编译原理与设计

计卫星　　王贵珍

北京理工大学　计算机学院

词法分析：设计与实现

- 单词与属性字
  - 单词是语言中具有**独立意义**的**最小语法单位**
  - 单词是受**词法规则**制约

例如，

…A*B…，单词是“A”、“*”和“B”。

…int int1…,单词是“int”和“int1”。

…A++*B…,单词是“A”、“++”、“*”和“B”。

类比，

我爱吃苹果，单词是…

词法分析：设计与实现

- 单词与属性字
  - FORTRAN 实型常数定义
    - F型: ± a. ， ±.b ， ±a.b
    - E型: ±a.E±d , ±a.bE±d , ±aE±d , ±.bE±d
    - D型: ±a.D±d , ±a.bD±d , ±aD±d , ±.bD±d

例如， **-3.　+.5　-3.5**

-3.E+6　-3.5E+6　-3E-3　-.5E-3

-3.D+6　-3.5D+6　-3D-3　-.5D-3

词法分析：设计与实现

- 常见单词分类
  - **关键字**(亦称保留字, 基本字等)：关键字一般是语言系统本身定义的, 通常是由字母组成的字符串。

    例如： C语言中的

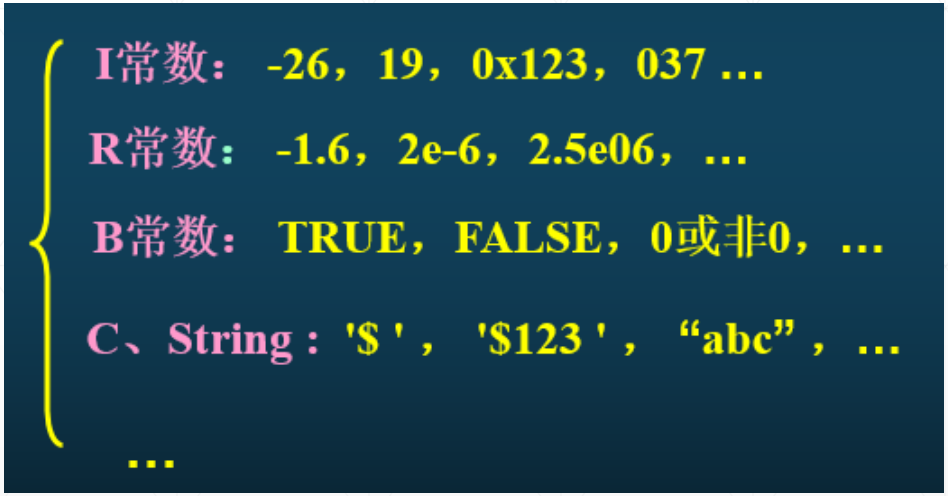
    int, for, break, static, char, switch, unsigned

    等， 关键字一般关联到语句的性质 。

  - **标识符**：用来表示各类名字的标识。如变量名、数组名、结构名、函数名等。

词法分析：设计与实现

- 常见单词分类
  - **常量**：如整型常数, 实型常数,不同进制的常数, 布尔常数,字符及字符串常数等。



  - Python中的单引号和双引号

词法分析：设计与实现

- 常见单词分类
  - **运算符**：表示程序中算术运算, 逻辑运算, 字符及位串等运算的确定字符(或串)。

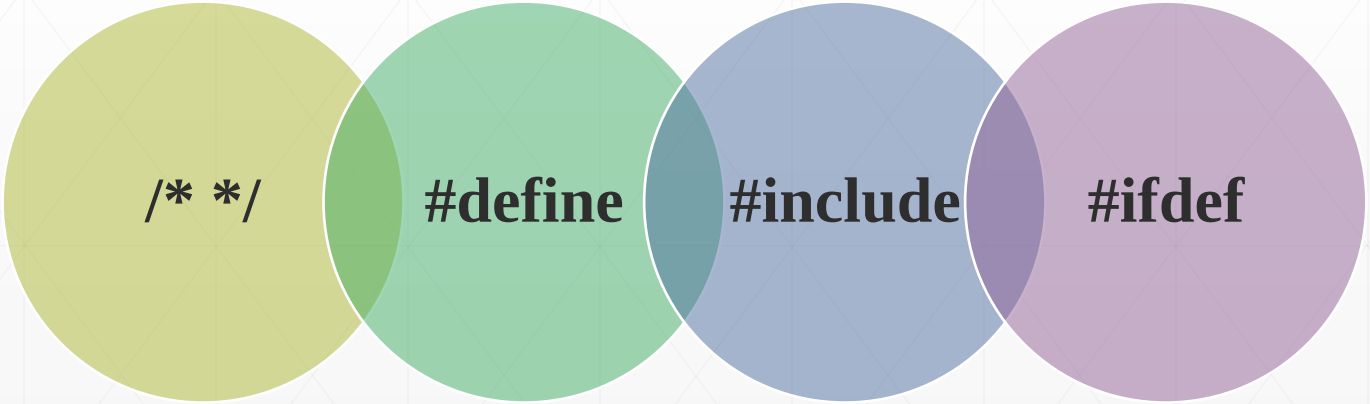
例如，各类语言较通用的＋，－，*，/，**，
＜＝，＞＝，＜，＞等。
还有一些语言特有的运算符，如C语言中的++，
？：，&，％＝等。
FORTRAN 语言中的 .AND.，.NOT.，.OR.。

sizeof

  - **界限符**：如逗号, 分号, 括号, 单双引号等。

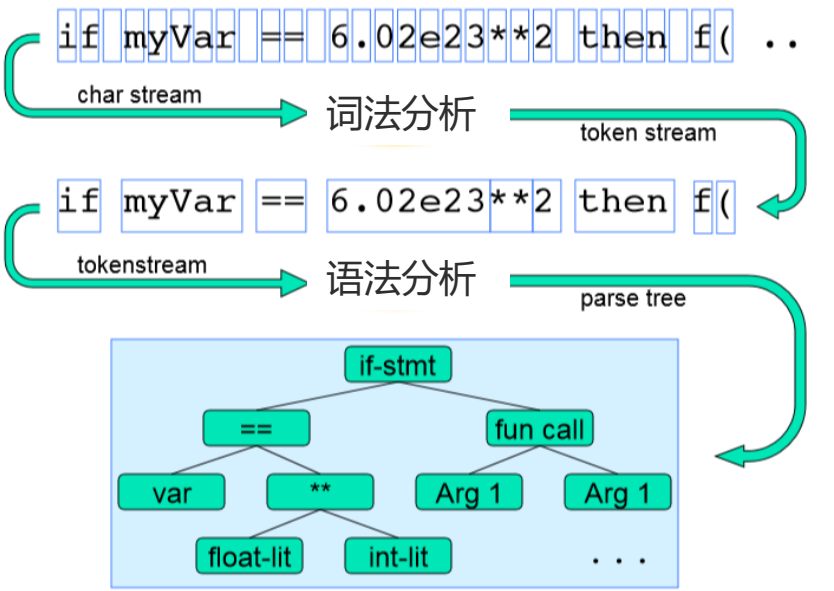
词法分析：设计与实现

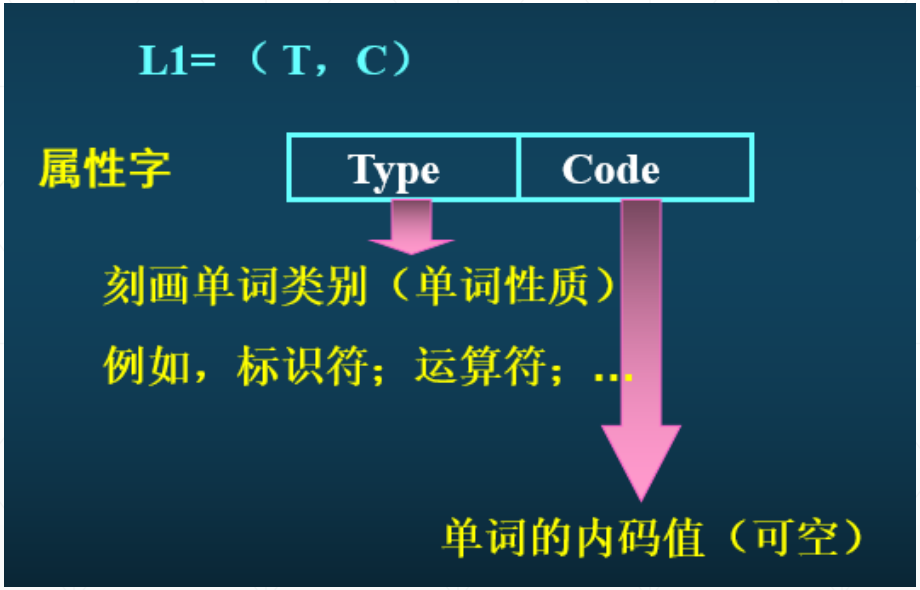
- 常见单词分类
  - 同一个字符开头跨单词类
    - -2.6， --j,->,a-b,-=
    - 0.0, 0x8a, 037
  - 非单词成分和预处理部分



词法分析：设计与实现

- 属性字（Token）





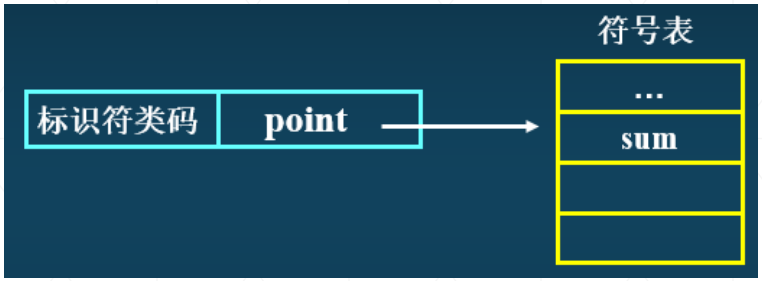
词法分析：设计与实现

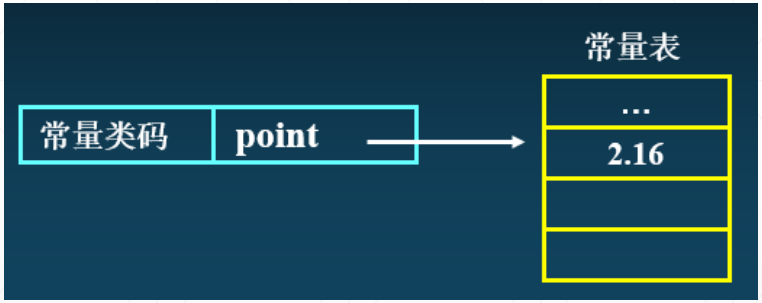
- 属性字（Token）
  - 早期设计

  - 视具体情况而定

词法分析：设计与实现

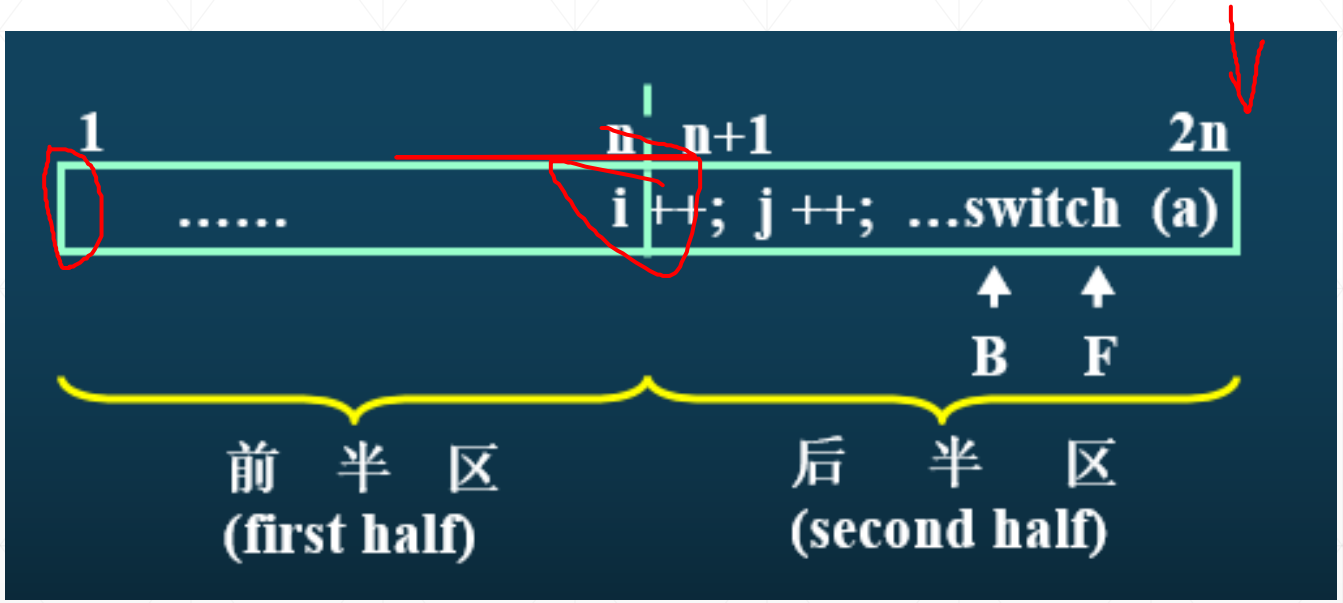
- 属性字（Token）设计
  - 标识符：类码+符号表
  - 关键字：类码
  - 常量：类码+符号表
  - 运算符：类码
  - 界限符：类码





词法分析：设计与实现

- 输入组织
  - 早期：对半互补缓冲区



词法分析：设计与实现

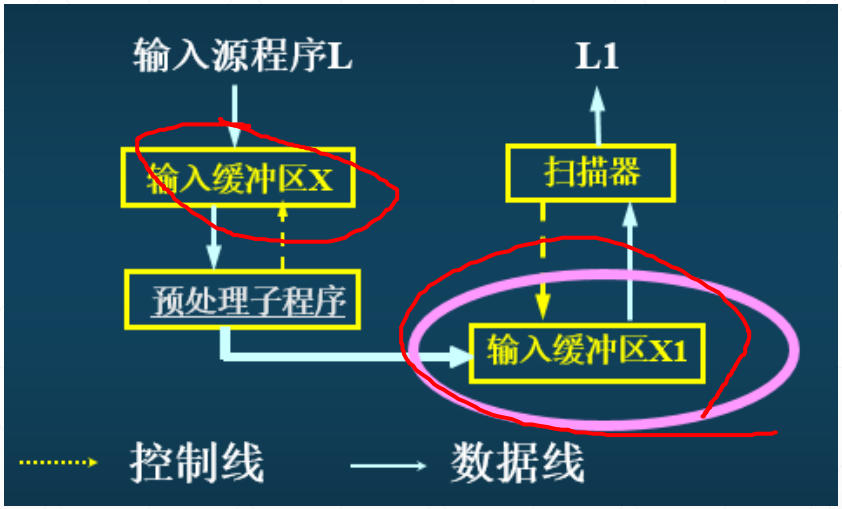
- 输入组织

两半区互补功能算法:

```
if ( F= 前 “ eof ”)
   { 重新分配、装入后半区;F++ };
else  if  ( F= 后 “ eof ”)
   {重新分配、装入前半区；  F=1};
else  F++;
```

词法分析：设计与实现

- 输入组织
  - 预处理的作用
    - 滤掉注释
    - 宏替换
    - 文件包含的嵌入
    - 条件编译处理

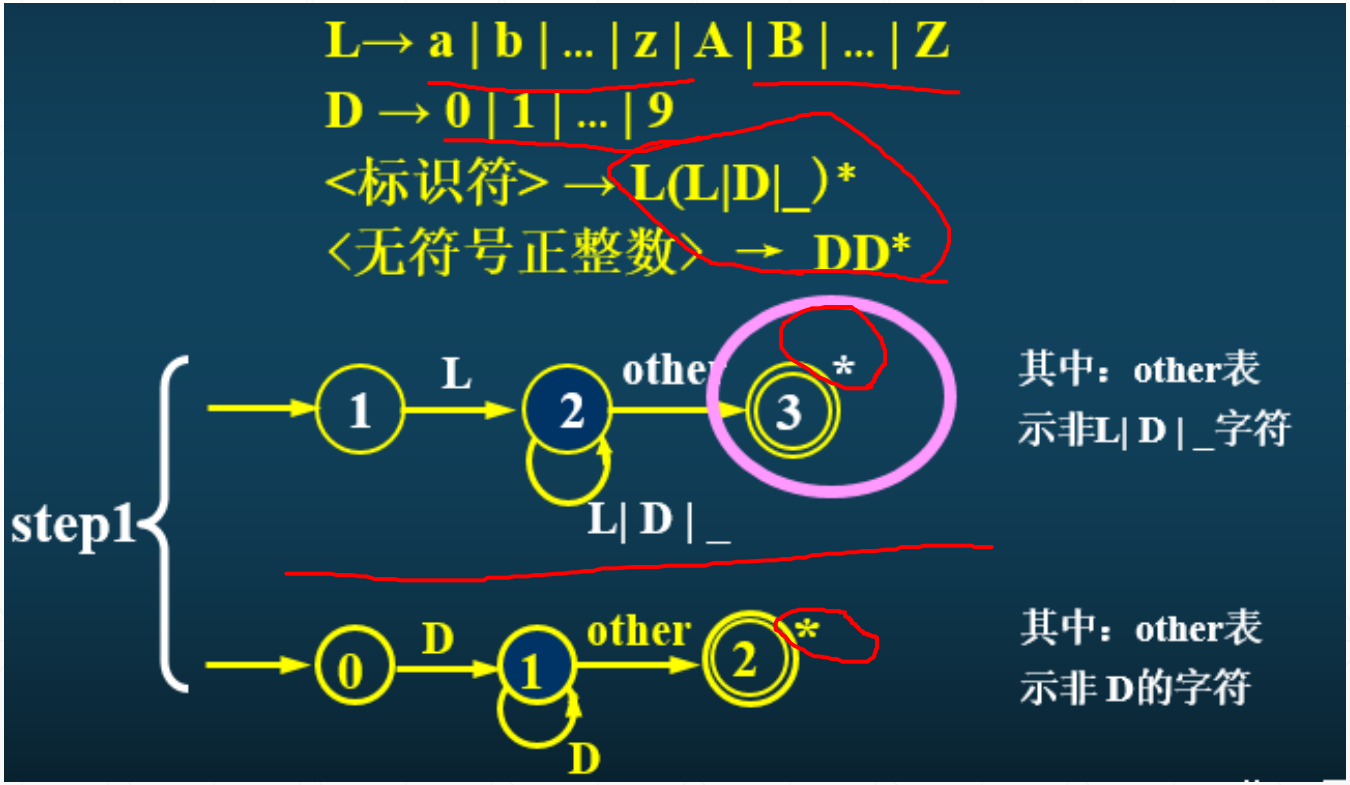


词法分析：设计与实现

- 词法分析器设计
  - 对语言的各类单词分别构造状态图
  - 将各类状态图合并，构成一个能识别该语言所有单词的状态图
  - 将各类单词的状态图的初态合并为惟一初态
  - 调整冲突编号

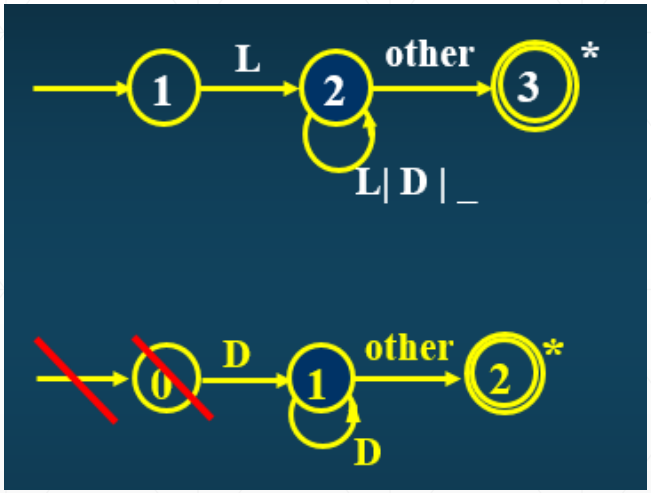
词法分析：设计与实现

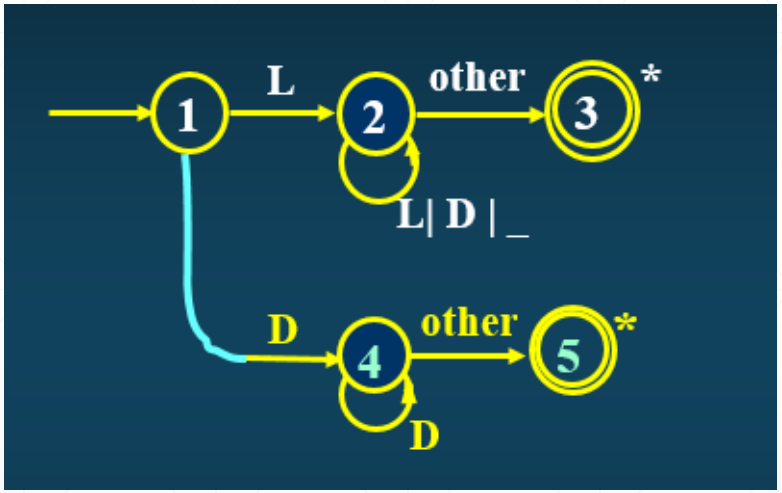
- 词法分析器设计



词法分析：设计与实现

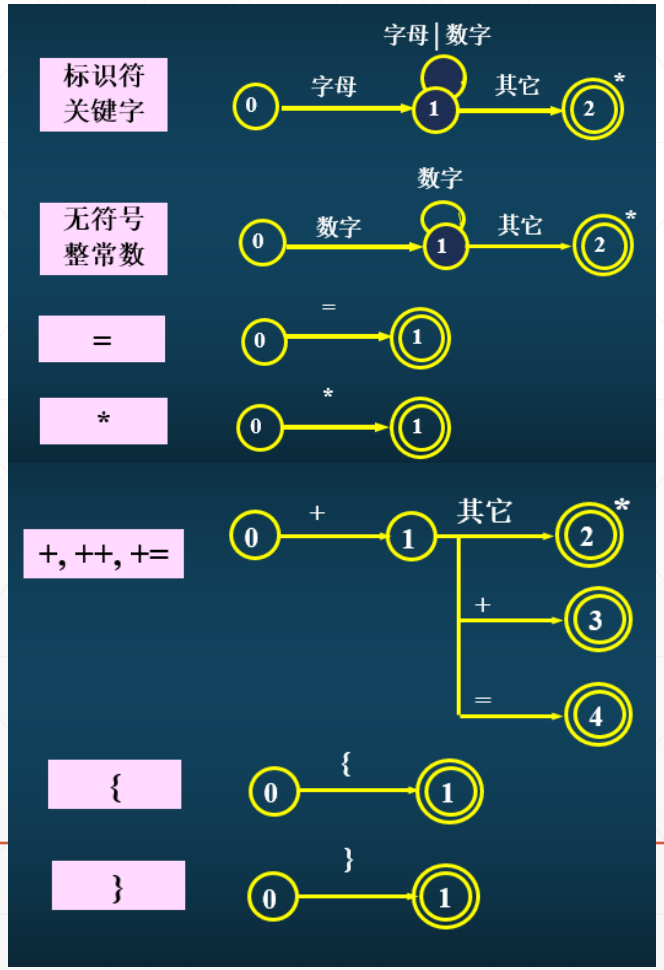
- 词法分析器设计

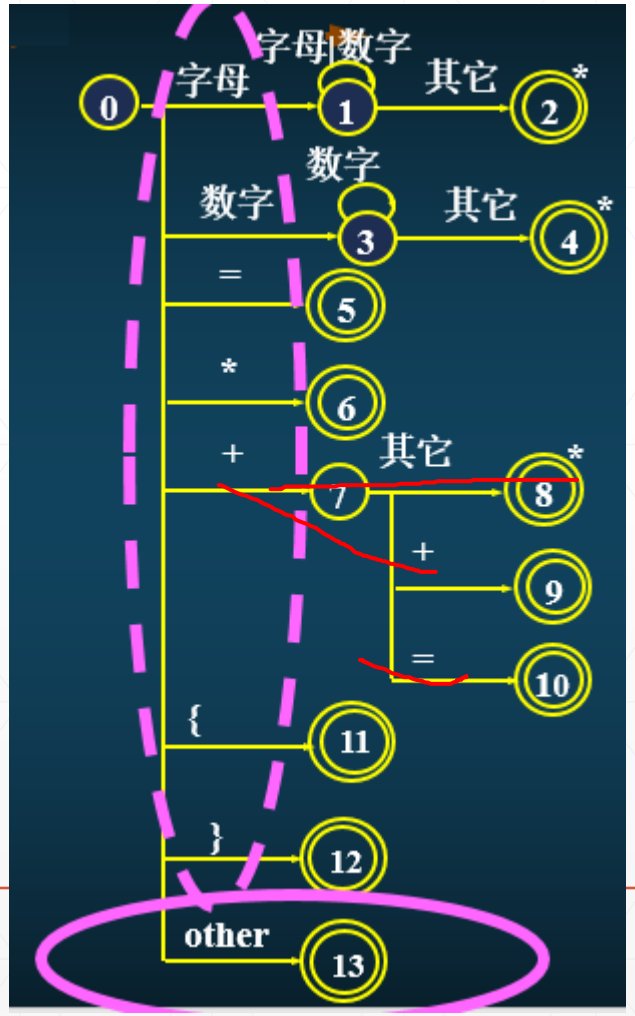




词法分析：设计与实现

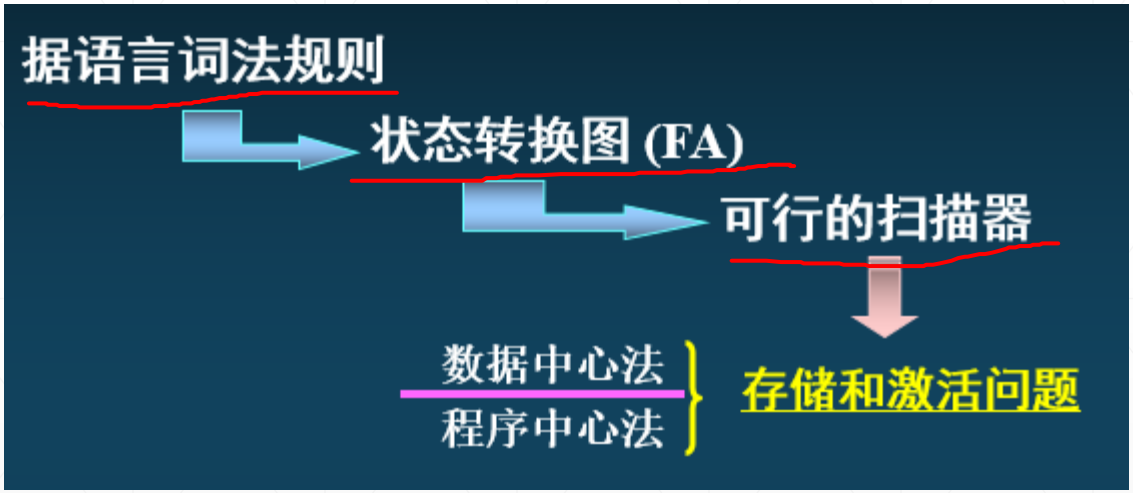
- 词法分析器设计





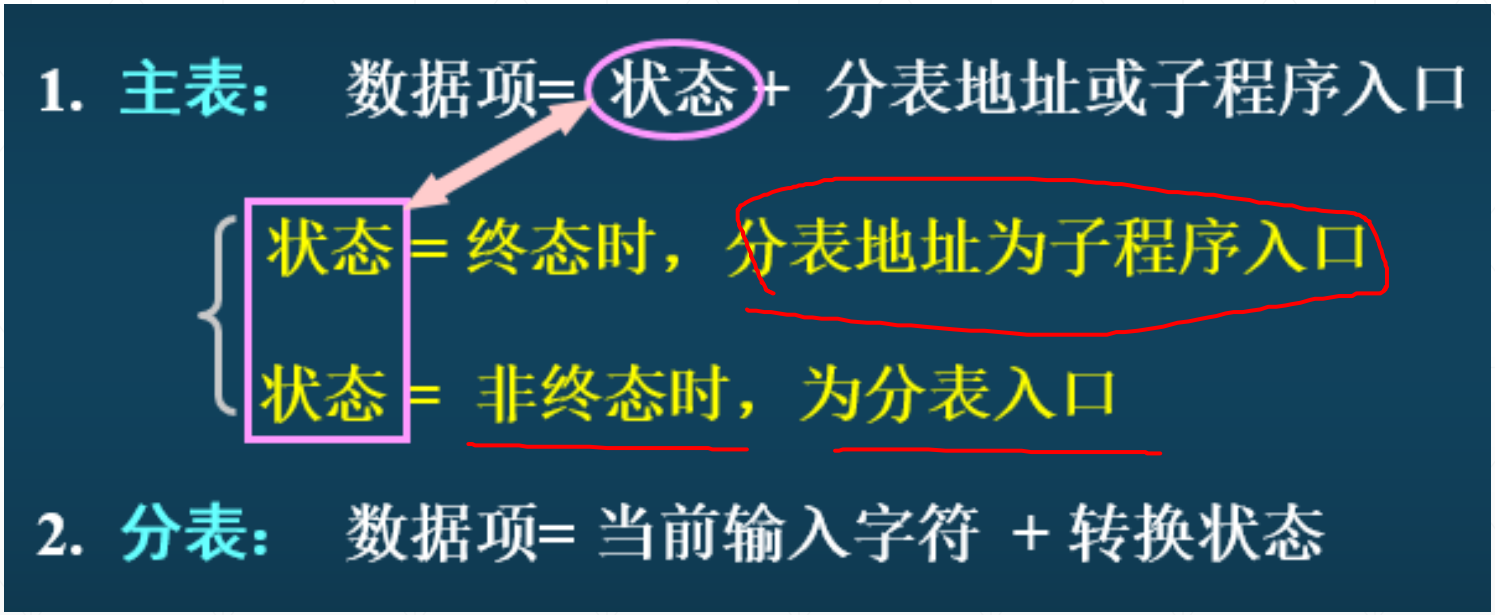
词法分析：设计与实现

- 词法分析器实现



词法分析：设计与实现

- 词法分析器实现:数据中心法



词法分析：设计与实现

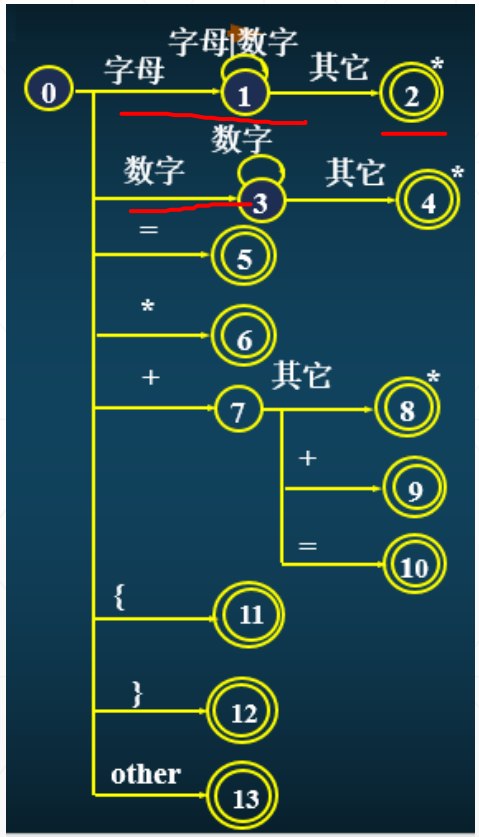
- 词法分析器实现:数据中心法

主 表

状 态	分表地址
0	(1)
1	(2)
2*	(标识符保留字)
3	(3)
4*	子(整常数)
5*	子(=)
6*	子(*)
7	(4)
8*	子(+)
9*	子(++)
10*	子(+=)
11*	子({)
12*	子(})
13*	Error

分表 (1)

输入字符	转换状态
字母	1
数字	3
=	5*
*	6*
+	7
{	11
}	12
Other	13



词法分析：设计与实现

- 词法分析器实现:数据中心法

分表 (1)

输入字符	转换状态
字母	1
数字	3
=	5*
*	6*
+	7
{	11
}	12
Other	13

分表 (2)

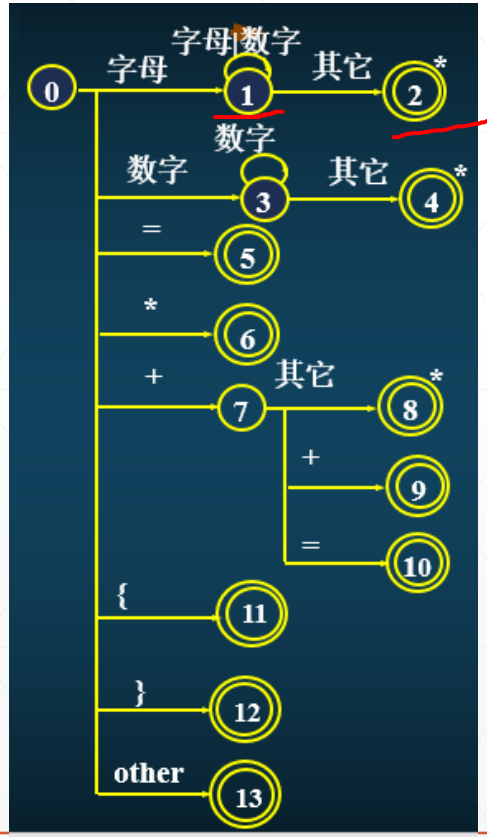
输入字符	转换状态
字母	1
数字	1
Other	2*

分表 (3)

输入字符	转换状态
数字	3
Other	4*

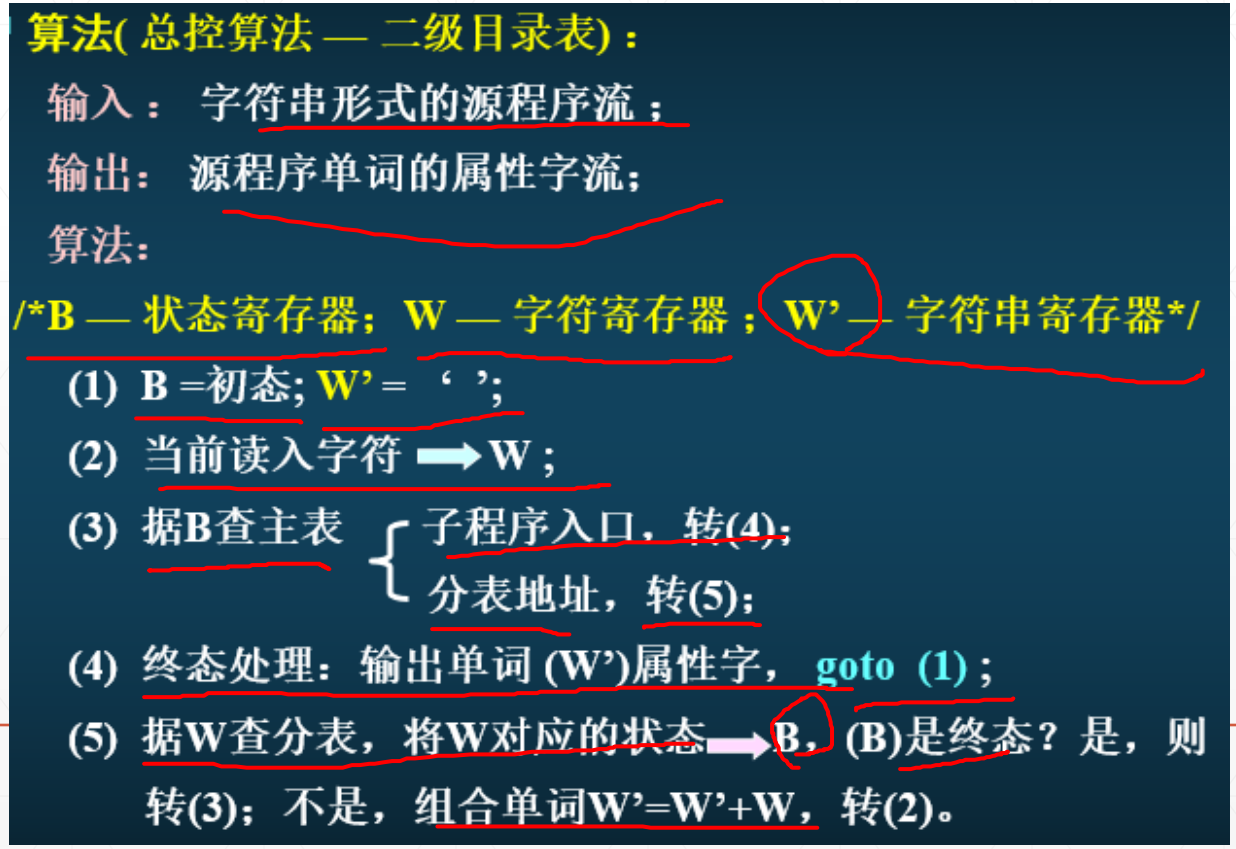
分表(4)

输入字符	转换状态
+	9*
=	10*
Other	8*



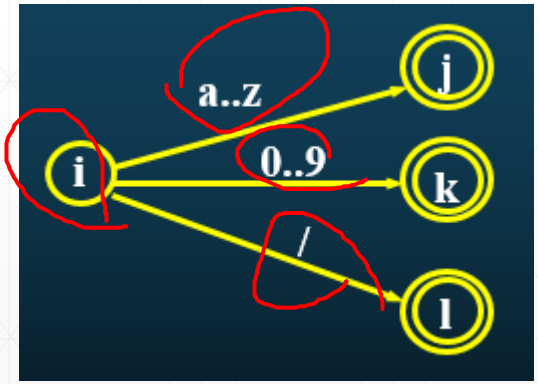
词法分析：设计与实现

- 词法分析器实现:数据中心法
  - 总控程序



词法分析：设计与实现

- 词法分析器实现:程序中心法
  - 编写一函数或直接跟踪状态图从初态开始的转换完成所有分支的跟踪来编写程序。

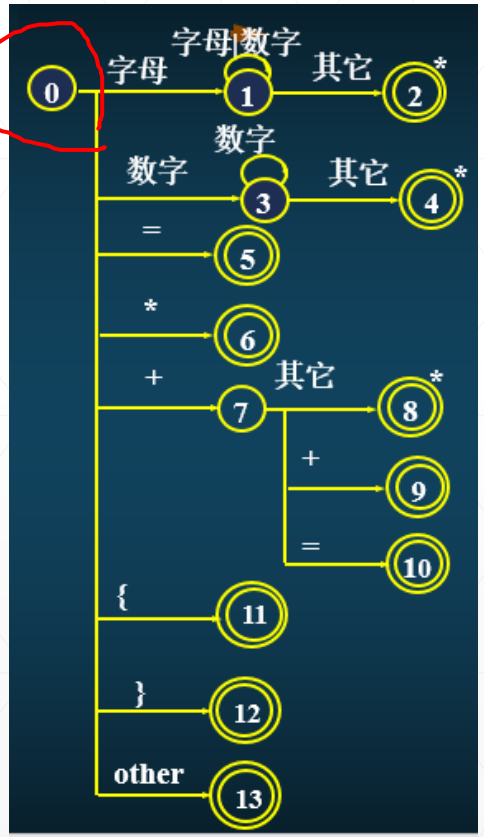



```
char char1;
  { char1=nextchar(   );
    if (state=i )
        switch (char1)
        {
          case 'a'…'z': J (chartype, char1); break;
          case '0'…'9': K (chartype, char1); break;
          case ' / '     : L (chartype, char1); break;
          default: error;
        }
  }
```

词法分析：设计与实现

- 词法分析器实现:程序中心法

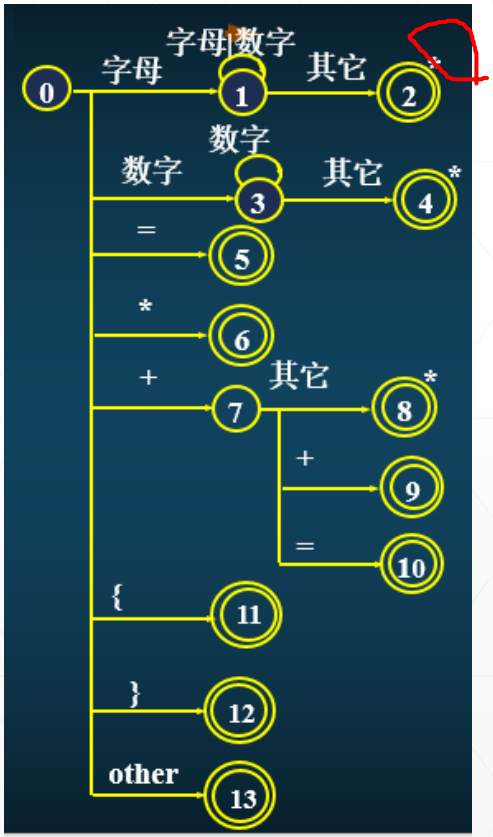
```
int state = 0;
enum lettet ('a'..'z');
enum number ('0'..'9');
char char1;
   while(1)
  {  char1 = nextchar( );
     switch(state)
     {
     case 0: switch (char1)
        {
          case  'a'…'z'   :  state = 1; break;
          case  '0'…'9'   :  state = 3; break;
          case  '='       :     state = 5; break;
          case  '*'       :     state = 6; break;
          case  '+'       :     state = 7; break;
          case  '{ '      :     state = 11; break;
          case  '} '      :     state = 12; break;
           default        :     state =13;
        } break;
```



词法分析：设计与实现

- 词法分析器实现:程序中心法

```
case 1: { while (char1 == letter||number)
              char1 = nextchar();
         state = 2; } break;
case 2: { untread(); return (02,value) or return
         (01,value);} break;
                  /* 函数untread()功能是回退一个已读进的字符；
                     属性01表示关键字；属性02表示标识符 */
 case 3:  {while (char1 == number)   char1 = nextchar();
           state = 4;}  break;
 case 4: {untread(); return (03 ,value);}break;
                  /*  属性03表示无符号整常数  */
  case 5:  return (04,  );break;
                  /*  属性04表示 “=”  */
  case 6:  return (05,  );break;
                  /*  属性05表示 “*”  */
```



词法分析：设计与实现

- 词法分析器实现:程序中心法

```
case 7: { char1 = nextchar( );
          if (char1 == '+') state = 9;
            else  if (char1==”=”)  state = 10;
                  else state=8;} break;
case 8 : {untread(); return(08, );} break;  /*  属性08表示 “+”    */
case  9:  return (09,  ); break;            /*  属性09表示 “++”  */
case 10:  return (12,  ); break;            /*  属性12表示 “+=”  */
case 11:  return (10,  ); break;            /*  属性10表示 “{”    */
case 12:  return (11,  ); break;            /*  属性11表示 “}”    */
case 13 : error;                        /*  error 是语法错处理函数   */
}
}
```