编译原理

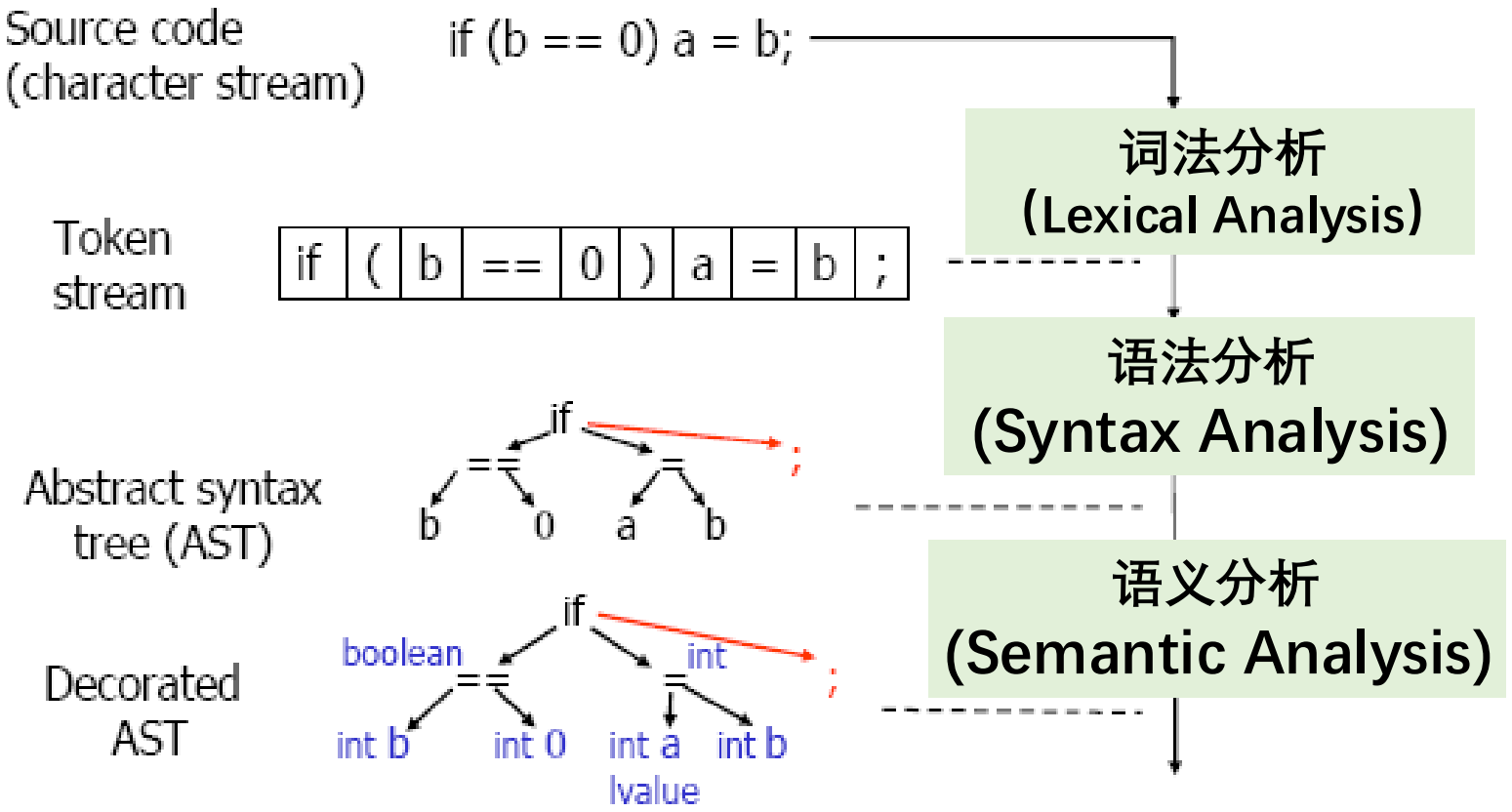
--词法分析

刘爽
中国人民大学信息学院

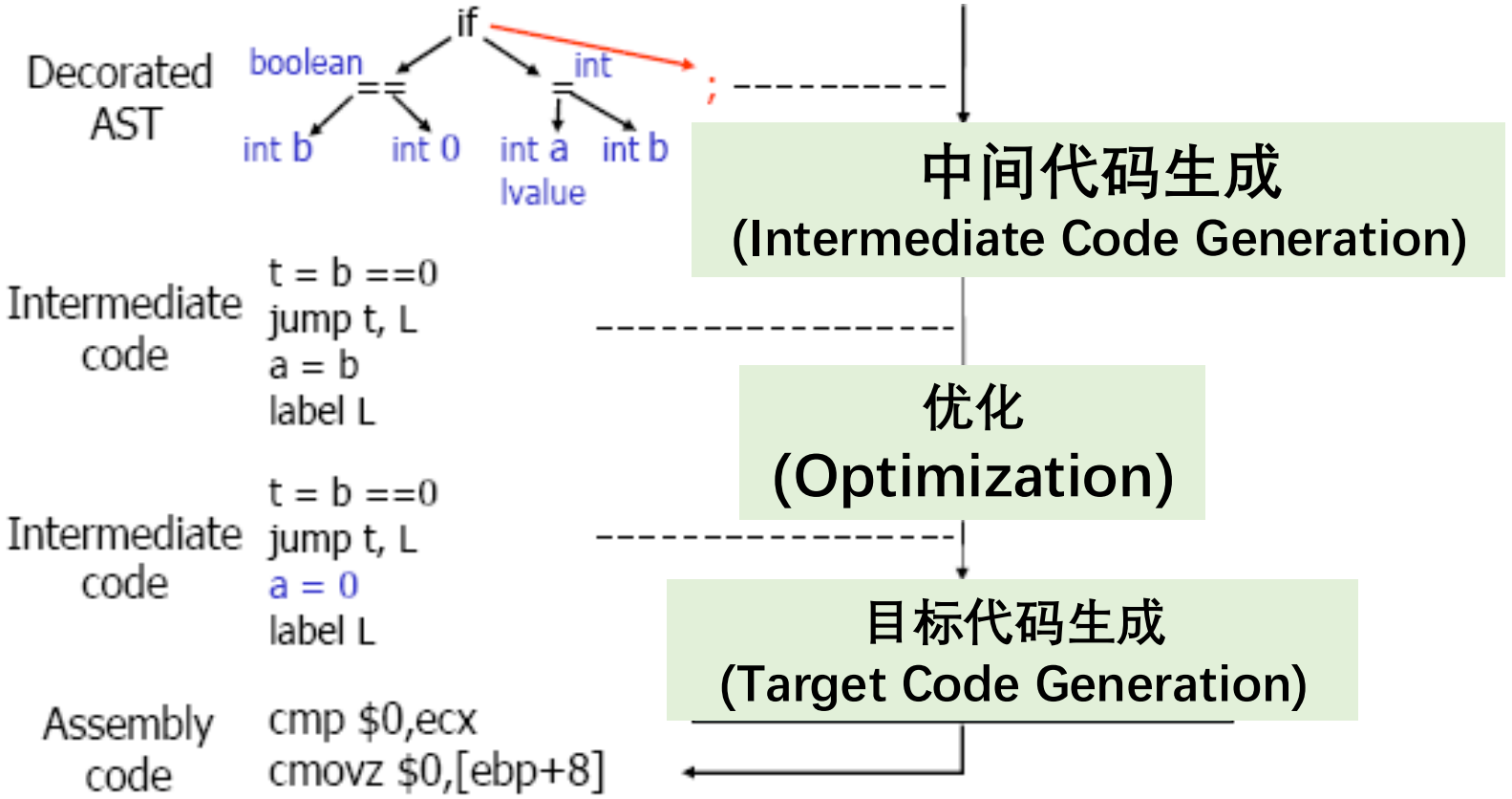
Outline

- 词法分析器的功能
- 单词种类及词法分析程序的输出形式
- 状态转换图
- 词法分析程序的设计与实现

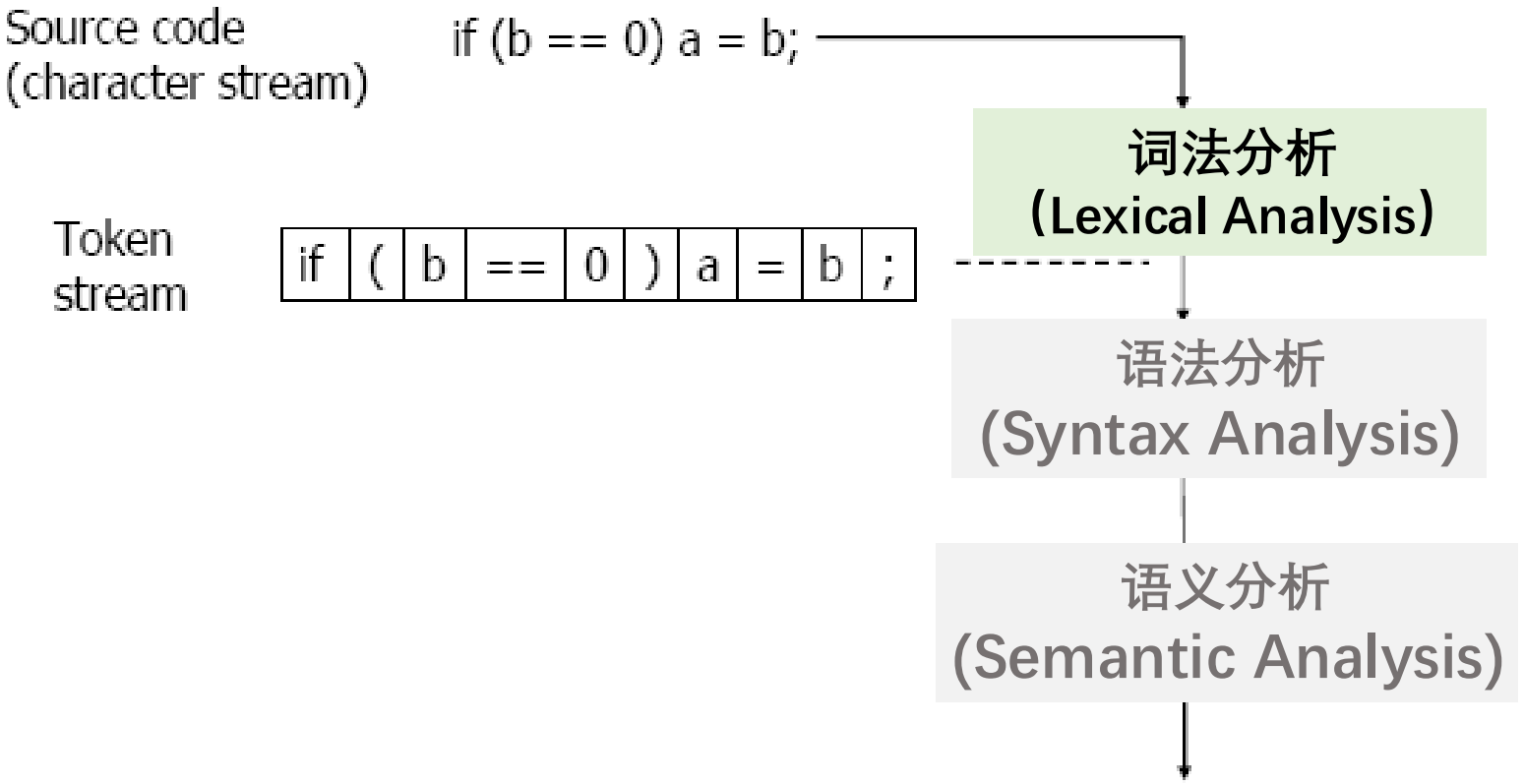
编译器结构


Source code
(character stream)
if (b == 0) a = b;
词法分析
(Lexical Analysis)
Token
stream
if (b == 0) a = b ;
语法分析
(Syntax Analysis)
Abstract syntax
tree (AST)
if
==
=
;
b
0
a
b
语义分析
(Semantic Analysis)
Decorated
AST
if
boolean
int
==
=
;
int b
int 0
int a
int b
lvalue

编译器结构

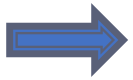

Decorated AST
if
boolean
==
int
=
;
int b
int 0
int a
lvalue
int b
Intermediate code
t = b ==0
jump t, L
a = b
label L
Intermediate code
t = b ==0
jump t, L
a = 0
label L
Assembly code
cmp $0,ecx
cmovz $0,[ebp+8]
中间代码生成
(Intermediate Code Generation)
优化
(Optimization)
目标代码生成
(Target Code Generation)

第一步：词法分析


Source code
(character stream)
if (b == 0) a = b;
词法分析
(Lexical Analysis)
Token
stream
if
(
b
==
0
)
a
=
b
;
语法分析
(Syntax Analysis)
语义分析
(Semantic Analysis)

程序

```
int main( ) ‘\n’ { int
  count=read( ); ‘\n’ //if
  number of entries read is
  greater than 1 ‘\n’ //then
  sort( ) and compact( ) ‘\n’
  if (count > 1) {  sort( );
  compact( ); } ‘\n’ if (count
  ==0) ‘\n’          count <<
  “no sales for this
  month\n”; ‘\n’ else write( );
  ‘\n’  return 0; ‘\n’ }
```



```
int main( )
{  int count=read( );
    //if number of entries read is greater than 1
   //then sort( ) and compact( )
    if (count > 1) {  sort( ); compact( ); }
    if (count ==0)
       count << “no sales for this month\n”;
    else write( );
    return 0;
}
```

词法分析器

主要功能:

- 读取源程序并生成单词符号
- 过滤掉注释和空白
- 关联编译器生成的错误消息与源程序的位置

主要阶段:

- 扫描阶段
- 词法分析阶段

- 词法分析器不能从全局的角度考察源程序，所以能在词法分析阶段发现的错误是有限的。主要是不符合合法标识符拼写的错误。

- 例如，如果词法分析器第一次碰到如下的C语言程序：fi(a==f(x))…

词法分析器无法区别fi究竟是关键字if的错误拼写还是一个未声明的函数标识符。由于fi是合法的标识符，词法分析器必须返回该标识符的记号，让编译器的其他阶段去处理这种错误。

Outline

- 词法分析器的功能
- 单词种类及词法分析程序的输出形式
- 状态转换图
- 词法分析程序的设计与实现

一些术语

- 词法单元(token)/单词符号：由一个词法单元名（单词种别）和一个可选的属性值组成，是词法分析器的输出。
- 词素(lexeme)：源程序中的一个字符序列，与某个词法单元的模式匹配，被词法分析器识别为该词法单元的一个实例。
- 模式(pattern)：描述了一个词法单元的词素可能具有的形式。

词法单元名	模式（非正式描述）	词素（lexemes）
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	< or > or <= or >= or == or !=	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but ", surrounded by "'s	`"core dumped"`

一些术语 (cont)

- 标识符 (Identifier): 表示各种名字，如x, str
- 关键字 (Keywords): 程序语言定义的具有固定意义的标识符，如if, else, while, for, break
- 常数 (Constant): 包括整型(Integer)，浮点型 (Floating-point)，布尔型(Boolean)，字符串(String) 等
- 符号 (Symbols): 包括运算符 +, *，++, <<, <, <=, 界符{, }, [,] 等
- Comments: /* bla bla bla */

单词符号：　（种别，属性）

单词符号分类

分类方法:

- 关键字（保留字）：可将其全体视为一种，也可以一字一种。后者更方便。
- 标识符：可以是一种，也可以按类型分种，一般统归为一种
- 常数：宜按照类型（整型、实型、布尔型等）分种
- 运算符：可采用一字一种的分法。也可以把具有一定共性的运算符视为一种
- 界符：一般用一符一种的分法。

单词符号	种别编码	助忆符
WHILE	1	$WHILE
IF	2	$IF
DO	3	$DO
STOP	4	$STOP
END	5	$END
标识符	6	$ID
常数	7	$INT
=	8	$ASSIGN
+	9	$PLUS
*	10	$STAR
**	11	$POWER
,	12	$COMMA
(	13	$LPAR
)	14	$RPAR

词法分析器的输出形式

- 词法分析器所输出的词法单元（单词符号）常表示成如下二元式:
  （词法单元名/单词种别，属性值）
  - 单词符号 种别：
    1）关键字（保留字，基本字）**if else**
    2）标识符，变量名，函数名 **Add()**
    3）常数 **0 3.14 TRUE**
    4）运算符 **+ - * /**
    5）界符 **, ; ()**

单词符号的种别分类并非固定，主要为处理方便。种别通常用整数编码，为方便记忆可以使用助记符。对于包含多个单词符号的种别，需要给出属性信息。

- 属性信息是指单词符号的特性或特征，属性值则是反应特性和特征的值。
- 例如：对于某个标识符，常将存放它的有关信息的符号表项的指针作为其属性值；对于某个常数，则将存放它的常数表项的指针作为其属性值。

词法分析输出—例子

- 输入：C++语言的代码：while (i == j) i++;
- 经词法分析器处理后，输出为：

<$WHILE, - >
<$LPAR, - >
<$ID, 指向i的符号表项的指针 >
< ==, ->
<$ID, 指向j的符号表项的指针>
<$RPAR, ->
<$ID, 指向i的符号表表项的指针>
< ++, ->
< ;, ->

单词符号	种别编码	助忆符
WHILE	1	$WHILE
IF	2	$IF
DO	3	$DO
STOP	4	$STOP
END	5	$END
标识符	6	$ID
常数	7	$INT
=	8	$ASSIGN
+	9	$PLUS
*	10	$STAR
**	11	$POWER
,	12	$COMMA
(	13	$LPAR
)	14	$RPAR

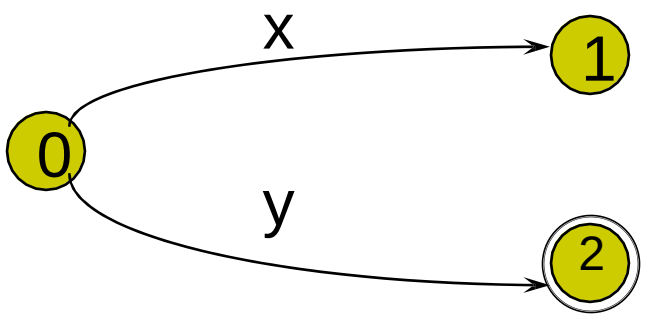
思考

- 词法分析器的输出是什么？

Outline

- 词法分析器的功能
- 单词种类及词法分析程序的输出形式
- 状态转换图
- 词法分析程序的设计与实现

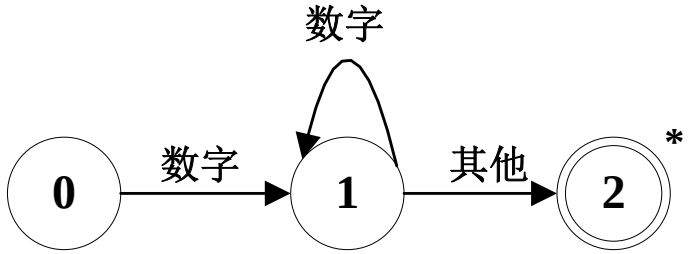
状态转换图



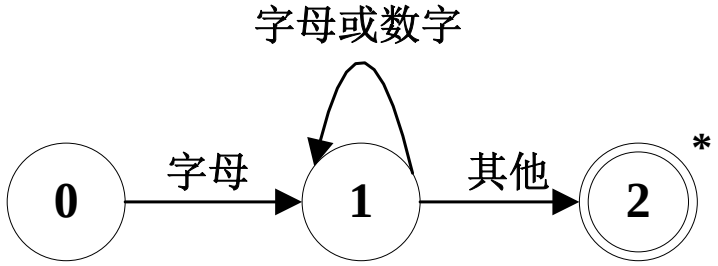
- **状态转换图**是有限方向图；
  - 结点代表状态，用圆圈表示；
  - 状态之间用**箭弧**连接；
  - 箭弧上的**标记**（字符）代表在射出结点（即箭弧始结点）状态下可能出现的输入字符或字符类。
- 一张转换图只包含有限个状态（即有限个结点），其中一个被认为是初始状态，至少一个最终状态/接受状态（用双圆圈表示）。
- 一个状态转换图可用于识别（或接受）一定的字符串。

状态转换图示例

识别整数的转换图



识别标识符的转换图



识别过程

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

#. 在遇到读入的字符是分隔符（如空格，换行符等）时，若当前状态是终止状态，说明读入的字符组成一单词符号；否则说明输入不符合词法规则。

- 终态结点上打了一个星号*，意味着多读进了一个不属于标识符部分的字符，应该把它退还给输入串。
- 如果在初始状态的输入不能符合任何一条转换路径，那么就说明这个转换图工作不成功，不能识别出标识符。

状态转换图实例

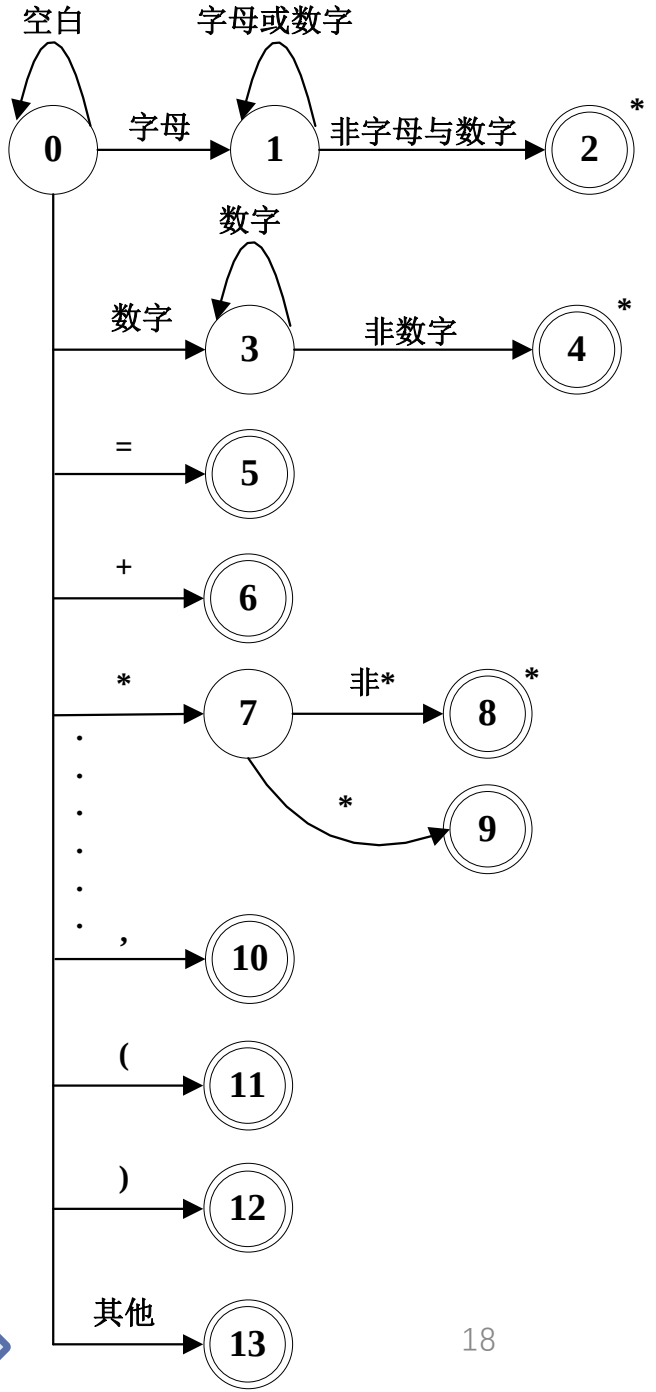
假设条件:

1. 关键字都是保留字，不允许使用他们作为自己定义的标识符;

2. 将关键字作为一类特殊标识符来处理。把它们预先安排在一张(保留字)表格中;

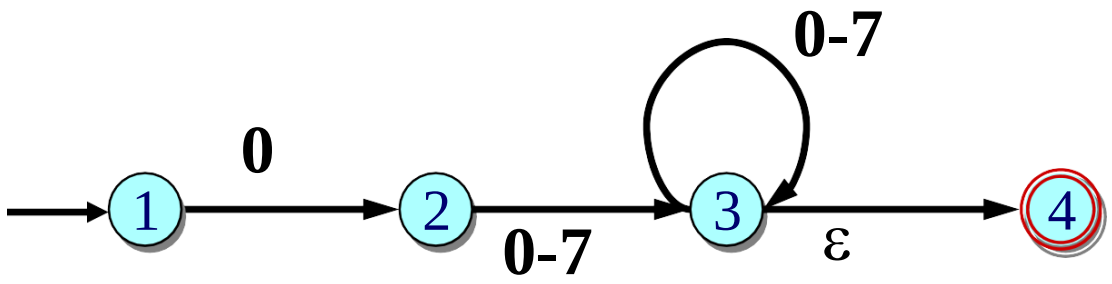
3. 如果关键字、标识符和常数之间没有确定的运算符或界符做间隔，则必须至少用一个空白符做间隔。

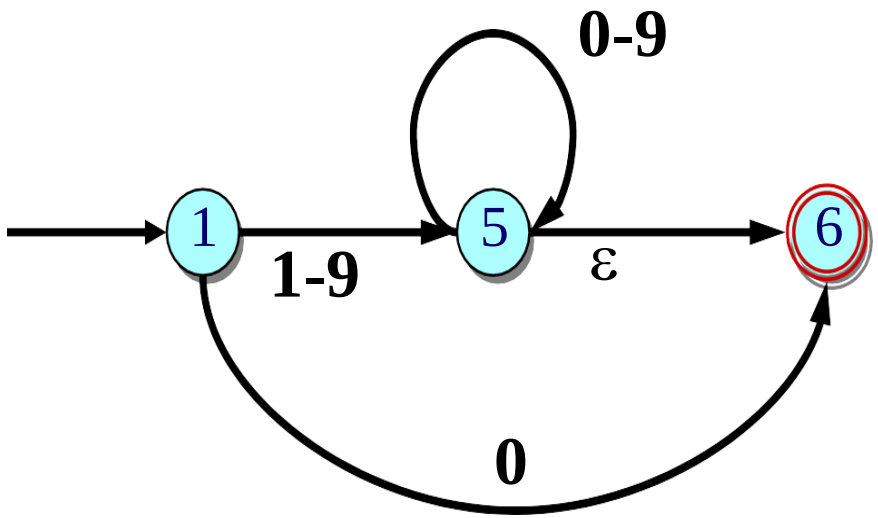
单词符号	种别编码	助忆符
DIM	1	$DIM
IF	2	$IF
DO	3	$DO
STOP	4	$STOP
END	5	$END
标识符	6	$ID
常数	7	$INT
=	8	$ASSIGN
+	9	$PLUS
*	10	$STAR
**	11	$POWER
,	12	$COMMA
(	13	$LPAR
)	14	$RPAR



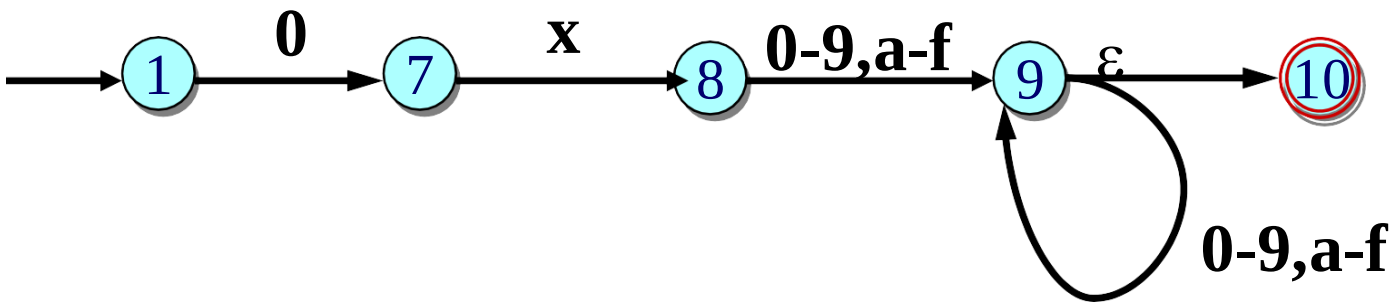
一个识别某简单语言所有单词符号的状态转换图

例：识别不同进制数的状态图





例：识别不同进制数的状态图 (cont)

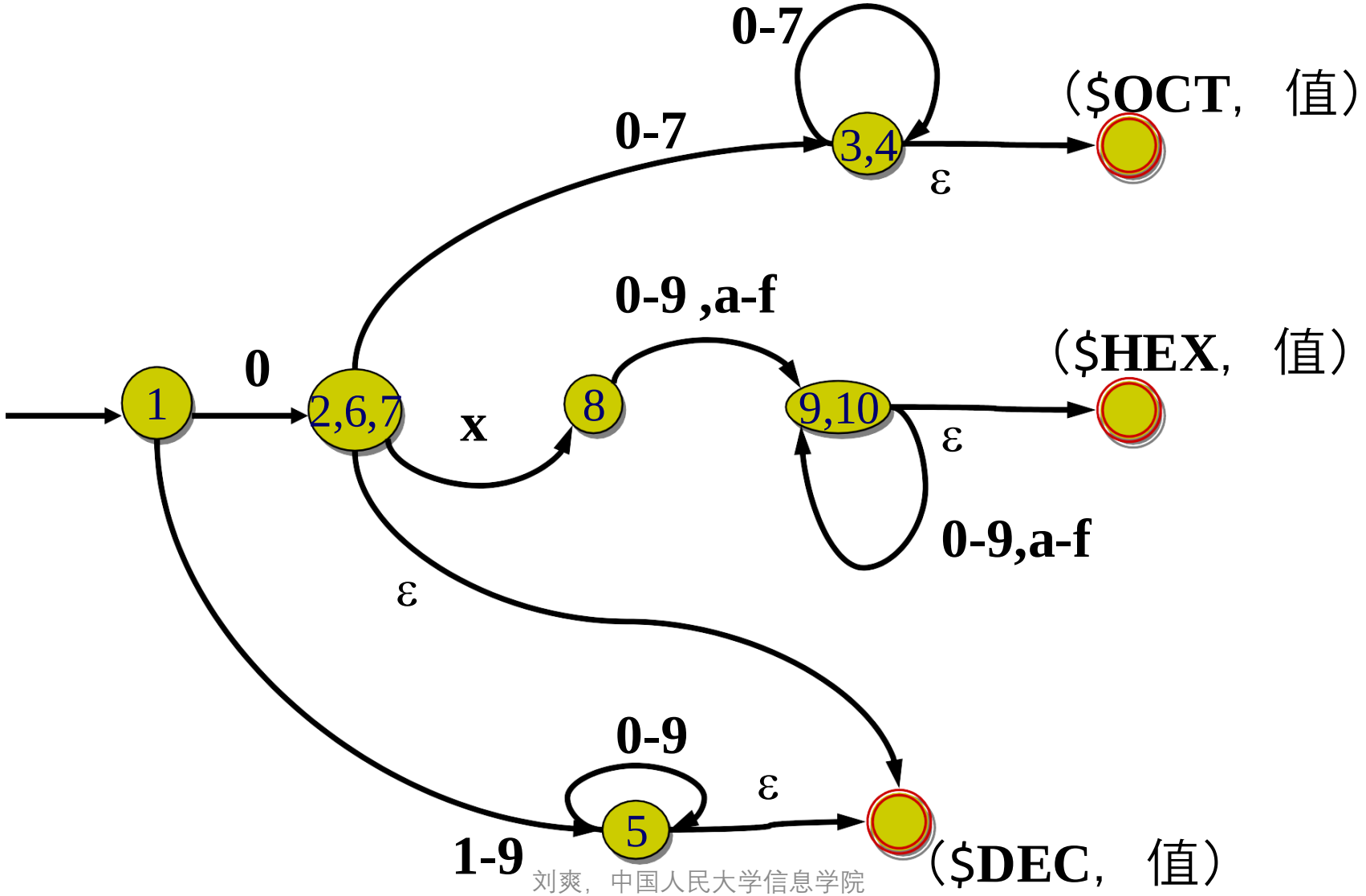


1、从开始状态出发；
2、选择输入符号，**构成目**标状态集
3、从新状态集出发，**重复1、2**

} **状**态图合并

自学

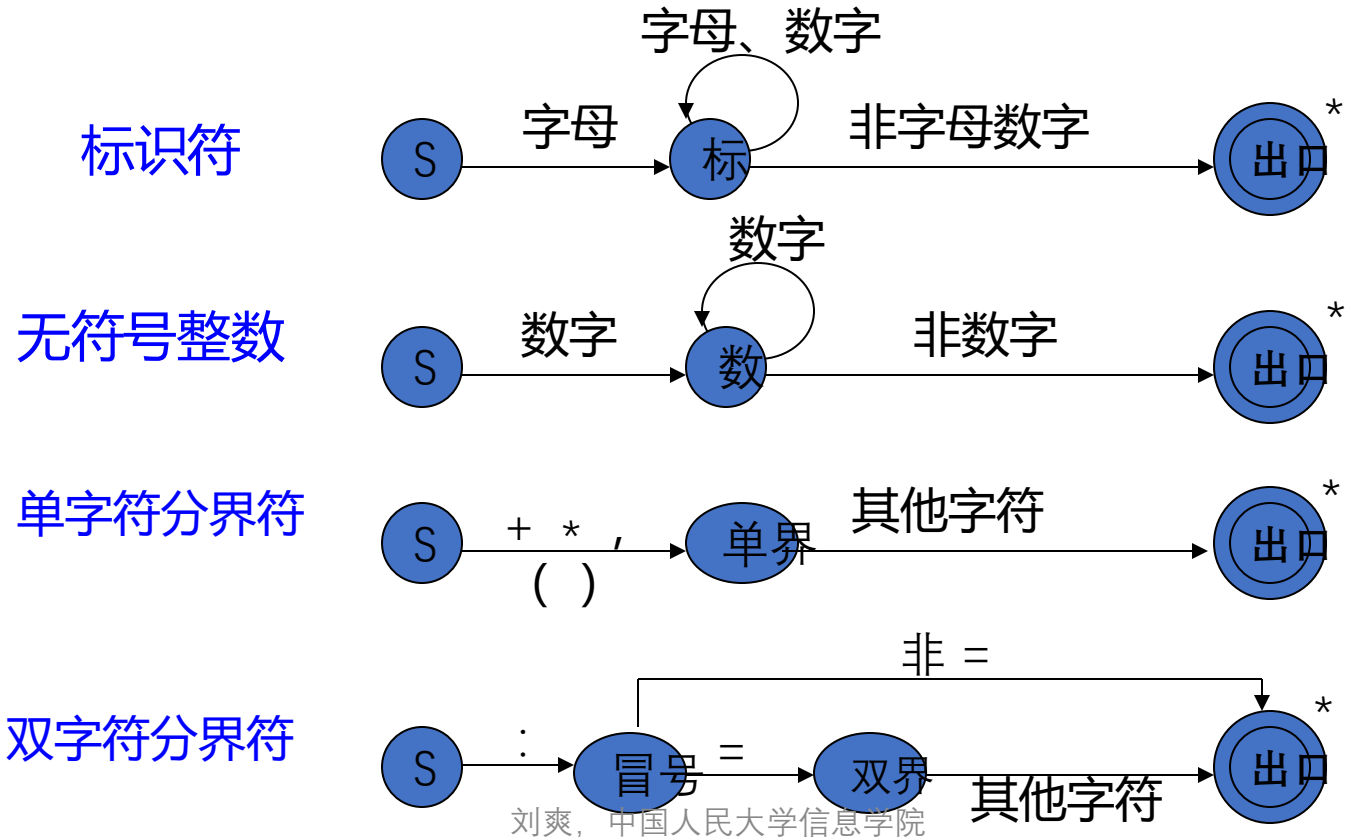
例：C语言无符号整数识别的状态图

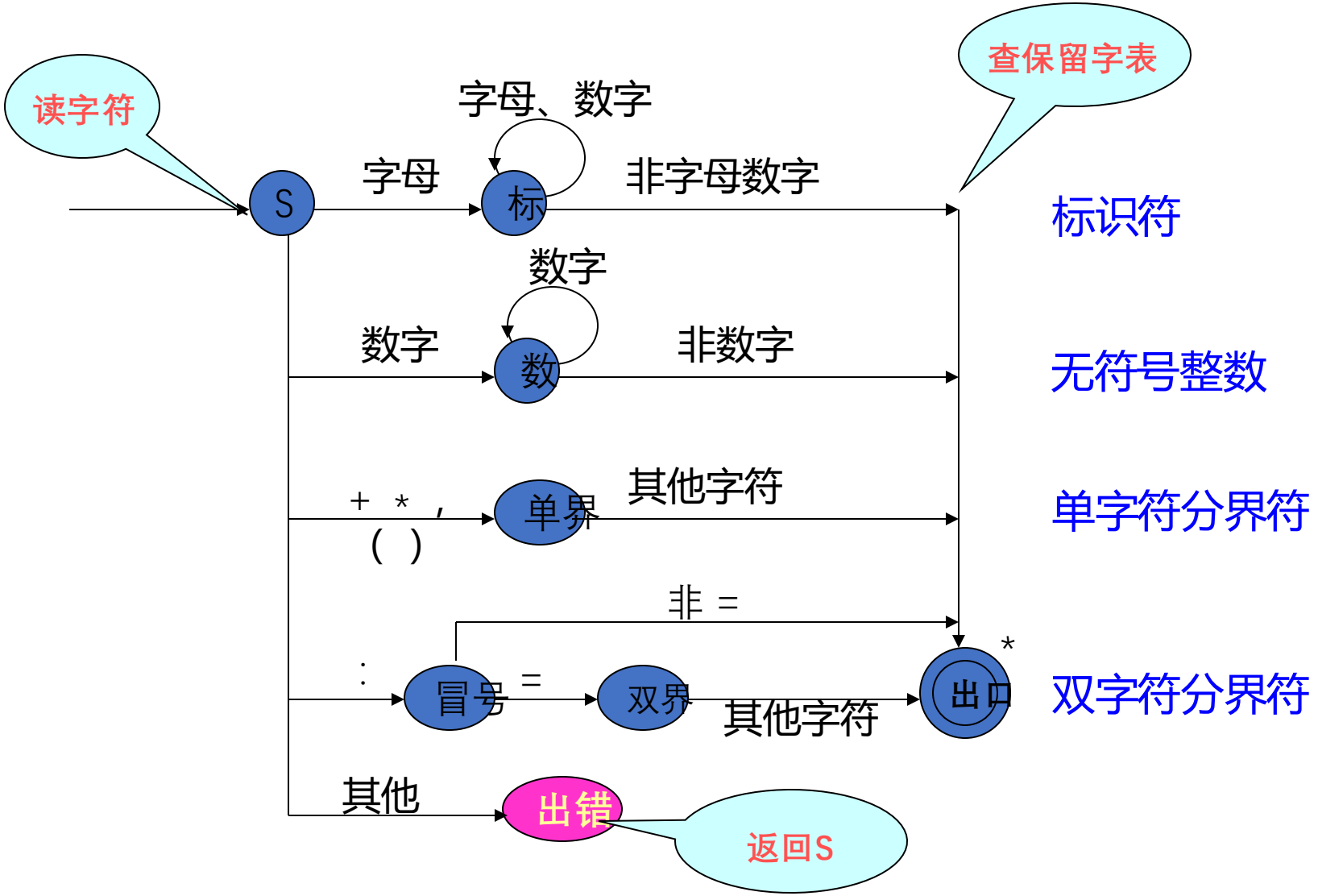


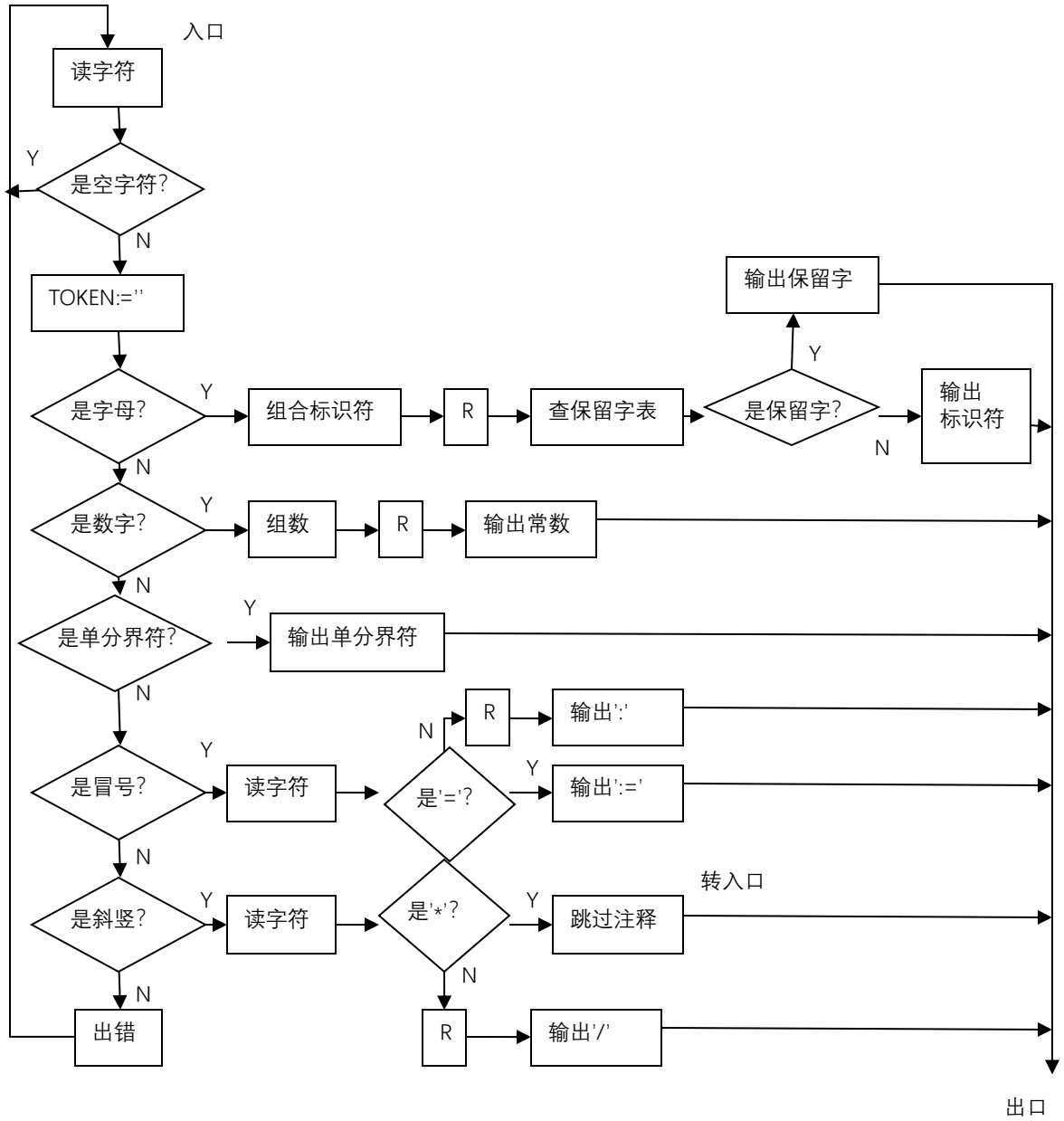
练习

- 画出下列描述对应的状态转换图
  - 1. 识别二进制的奇数
  - 2. 识别十进制的偶数

- 文法：1. <标识符>::= 字母 | <标识符>字母 | <标识符>数字
  2. <无符号整数>::=数字 | <无符号整数>数字
  3. <单字符分界符>::= : | + | * | , | (|)
  4. <双字符分界符>::= <冒号>=
  5. <冒号>::= ：




读字符
查保留字表
字母、数字
字母
非字母数字
S
标
标识符
数字
数字
非数字
数
无符号整数
其他字符
+ * ,
( )
单界
单字符分界符
非 =
*
:
=
冒号
双界
出口
双字符分界符
其他字符
其他
出错
返回S

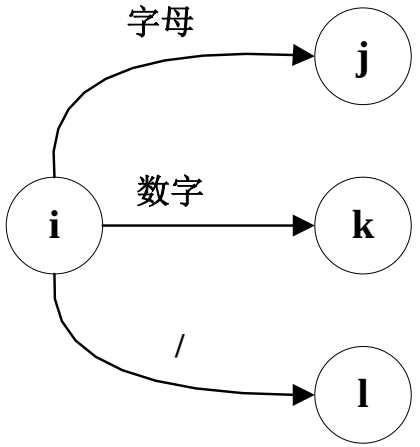

入口
读字符
是空字符?
Y
N
TOKEN:=''
是字母?
Y
N
组合标识符
R
查保留字表
是保留字?
Y
N
输出保留字
输出
标识符
是数字?
Y
N
组数
R
输出常数
是单分界符?
Y
N
输出单分界符
是冒号?
Y
N
读字符
是'='?
N
Y
R
输出':'
输出':='
是斜竖?
Y
N
读字符
是'*'?
Y
N
转入口
跳过注释
出错
R
输出'/'
出口

状态转换图的实现

- 程序实现:每个状态结点对应一段程序.

1) 不含回路的分叉结点:

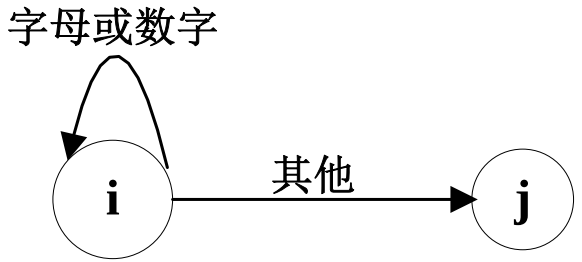
- CASE 或者 IF--THEN—ELSE




```
GetChar()
if (IsLetter())
    {状态j的对应程序段}
else if (IsDigit())
    {状态k的对应程序段}
else if (ch == '/')
    {状态l的对应程序段}
else
    {错误处理}
```

2) 含回路的分叉结点:

- WHILE




```
GetChar();
while (IsLetter() or IsDigit())
      GetChar();
 状态j的对应程序段
```

3) 终点结:

- RETURN(Code, Value)： 返回调用者

3型文法：

（左线性）

P： U ::= t

或 U ::= Wt

其中 U、W∈Vn

t∈Vt

（右线性）

P： U ::= t

或 U ::= tW

其中 U、W∈Vn

t∈Vt

3型文法称为正则文法。它是对2型文法进行进一步限制。

3型语言：L3　又称正则语言、正则集合
这种语言可以由有穷自动机接受。

3.3 正则文法和状态图

- **状态图的画法（根据文法画出状态图）**

例如：正则文法

Z::= U0 |V1

U ::=Z1 |1

V ::=Z0 | 0

左线性文法。该文法所定义的语言为：

$L(G[Z]) = \{ B^n \mid n>0 \}$，其中 B= {01,10}

例：正则文法

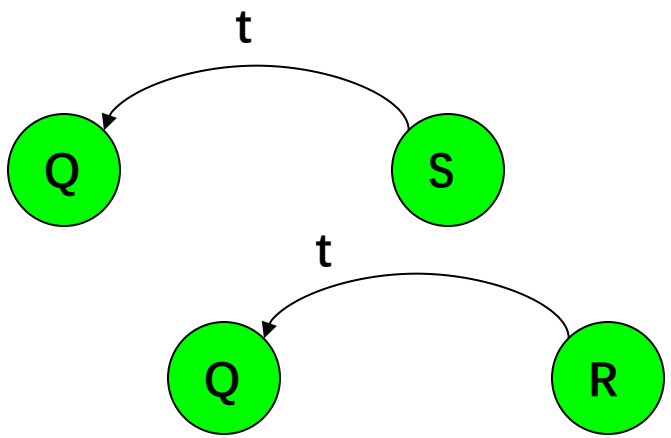
Z::= U0 |V1

U ::=Z1 |1

V ::=Z0 | 0

左线性文法的状态图的画法：

1. **令G的每个非终结符都是一个状态，识别符号为终止状态；**
2. **设一个开始状态S；**
3. **若Q::=t，Q ∈Vn, t∈Vt, 则：**



4. **若Q::=Rt, Q、R∈Vn,t ∈Vt, 则：**

5. **按自动机方法，可加上开始状态和终止状态标志。**

词法规则 → 状态图 → 词法分析程序

1. 每个非终结符设一个状态，识别符号为终止状态；
2. 设一个开始状态S；
3. 若Q::=t，Q ∈Vn,t ∈Vt,
4. 若Q::=Rt, Q、R∈Vn,t∈Vt,
5. 加上开始状态和终止状态标志

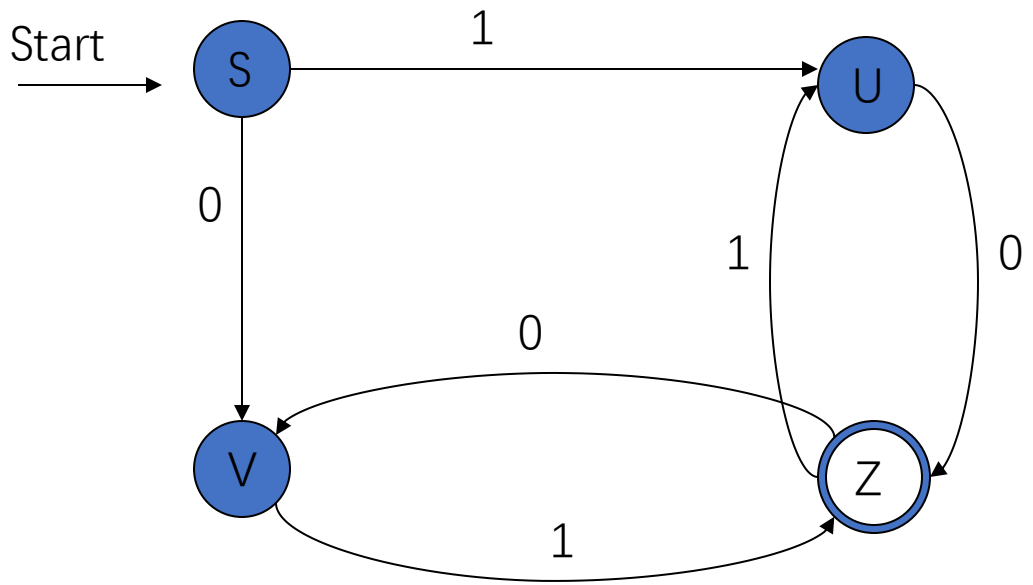
例如：正则文法

Z::= U0 |V1

U ::=Z1 |1

V ::=Z0 | 0

其状态图为：



Outline

- 词法分析器的功能
- 单词种类及词法分析程序的输出形式
- 状态转换图
- 词法分析程序的设计与实现

扫描器的设计与实现

- 词法的扫描器的实现:
- 子程序 scan( )
  - 输入：字符流
  - 输出:
    - code: 单词种别编码
    - value: 单词符号属性值
- 《程序设计语言编译原理（第三版）》section 3.2.4

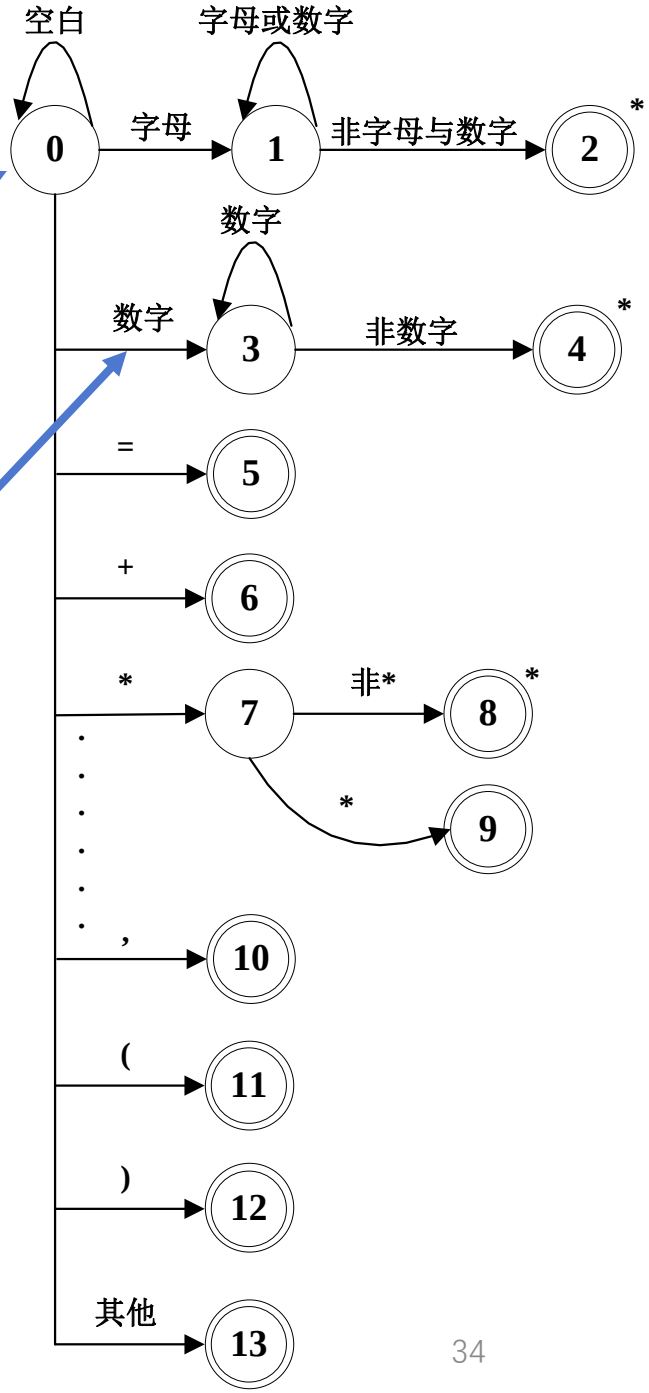
扫描器的设计与实现

- **数据结构**
  - ch　　字符变量，存放当前输入字符
  - token　　字符数组，存放构成单词符号的字符串
  - code　　单词符号种别
  - value　　单词符号属性
- **子例程**
  - isKeyword(token): 判别 token是关键字？返回关键字种别 或 -1
  - Lookup(token): 将 token 存入符号表，返回入口指针
  - getchar(): 从输入缓冲区中读入一个字符放入ch

单词符号	种别编码	助忆符
DIM	1	$DIM
IF	2	$IF
DO	3	$DO
STOP	4	$STOP
END	5	$END
标识符	6	$ID
常数	7	$INT
=	8	$ASSIGN
+	9	$PLUS
*	10	$STAR
**	11	$POWER
,	12	$COMMA
(	13	$LPAR
)	14	$RPAR

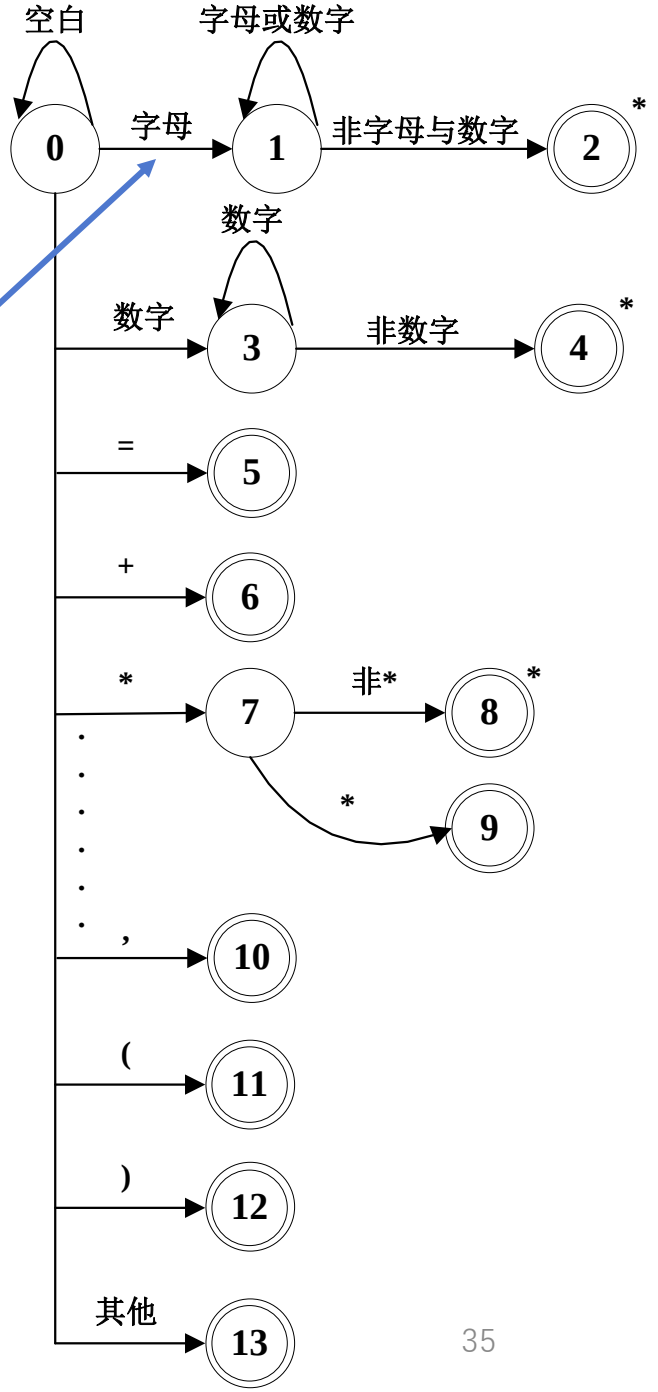
状态图的实现算法

```
getchar()
        WHILE ch 是空格
                DO getchar();     //跳过空格
        CASE ch OF
          isdigit(ch) :
             ch→token; getchar();
             WHILE isdigit(ch) DO
                ch→token; getchar();
             输入指针回退一个字符;
          Lookup(token)→value
          返回 （$INT, value）
```



状态图的实现算法

```
isalpha(ch) :
    ch→token; getchar();
    WHILE isalpha(ch) OR isdigit(ch)
        DO ch→token; getchar();
    输入指针回退一个字符;
    key = isKeyword(token);
    IF key≥0   THEN     返回 key
    Lookup(token)→value;
    返回 ($ID, value)
```

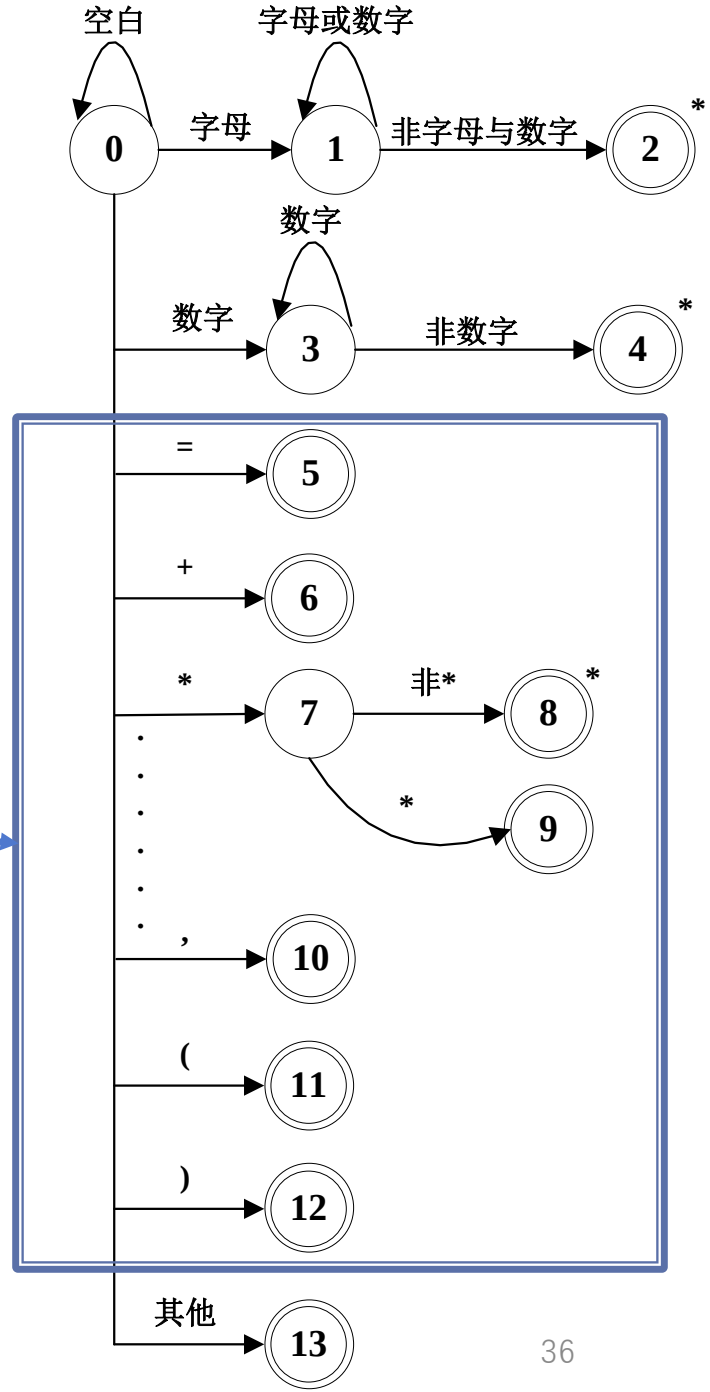


状态图的实现算法

```
‘=’:    返回 ($ASN, -)
‘+’:    返回 ($ADD, -)
'-':    返回 ($SUB, -)
'*':    返回 ($MUL, -)
'/':    返回 ($DIV, -)
'=':    返回 ($EQ, -)
'>':    返回 ($GT, -)
'<':    返回 ($LT, -)
'(':    返回 ($LP, -)
')':    返回 ($RP, -)
';':    返回 ($SEMI, -)
```

其它： 出错处理

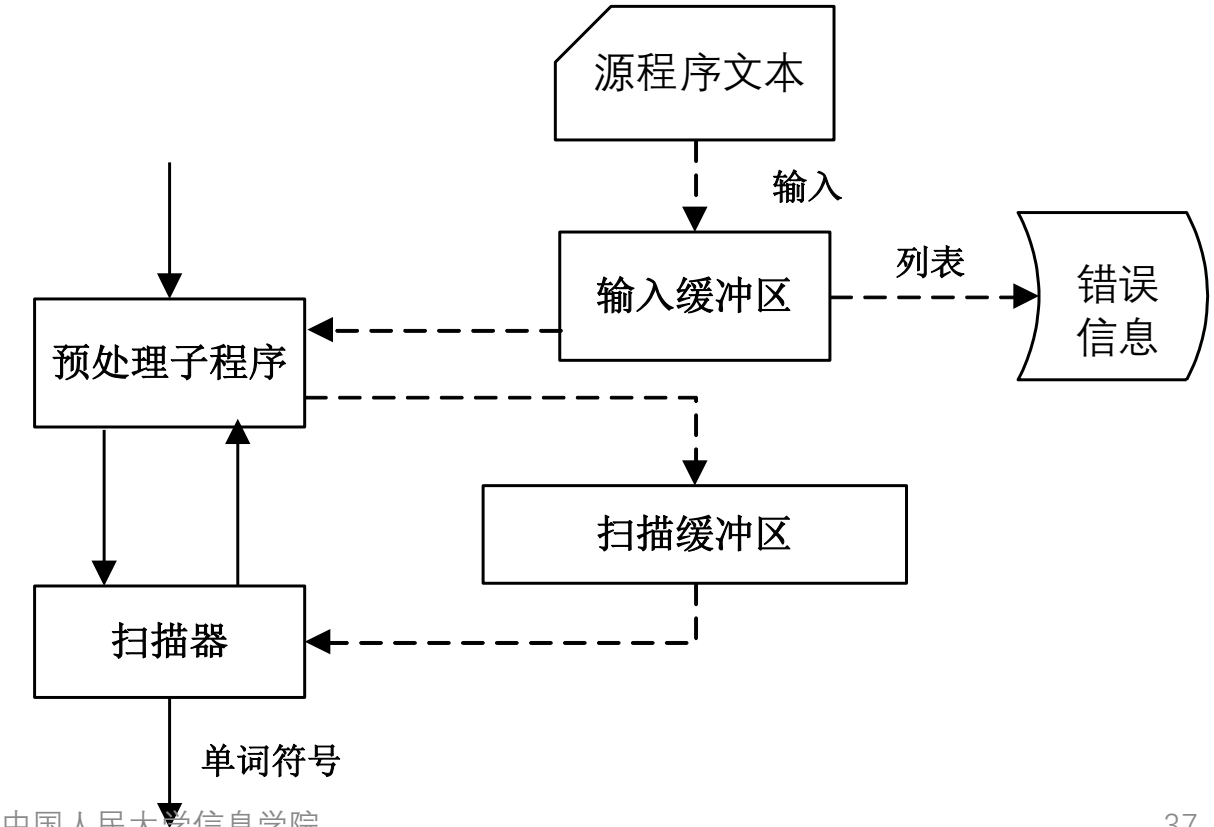
END OF CASE



词法分析器结构图

- 预处理:
  - 针对空白符、跳格符、回车符和换行符等编辑性字符的处理。
  - 出错信息的列表打印
- 扫描器
  - 输入预处理后的字符串
  - 输出单词符号

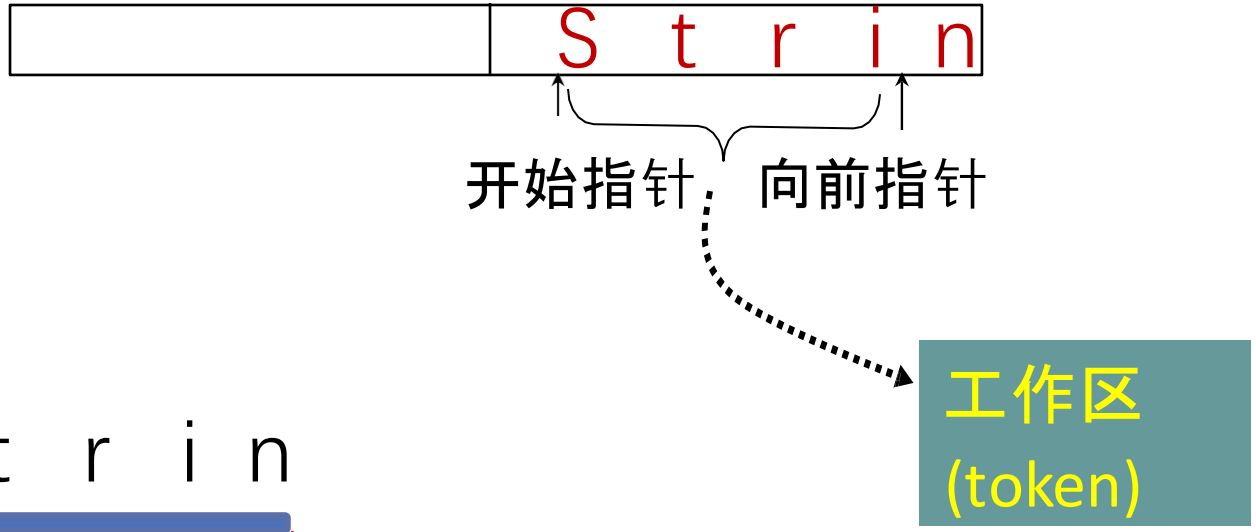
将词法分析器作为独立的子程序

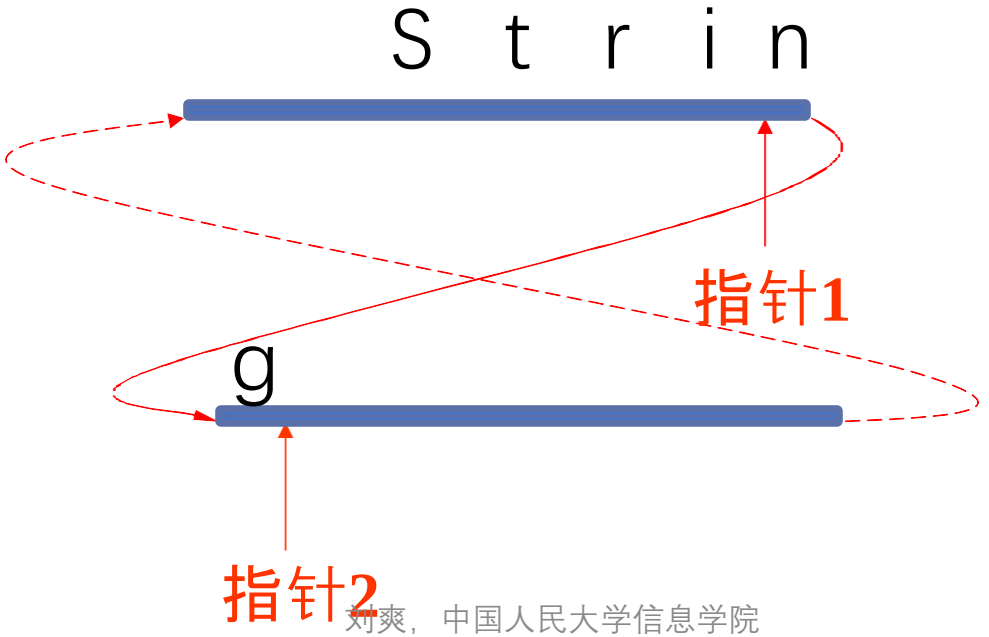


关于扫描缓冲区

- N个字符（如1024）

- 双缓冲区问题
  - 功能1：并行
  - 功能2：捻接





关键字的识别—超前搜索

- 像FORTRAN这样的语言，关键字不加以特殊保护，关键字和用户自定义的标识符或标号之间没有特殊的界符做间隔。这使得关键字的识别甚为麻烦。

1. DO99K = 1, 10
2. IF(5.EQ.M) I = 10
3. DO99K = 1.10
4. IF(5.EQ.M) = 55

  - 其中，语句1和2是DO和IF语句，他们都是以基本字开头的，语句3和4是赋值语句，都是以用户自定义的标识符开头的。
  - 需要超前搜索进行区别。

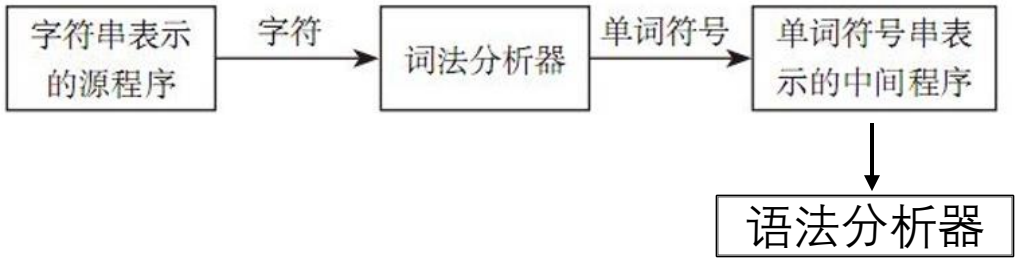
其他单词符号的识别

- 标识符的识别
  - 多数语言的标识符是字母开头的“字母/数字”串，而且在程序中标识符的出现都后跟着算符或者界符，因此标识符的识别没有太大的困难。
- 常数的识别
  - 基本来讲常数的识别也是很直接的，有些需要用到超前搜索。

    如5.E08和5.EQ.M，需要搜索到第四个字符
- 算符和界符的识别
  - 词法分析器将那些多个字符复合成的算符和界符(如++、--、>=)拼合成一个单词符号。因为这些单词符号是不可分的整体，若分划开来，便失去了原来的意义。这里需要超前搜索。

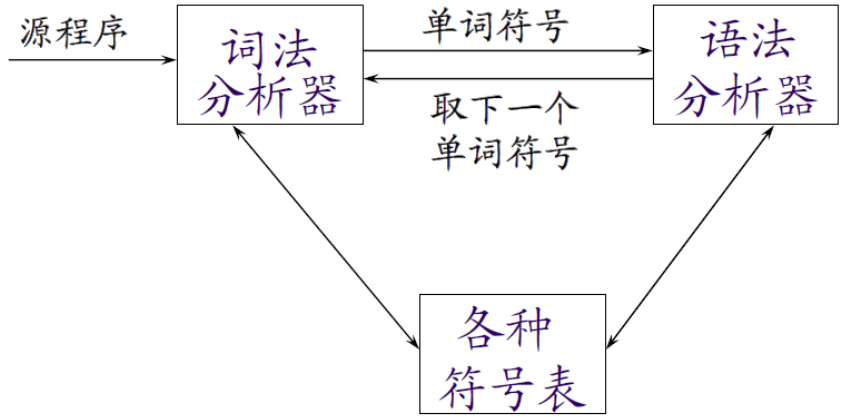
词法分析器在编译器中的位置

- 词法分析是编译的第一阶段。其任务是读入源程序输入字符，将它们组成词素，生成并输出一个词法单元（单词符号）序列。
- 词法分析可以是编译过程中的单独一遍，在语法分析前进行，如图（a）所示。也可以作为一个子程序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词法单元供语法分析使用，如图（b）所示。



（a）词法分析器作为一遍

结构清晰，但效率较差



（b）词法分析器作为一个子程序

结构较为复杂，但效率高

词法分析中的错误恢复

- 在词法分析阶段对错误进行纠正的假设前提：大多数词法错误是多、漏或错了一个字符或者相邻的两个字符错位的结果
- 最简单的错误恢复策略是“紧急方式”恢复，即反复删除掉剩余输入最前面的字符，直到词法分析器能发现一个正确的记号为止
- 其他的恢复动作包括：删除一个多余的字符、插入一个遗漏的字符、用一个正确的字符代替一个不正确的字符、交换两个相邻的字符
- 对错误进行修补最直观的策略是看：剩余输入的前缀能否通过上面的一个变换变成一个合法的词素

Summary

- 词法分析器的功能
- 单词种类及词法分析程序的输出形式
- 状态转换图
- 词法分析程序的设计与实现

阅读材料：《程序设计语言编译原理（第3版）》，陈火旺等编著，国防工业出版社，2004年，第三章
《编译原理与技术》张莉等编著，第三章