

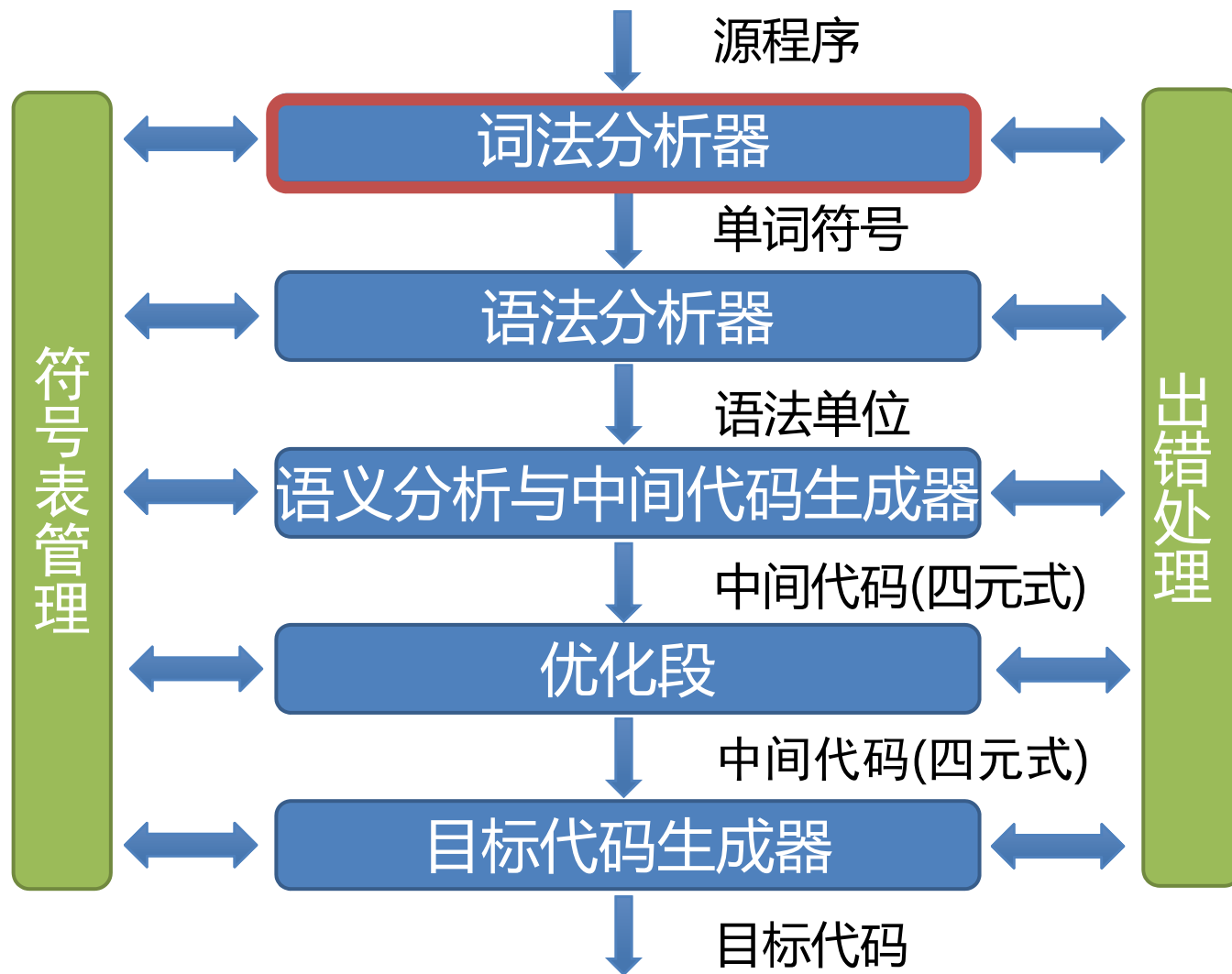
编译原理

词法分析器的设计--状态转换图

编译原理

词法分析概述

编译程序总框



词法分析

- ▶ 词法分析器的设计
- ▶ 正规表达式与有限自动机
- ▶ 词法分析器的自动产生--LEX

词法分析的任务

▶ 词法分析的任务

- ▶ 从左至右逐个字符地对源程序进行扫描，产生一个个单词符号

▶ 词法分析器(Lexical Analyzer)

- ▶ 扫描器(Scanner)
- ▶ 执行词法分析的程序

词法分析器的功能

► 功能

- 输入源程序、输出单词符号

► 单词符号的种类

- **基本字**：如 begin, repeat, for, ...
- **标识符**：用来表示各种名字，如变量名、数组名和过程名
- **常数**：各种类型的常数
- **运算符**：+, -, *, /, ...
- **界符**：逗号、分号、括号和空白

词法分析器的输出

- ▶ 输出的单词符号的表示形式
 - ▶ (单词种别, 单词自身的值)
- ▶ 单词种别通常用整数编码表示
 - ▶ 若一个种别只有一个单词符号, 则种别编码就代表该单词符号。假定基本字、运算符和界符都是一符一种。
 - ▶ 若一个种别有多个单词符号, 则对于每个单词符号, 给出种别编码和自身的值。
 - ▶ 标识符单列一种; 标识符自身的值表示成按机器字节划分的内部码
 - ▶ 常数按类型分种; 常数的值则表示成标准的二进制形式

例 PASCAL程序

► if (5 = m) goto 100

► 输出单词符号

if (34, -)

左括号 (2, -)

整常数 (20, '5' 的二进制)

等号 (6, -)

标识符 (26, 'm')

右括号 (16, -)

goto (30, -)

标号 (19, '100' 的二进制)

例 C程序

▶ while (i>=j) i--;

▶ 输出单词符号

< while, - >

< (, - >

< id, 指向i的符号表项的指针 >

< >=, - >

< id, 指向j的符号表项的指针 >

<), - >

< id, 指向i的符号表项的指针 >

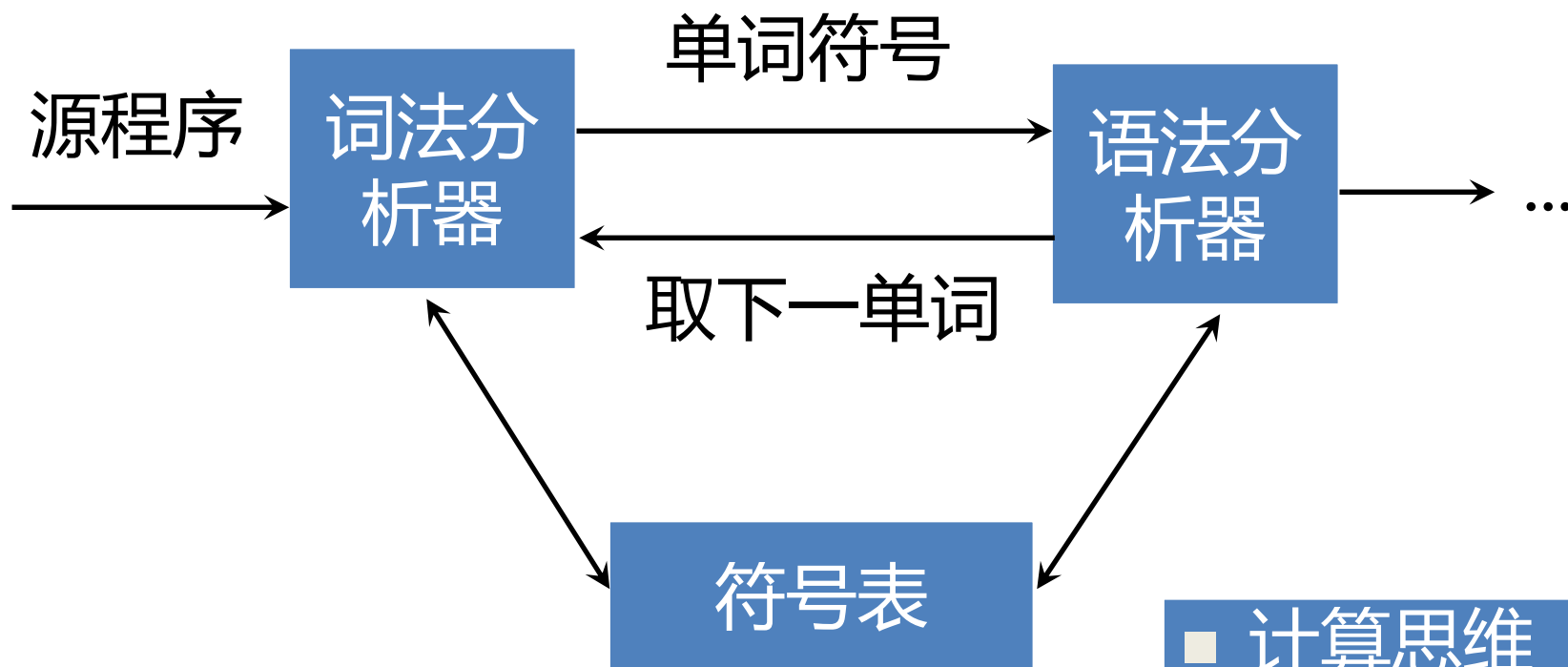
< --, - >

< ;, - >

词法分析器作为一个独立子程序

- ▶ 词法分析作为一个独立的阶段
 - ▶ 结构简洁、清晰和条理化，有利于集中考虑词法分析一些枝节问题
- ▶ 但不一定不作为单独的一遍
 - ▶ 将其处理为一个子程序

词法分析器在编译器中地位



■ 计算思维

□ 分解

□ 权衡

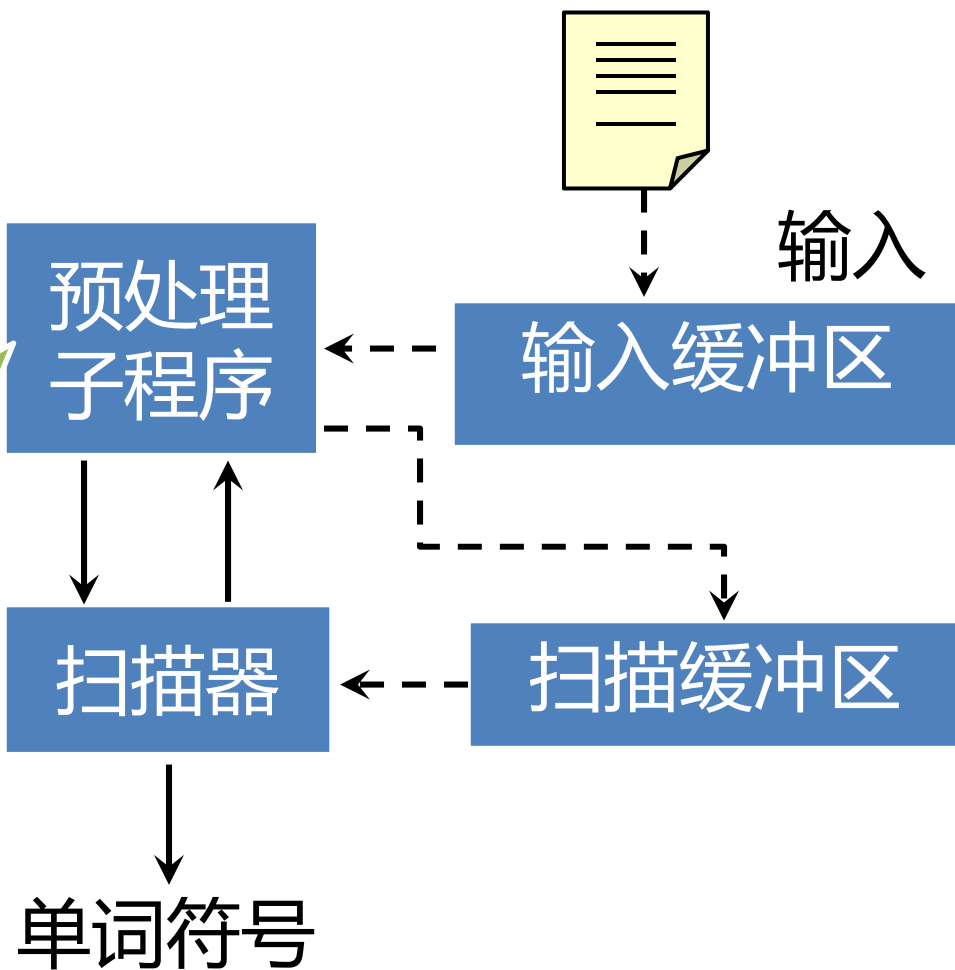
编译原理

词法分析器的结构

- 计算思维
 - 分解

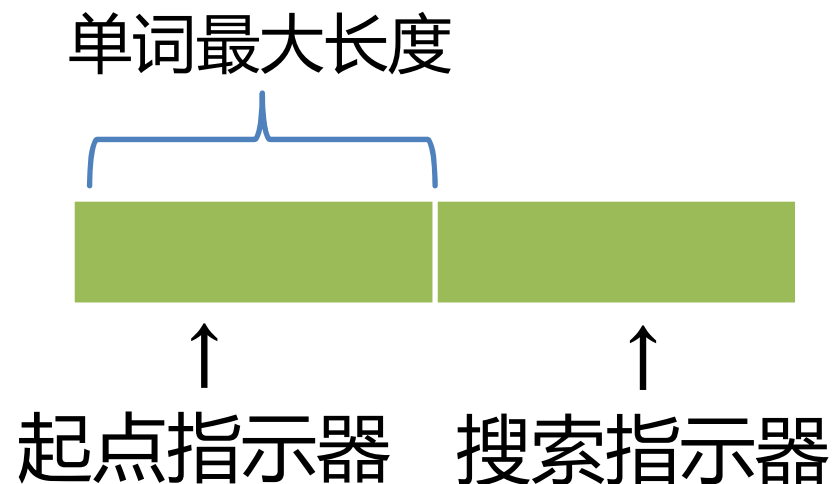
词法分析器的结构

- 剔除无用的空白、跳格、回车和换行等编辑性字符
- 区分标号区、捻接续行和给出句末符等

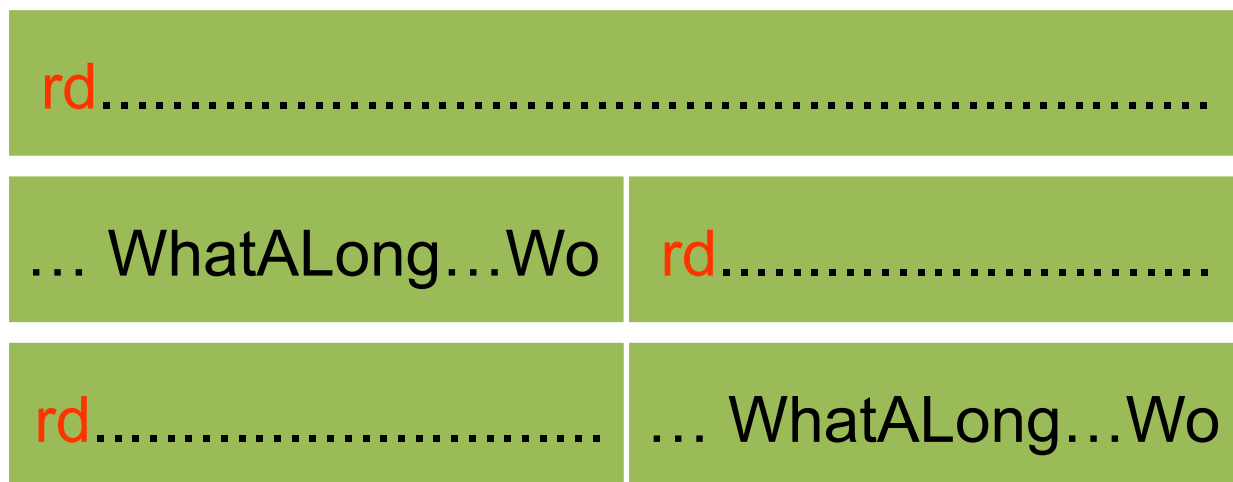


扫描缓冲区

- ▶ 两个半区互补使用



WhatALong...Word



编译原理

超前搜索

单词符号的识别:超前搜索

- ▶ 基本字识别

- ▶ 例如

DO99K=1, 10

DO 99 K = 1, 10

DO99K=1.10

IF (5.EQ.M) GOTO 55

IF(5.EQ.M)GOTO55

IF(5)=55

- ▶ 需要超前搜索才能确定哪些是基本字

单词符号的识别:超前搜索

▶ 标识符识别

- ▶ 字母开头的字母数字串, 后跟界符或算符

▶ 常数识别

- ▶ 识别出算术常数并将其转变为二进制内码表示

5.EQ.M

5.E08

▶ 算符和界符的识别

- ▶ 把多字符组成的算符和界符拼合成一个单词符号

:=, **, .EQ., ++, --, >=

几点限制——不必使用超前搜索

- ▶ 所有基本字都是保留字;用户不能用它们作自己的标识符
- ▶ 基本字作为特殊的标识符来处理, 使用保留字表
- ▶ 如果基本字、标识符和常数(或标号)之间没有确定的运算符或界符作间隔, 则必须使用一个空白符作间隔

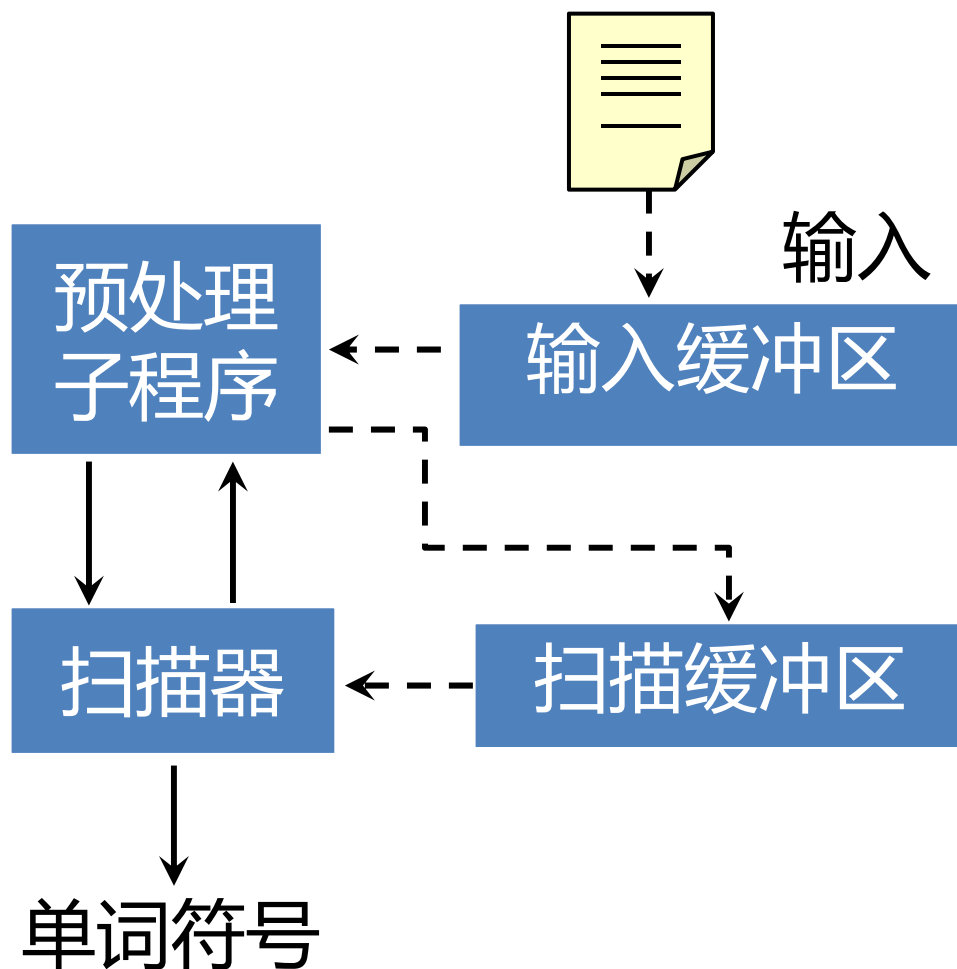
DO99K=1, 10

要写成 DO 99 K=1, 10

编译原理

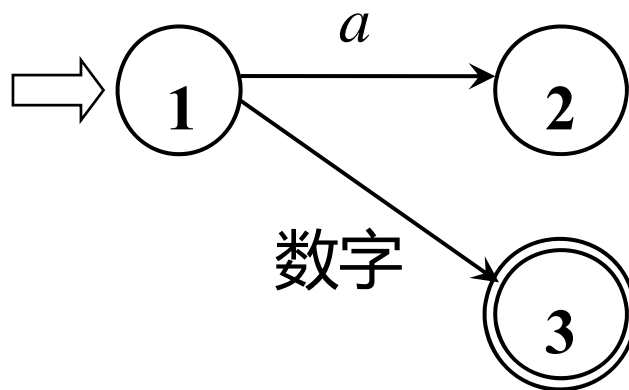
状态转换图

词法分析器的结构



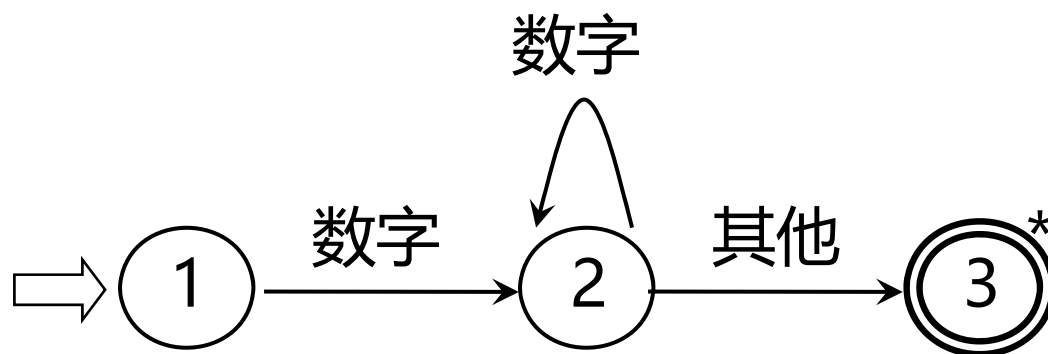
状态转换图

- ▶ 状态转换图是一张有限方向图
 - ▶ 结点代表状态，用圆圈表示
 - ▶ 状态之间用箭弧连结，箭弧上的标记(字符)代表射出该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，其中有一个为初态，至少要有有一个终态



状态转换图

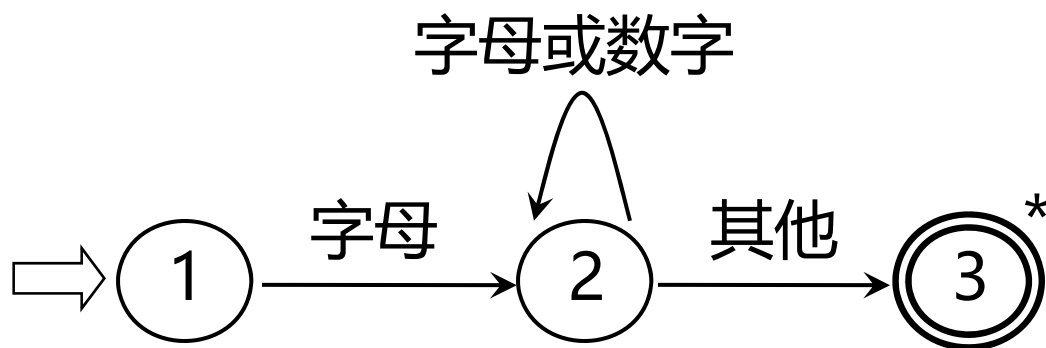
- ▶ 状态转换图可用于识别(或接受)一定的字符串
- ▶ 若存在一条从初态到某一终态的道路, 且这条路上所有弧上的标记符连接成的字等于 α , 则称 α 被该状态转换图所识别(接受)



识别整常数的状态转换图

状态转换图

- ▶ 状态转换图可用于识别(或接受)一定的字符串
- ▶ 若存在一条从初态到某一终态的道路, 且这条路上所有弧上的标记符连接成的字等于 α , 则称 α 被该状态转换图所识别(接受)



识别标识符的状态转换图

词法分析器的设计示例

► 一个简单的程序设计语言

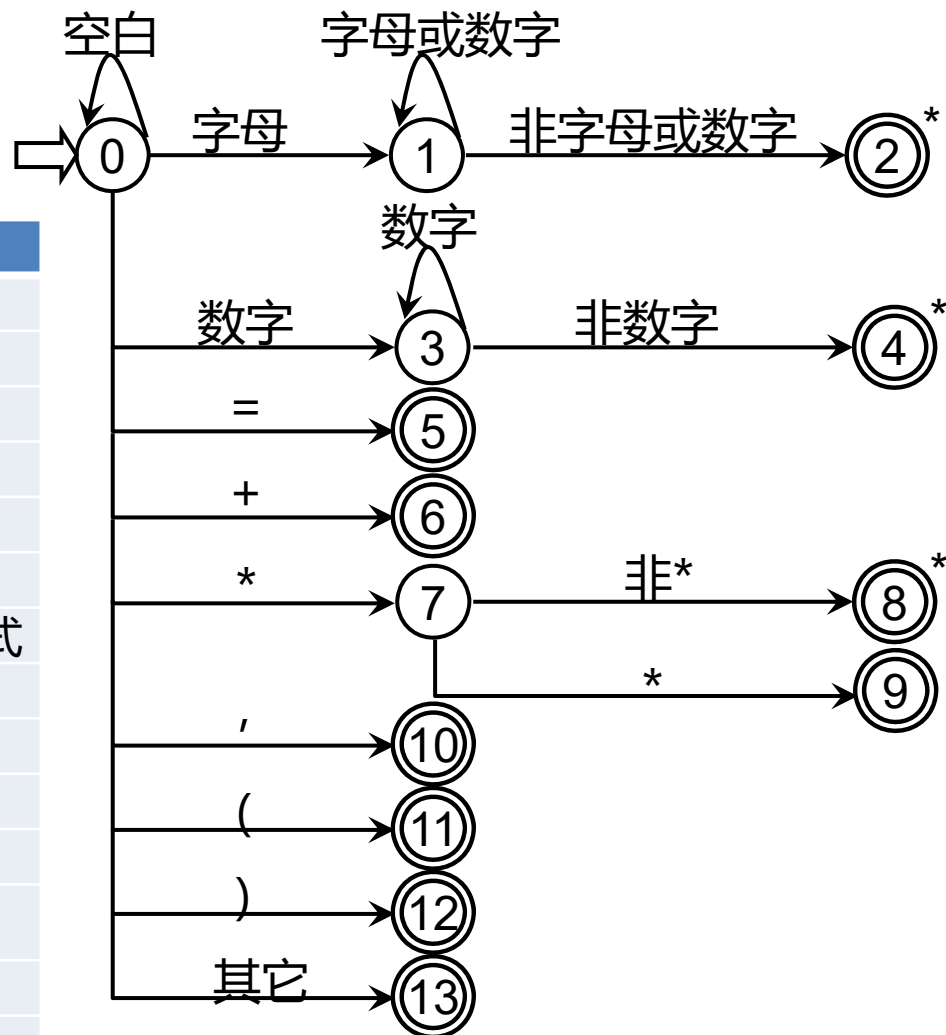
► 单词表

单词符号	种别编码	助忆符	内码值
DIM	1	\$DIM	-
IF	2	\$IF	-
DO	3	\$DO	-
STOP	4	\$STOP	-
END	5	\$END	-
标识符	6	\$ID	内部字符串
常数(数)	7	\$INT	标准二进制形式
=	8	\$ASSIGN	-
+	9	\$PLUS	-
*	10	\$STAR	-
**	11	\$POWER	-
,	12	\$COMMA	-
(13	\$LPAR	-
)	14	\$RPAR	-

词法分析器的设计示例

► 设计状态转换图

单词符号	种别编码	助忆符	内码值
DIM	1	\$DIM	-
IF	2	\$IF	-
DO	3	\$DO	-
STOP	4	\$STOP	-
END	5	\$END	-
标识符	6	\$ID	内部字符串 标准二进制形式
常数(数)	7	\$INT	
=	8	\$ASSIGN	-
+	9	\$PLUS	-
*	10	\$STAR	-
**	11	\$POWER	-
,	12	\$COMMA	-
(13	\$LPAR	-
)	14	\$RPAR	-



几点限制——不必使用超前搜索

- ▶ 所有基本字都是保留字;用户不能用它们作自己的标识符
- ▶ 基本字作为特殊的标识符来处理, 使用保留字表
- ▶ 如果基本字、标识符和常数(或标号)之间没有确定的运算符或界符作间隔, 则必须使用一个空白符作间隔

DO99K=1, 10

要写成 DO 99 K=1, 10

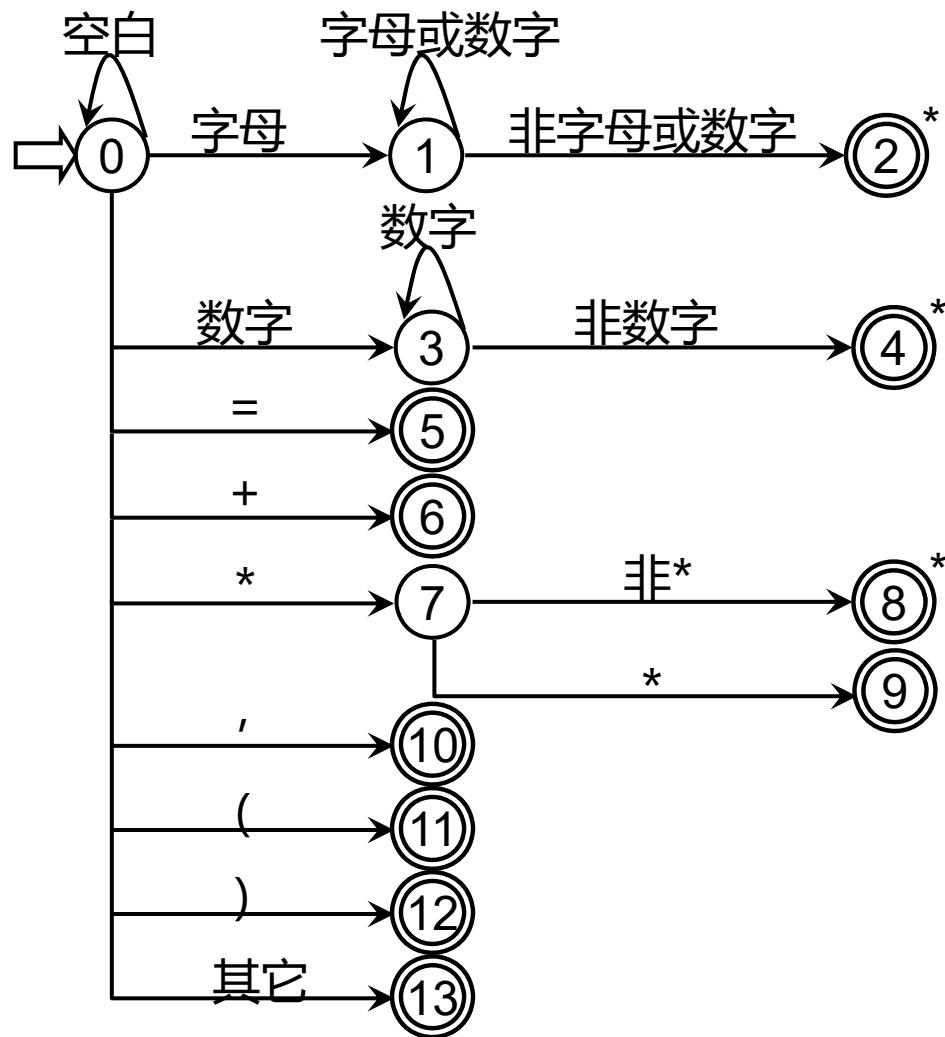
编译原理

状态转换图的程序实现

词法分析器的设计示例

▶ 状态转换图的代码实现

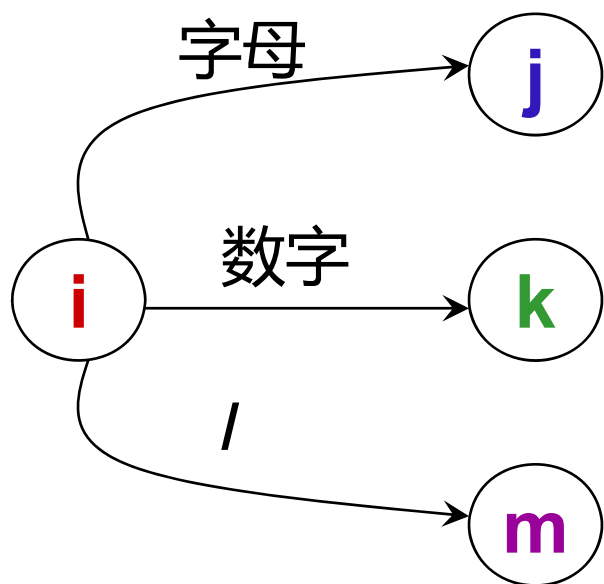
- ▶ 每个状态结点对应一小段程序



状态转换图的实现

▶ 不含回路的分叉结点

- ▶ 可用一个CASE语句或一组IF-THEN-ELSE语句实现



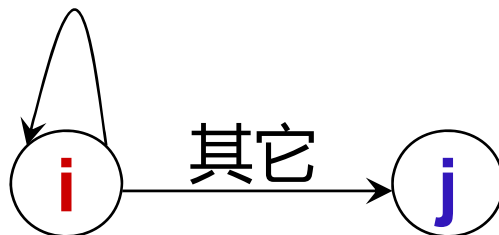
```
GetChar( );  
if (IsLetter( ))  
    {...状态j的对应程序段...;}  
else if (IsDigit( ))  
    {...状态k的对应程序段...;}  
else if (ch=='/')  
    {...状态m的对应程序段...;}  
else  
    {...错误处理...;}
```

状态转换图的实现

▶ 含回路的状态结点

- ▶ 对应一段由**WHILE**结构和**IF**语句构成的程序

字母或数字

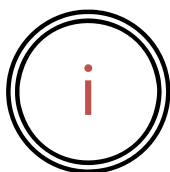


```
GetChar( );  
while (IsLetter( ) or IsDigit( ))  
    GetChar( );  
...状态j的对应程序段...
```

状态转换图的实现

▶ 终态结点

- ▶ 表示识别出某种单词符号，对应返回语句



RETURN (C, VAL)

其中，C为单词种别，VAL为单词自身值

状态转换图的实现

▶ 全局变量与过程

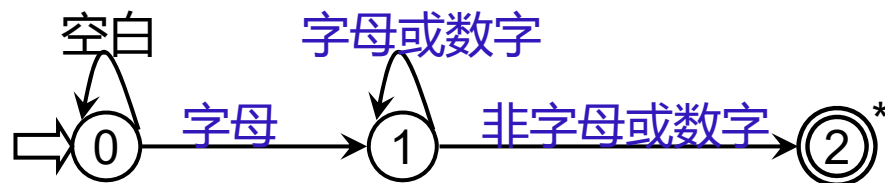
- ▶ **ch** 字符变量，存放最新读入的源程序字符
- ▶ **strToken** 字符数组，存放构成单词符号的字符串
- ▶ **GetChar** 子程序过程，把下一个字符读入到 ch 中
- ▶ **GetBC** 子程序过程，跳过空白符，直至 ch 中读入一非空白符
- ▶ **Concat** 子程序，把ch中的字符连接到 strToken

状态转换图的实现

▶ 全局变量与过程

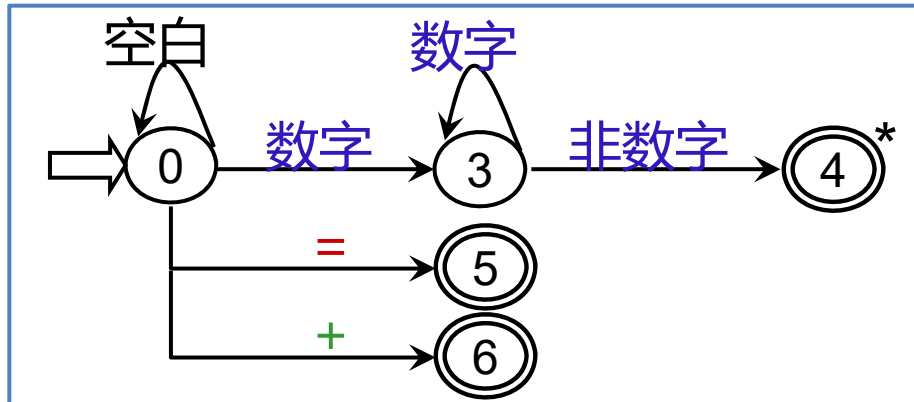
- ▶ **IsLetter**和 **IsDisgital** 布尔函数，判断ch中字符是否为字母和数字
- ▶ **Reserve** 整型函数，对于 strToken 中的字符串查找保留字表，若它是保留字则给出它的编码，否则回送0
- ▶ **Retract** 子程序，把搜索指针回调一个字符位置
- ▶ **InsertId** 整型函数，将strToken中的标识符插入符号表，返回符号表指针
- ▶ **InsertConst** 整型函数过程，将strToken中的常数插入常数表，返回常数表指针

词法分析器的实现



```
int code, value;
strToken := ""; /*置strToken为空串*/
GetChar(); GetBC();
if (IsLetter())
begin
    while (IsLetter() or IsDigit())
    begin
        Concat(); GetChar();
    end
    Retract();
    code := Reserve();
    if (code = 0)
    begin
        value := InsertId(strToken);
        return ($ID, value);
    end
    else
        return (code, -);
end
```

词法分析器的实现



```
else if (IsDigit())
begin
    while (IsDigit())
    begin
        Concat( ); GetChar( );
    end
    Retract();
    value := InsertConst(strToken);
    return($INT, value);
end
else if (ch = '=') return ($ASSIGN, -);
else if (ch = '+') return ($PLUS, -);
```

词法分析器的实现

```
else if (ch == '*')
```

```
begin
```

```
    GetChar();
```

```
    if (ch == '*') return ($POWER, -);
```

```
    Retract(); return ($STAR, -);
```

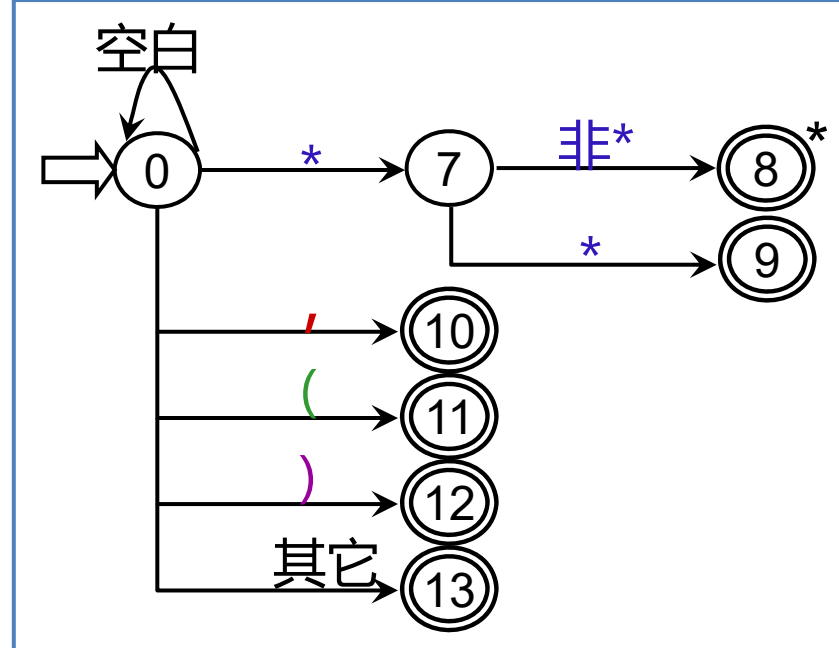
```
end
```

```
else if (ch == ',') return ($COMMA, -);
```

```
else if (ch == '(') return ($LPAR, -);
```

```
else if (ch == ')') return ($RPAR, -);
```

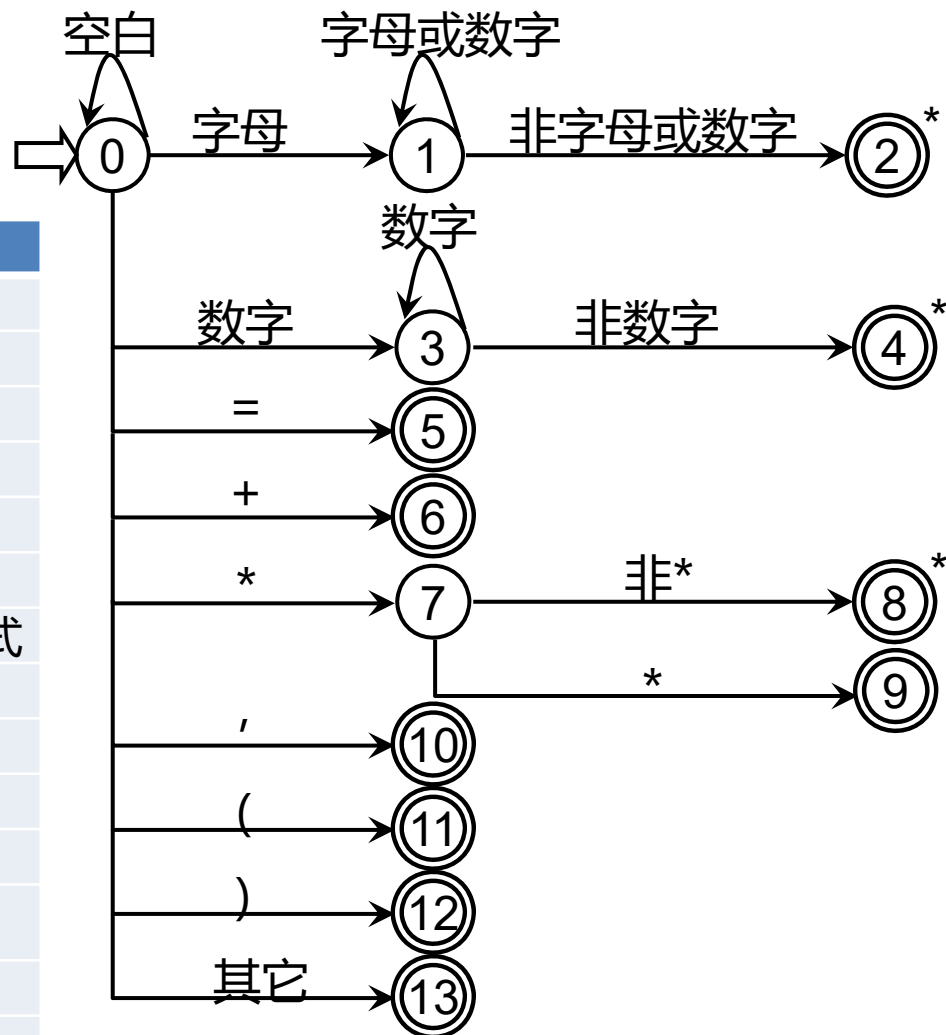
```
else ProcError( );    /* 错误处理 */
```



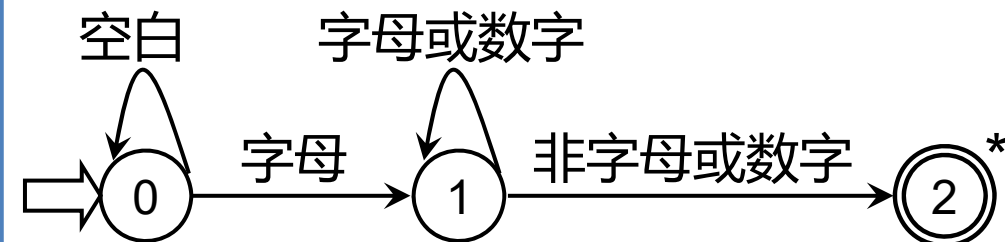
词法分析器的设计示例

► 设计状态转换图

单词符号	种别编码	助忆符	内码值
DIM	1	\$DIM	-
IF	2	\$IF	-
DO	3	\$DO	-
STOP	4	\$STOP	-
END	5	\$END	-
标识符	6	\$ID	内部字符串 标准二进制形式
常数(数)	7	\$INT	
=	8	\$ASSIGN	-
+	9	\$PLUS	-
*	10	\$STAR	-
**	11	\$POWER	-
,	12	\$COMMA	-
(13	\$LPAR	-
)	14	\$RPAR	-



将状态图的代码



▶ 变量curState用于保存现有的状态

▶ 用二维数组表示状态图：state

只是个框架，还有很多细节需要考虑！
是否有自动的方法产生词法分析程序？

```
curState = 初态
GetChar();
while( stateTrans[curState][ch]有定义){
    //存在后继状态，读入、拼接
    Concat();
    //转换入下一状态，读入下一字符
    curState= stateTrans[curState][ch];
    if curState是终态 then 返回strToken中的单词
    GetChar();
}
```

编译原理

小结

小结

▶ 词法分析器的功能

- ▶ 输入源程序、输出单词符号

▶ 词法分析器的设计

- ▶ 给出程序设计语言的单词规范——单词表
- ▶ 对照单词表设计识别该语言所有单词的状态转换图
- ▶ 根据状态转换图编写词法分析程序

是否有自动的方法产生词法分析程序？