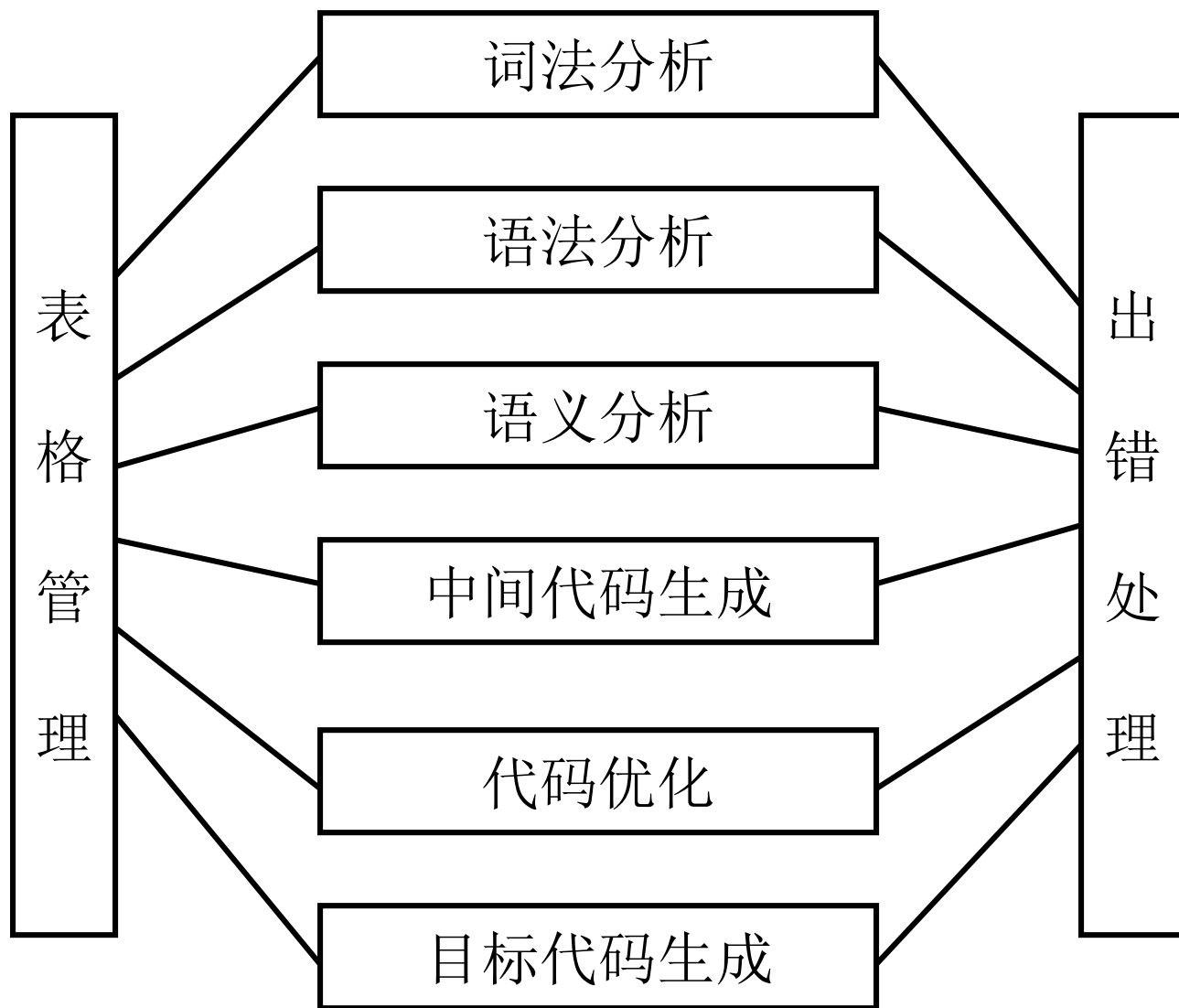


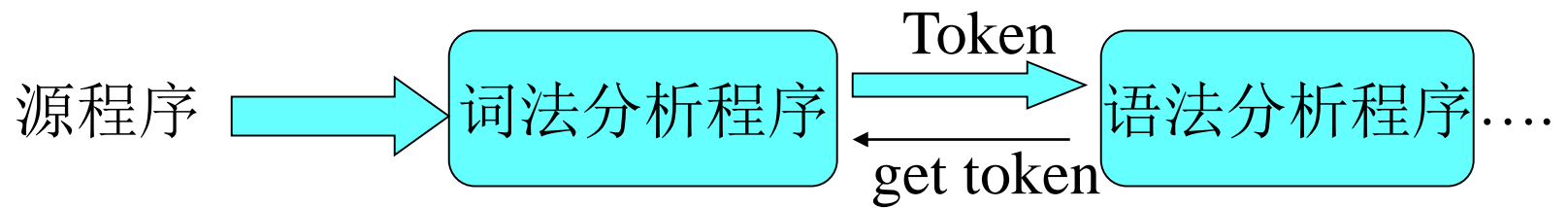
第3章 词法分析

词法分析



- 词法分析是编译过程中的一个阶段，在语法分析前进行。也可以和语法分析结合在一起作为一遍，由语法分析程序调用词法分析程序来获得当前单词供语法分析使用

词法分析程序和语法分析程序的关系



词法分析的主要功能

- **功能：分析输入的源程序，输出单词符号**
把构成源程序的字符串转换成单词符号的序列，并构造符号表, 记录其属性
- **单词符号(token)形式**
 - 按照最小语义单位设计
 - 通常表示为二元组：
(单词类别, 属性值) (类号, 内码)

词法分析程序的其他任务

- 过滤掉空格、跳过注释、换行符
- 追踪换行标志，
- 复制出错源程序，
- 宏展开，

单词符号的表示

- 常用单词类别

- 各关键字(保留字、基本字)，各种运算符，各种分界符——各用一个类别码标识
- 标识符——用一个类别码标示
- 常量(整数，实数，字符串等)——用一个类别码标示

- 属性(值)——单词的值

- 常数的值，标识符的名字等
- 保留字、运算符、分界符的属性值可以省略

词法分析中的表格

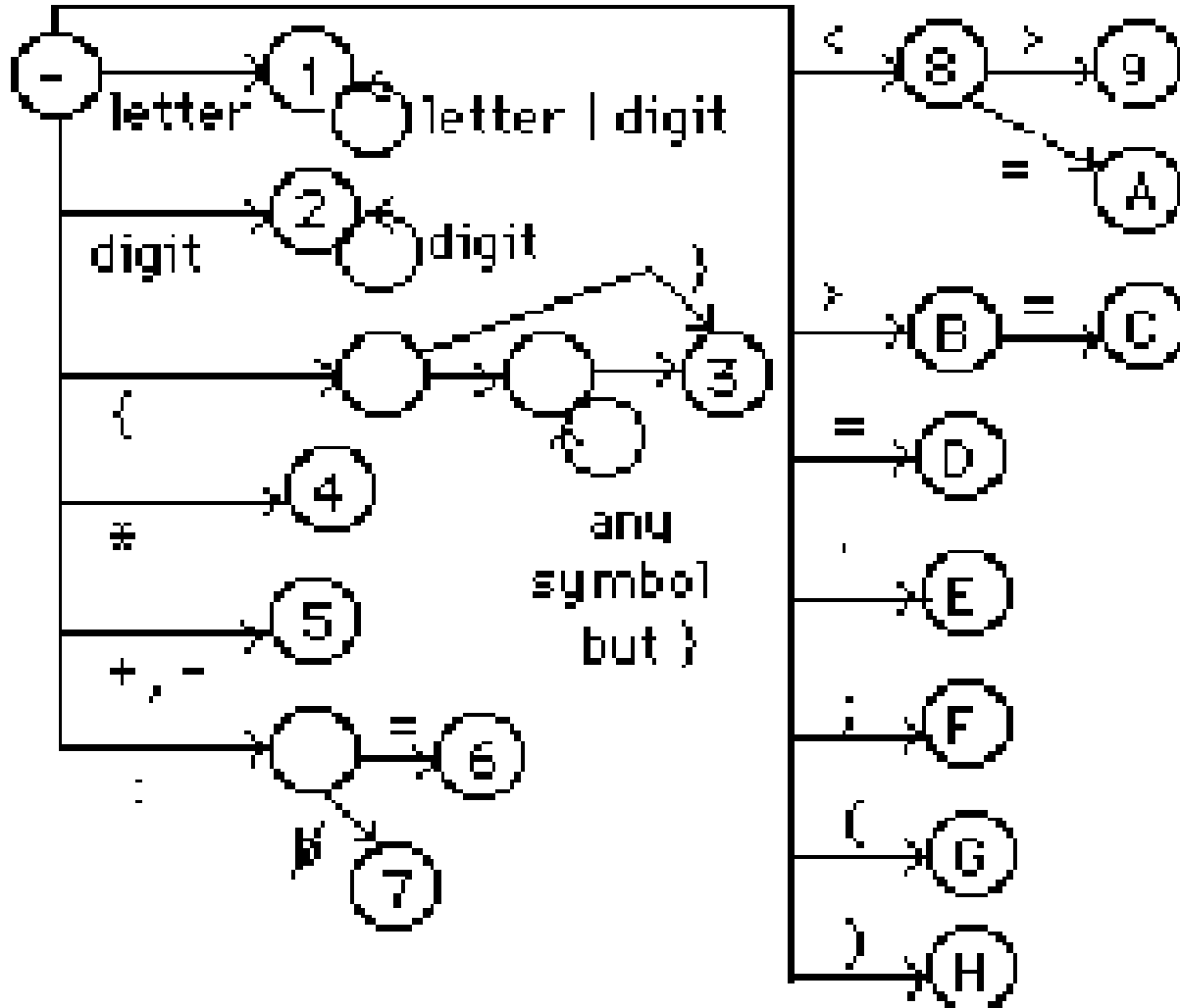
- 保留字表
- 常量表
- 标识符表*
- 自定义函数名表
- 标准函数名表
- 标号表

源程序示例

```
main( ) {  
    printf(“hello”);  
}
```

```
double p, q;  
double r, s, t, z;  
    // convert to canonical form  
r = b/a;  
s = c/a;  
t = d/a;  
p = s-(r*r)/3.0;  
q = (2*r*r*r)/27.0-(r*s)/3.0+t;  
double D, phi, R;  
R = bgSign(q)*sqrt(fabs(p)/3.0);  
D = pow(p/3.0,3)+pow(q/2,2);
```

```
while (j<=ir)
{
    if (j<ir && ra[j-1]<ra[j+1-1])
        j++;
    if (rra<ra[j-1])
    {
        ira[i-1] = ira[j-1];
        ra[i-1] = ra[j-1];
        i = j;
        j <<= 1;
    }
    else
        j = ir+1;
}
```



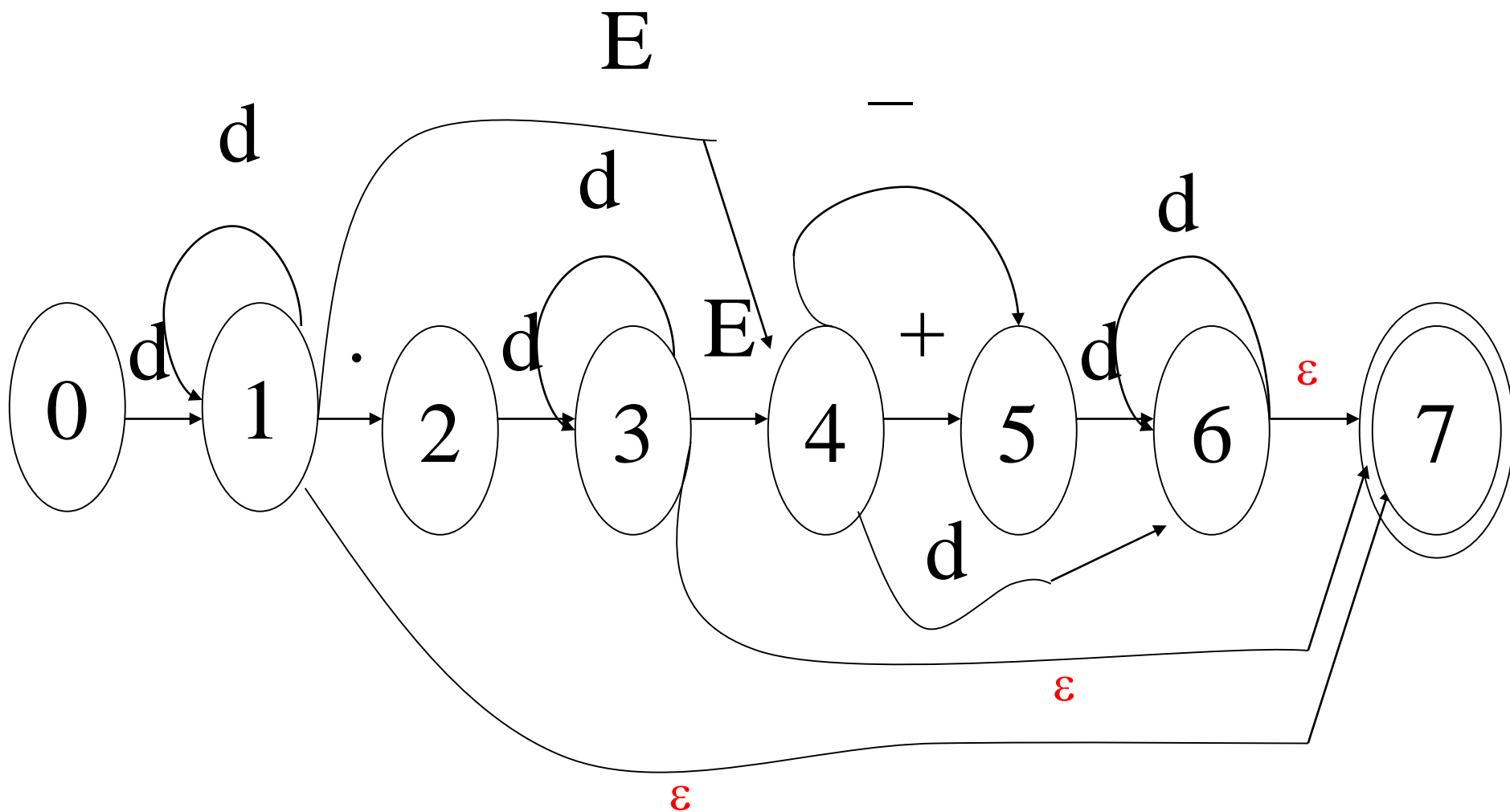
词法分析程序的构造

- 词法规则（正规文法）
- 正则表达式
- 自动机（状态转换图）
- 程序实现自动机状态转换

无正负号（实）数

- $dd^*(\bullet dd^* \mid \varepsilon)(e(+ \mid - \mid \varepsilon)dd^* \mid \varepsilon)$

$dd^*(\bullet dd^* \mid \varepsilon)(e(+ \mid - \mid \varepsilon)dd^* \mid \varepsilon)$



C语言风格的注释

- 自动机？

利用状态转换图识别单词

1. 从初态出发
2. 读入一字符
3. 按当前字符转入下一状态
4. 重复 2,3 直到无法继续转移

程序实现:

state=1;

while state=1,2,3,4 *do*

{ *case* state *of*

1: *if*(inputchar=='/') {getchar; state=2;}
 else {goto other state or error}

2: *if*(inputchar=='*') {getchar; state=3;}
 else {goto other state or error}

3: *if*(inputchar=='*') {getchar; state=4;}
 else {getchar}/*stay at state 3*/

4: *if*(inputchar=='/') {getchar; state=5;}
 else if (inputchar=='*') {getchar;}
 else {getchar; state=3;}

}

if(state==5) accept; *else* error;

状态转换矩阵

	/	*	other	终止状态
1	2			No
2		3		No
3	3	4	3	No
4	5	4	3	No
5				Yes

state=1;

ch=getchar;

while not accept(state) *and not* error(state) *do*

{

newstate=T[state][ch];

if not error(newstate) *then* ch=getchar;

else break;

state=newstate;

}

if accept(state) *then* OK;

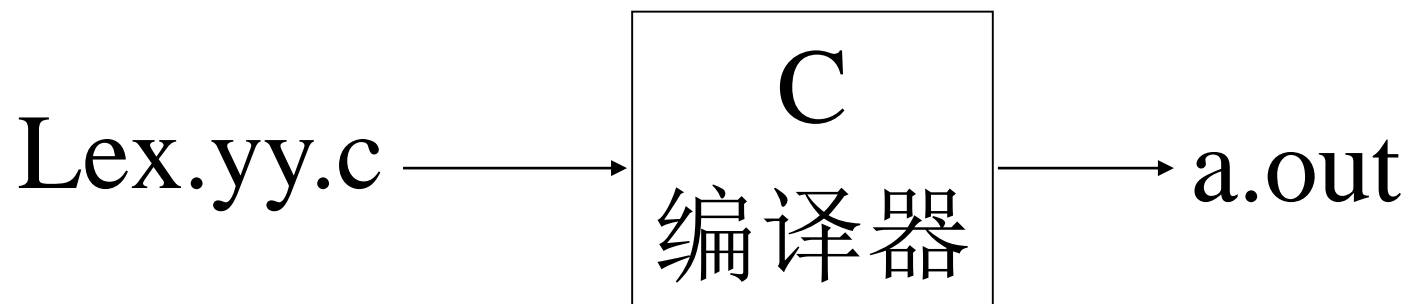
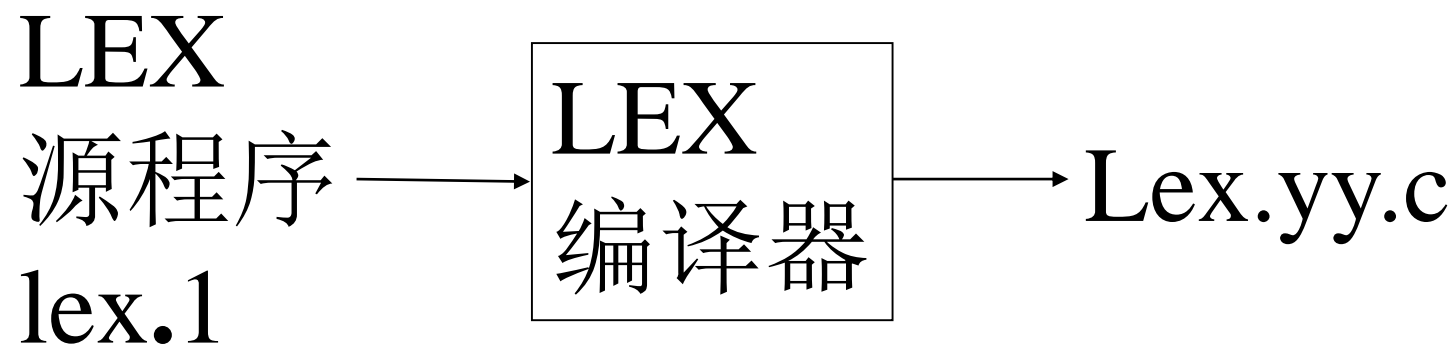
词法分析程序的自动构造工具LEX 简介

一、原理

单词的结构用正则式描述

正则式 \Rightarrow NFA \Rightarrow DFA \Rightarrow min DFA

二、用LEX建立词法分析程序的过程



三、lex源程序

- lex源程序由三部分组成
- 说明部分
- %%
- 翻译规则（转换规则）
- %%
- 辅助过程

$$D_1 \longrightarrow R_1$$
$$D_2 \longrightarrow R_2$$
$$\vdots$$
$$D_n \longrightarrow R_n$$

其中：

R_1, R_2, \dots, R_n 为正则表达式

D_1, D_2, \dots, D_n 为正则表达式名字，称简名

例：标识符：

letter $\rightarrow A|B|\dots|Z$

digit $\rightarrow 0|1|\dots|9$

iden $\rightarrow \text{letter}(\text{letter}|\text{digit})^*$

带符号整数：

integer $\rightarrow \text{digit}(\text{digit})^*$

sign $\rightarrow +|-|\epsilon$

signinteger $\rightarrow \text{sign integer}$

翻译规则的形式为：

• p_1 { 动作₁ }

• p_2 { 动作₂ }

•... ...

• p_n { 动作_n }

- 每个 p_i 是正则式的名字，每个动作 i 是正则式 p_i 识别某类单词时词法分析器应执行动作的程序段
- 用C书写
- 其基本动作是返回单词的类别编码和单词值

例：LEX 源程序

AUXILIARY DEFINITIONS /*辅助定义*/

letter \rightarrow A|B|...|Z

digit \rightarrow 0|1|...|9

% %

RECOGNITION RULES /*识别规则*/

1.BEGIN {RETURN(1,—) }

2.END {RETURN(2,—) }

3.FOR {RETURN(3,—) }

RETURN是LEX过程，该过程将单词传给
语法分析程序 RETURN (C, LEXVAL)

- 其中C为单词类别编码，
- 而LEXVAL：
 - 标识符：TOKEN（字符数组）
 - 整常数：DTB
 - 数值转换函数，将TOKEN中的数字串转换二进制
 - 其他单词：无定义

4.DO	{RETURN(4,—) }
5.IF	{RETURN(5,—) }
6.THEN	{RETURN(6,—) }
7.ELSE	{RETURN(7,—) }
8.letter(letter digit)*	{RETURN(8,TOKEN) }
9.digit(digit)*	{RETURN(9,DTB) }
10. :	{RETURN(10,—) }
11. +	{RETURN(11,—) }
12. “*”	{RETURN(12,—) }

13. ,	{RETURN(13,—) }
14. “ (“	{RETURN(14,—) }
15. “) ”	{RETURN(15,—) }
16. :=	{RETURN(16,—) }
17. =	{RETURN(17,—) }

- **辅助过程**是动作需要的，这些过程用C书写，可以分别编译并置于词法分析器中


```
%{
/*  Lex program to convert uppercase to
    lowercase except inside comments
*/
#include <stdio.h>
#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
```

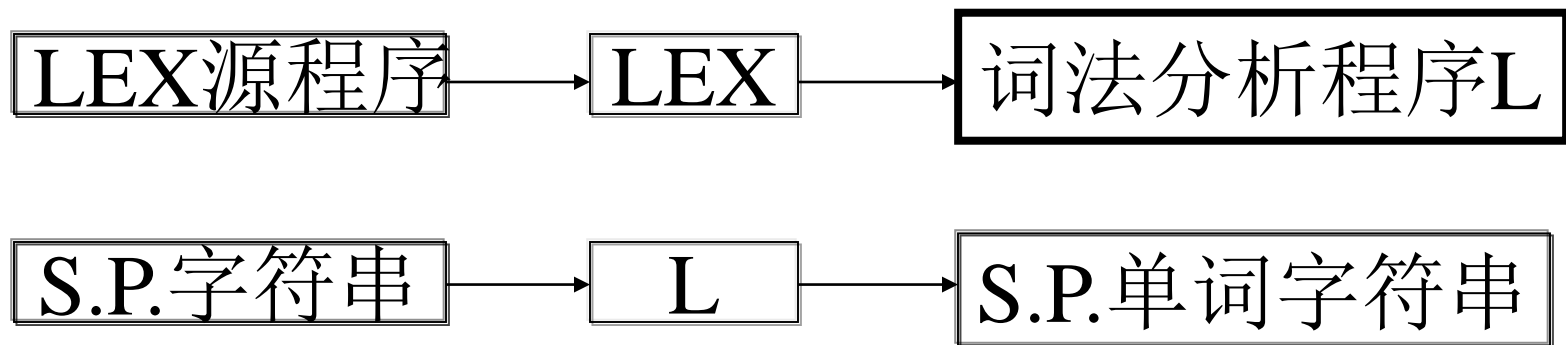
```
#define TRUE 1
#endif
%}
%%
[A-Z]  {putchar(tolower(yytext[0]));
        /* yytext[0] is the single
           uppercase char found */
      }
"/*"   { char c ;
        int done = FALSE;
        ECHO;
        do
        { while ((c=input())!='*')
            putchar(c);
            putchar(c);
            while((c=input())=='*')
                putchar(c);
            putchar(c);
            if (c == '/') done = TRUE;
        } while (!done);
      }
%%
void main(void )
{ yylex();}
```

表2-3 一些Lex内部名字

Lex 内部名字	含义/使用
<code>lex.yy.c</code> 或 <code>lexyy.c</code>	Lex 输出文件名
<code>yylex</code>	Lex 扫描例程
<code>yytext</code>	当前行为匹配的串
<code>yyin</code>	Lex 输入文件 (缺省: <code>stdin</code>)
<code>yyout</code>	Lex 输出文件 (缺省: <code>stdout</code>)
<code>input</code>	Lex 缓冲的输入例程
<code>ECHO</code>	Lex 缺省行为 (将 <code>yytext</code> 打印到 <code>yyout</code>)

资料来自于：

K. C. Loudon, 编译原理及实践, 冯博琴译,
机械工业出版社



LEX的实现

LEX的功能是根据LEX源程序构造一个词法分析程序，

该词法分析器实质上是一个有穷自动机。

LEX生成的词法分析程序有两部分组成：

词法分析程序

状态转换矩阵
(DFA)

控制执行程序

∴LEX的功能是根据LEX源程序生成状态转换矩阵和控制程序

LEX的处理过程:

. 扫描每条识别规则 P_i 构造一相应的非确定有穷自动机 M_i

.. 将各条规则的有穷自动机 M_i 合并成一个新的NFA M

...确定化 $NFA \Rightarrow DFA$

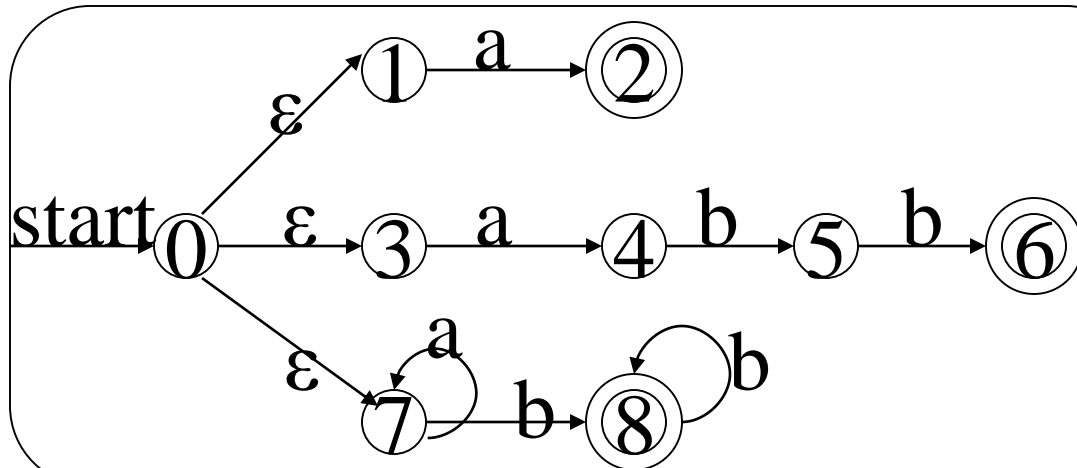
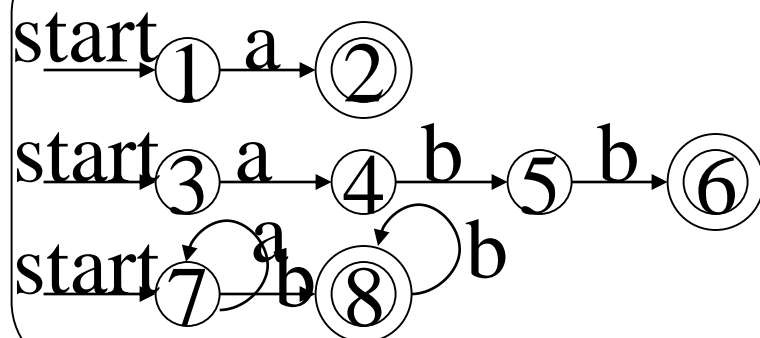
即生成该DFA的状态转换矩阵和控制执行程序

例： LEX源程序，

a { }

abb { }

a* bb* { }



状态		a	b	所识别的单词
初态	{0,1,3,7}	{2,4,7}	{8}	a
终态	{2,4,7}	{7}	{5,8}	
终态	{8}	\varnothing	{8}	
	{7}	{7}	{8}	a*bb*
终态	{5,8}	\varnothing	{6,8}	
终态	{6,8}	\varnothing	{8}	abb

- LEX是一个通用的工具，用它可以生成各种语言的语法分析程序，只需要根据不同的语言书写不同的LEX源文件
- LEX不但能自动生成词法分析器，而且也可以产生多种模式识别器及文本编辑程序等

词法分析技术其他应用

- 查询语言, 信息检索系统
 - 识别由正规式描述的字符串
- 命令语言
 - 识别命令格式
- 报文的处理
 - 识别报文格式
- 应用范围:
 - 数据格式可以用三型文法描述

Questions

- 如何编程实现NFA2DFA?
- 如何编程实现DFA最小化?