

编译原理与技术

第3章：词法分析

王吴凡

北京邮电大学计算机学院

主页：cswwf.github.io

邮箱：wufanwang@bupt.edu.cn

词法分析

- 词法分析任务由词法分析程序完成
- 词法分析程序的作用：
 - 扫描源程序字符流
 - 按照源语言的词法规则识别出各类单词符号
 - 产生用于语法分析的记号序列
- 词法检查
- 与用户接口的一些任务：
 - 跳过源程序中的注释和分隔单词的空格
 - 记录单词符号在源程序中的位置
- 创建符号表(需要的话)
把识别出来的标识符插入符号表中

教学内容、目标与要求

■ 教学内容

- 词法分析程序与语法分析程序的关系
- 词法分析程序的输入与输出
- 记号的描述与识别
- 词法分析程序的设计与实现

■ 教学目标与要求

- 掌握单词符号的形式化描述和识别
- 掌握词法分析程序的设计与实现方法

内容目录

3.1 词法分析程序与语法分析程序的关系

3.2 词法分析程序的输入与输出

3.3 记号的描述和识别

3.4 词法分析程序的设计与实现

3.5 软件工具 LEX (自学)

小结

3.1 词法分析程序与语法分析程序的关系

■ 三种关系：

- 词法分析程序作为独立的一遍
- 词法分析程序作为语法分析程序的子程序
- 词法分析程序与语法分析程序作为协同程序

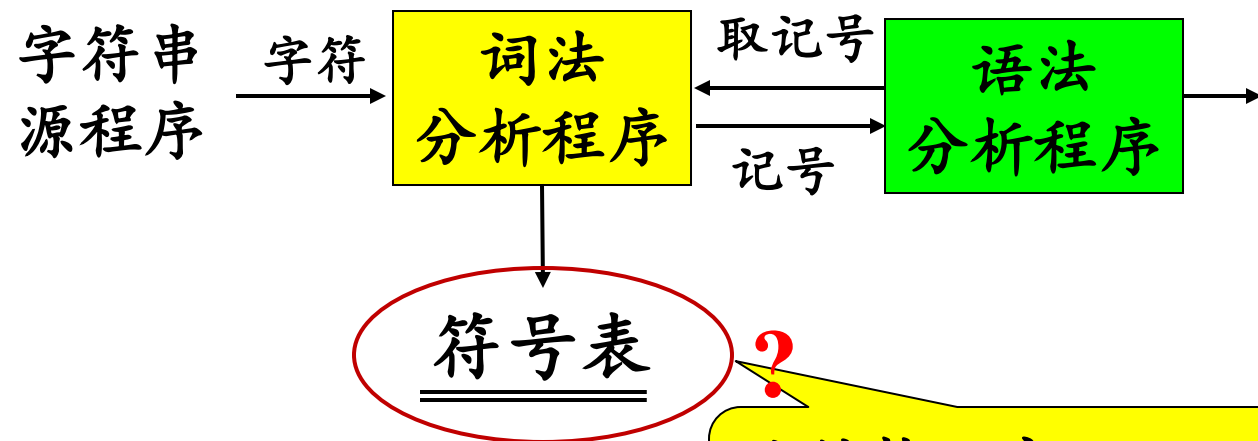
词法分析程序与语法分析程序的关系

■ 词法分析程序作为独立的一遍



- 输出放入一个中间文件
 - 磁盘文件
 - 内存文件

■ 词法分析程序作为语法分析程序的子程序



?
块结构语言 No!
非块结构语言 Yes

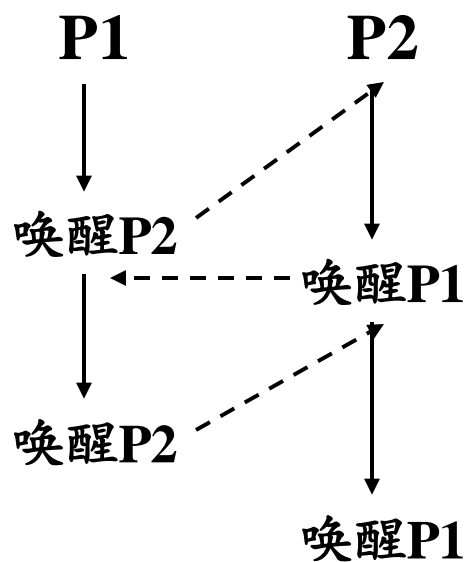
- 避免了中间文件
- 省去了取送符号的工作
- 有利于提高编译程序的效率

词法分析程序与语法分析程序的关系

■ 词法分析程序与语法分析程序作为协同程序

□ 协同程序：

如果两个或两个以上的程序，它们之间交叉地执行，这些程序称为协同程序。



■ 分离词法分析程序的好处

□ 可以简化设计

➤ 词法分析程序很容易识别并去除程序功能无关的空格、注释，使语法分析程序致力于语法分析，结构清晰，易于实现。

□ 可以改进编译程序的效率

➤ 利用专门的读字符和处理记号的技术构造更有效的词法分析程序。

□ 可以加强编译程序的可移植性

➤ 在词法分析程序中处理特殊的或非标准的符号。

3.2 词法分析程序的输入与输出

- 一、词法分析程序的实现方法
- 二、设置缓冲区的必要性
- 三、配对缓冲区
- 四、词法分析程序的输出

一、词法分析程序的实现方法

■ 利用词法分析程序自动生成器

- 从基于正规表达式的规范说明自动生成词法分析程序。
- 生成器提供用于源程序字符流读入和缓冲的若干子程序

■ 利用传统的系统程序设计语言来编写

- 利用该语言所具有的输入/输出能力来处理读入操作

■ 利用汇编语言来编写

- 直接管理源程序字符流的读入

二、设置缓冲区的必要性

- 为了得到某一个单词符号的确切性质，需要超前扫描若干个字符。

- 有合法的FORTRAN语句：

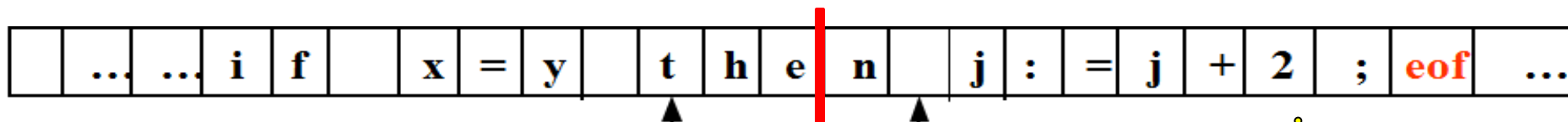
DO 99 K = 1, 10 和 DO 99 K = 1.10



- 为了区别这两个语句，必须超前扫描到等号后的第一个分界符处。
- C语言中：==、/*、//、++、for_loop

三、配对缓冲区

- 缓冲区分为大小相同的两半，每半含N个字符，N=1KB或4KB



- 测试过程:

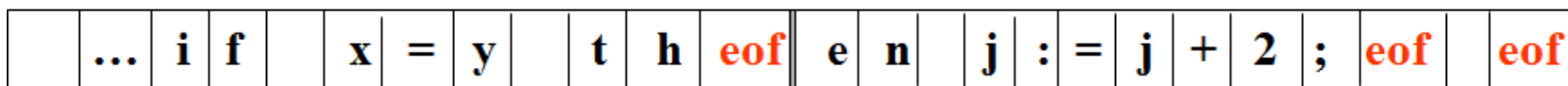
IF (向前指针在左半区的终点) {
 读入字符串，填充右半区;
 向前指针前移一个位置;
}

ELSE IF (向前指针在右半区的终点) {
 读入字符串，填充左半区;
 向前指针移到缓冲区的开始位置;
}

ELSE 向前指针前移一个位置;

单缓冲区是否可行?

每半区带有结束标记的缓冲区



开始指针 向前指针

■ 测试过程:

向前指针前移一个位置;

IF (向前指针指向 eof) {

IF (向前指针在左半区的终点) {

 读入字符串, 填充右半区;

 向前指针前移一个位置;

 };

ELSE IF (向前指针在右半区的终点) {

 读入字符串, 填充左半区;

 向前指针指向缓冲区的开始位置;

 };

ELSE 终止词法分析;

}

四、词法分析程序的输出：记号

■ 记号、模式和单词

- 记号：某一类单词符号的类别编码，如标识符的记号为id，常数的记号为num等。
- 模式：某一类单词符号的构词规则，如标识符的模式是“由字母开头的字母数字串”。
- 单词：某一类单词符号的一个特例，如position是标识符。

■ 记号的属性

- 词法分析程序在识别出一个记号后，要把与之有关的信息作为它的属性保留下来。
- 记号影响语法分析的决策，属性影响记号的翻译。
- 在词法分析阶段，对记号只能确定一种属性
 - 标识符：单词在符号表中的入口指针
 - 常数：它所表示的值
 - 关键字：（一符一种、或一类一种）
 - 运算符：（一符一种、或一类一种）
 - 分界符：（一符一种、或一类一种）

total:=total+rate*4 的词法分析结果

<id, 指向标识符total在符号表中的入口的指针>

<assign_op, - >

<id, 指向标识符total在符号表中的入口的指针>

<plus_op, - >

<id, 指向标识符rate在符号表中的入口的指针>

<mul_op, - >

<num, 整数值4>

3.3 记号的描述和识别

- 识别单词是按照记号的模式进行的，一种记号的模式匹配一类单词的集合。
 - 为设计词法程序，对模式要给出规范、系统的描述。
- 正规表达式和正规文法是描述模式的重要工具。
 - 同等表达能力
 - 表达式：清晰、简洁
 - 文法：便于识别

一、词法与正规文法

二、记号的文法


三、状态转换图与记号的识别

一、词法与正规文法

文法?

- 把源语言的文法 G 分解为若干子文法:

G_0 、 G_1 、 G_2 、...、 G_n



语法

词法

- 词法: 描述语言的标识符、常数、运算符和标点符号等记号的文法

—— 正规文法

- 语法: 借助于记号来描述语言的结构文法

—— 上下文无关文法

二、记号的文法

- 标识符
- 常数
 - 整数
 - 无符号数
- 运算符
- 分界符
- 关键字

标识符

- 假设标识符定义为“由字母打头的、由字母或数字组成的符号串”

- 描述标识符集合的**正规表达式**:

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

正规表达式?

- 表示标识符集合的正规定义式:

$\text{letter} \rightarrow A | B | \dots | Z | a | b | \dots | z$

$\text{digit} \rightarrow 0 | 1 | \dots | 9$

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

标识符

■ 把正规定义式转换为相应的正规文法

$$\begin{aligned} & (\text{letter} \mid \text{digit})^* \\ &= \varepsilon \mid (\text{letter} \mid \text{digit})^+ \\ &= \varepsilon \mid (\text{letter} \mid \text{digit})(\text{letter} \mid \text{digit})^* \\ &= \varepsilon \mid \text{letter}(\text{letter} \mid \text{digit})^* \mid \text{digit}(\text{letter} \mid \text{digit})^* \\ &= \varepsilon \mid (\text{A} \mid \dots \mid \text{Z} \mid \text{a} \mid \dots \mid \text{z})(\text{letter} \mid \text{digit})^* \\ &\quad \mid (0 \mid \dots \mid 9)(\text{letter} \mid \text{digit})^* \\ &= \varepsilon \mid \text{A}(\text{letter} \mid \text{digit})^* \mid \dots \mid \text{Z}(\text{letter} \mid \text{digit})^* \\ &\quad \mid \text{a}(\text{letter} \mid \text{digit})^* \mid \dots \mid \text{z}(\text{letter} \mid \text{digit})^* \\ &\quad \mid 0(\text{letter} \mid \text{digit})^* \mid \dots \mid 9(\text{letter} \mid \text{digit})^* \end{aligned}$$

■ 标识符的正规文法

$$\begin{aligned} id &\rightarrow \text{A rid} \mid \dots \mid \text{Z rid} \mid \text{a rid} \mid \dots \mid \text{z rid} \\ rid &\rightarrow \varepsilon \mid \text{A rid} \mid \text{B rid} \mid \dots \mid \text{Z rid} \\ &\quad \mid \text{a rid} \mid \text{b rid} \mid \dots \mid \text{z rid} \\ &\quad \mid 0 rid \mid 1 rid \mid \dots \mid 9 rid \end{aligned}$$

■ 一般写作:

$$\begin{aligned} id &\rightarrow \text{letter rid} \\ rid &\rightarrow \varepsilon \mid \text{letter rid} \mid \text{digit rid} \end{aligned}$$

常数——整数

- 描述整数结构的正规表达式为：

$$(\text{digit})^+$$

- 对此正规表达式进行等价变换：

$$(\text{digit})^+ = \text{digit}(\text{digit})^*$$

$$(\text{digit})^* = \varepsilon \mid \text{digit}(\text{digit})^*$$

- 整数的正规文法：

digits \rightarrow *digit remainder*

remainder $\rightarrow \varepsilon \mid \text{digit remainder}$

常数——无符号数

- 无符号数的正规表达式为：
 $(\text{digit})^+ (.(\text{digit})^+)? (\text{E}(+|-)?(\text{digit})^+)?$

- 正规定义式为

$\text{digit} \rightarrow 0 \mid 1 \mid \dots \mid 9$

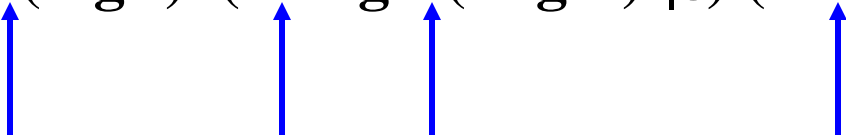
$\text{digits} \rightarrow \text{digit}^+$

$\text{optional_fraction} \rightarrow (.\text{digits})?$

$\text{optional_exponent} \rightarrow (\text{E}(+|-)?\text{digits})?$

$\text{num} \rightarrow \text{digits optional_fraction optional_exponent}$

把正规定义式转换为正规文法

$$\begin{aligned} & (\text{digit})^+ (.(\text{digit})^+)? (E(+|-)?(\text{digit})^+)? \\ & = (\text{digit})^+ (.(\text{digit})^+ | \varepsilon) (E(+|-|\varepsilon)(\text{digit})^+ | \varepsilon) \\ & = \text{digit}(\text{digit})^* (.\text{digit}(\text{digit})^* | \varepsilon) (E(+|-|\varepsilon)\text{digit}(\text{digit})^* | \varepsilon) \end{aligned}$$


num1 表示无符号数的第一个数字之后的部分

num2 表示小数点以后的部分

num3 表示小数点后第一个数字以后的部分

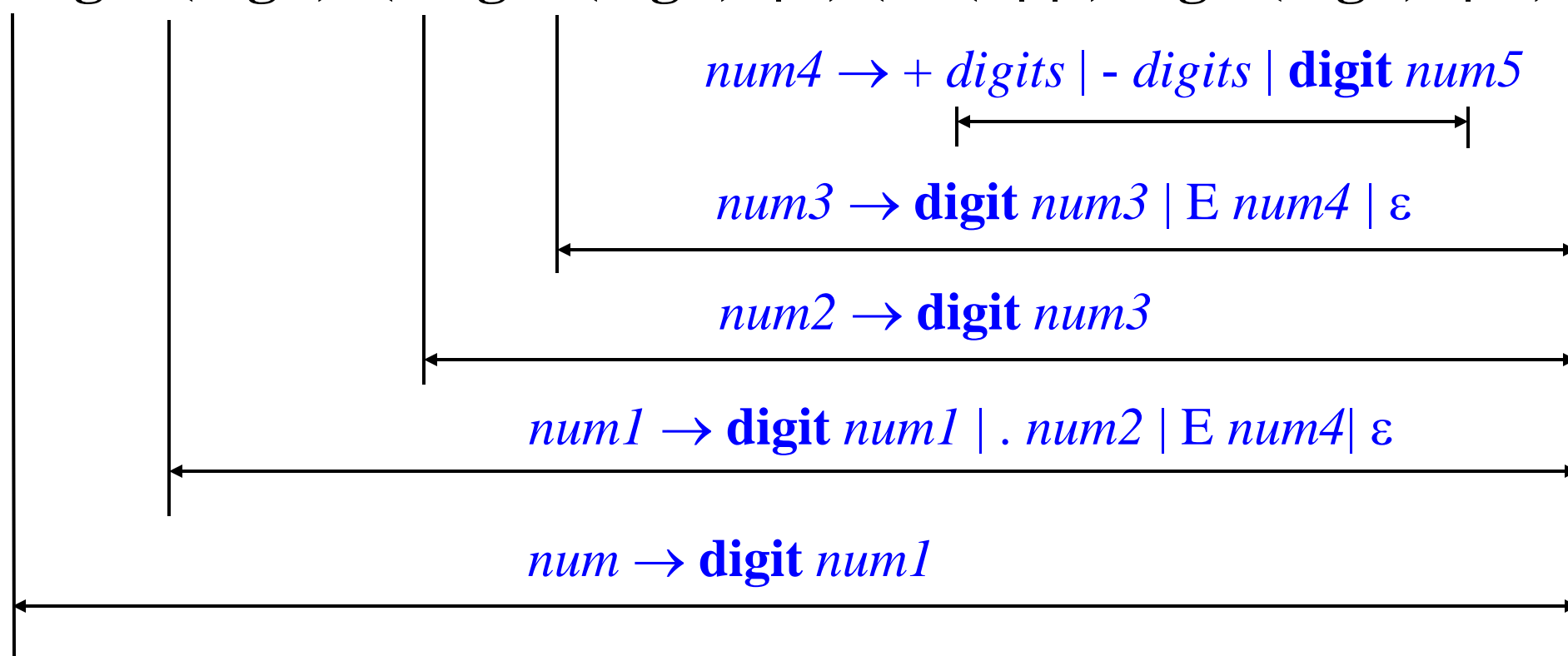
num4 表示E之后的部分

num5 表示 $(\text{digit})^*$

digits 表示 $(\text{digit})^+$

无符号数分析图

$\text{digit } (\text{digit})^* (.\text{digit } (\text{digit})^* | \varepsilon) (E (+|-|\varepsilon) \text{digit } (\text{digit})^* | \varepsilon)$



num5 表示 $(\text{digit})^*$

digits 表示 $(\text{digit})^+$

$\text{digits} \rightarrow \text{digit } \text{num5}$

$\text{num5} \rightarrow \text{digit } \text{num5} \mid \varepsilon$

无符号数的正规文法

$num \rightarrow digit\ num1$

$num1 \rightarrow digit\ num1$

| $.num2$

| $E\ num4$

| ϵ

$num2 \rightarrow digit\ num3$

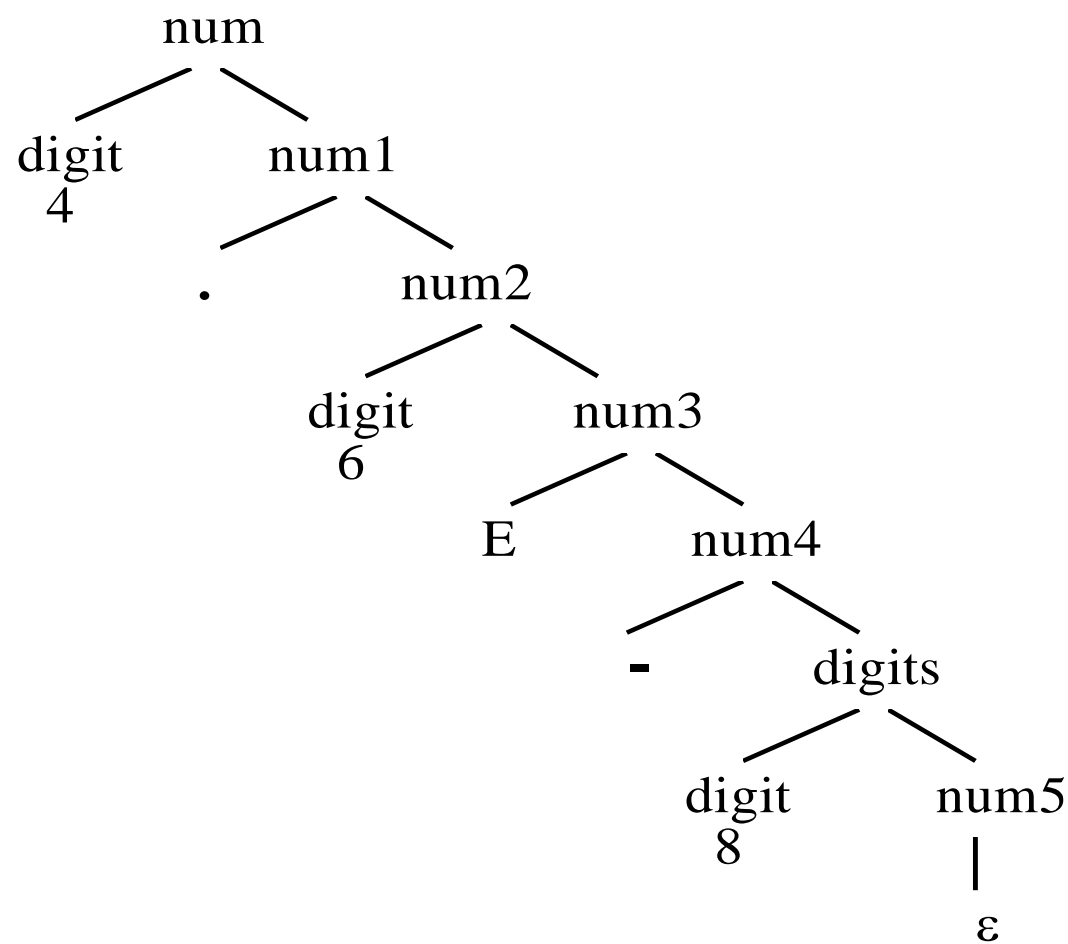
$num3 \rightarrow digit\ num3 \mid E\ num4 \mid \epsilon$

$num4 \rightarrow +\ digits \mid -\ digits \mid digit\ num5$

$digits \rightarrow digit\ num5$

$num5 \rightarrow digit\ num5 \mid \epsilon$

■ 4. 6E-8 的分析树



运算符

- 关系运算符的正规表达式为：

$< | <= | = | <> | >= | >$

- 正规定义式：

$\text{relop} \rightarrow < | <= | = | <> | >= | >$

- 关系运算符的正规文法：

$\text{relop} \rightarrow < | < \text{equal} | = | < \text{greater} | > | > \text{equal}$

$\text{greater} \rightarrow >$

$\text{equal} \rightarrow =$

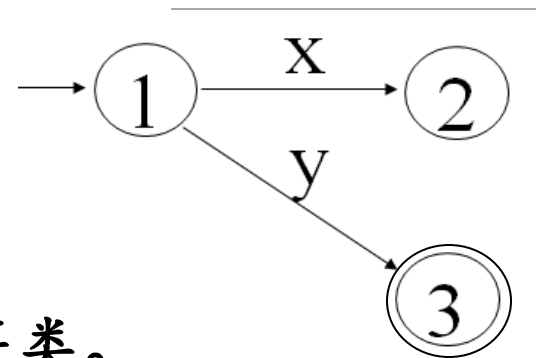
三、状态转换图与记号的识别

- 状态转换图
- 利用状态转换图识别记号
- 为线性文法构造相应的状态转换图
 - 状态集合的构成
 - 状态之间边的形成

状态转换图

■ 状态转换图是一张有限的方向图

- 图中结点代表状态，用圆圈表示。
- 状态之间用有向边连接。
- 边上的标记代表在射出结状态下，可能出现的输入符号或字符类。



■ 标识符的状态转换图

- 标识符的文法产生式:

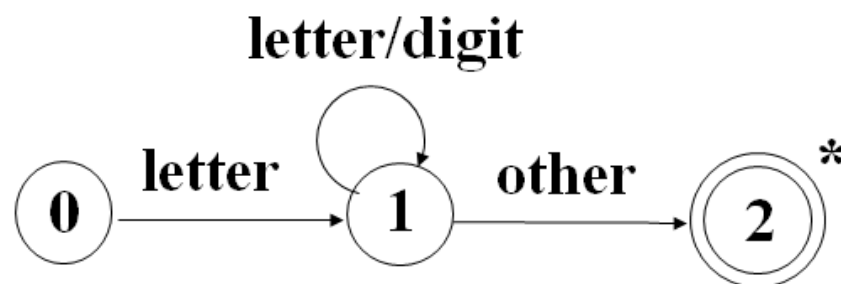
$id \rightarrow \text{letter } rid$

$rid \rightarrow \varepsilon \mid \text{letter } rid \mid \text{digit } rid$

- 标识符的状态转换图:

■ 利用状态转换图识别记号

- 语句 DO99K=1.10 中标识符 DO99K



为线性文法构造相应的状态转换图

■ 状态集合的构成

- 对文法G的每一个非终结符号设置一个对应的状态
- 文法的开始符号对应的状态称为初态
- 增加一个新的状态，称为终态。

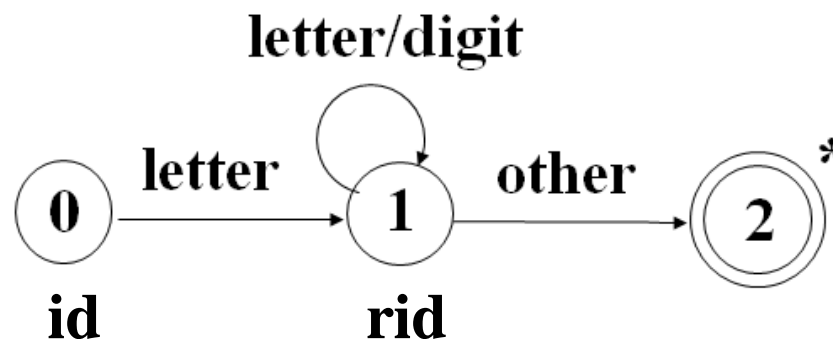
■ 状态之间边的形成

- 对产生式 $A \rightarrow aB$ ，从A状态到B状态画一条标记为a的边
- 对产生式 $A \rightarrow a$ ，从A状态到终态画一条标记为a的边
- 对产生式 $A \rightarrow \varepsilon$ ，从A状态到终态画一条标记为 ε 的边

■ 标识符的正规文法

$id \rightarrow \text{letter } rid$

$rid \rightarrow \varepsilon \mid \text{letter } rid \mid \text{digit } rid$



无符号数的右线性文法的状态转换图

$num \rightarrow \text{digit } num1$

$num1 \rightarrow \text{digit } num1 \mid . num2 \mid E num4 \mid \varepsilon$

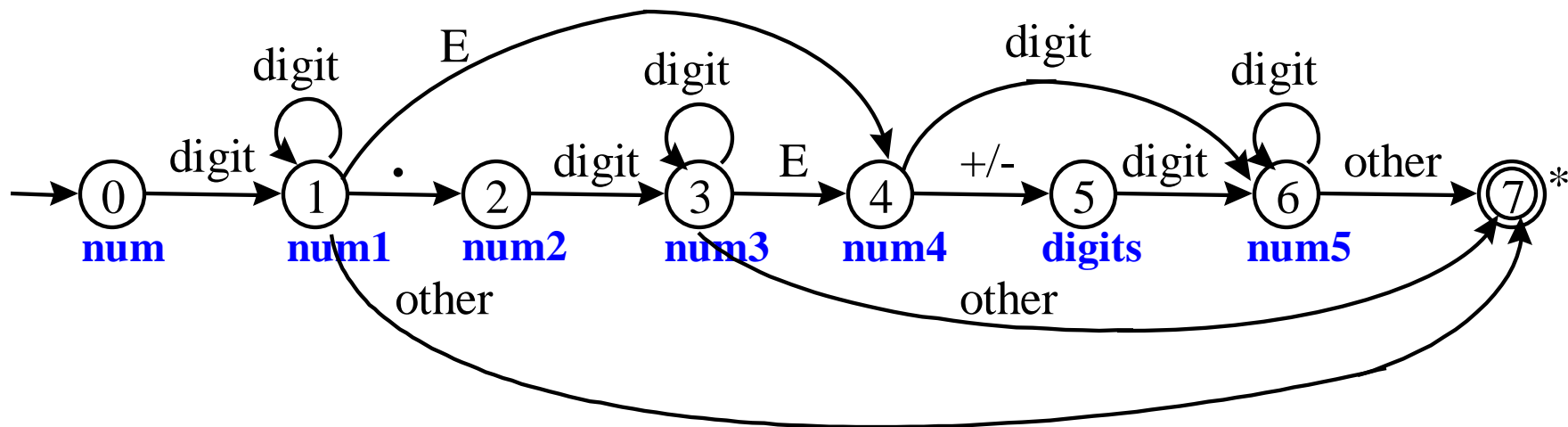
$num2 \rightarrow \text{digit } num3$

$num3 \rightarrow \text{digit } num3 \mid E num4 \mid \varepsilon$

$num4 \rightarrow + \text{digits} \mid - \text{digits} \mid \text{digit } num5$

$\text{digits} \rightarrow \text{digit } num5$

$num5 \rightarrow \text{digit } num5 \mid \varepsilon$



3.4 词法分析程序的设计与实现

一、文法及状态转换图

1. 语言说明
2. 记号的正规文法
3. 状态转换图

二、词法分析程序的构造

三、词法分析程序的实现

1. 输出形式
2. 设计全局变量和过程
3. 编制词法分析程序

一、文法及状态转换图

■ 语言说明

标识符：以字母开头的、后跟字母或数字组成的符号串。

保留字：标识符的子集。

无符号数：同PASCAL语言中的无符号数。

关系运算符：<、<=、=、<>、>=、>。

标点符号：+、-、*、/、(、)、:、'、; 等。

赋值号：:=

注释标记：以 '/*' 开始，以 '*/' 结束。

单词符号间的分隔符：空格

记号的正规文法

■ 标识符的文法

$id \rightarrow \text{letter } rid$

$rid \rightarrow \varepsilon \mid \text{letter } rid \mid \text{digit } rid$

■ 无符号整数的文法

$digits \rightarrow \text{digit } remainder$

$remainder \rightarrow \varepsilon \mid \text{digit } remainder$

■ 无符号数的文法

$num \rightarrow \text{digit } num1$

$num1 \rightarrow \text{digit } num1 \mid . num2 \mid E num4 \mid \varepsilon$

$num2 \rightarrow \text{digit } num3$

$num3 \rightarrow \text{digit } num3 \mid E num4 \mid \varepsilon$

$num4 \rightarrow + digits \mid - digits \mid \text{digit } num5$

$digits \rightarrow \text{digit } num5$

$num5 \rightarrow \text{digit } num5 \mid \varepsilon$

记号的正规文法

■ 关系运算符的文法

$relop \rightarrow < | <equal | = | <greater | > | >equal$

$greater \rightarrow >$

$equal \rightarrow =$

■ 赋值号的文法

$assign_op \rightarrow :equal$

$equal \rightarrow =$

■ 标点符号的文法

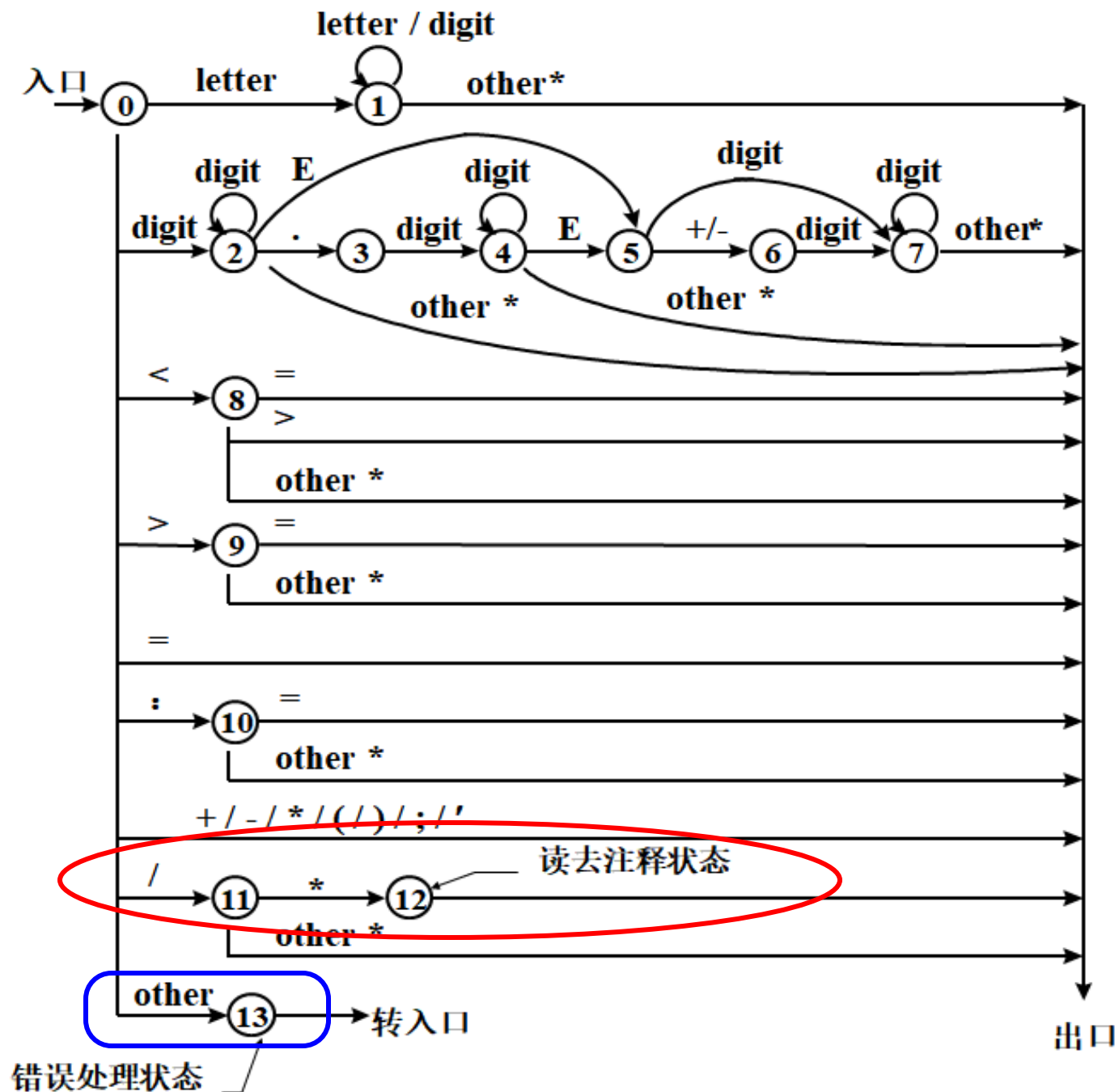
$single \rightarrow + | - | * | / | (|) | : | ' | ;$

■ 注释头符号的文法

$note \rightarrow / star$

$star \rightarrow *$

状态转换图



二、词法分析程序的构造

- 把语义动作添加到状态转换图中，使每一个状态都对应一小段程序。
- 程序功能：如状态有若干条射出边，则读一个字符，根据读到的字符，选择标记与之匹配的边到达下一个状态。
- 状态0：读进一个字符。若读入的字符是一个空格（包括blank、tab、enter）就跳过它，直到读进一个非空字符为止。然后，根据非空字符转相应的状态。
- 在标识符状态：识别并组合出一个标识符之后，还必须加入一些动作，如查关键字表，确定识别出的单词符号是否为关键字，并输出相应的记号。
- 在“<”状态，若读进的下一个字符是“=”，则输出关系运算符“<=”；若读进的下一个字符是“>”，则输出关系运算符“<>”；否则输出关系运算符“<”。

三、词法分析程序的实现

- 输出形式
- 设计全局变量和过程
- 编制词法分析程序

输出形式

- 利用翻译表，将识别出的单词的记号以二元式的形式加以输出
- 二元式的形式：
 <记号，属性>
- 翻译表：

| 正规表达式 | 记号 | 属性 |
|-------|-----------|---------------|
| if | if | - |
| then | then | - |
| else | else | - |
| id | id | 符号表入口指针 |
| num | num | 常数表入口指针 / val |
| < | relop | LT |
| <= | relop | LE |
| = | relop | EQ |
| <> | relop | NE |
| > | relop | GT |
| >= | relop | GE |
| := | assign-op | - |
| + | + | - |
| - | - | - |
| * | * | - |
| / | / | - |
| (| (| - |
|) |) | - |
| , | , | - |
| ; | ; | - |
| : | : | - |

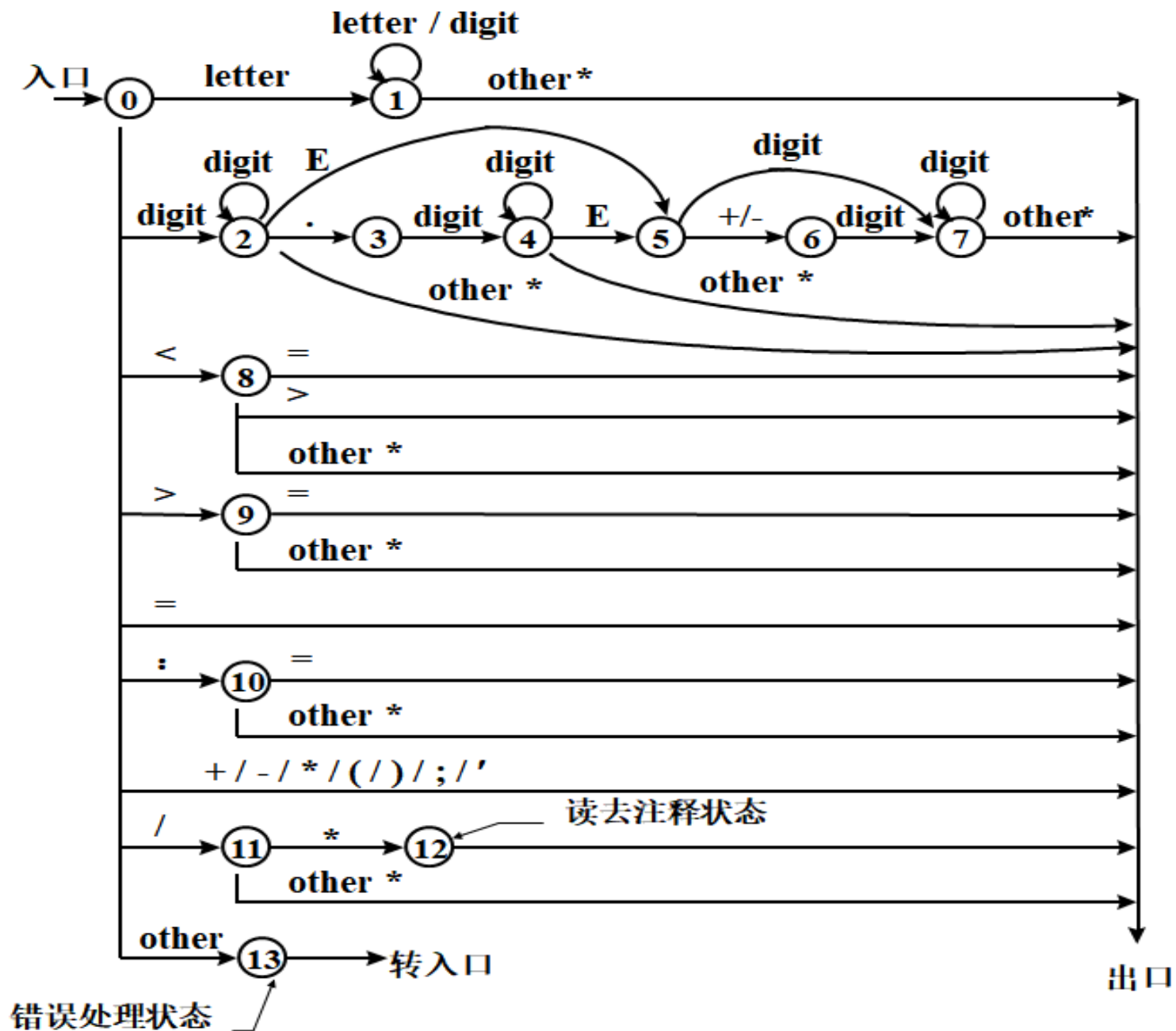
设计全局变量和过程

- (1) **state**: 整型变量, 当前状态指示。
- (2) **C**: 字符变量, 存放当前读入的字符。
- (3) **token**: 字符数组, 存放当前正在识别的单词字符串。
- (4) **buffer**: 字符数组, 输入缓冲区。
- (5) **forward**: 字符指针, 向前指针。
- (6) **lexemebegin**: 字符指针, 指向buffer中当前单词的开始位置。
- (7) **get_char()**: 每调用一次, 返回forward指向的buffer中的字符, 并把它放入变量C中, 然后, 移动forward, 使之指向下一个字符。
- (8) **get_nbc()**: 检查C中的字符是否为空格, 若是, 则反复调用get_char(), 直到C中进入一个非空字符为止。
- (9) **cat()**: 把C中的字符连接在token中的字符串后面。
- (10) **iskey**: 整型变量, 值为-1, 表示识别出的单词是用户自定义标识符, 否则, 表示识别出的单词是关键字, 其值为关键字的记号。

设计全局变量和过程

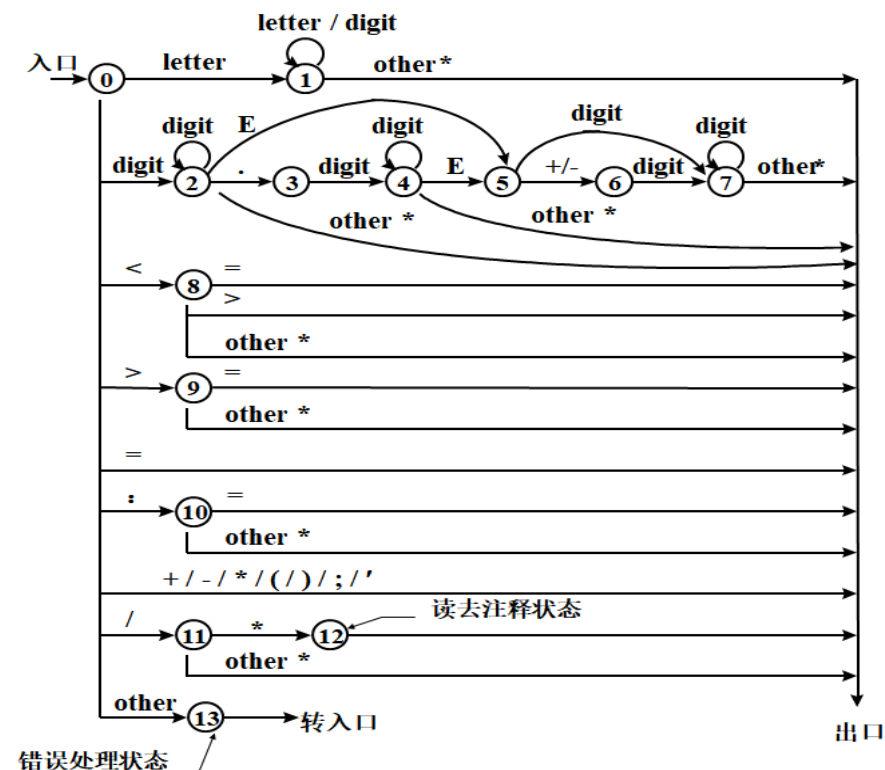
- (11) **letter()**: 判断C中的字符是否为字母, 若是则返回true, 否则返回false。
- (12) **digit()**: 判断C中的字符是否为数字, 若是则返回true, 否则返回false。
- (13) **retract()**: 向前指针forward后退一个字符。
- (14) **reserve()**: 根据token中的单词查关键字表, 若token中的单词是关键字, 则返回值该关键字的记号, 否则, 返回值“-1”。
- (15) **SToI()**: 将token中的字符串转换成整数。
- (16) **SToF()**: 将token中的字符串转换成浮点数。
- (17) **table_insert()**: 将token中的单词插入符号表, 返回其在符号表中的位置指针。
- (18) **error()**: 对发现的错误进行相应的处理。
- (19) **return()**: 将识别出来的单词的记号返回给调用程序。

编制词法分析程序



词法分析程序（类C语言描述）

```
state=0;
DO {
  SWITCH ( state ) {
    CASE 0:  // 初始状态
      token=' ' ;    get_char() ;    get_nbc();
      SWITCH ( C ) {
        CASE 'a': CASE 'b': ... CASE 'z': state=1; break; //设置标识符状态
        CASE '0': CASE '1': ... CASE '9': state=2; break; //设置常数状态
        CASE '<': state=8; break; //设置 '<' 状态
        CASE '>': state=9; break; //设置 '>' 状态
        CASE ':': state=10; break; //设置 ':' 状态
        CASE '/': state=11; break; //设置 '/' 状态
        CASE '=': state=0; return(relop, EQ); break; //返回 '=' 的记号
        CASE '+': state=0; return('+', -); break; //返回 '+' 的记号
        CASE '-': state=0; return('-', -); break; //返回 '-' 的记号
        CASE '*': state=0; return('*', -); break; //返回 '*' 的记号
        CASE '(': state=0; return('(', -); break; //返回 '(' 的记号
        CASE ')': state=0; return(')', -); break; //返回 ')' 的记号
        CASE ';': state=0; return(';', -); break; //返回 ';' 的记号
        CASE '\\': state=0; return('\\', -); break; //返回 '\\' 的记号
        default: state=13; break; //设置错误状态
      }
    };
  break;
}
```



词法分析程序

CASE 1:

```
cat();  
get_char();
```

```
IF ( letter() || digit() ) state=1;
```

```
ELSE {
```

```
    retract();
```

```
    state=0;
```

```
    iskey=reserve(); // 查关键字表
```

```
    IF ( iskey!=-1 )
```

```
        return (iskey, -); // 是关键字
```

```
    ELSE { // 识别出的是用户自定义标识符
```

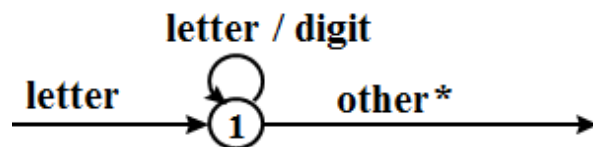
```
        identry=table_insert();
```

```
        return(ID, identry);
```

```
    };
```

```
};
```

```
break;
```



CASE 2:

```
cat();
```

```
get_char();
```

```
SWITCH ( C ) {
```

```
    CASE '0':
```

```
    CASE '1':
```

```
        ⋮
```

```
    CASE '9': state=2; break;
```

```
    CASE '.': state=3; break;
```

```
    CASE 'E': state=5; break;
```

```
    DEFAULT: // 识别出整常数
```

```
        retract();
```

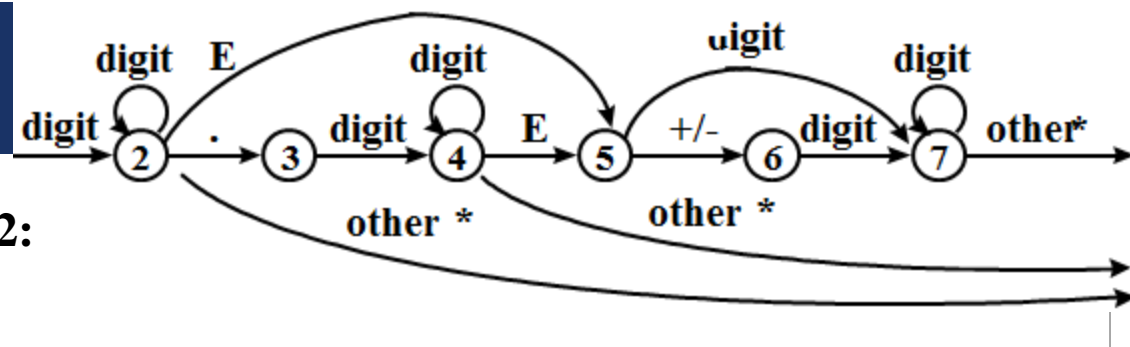
```
        state=0;
```

```
        return(NUM, SToI(token)); // 返回整数
```

```
        break;
```

```
};
```

```
break;
```



本章小结

- 词法分析器的作用
- 与语法分析器的关系
 - 独立、子程序、协同程序
- 配对缓冲区
 - 必要性、算法
- 记号
 - 记号、模式、单词
 - 属性
 - 二元式形式：<记号，属性>
 - 描述：
 正规表达式、正规文法
 - 识别：状态转换图
- 词法分析器的设计与实现
 - 各类单词符号的正规表达式
 - 各类单词符号的正规文法
 - 构造与文法相应的状态转换图
 - 合并为词法分析器的状态转换图
 - 增加语义动作，构造词法分析器的程序框图
 - 确定输出形式、设计翻译表
 - 定义变量和过程
 - 编码实现

学习任务

■ 作业要求

- 理解正规表达式、正规文法和DFA之间的等价性；
- 理解单词符号的识别过程。

■ 研究型学习

- 词法错误及注释的识别与处理



程序设计1

■ 题目：C语言词法分析程序的设计与实现

■ 实验内容及要求：

1. 可以识别出用C语言编写的源程序中的每个单词符号，并以记号的形式输出每个单词符号。
2. 可以识别并跳过源程序中的注释。
3. 可以统计源程序中的语句行数、各类单词的个数、以及字符总数，并输出统计结果。
4. 检查源程序中存在的词法错误，并报告错误所在的位置。
5. 对源程序中出现的错误进行适当的恢复，使词法分析可以继续进行，对源程序进行一次扫描，即可检查并报告源程序中存在的所有词法错误。

■ 实现方法要求：分别用以下两种方法实现。

方法1：编写LEX源程序，利用LEX编译程序自动生成词法分析程序。

方法2：采用C/C++作为实现语言，手工编写词法分析程序。

实验报告要求

■ 内容：

- 实验题目、要求
- 程序设计说明
- 源程序
- 可执行程序
- 测试报告：输入、运行结果、分析说明

■ 提交：

- 个人资料打包
- 命名规则：班级-学号-姓名

■ 云空间线上提交。

★ 文法及其形式定义

■ **文法**：所谓文法就是描述语言的语法结构的形式规则。

■ 任何一个文法都可以表示为一个**四元组** $G=(V_T, V_N, S, \phi)$

V_T 是一个非空的有限集合，它的每个元素称为**终结符号**。

V_N 是一个非空的有限集合，它的每个元素称为**非终结符号**。

$$V_T \cap V_N = \phi$$

S 是一个特殊的非终结符号，称为文法的**开始符号**。

ϕ 是一个非空的有限集合，它的每个元素称为**产生式**。

产生式的形式为： $\alpha \rightarrow \beta$

“ \rightarrow ”表示“**定义为**”（或“**由.....组成**”）

$$\alpha, \beta \in (V_T \cup V_N)^*, \alpha \neq \epsilon$$

左部相同的产生式 $\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \dots, \alpha \rightarrow \beta_n$ 可以缩写

$$\alpha \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$$

“ $|$ ”表示“**或**”，每个 $\beta_i (i=1, 2, \dots, n)$ 称为 α 的一个**候选式**

★文法的分类

- 根据对产生式施加的限制不同，定义了四类文法和相应的四种形式语言类。

| 文法类型 | 产生式形式的限制 | 文法产生的语言类 |
|--------------------------|--|-------------------|
| 0型文法 | $\alpha \rightarrow \beta$ 其中 $\alpha, \beta \in (V_T \cup V_N)^*$ $ \alpha \neq 0$ | 0型语言 |
| 1型文法，即 上下文有关文法 | $\alpha \rightarrow \beta$ 其中 $\alpha, \beta \in (V_T \cup V_N)^*$ $ \alpha \leq \beta $ | 1型语言，即 上下文有关语言 |
| 2型文法，即 上下文无关文法 | $A \rightarrow \beta$ 其中 $A \in V_N, \beta \in (V_T \cup V_N)^*$ | 2型语言，即 上下文无关语言 |
| 3型文法，即 正规文法 (线性文法) | $A \rightarrow a$ 或 $A \rightarrow aB$ (右线性)，或 $A \rightarrow a$ 或 $A \rightarrow Ba$ (左线性) 其中 $A, B \in V_N, a \in V_T \cup \{\epsilon\}$ | 3型语言，即 正规语言 |

★ 上下文无关文法及相应的语言

- 所定义的语法单位(或称语法实体)完全独立于这种语法单位可能出现的上下文环境。
- 现有程序设计语言中, 许多语法单位的结构可以用上下文无关文法来描述。
- 例: 描述算术表达式的文法G:

$G = (\{ i, +, -, *, /, (,) \}, \{ \langle \text{表达式} \rangle, \langle \text{项} \rangle, \langle \text{因子} \rangle \}, \langle \text{表达式} \rangle, \varphi)$

其中 φ :

$\langle \text{表达式} \rangle \rightarrow \langle \text{表达式} \rangle + \langle \text{项} \rangle \mid \langle \text{表达式} \rangle - \langle \text{项} \rangle \mid \langle \text{项} \rangle$

$\langle \text{项} \rangle \rightarrow \langle \text{项} \rangle * \langle \text{因子} \rangle \mid \langle \text{项} \rangle / \langle \text{因子} \rangle \mid \langle \text{因子} \rangle$

$\langle \text{因子} \rangle \rightarrow (\langle \text{表达式} \rangle) \mid i$

语言 $L(G)$ 是所有包括加、减、乘、除四则运算的算术表达式的集合。

★ 文法书写约定

■ 终结符号

- 次序靠前的小写字母，如：a、b、c
- 运算符号，如：+、-、*、/
- 各种标点符号，如：括号、逗号、冒号、等于号
- 数字1、2、...、9
- 正体字符串，如：id、begin、if、then

■ 非终结符号

- 次序靠前的大写字母，如：A、B、C
- 大写字母S常用作文法的开始符号
- 小写的斜体符号串，如：*expr*、*term*、*factor*、*stmt*

★ 文法书写约定

■ 文法符号

- 次序靠后的大写字母，如：X、Y、Z

■ 终结符号串

- 次序靠后的小写字母，如：u、v、...、z

■ 文法符号串

- 小写的希腊字母，如： α 、 β 、 γ 、 δ

- 可以直接用产生式的集合代替四元组来描述文法，
第一个产生式的左部符号是文法的开始符号。

★ 正规表达式

- 用正规表达式可以精确地定义集合，如某语言的标识符，由字母开头、由字母或数字组成的符号串，正规表达式：

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

定义：字母表 Σ 上的正规表达式

- (1) ϵ 是正规表达式，它表示的语言是 $\{\epsilon\}$
- (2) 如果 $a \in \Sigma$ ，则 a 是正规表达式，它表示的语言是 $\{a\}$
- (3) 如果 r 和 s 都是正规表达式，分别表示语言 $L(r)$ 和 $L(s)$ ，则：
 - 1) $(r)|(s)$ 是正规表达式，表示的语言是 $L(r) \cup L(s)$
 - 2) $(r)(s)$ 是正规表达式，表示的语言是 $L(r)L(s)$
 - 3) $(r)^*$ 是正规表达式，表示的语言是 $(L(r))^*$
 - 4) (r) 是正规表达式，表示的语言是 $L(r)$

- 正规表达式表示的语言叫做正规集。

★ 正规表达式的书写约定

- 一元闭包 ‘*’ 具有最高优先级，并且遵从左结合
- 连接运算的优先级次之，遵从左结合
- 并运算 ‘|’ 的优先级最低，遵从左结合

例：如果 $\Sigma = \{a, b\}$ ，则有：

正规表达式 $a|b$ 表示集合 $\{a, b\}$

$(a|b)(a|b)$ 表示： $\{aa, ab, ba, bb\}$

a^* 表示：由0个或多个a组成的所有符号串的集合

$a|a^*b$ 表示：a和0个或多个a后跟一个b的所有符号串的集合

$(a|b)^*$ 表示：由a和b构成的所有符号串的集合

$(a^*|b^*)^*$

- 如果两个正规表达式r和s表示同样的语言，即 $L(r)=L(s)$ ，则称r和s等价，写作 $r=s$ 。

如： $(a|b)=(b|a)$

★ 正规表达式遵从的代数定律

| 定律 | 说明 |
|----------------------------------|---------------------------|
| $r s=s r$ | “并”运算是可交换的 |
| $r (s t)=(r s) t$ | “并”运算是可结合的 |
| $(rs)t=r(st)$ | 连接运算是可结合的 |
| $r(s t)=rs rt$ $(s t)r=sr tr$ | 连接运算对并运算的分配 |
| $\epsilon r=r, r\epsilon=r$ | 对连接运算而言, ϵ 是单位元素 |
| $r^*=(r \epsilon)^*$ | $*$ 和 ϵ 之间的关系 |
| $r^{**}=r^*$ | $*$ 是等幂的 |
| $r^*=r^+ \epsilon, r^+=rr^*$ | $+$ 和 $*$ 之间的关系 |

★ 正规定义式

定义：令 Σ 是字母表，正规定义式是如下形式的定义序列：

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

...

$d_n \rightarrow r_n$

其中 d_i 是不同的名字， r_i 是 $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$ 上的正规表达式。

例：Pascal语言的无符号数可用如下的正规表达式来描述：

$\text{digit}^+(\text{.digit}^+|\epsilon)(\text{E}(+|-|\epsilon)\text{digit}^+|\epsilon)$

正规定义式：

$\text{digit} \rightarrow 0|1|\dots|9$

$\text{digits} \rightarrow \text{digit digit}^*$

$\text{optional_fraction} \rightarrow \text{.digits}|\epsilon$

$\text{optional_exponent} \rightarrow (\text{E}(+|-|\epsilon)\text{digits})|\epsilon$

$\text{num} \rightarrow \text{digits optional_fraction optional_exponent}$

★表示的缩写

- 引入正闭包运算符 ‘+’
 - $r^* = r^+ | \epsilon$ 、 $r^+ = rr^*$
 - $\text{digits} \rightarrow \text{digit}^+$
- 引入可选运算符 ‘?’
 - $r? = r | \epsilon$
 - $\text{optional_fraction} \rightarrow (. \text{digits})?$
 - $\text{optional_exponent} \rightarrow (E(+|-)? \text{digits})?$
- 引入表示 ‘[...]’
 - 字符组[abc], 表示正规表达式a|b|c
 - $\text{digit} \rightarrow [0-9]$
 - 标识符的正规表达式: $[A-Za-z][A-Za-z0-9]^*$

★ 正规文法的产生式和正规定义式中的正规定义

- 两个不同的概念，具有不同的含义。

- 产生式：

- 左部是一个非终结符号，右部是一个符合特定形式的文法符号串 α
- α 中的非终结符号可以与该产生式左部的非终结符号相同，即允许非终结符号的递归出现。

- 正规定义：

- 左部是一个名字，右部是一个正规表达式
- 表达式中出现的名字是有限制的，即只能是此定义之前已经定义过的名字。