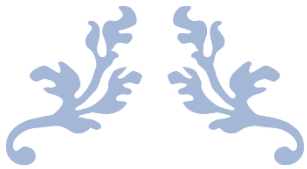


编译原理实验报告

实验一：词法分析

2022-12-1

计算机科学与工程学院

09020312 陈鑫

目录

一、实验目的

程序的编译是从输入源程序到输出目标代码，其中，词法分析是编译的第一个阶段，是 对编译程序至关重要。词法分析就是扫描源程序字符串，按词法规则识别正确的单词，并转 换成统一规格交语法分析使用。通过自己编程实现词法分析，我们能更深入地理解词法分析 的过程，并提高自己的编程能力。

二、实验内容

（1）自选语言或某语言的子集，对该语言的代码进行词法分析
（2）词法实验部分要求写出该语言的词法分析程序。要求识别出词法分析单元，给出词法单元的分类，按照不同类别填入相应的表中，表中给出词法单元在源程序中的位置。
具体步骤（手工部分）：
（1）词法的产生式（或 regular expression）转换成 NFA
（2）NFA 确定化为 DFA
（3）DFA 最小化
（4）根据最小化后的 DFA 写出词法分析程序。
（5）要求：输入一系列正规表达式，输出它们的 token 序列，可以包括错误处理。

三、实验原理

词法分析是从左向右一个字符一个字符地读入源程序，扫描每行源程序的符号，依据词法规则，识别单词。执行词法分析的程序称为词法分析器，将给定的程序通过词法分析器，识别出一个个单词符号。单词符号常采用统一的二元式表示：（单词种别码，单词符号自身的值），单词种别码是语法分析所需要的信息，而单词符号自身属性值是编译其他阶段需要的信息。

四、实验设计

本次实验分析的语言是 c 语言的一个子集,详细如下：

（1）关键字：

"begin","char","int","float","double","const","if","then","else","while","do",

"for","break","continue","sizeof",”void”,”return”,"end"

（2）运算符：

=、==、>、<、>=、<=、+、-、*、/

（3）界限符：

( ) [ ]{ } , : ; 等

（4）标识符（ID）：

用字母、数字、下划线的连接用来表示变量名、过程名等的单词称为标识符（不能以数字开头、不与关键字重名、不包含$、#等无法识别的字符）

（5）常量（NUM）：

整数、小数、浮点数。

（6）词法分析阶段通常忽略空格、制表符和换行符等。

根据以上的分类与分析，设置该语言中的单词符号及种别编码如下表。

表 1 单词符号及种别编码

单词符号	种别编码	单词符号	种别编码
begin	1	=	31
char	2	==	32
int	3	>	33
float	4	<	34
double	5	>=	35
const	6	<=	36
if	7	+	37
then	8	-	38
else	9	*	39

while	10	/	40
do	11	(	41
for	12	)	42
break	13	[	43
continue	14	]	44
sizeof	15	{	45
void	16	}	46
return	17	,	47
end	18	:	48
ID 标识符	20	;	49
NUM 常量	30		

五、状态图分析

5.1 关键字

本次设计中关键字有 18 个，分别包括：
"begin","char","int","float","double","const","if","then","else","while","do",
"for","break","continue","sizeof",”void”,”return”,"end",。

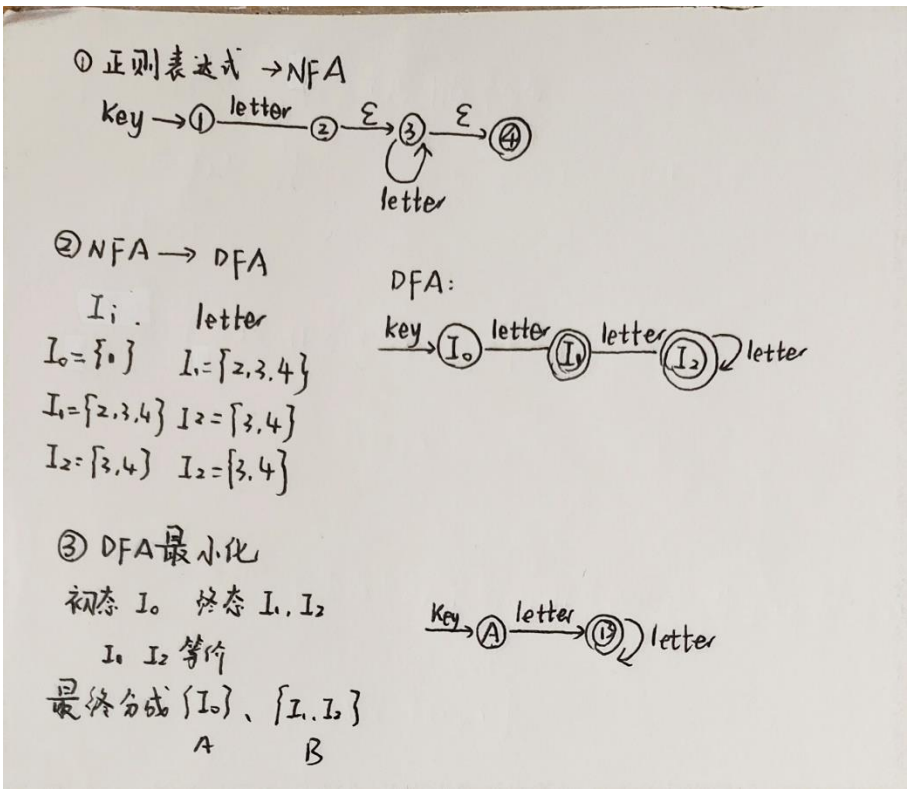
关键字都是由小写字母组成，在程序中，将 18 个关键字保存在一个 string 类型的 vector 中，然后做一次循环，将字符串逐个与 18 个关键字对比，相同则取出对应的种别编码，存入事先设计好的 vector 中。

关键字的正规表达式为：

$$keyID -> (letter)^{+}$$

$$letter -> [a-z]$$

按照正则表达式—>NFA—>DFA—>最小化 DFA 的顺序，过程如图所示：



5.2 运算符

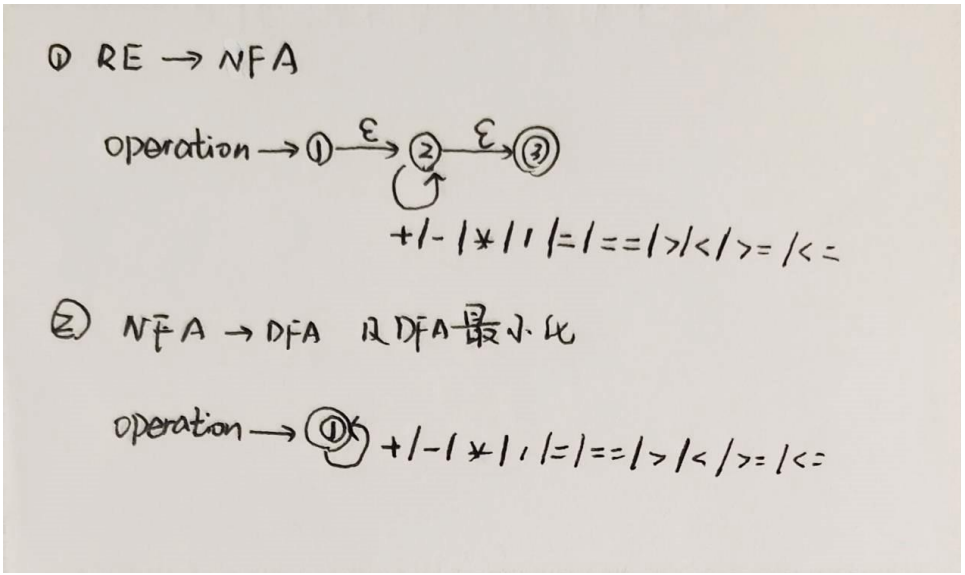
本实验设计了十个运算符，分别为：

=、==、>、<、>=、<=、+、-、*、/

运算符表达式为：

$$operation \rightarrow (= \mid == \mid > \mid < \mid \geq \mid \leq \mid + \mid - \mid * \mid /)^*$$

按照正则表达式—>NFA—>DFA—>最小化 DFA 的顺序，过程如图所示：



5.3 界限符

本实验设计的界限符有：

( ) [ ] { } , : ;

界限符表达式为：

$$delimiter -> ((|)|[|]|\{|\}|,|:|;)^*$$

从正则表达式—>NFA—>DFA—>最小化 DFA 的图示与运算符图示流程相同。

5.4 标识符

用字母、数字、下划线的连接用来表示变量名、过程名等的单词称为标识符（不能以数字开头、不与关键字重名、不包含$、#等无法识别的字符）。

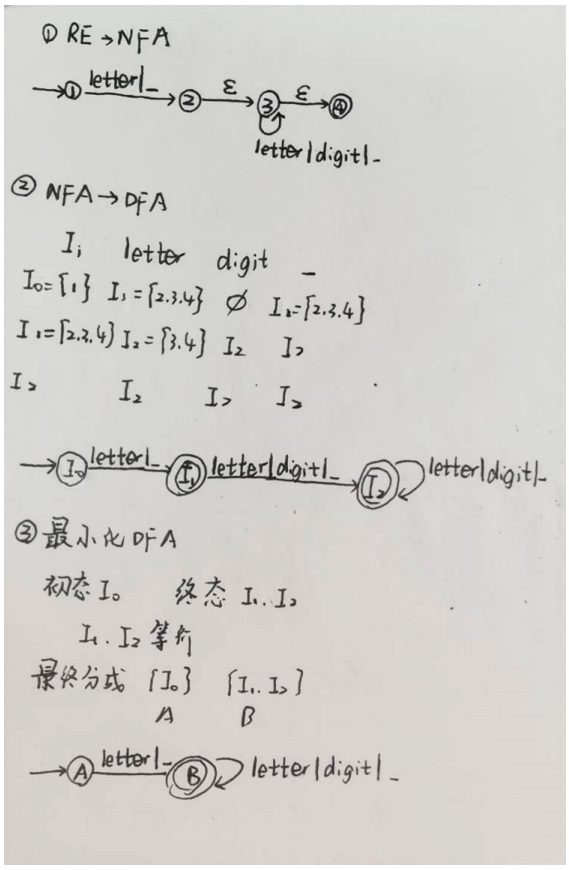
标识符的正规表达式为：

$$ID -> (letter | \_)(letter | digit | \_)^*$$

$$letter -> [a-zA-Z]$$

$$digit -> [0-9]$$

按照正则表达式—>NFA—>DFA—>最小化 DFA 的顺序，过程如图所示：



5.5 常量

将小数，整数，浮点数三类归并后分为无符号数和有符号数。

因为有符号数，除了开始有符号外，之后的判断与无符号数是一致的。所以在这里只针对无符号数的正规表达式构造 NFA 和 DFA。

无符号数正规表达式：

$$NUM -> \ digits\ op\_fra\ op\_exp$$

其中：

$$digits -> \ d(d)^*$$

$$d -> \ [0-9]$$

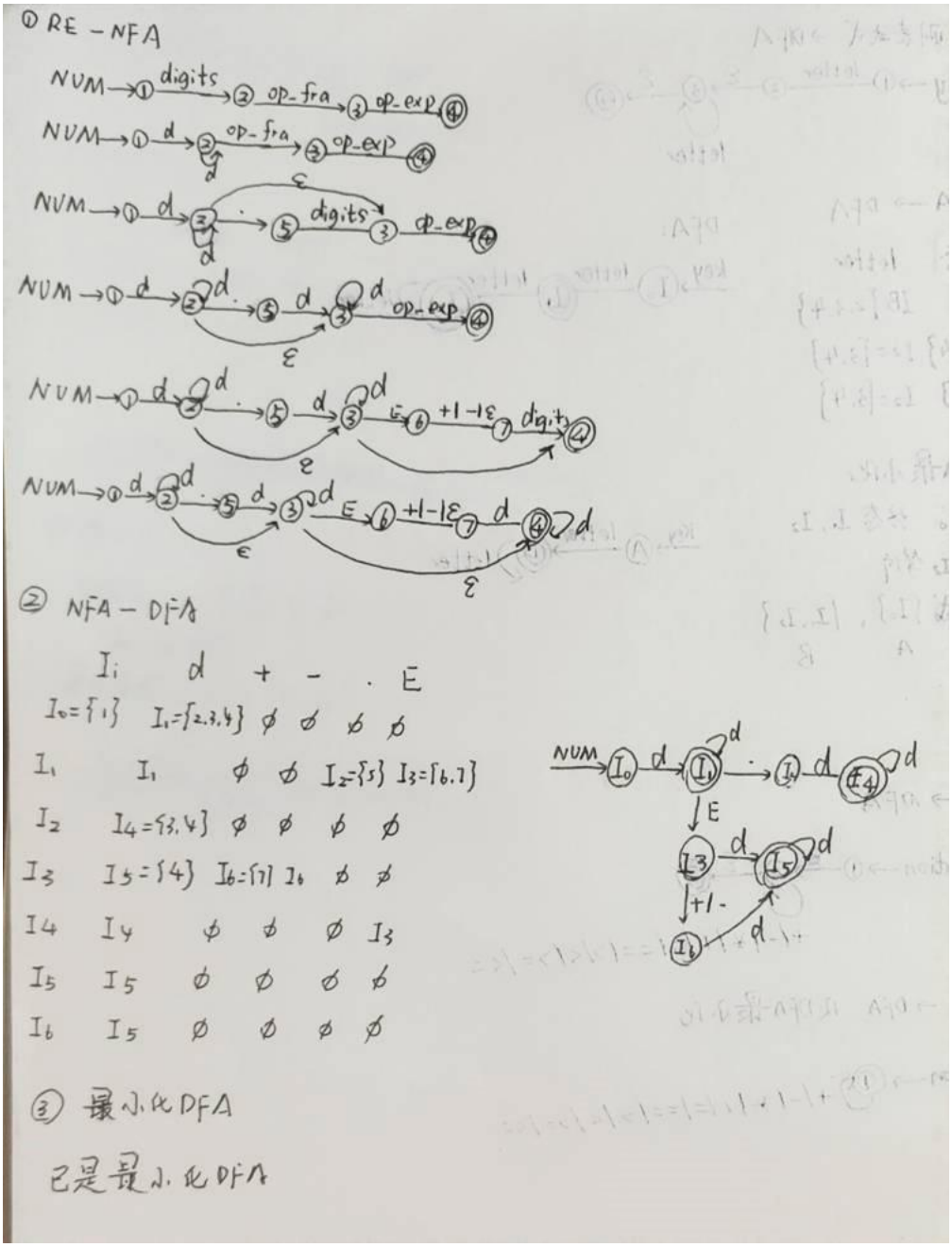
$$op\_exp -> \ (\ E\ (+|-|\varepsilon)\ digits) \mid \varepsilon$$

将上述表达式整合得无符号数 NUM 的正规表达式为：

$$NUM -> d(d)*|(d(d)*|\varepsilon)((.d(d)*)(E(+|-|\varepsilon)d(d)*|\varepsilon)|(E(+|-|\varepsilon)d(d)*))$$

$$d -> \ [0-9]$$

按照正则表达式—>NFA—>DFA—>最小化 DFA 的顺序，过程如图所示：



对于有符号数，只需要在 1 状态之前多了一个加号和减号的判断，其余不变。

六、实验结果

这里对文件中的一段程序进行词法分析，源程序及其词法分析结果如下：
输入：

```
begin
int main()
{
int a[12];
int b=-15;
double c=20.22
int c_x=3.1415926
char array[10];
int _x=a;
if(_x==a)
{
while(a)
a=a*3;
}
###
return 0;
}
end
```

输出 token 序列：

```
词法分析测试结果如下:
< 单词种别码, 单词本身 >          //所属类别
<   1     ,    begin     >          //关键字
<   3     ,    int       >          //关键字
<   20    ,    main      >          //标识符
<   41    ,    (         >          //界限符
<   42    ,    )         >          //界限符
<   45    ,    {         >          //界限符
<   3     ,    int       >          //关键字
<   20    ,    a         >          //标识符
<   43    ,    [         >          //界限符
<   30    ,    12        >          //常量
<   44    ,    ]         >          //界限符
<   49    ,    ;         >          //界限符
<   3     ,    int       >          //关键字
<   20    ,    b         >          //标识符
<   31    ,    =         >          //运算符
<   30    ,    -15       >          //常量
<   49    ,    ;         >          //界限符
<   5     ,    double    >          //关键字
<   20    ,    c         >          //标识符
<   31    ,    =         >          //运算符
<   30    ,    20.22     >          //常量
<   3     ,    int       >          //关键字
<   20    ,    c_x       >          //标识符
<   31    ,    =         >          //运算符
<   30    ,    3.1415926 >          //常量
<   2     ,    char      >          //关键字
<   20    ,    array     >          //标识符
<   43    ,    [         >          //界限符
<   30    ,    10        >          //常量
<   44    ,    ]         >          //界限符
<   49    ,    ;         >          //界限符
<   3     ,    int       >          //关键字
<   20    ,    _x        >          //标识符
<   31    ,    =         >          //运算符
<   20    ,    a         >          //标识符
<   49    ,    ;         >          //界限符
<   7     ,    if        >          //关键字
<   41    ,    (         >          //界限符
<   20    ,    _x        >          //标识符
<   32    ,    ==        >          //运算符
<   20    ,    a         >          //标识符
<   42    ,    )         >          //界限符
<   45    ,    {         >          //界限符
<   10    ,    while     >          //关键字
<   41    ,    (         >          //界限符
<   20    ,    a         >          //标识符
<   42    ,    )         >          //界限符
<   20    ,    a         >          //标识符
<   31    ,    =         >          //运算符
<   20    ,    a         >          //标识符
<   39    ,    *         >          //运算符
<   30    ,    3         >          //常量
<   49    ,    ;         >          //界限符
<   46    ,    }         >          //界限符
<   -1    ,    ###       >          //无法识别的符号
<   17    ,    return    >          //关键字
<   30    ,    0         >          //常量
<   49    ,    ;         >          //界限符
<   46    ,    }         >          //界限符
词法分析成功! 共1个无法识别的符号。
```

发现输出完全符合预期，即词法分析成功。

七、实验心得

通过本次实验，我实现了一个简单的 c 语言子集的词法分析器的编程，对程序编译中词法

分析这一阶段有了更深的理解，对词法分析的相关知识点也进行了很好的复习，同时也提升了自身的编程能力