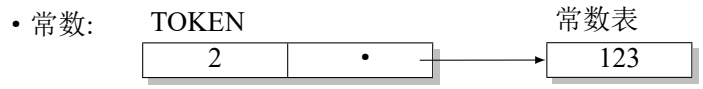
在词法分析器中，单词 Token 采用二元组的表示形式，一个是类，表示语法信息，也即分类码，一个是值，表示语义信息，实际上存储的不是具体值，而是一个指向符号表的指针，符号表中记录上下界、类型等信息，这部分内容会在后面进行详述。

我们先介绍 Token 的表示，如图4.3所示。

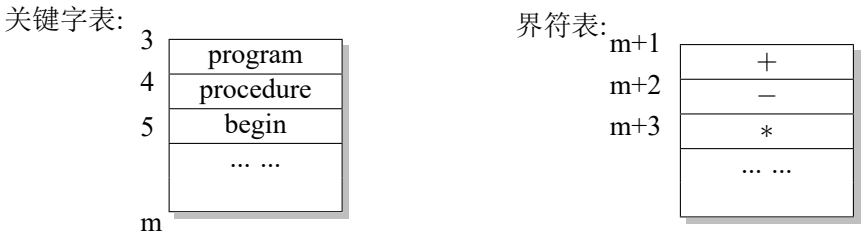
假如是一个标识符，对应的 Token，类码写 i，第二项的指针指向标识符表，标识符表其实是符号表内容的一部分，存储标识符的内容。常数对应的 Token 也是两部分构成，第一项是它的类码 c，第二项指向存储常数内容的表，我们称之为常数表。考虑到前面长度统一的要求，例如双精度和单精度表示长度不同，字符型和字符串型长度也不同，所以常数存储在常数表，而不是直接放在 Token 表示的第二项中。关键字对应的 Token，类码用 k 表示，第二项指向关键字表。类似地，对于界符对应的 Token，类码用 p 表示，第二项指向界符表。我们发现，不同 Token 的表示形式，除了类码外结构基本一致，其中指针分别指向对应的表。类码并不是严格规定的，可以自行设计，这里我们习惯用缩写进行定义，而在程序设计时，可以按照 1，2，3，…… 进行定义。

根据上述分析，给出各类单词对应的 Token 表示的详细设计方案：





其中：i——从 3 开始编码，一个单词一个码！习惯上，关键字先编，其余后编。



4.2 词法分析程序的设计

4.2.1 词法分析程序功能划分

词法分析器的实现，一般有两种方式，第一种方式是独立一遍的扫描器，是将整段程序，输入扫描器，得到 Token 序列，也就是单词串，如图4.4所示。

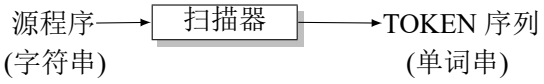


图 4.4: 独立一遍的扫描器

第二种方式是考虑下游系统，将扫描器作为语法分析器的子程序，如图4.5所示。此种方式在收到语法分析器发出的取单词指令后，从源程序中取一个单词并执行，然后将得到的 Token 返回给语法分析器，语法分析器会不断执行这个过程，最终得到整个分析的结果。换言之，Token 序列并没有在单步执行扫描器后，显性表示，而是在语法分析过程中逐步生成，我们把这种方式称为语法制导。在这种方式下，词法分析器不是独立的，而是作为依附于语法分析的一个装置，或一个配件。

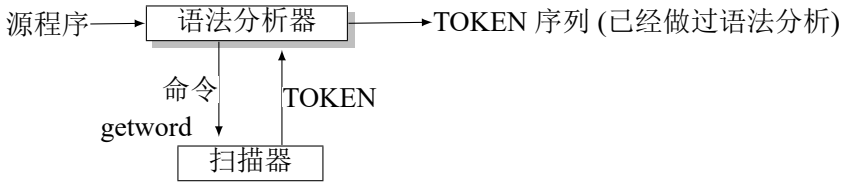


图 4.5: 作为语法分析器的子程序的扫描器

下面比较这两种方法的优缺点，第一种方式实现简单，词法分析和语法分析相互分离，耦合小，易于调试；第二种方式通过语法驱动，适合语法分析的方式，耦合大。语法驱动的方式读入一次源程序，可同时完成词法分析和语法分析，效率要高于第一种方式。

4.2.2 一个简单词法分析器的实现

词法分析器的核心由两部分构成，第一个是识别器，顾名思义，识别源程序中一个一个单词。第二部分是翻译器，根据识别器所识别出的对象，完成从单词串到单词的 Token 串的翻译。

分析单词的构词规则不难发现，单词可以用正规语言描述，字母和数字等是单词正规语言对应的终结符集，而符合构词规则的标识符、关键字和常数等便是该语言定义的符号串。例如，标识符对应的正规语言为 $L_1 = \{l(l|d)^n, n \geq 0\}$，其中 l 表示字母，d 表示数字。**有限状态自动机是识别正规语言的主要方式，因此构建识别器的核心是构造能够识别源程序中各种单词的有限状态自动机。**

翻译器是将识别器识别出的单词转换为统一的机内表示 Token。单词 Token 包括分类码和单词语义信息

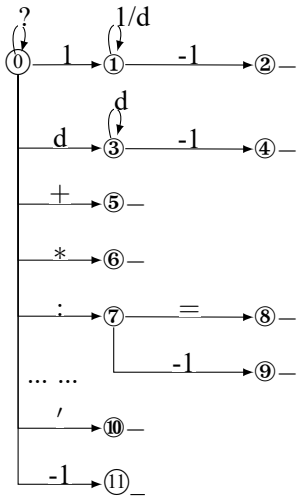


图 4.6: 类 Pascal 语言的 Token 扫描器

单词	编码
program	3
procedure	4
begin	5
end	6
while	7
do	8
+	9
*	10
:	11
:=	12
=	13
,	14
;	15

图 4.7: 关键字和界符表

两部分，因此翻译器的目标便是分析出单词的类别及其语义信息，值得注意的是，在词法分析中，我们获取单词的语义信息主要是单词名字，如标识符名字，其它语义信息将在后续符号表和中间代码生成环节中逐渐扩充。**明确了翻译器的目标（即任务），我们可通过为自动机的状态节点设计相应的语义动作函数，来生成单词的分类码和语义信息，即单词 Token。**

首先，我们设计一个简单的类 Pascal 语言的 Token 扫描器，如图4.6所示，该扫描器的目标是识别四类单词并转换成单词 Token。

下面先介绍自动机描述中的一些符号表示。第一，用“l”表示字母，用“d”表示数字，图中的“.”、“-”都是界符。事实上，整个符号表包括字母、数字和其他一些符号。我们假设用 # 表示程序结束，一旦遇到 #，表示程序结束。

第二，“?”表示空格、回车和换行，这三者构成我们常说的一些格式的变换。自动机的这一部分保证了会把和内容无关的空格、回车和换行过滤掉，不影响后面识别的结果。

第三，带有“-”号标记的状态是结束态，表示已经识别出一个单词。

第四，……表示省略了其他界符的处理。

接下来，我们从不同路径看一下上述的自动机实现的功能。

自动机中第一条路径，这一部分自动机识别出关键字和标识符。对于关键字和标识符，我们要求起始字符是字母，不能以数字等作为起始。我们现在使用的绝大多数编程语言都是这样约定的，目的是为了避免歧义。

自动机中第二条路径，没有小数点，识别出来的是无符号整数。

其余路径用于识别 +、*、:= 等界符。

给定一种语言以及该语言定义，就可以相应画出识别 Token 的自动机。

一个简单的词法分析装置如下，给定一个源程序字符串，通过识别器识别得到一串单词，通过为相应状态节点设计**语义动作**函数，可在识别出单词的同时把单词转化为机内表示 Token，也就是确定单词的类码以及语义。

在图4.6所示的 Token 扫描器中，**状态 2** 表明自动机已识别出一个以字母开头后面跟着字母和数字组成的单词，但需要进一步确定该单词是关键字还是标识符。因此，**状态 2** 的语义动作可表述为，查找关键字表，判断是否为系统保留字，如果是，就返回这个关键字所对应的 Token；如果没有查到，则在符号表中查找当前

标识符，看是否为已经定义的标识符，若找到则说明该标识符已经定义过，若没有则在符号表中增加这个新的标识符，并返回相应的 Token 表示。**状态 2** 对应的语义动作伪代码如下（即单词转换为 Token 的伪代码）：

```
1   IDorKeyToken (strTOKEN)                    //strTOKEN中保存的是已识别出的单词
2     begin
3       code:=Reserve(strTOKEN);               //查关键字表，查到时返回其编码，否则，返回0
4         if (code=0)                          //未查到，因此strTOKEN中的串为标识符
5           begin
6             value:=InsertID(strTOKEN);       //将strTOKEN插入符号表，返回当前位置
7             return(1,value);                 //生成一个标识符TOKEN
8           end
9         else                                 //查到了，因此strTOKEN中的串为关键字
10          return(code,_);                    //生成一个关键字TOKEN
11  end
```

其中：关于两个语义动作函数的说明如下，

- Reserve()：查 strTOKEN 中的字符串是否在关键字表中，查到了，返回其编码，否则，返回 0；
- InsertID()：用 strTOKEN 中的标识符查符号表，查到了，返回位置指针，否则，将此标识符插入到符号表中，并返回当前位置指针。

状态 4 表明自动机已识别出一个常数单词，图4.6给出的常数识别较为简单，只能识别是否为无符号整数。实际上还可识别出带小数点表示的浮点数，浮点数还可以通过科学计数法，用 e 指数的形式表示，形如 6.26e12。不难发现，采用科学计数法表示的浮点数其实是一个用字符串形式表示的常数，生成 Token 时需要将其转换为数值型常数，此种情况下，常数处理对应的也是一个自动机，常数处理是将一个符号串表达的常数，转化为计算机内部真正的表示，一个数值型常数，再填入常数表，得到的常数 Token 就会指向这个表。此处，先给出4.6中**状态 4** 对应的语义动作伪代码，关于字符型常数转换为数值型常数的处理，将在下一节做详细介绍。

```
1   ConstToken(strTOKEN);                      //strTOKEN中保存的是已识别出的单词
2     begin
3       value:=InsertConst(strTOKEN);          //将strTOKEN插入常数表，返回当前位置
4       return(2,value);                       //生成一个常数TOKEN
5     end
```

其中：关于语义动作函数的说明如下，

- InsertConst()：用 strTOKEN 中的常数查常数表，查到了，返回位置指针，否则，将此常数插入到常数表中，并返回当前位置指针。

状态 5、6、8、9、10 用于识别界符，相应的语义动作为：查找界符表，如果查得到，确定是界符，返回相应的 Token，如果查不到，则返回一个错误，表示输入程序出错。日常程序编译时报无法识别的错误，可能就是这样的原因。**状态 5、6、8、9、10 的语义动作**伪代码如下：

```
1    PuncToken(strTOKEN)              //strTOKEN中保存的是已识别出的单词
2      begin
3        code:=Bound(strTOKEN);       //查界符表，查到时返回其编码，否则，↩
             返回0
4        if (code=0)                    //未查到，因此是非法字符
5          ProcError();                 //错误处理
6        else                          //查到了，因此是界符
7          return(code,_);              //生成一个界符TOKEN
8      end
```

其中：关于语义动作函数的说明如下，

- Bound()：查 strTOKEN 中的字符串是否在界符表中，查到了，返回其编码，否则，返回 0。

基于上述分析，给出完整的词法分析器伪代码，详见本章附录 1。

综上，可以看出，设计一个简单的词法分析器主要包括三个步骤：

1. 分析单词的构词规则，明确单词的形式化描述方法。高级程序语言中单词可用正规语言描述；
2. 构造有限状态自动机，用于识别单词；
3. 根据词法分析的目标（将单词转换为机内表示），设计语义动作，并插入自动机的相应状态处。

4.2.3 词法分析示例

这里给出一个 Pascal 程序片段，下面给出采用上一小节介绍的词法分析方法，生成该程序片段对应的 Token 序列的详细过程。

Pascal 程序片段如下：

```
1    x1:=x1+1;
2    begin
3      yy:=5;
4      zz:=yy*x1;
5    end;
6    while (x1=20) do
7      x1:=yy;
```

首先，对于这样一段程序，我们需要关键字表，含有程序预定义好、具有特殊语义的一些关键字，如表中给出的 Begin，End 等，我们可以对关键字进行编码。需要界符表，包括识别出来的界符，如:;、:= 等，同样可以进行编码。还需要标识符表，保存变量名，到后续符号表的学习，会进一步了解变量类型，物理地址等。还有常数表，保存程序中出现的常数。其中，关键字表和界符表是在编译程序之前由系统设定好的，而标识符表和常数表则是在处理用户输入程序过程中动态变化的。

下面，根据这段程序进行分析。给定源程序后，通过图 4.6 所示的扫描器，也就是词法分析部分，对它进行处理。开始的时候，假设指针指向整个源程序的第一个符号 x，然后来看整个流程。在我们取到 x 后，先

符号表 I

x_1
yy
zz

常数表 C

1
5
20

图 4.8: 符号表和常数表

通过识别器，来识别这个符号，当前起始状态是 0，读入 x，进入 1 状态，再读入数字 1，也就是图中在 1 状态时读入表示数字的 d，仍保持在 1 状态，后面是一个空格，不是字母或数字，则进入状态 2，当前单词识别结束，得到一个识别好的字符串 x1，在状态 2，执行对应的语义动作 IDorKeyToken (strTOKEN)，以判断该单词类别并生成相应的 Token。语义动作 IDorKeyToken (strTOKEN) 的执行过程可描述为，先查图 4.7 所示的关键字和界符表，表中没有名为 x1 的关键字，接着判断是否为标识符表里的内容。这里需要注意，在刚得到 x1，进行查表的时候，标识符表里是没有 x1 信息的，需要在标识符表中填写 x1 的信息，由此完成这个 Token 的识别，最终得到的结果是 (1,I1)，其中 1 表示 x1 的类型是标识符，I1 表示指向标识符表第一项。

继续执行，识别得到界符:=，进入状态 8，并执行状态 8 对应的语义动作 PuncToken(strTOKEN)。通过查图 4.7 所示的关键字和界符表，确定是界符，对应的代码为 12，因此得到该界符对应的 Token 信息 (12,_)。

再向下执行，可以再识别得到一个 x1，进入状态 2，执行对应的语义动作 IDorKeyToken (strTOKEN)。这个 x1 的 Token 表示结果是 (1,I1)，因为在识别过程中，进行查表的时候，标识符表里已经有 x1，会直接返回表里存在的索引，不需要填表。以此类推，可以得到 + 的 token 表示是 (9,_)。得到常数 1 的 token 表示过程中，需要填常数表，得到 (2,C1) 的表示。以此类推，最终得到本节一开始给出的程序片段对应的标识符表、常数表如图4.8：

上述 Pascal 程序片段对应的 Token 序列：

$(1, I1), (12, \_), (1, I1), (9, \_), (2, C1), (15, \_),$
$(5, \_),$
$(1, I2), (12, \_), (2, C2), (15, \_),$
$(1, I3), (12, \_), (1, I2), (10, \_), (1, I1), (15, \_),$
$(6, \_), (15, \_),$
$(7, \_), (1, I1), (13, \_), (2, C3), (8, \_),$
$(1, I1), (15, \_), (1, I2), (15, \_)$

4.3 算数常数处理机设计

从上一节可以看出，我们在识别单词有限状态自动机中，加入语义动作函数，便可得到一个简单的词法分析器。给定源程序，经过词法分析器分析后便可得到一个 Token 序列。但是需要注意的是，在简单词法分析器中对于实常数的处理，只考虑了最简单的形式，即无符号整数。如果源程序中的实常数是用字符串来描述的，比如将 6.26e12 赋值给 a 的语句，6.26e12 在源程序中是一个字符串，而在机器处理时，需要将字符型实常数 6.26e12 转换为数值型实常数 6.26×10^{12}，这里面临的问题是如何将字符型的量转化为数值型的量，即生成字符型实常数的机内表示 Token。

同样的，字符型实常数可以用正规语言来描述，因此，字符型实常数的识别仍可采用有限状态自动机来识别，同时在自动机中加入负责数值转换的语义动作，形成常数处理机，从而完成字符型实常数到数值型实常数的转换。

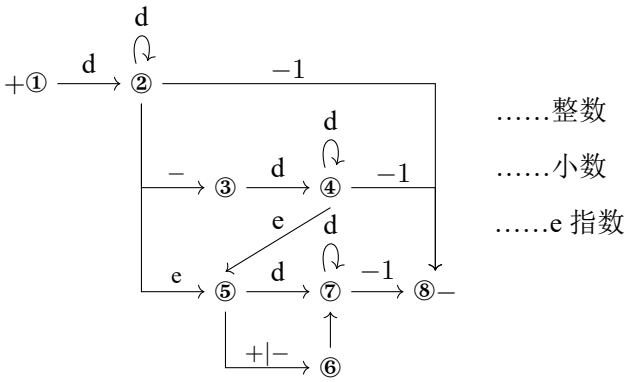


图 4.9: 识别实常数的自动机

4.3.1 识别器设计

一个实常数需要满足如下的形式，该形式不是唯一的表示形式，其中 e 指数部分是可选的。小数点前是整数部分，小数点后是小数部分。e 前面称之为尾数部分，e 后面称之为指数部分。

Pascal 常数的文法：画图　　例：128.67e-12

接下来，设计一个识别实常数的自动机，如图4.9所示。首先它的起始状态是 1，d 表示数字，“-1”泛指后继符，2 状态通过不断读入数字 d，来识别整数部分。我们约定至少要有一个数字，如果一个数字都没有，就只能停留在 1 状态，读入一个数字 d，就会到达 2 状态。对于 3 状态和 4 状态，这一路径用来识别小数，到 2 状态时是整数，读入一个小数点，跳转到 3 状态，此时知道这是一个小数，3 状态只能跳转到 4 状态，4 状态不断读入数字 d，作为小数部分，直到读到其他字符。如果不是 e，那么当前小数识别结束，如果是 e，就到达 5 状态，当然 5 状态也可以从 2 状态到达。表示指数的 e 出现在一个整数或者小数后边，是科学计数法的表示。对于 5 状态，或者走上面的路径，或者走下面的路径，下面的路径带加减号，表示幂指数的正负，如果不带加减号，读入数字 d，就默认指数部分是正数。到了 7 状态不断读入数字，作为指数部分，最后到达结束状态 8。

其中：“d”表示数字，“-1”泛指单词的后继符。

4.3.2 翻译器设计

从上一节介绍的简单扫描器可知，自动机可以识别出源程序中的单词，通过翻译器即相应的语义动作可完成单词到机内表示 Token 的转换。在常数处理机中也是同样的，通过图4.9所示的自动机可以识别出字符型表示的实常数，翻译器将其转换为统一的机内表示 Token。

单词 Token 包括分类码和单词语义信息两部分，因此翻译器的目标便是分析出单词的类别及其语义信息，在常数处理中，语义信息是指字符型实常数对应的数值型实常数。**明确了翻译器的目标（即任务），我们可通过为自动机的状态节点设计相应的语义动作函数，来生成单词的分类码和语义信息，即单词 Token。**

对于这一设计，我们约定，一个数由尾数和指数两部分组成，N 表示尾数部分数值，当读尾数部分时，读入一个数字，就将之前的尾数部分乘以 10，再加上当前读入 d 的数值；P 表示指数部分数值，和尾数部分计算方法类似，读入一个新的数字 d，就把当前的指数乘以 10 并加上 d 的数值，比如读入 8，0*10 再加上 8，就是 8，接着如果跟着 2，把 8 乘 10 再加上 2，就是 82。表示 N 时没有考虑小数点位置，我们还需要记录小数位数，用 m 表示小数位数，读入一位小数，就把 m 加 1。用 e 表示指数部分的符号变量，做一个简单的取值，正的用 1 表示，负的用-1 表示。综合以上，我们还能知道一个变量是整型或实型，用 t 表示是整型或实型，约定整数为 0，浮点数为 1。最终可得结果变量的计算公式：$NUM := N * 10^{e*P-m}$。

通过上述分析可知，为了计算常数值，引入了如下变量：

- N 表示尾数部分数值；
- P 表示指数部分数值；
- m 表示小数位数；
- e 表示指数部分的符号变量，指数为正用 1 表示，为负用-1 表示；
- t 表示是整型或实型，约定整数为 0，浮点数为 1。

为了完成转化的任务，我们在图4.9所示自动机的状态节点处设计语义动作函数，并假设在 i 状态结点处的语义动作为 q_i。

1. q_1：初始化，N:=P:=m:=t:=0；e:=1（表示默认是正数）；NUM:=0；
2. q_2：读入整数部分，N:=10*N+(d)，其中 (d) 表示将字符 d 转换为数字 d；
3. q_3：读入小数点，表示为小数，t:=1；
4. q_4：读入小数部分 N:=10*N+(d)，m:=m+1；
5. q_5：读入表示指数的 e，t:=1；
6. q_6：读入指数部分符号，如果读入负号，则 e:=-1；
7. q_7：读入指数部分，P:=10*P+(d)；
8. q_8：拼实常数，存入常数表并生成 Token，$NUM := N * 10^{e*P-m}$，Token=(2, C)，其中 C 为指向常数表中 NUM 的指针。

有了上述 8 个语义动作，我们执行前面的自动机，便能把字符串型的数值常量识别出来并自动翻译成一个计算机能够识别表示的数字。常数处理机的实现包括两部分，状态转换表设计和主控程序设计。根据图 4.9 所示自动机可以很容易得到状态转换表，如图 4.10(a) 所示。常数处理机主控程序则是在通用自动机主控程序上加入语义动作即可，即转换到某一状态时，要执行该状态对应的语义动作，如图 4.10(b) 所示，其中 Semfun(state) 表示执行状态 state 对应的语义动作。

接下来，通过一个具体的例子描述常数处理机的执行过程。假设待处理的字符串为 8.67e-12#，该字符串的处理过程如图 4.11 所示。

起始状态是 1 状态，执行 1 状态的语义动作 q1，进行初始化。读入数字 8，查状态表得知要变换到 2 状态。接下来，执行 2 状态的语义动作 q2，更新 N。指针向后移动，读入小数点，2 状态遇到小数点，经过查表，得知到达 3 状态。到达 3 状态，执行 3 状态的语义动作 q3，令 t=1，表示这是带小数的，接下来读入数字 6，3 状态读 6，到 4 状态。在 4 状态，执行 4 状态的语义动作 q4，处理尾数结果，并计算小数位数，根据计算公式，更新 n、m、p、e、t 的值，接下来，4 状态读入 7，仍然是 4 状态。这一步比较容易，同样更新 n、m、p、e、t 的值，再读入 e，4 状态读入 e，到达 5 状态。执行 5 状态的语义动作 q5，令 t=1，读入 “-”，进入 6 状态。执行 6 状态的语义动作 q6，令 e=-1。接下来读入数字 1，进入 7 状态。执行 7 状态的语义动作 q7，处理指数部分，根据公式更新 p 的值，读入数字 2，仍在 7 状态。同样执行 7 状态的语义动作 q7，更新 p 的值，最终读入 #，进入状态 8。执行语义动作 q8，得到数值转换结果 NUM=$867 * 10^{-14}$ 和机内表示 Token。

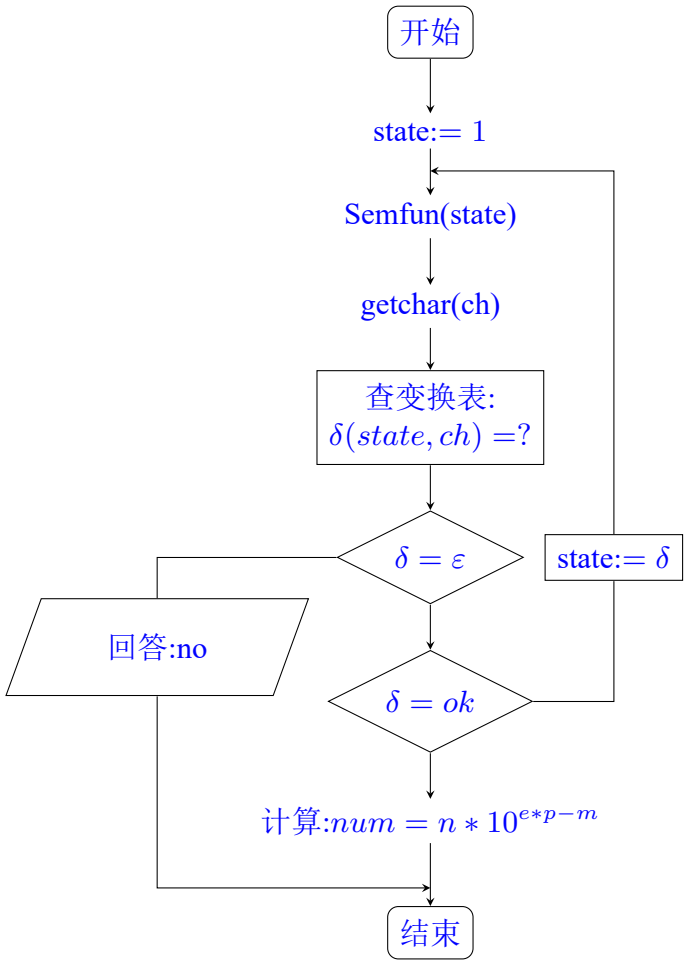
附录 1

完整的简单词法分析器伪代码如下：

＼	d	•	e	+|−	-1	
1	2	0	0	0	0	+
2	2	3	5	0	8	
3	4	0	0	0	0	
4	4	0	5	0	8	
5	7	0	0	6	0	
6	7	0	0	0	0	
7	7	0	0	0	8	
8	0	0	0	0	0	−

0—err

(a) 状态转换表



(b) 主控程序

图 4.10: 常数处理机状态表和主控程序

状态	翻译	n	m	p	e	t	符号	变换
1	q(1)	0	0	0	1	0	8	2
2	q(2)	8	0	0	1	0	.	3
3	q(3)	8	0	0	1	1	6	4
4	q(4)	86	1	0	1	1	7	4
4	q(4)	867	2	0	1	1	e	5
5	q(5)	867	2	0	1	1	-	6
6	q(6)	867	2	0	-1	1	1	7
7	q(7)	867	2	1	-1	1	2	7
7	q(7)	867	2	12	-1	1	#	ok

图 4.11: 常数处理过程示例

```
1int code,value;                                //TOKEN结构
2strTOKEN:=“ ”;                                 //字符数组，存放构成单词符号的字符串
3ch:=GetChar();                                 //读当前字符到ch
4ch:=GetBC();                                   //读一个非空字符到ch
5if (IsLetter(ch))                              // IsLetter(ch)---判断ch是否为字母
6    begin                                      //ch为字母，以下拼标识符或关键字
7    while (IsLetter(ch) or IsDigit(ch))        //拼单词
8        begin
9        Concat();                              //将ch中字符连接到strTOKEN中
10       ch:=GetChar();
11       end
12   Retract();                                 // Retract()---当前符号位置减1
13   code:=Reserve();                           //查关键字表，查到时返回其编码，否则，返回0
14   if (code=0)                                //未查到，因此strTOKEN中的串为标识符
15       begin
16       value:=InsertID(strTOKEN);             //将strTOKEN插入符号表，返回当前位置
17       return(1,value);                       //生成一个标识符TOKEN
18       end
19   else                                       //查到了，因此strTOKEN中的串为关键字
20   return(code,_);                            //生成一个关键字TOKEN
21   end
22else if (IsDigit(ch))                          //ch中不是字母，因此IsDigit(ch)---判断ch是否为数字
23   begin                                      //ch为数字，以下拼数字
24   while (IsDigit(ch))
25       begin
26       Concat();
27       ch:=GetChar();
28       end
29   Retract();
30   value:=InsertConst(strTOKEN);              //将strTOKEN插入常数表，返回当前位置
31   return(2,value);                           //生成一个常数TOKEN
32   end
33else                                           //判断ch是否是界符
34   begin
35   Concat();
36   if (ch=’:’)
37       begin
38       ch:=GetChar();
39       if (ch=’=’)
```

```
40              Concat();
41          else
42              Retract();
43          end
44code:=Bound();                  //查界符表，查到时返回其编码，否则，返回0
45if (code=0)                     //未查到，因此是非法字符
46     ProcError();                  //错误处理
47else                            //查到了，因此是界符
48     return(code,_);               //生成一个界符TOKEN
49end.
```