实验一：词法分析器编程　Lab1: Lexical Analyzer Programming

09019204
曹邹颖

一．实验目的 Motivation/Aim

本次词法分析器的编程实验目的在于：

1) 了解词法分析程序的任务是从左至右扫描源程序的字符串，按照此法规则（正规文法规则）识别出一个个正确的单词，并转换该单词成相应的二元式（类号、内码）交语法分析使用；
2) 掌握词法分析程序设计的原理和构造方法;
3) 对编译的基本概念、原理和方法有完整的和清楚的理解,并能正确地、熟练地运用。

二．实验内容 Content description

实验一要求如下：

a. 输入：字符流，REs（REs 的数量自行确定）
b. 输出：token 序列
c. 词类号自行定义
d. 可能包括错误处理

确定实验内容如下：

用 C++语言实现对 C++语言子集的源程序进行词法分析。通过输入源程序从左到右对字符串进行扫描和分解，依次输出各个单词的内部编码及单词符号自身值；若遇到错误则显示“Error”，然后跳过错误部分继续显示；同时进行标识符登记符号表的管理。

三．实验方法 Ideas/Methods

本次实验采用龙书的 3.4.4 小节——基于有限自动机的编程方法：

1) 首先自行定义一些正则表达式；
2) 将正则表达式（REs）转换为不确定有限自动机（NFAs），即对待分析的简单的词法（关键词/标识符/整常数/运算符/界限符等）先分别建立自己的 FA；
3) 再将这些 NFA 合并为单个 NFA，即将各个 FA 用 ε 产生式连接起来并设置一个唯一的开始符，终结符不合并；
4) 然后将该 NFA 转换为具有最小状态的 DFAo，即 NFA 的确定化加上 DFA 的简化；
5) 最后基于 DFAo 程序设计。

四．实验假设 Assumptions

待分析的简单词法：

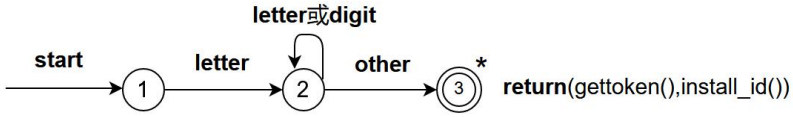
1) 关键词：

"auto"，"int"，"float"，"double"，"char"，"if"，"else"，"switch"，"case"，"break"，"continue"，"while"，"do"，"bool"...

2) 标识符：ID=letter(letter|digit)*

26 位小写字母，"fun"，"number"，"value"，"set"，"ptr"，"str"，"count"，"next"，"pre"，"words"，"keyword"，"num"，"fp"，"name"......

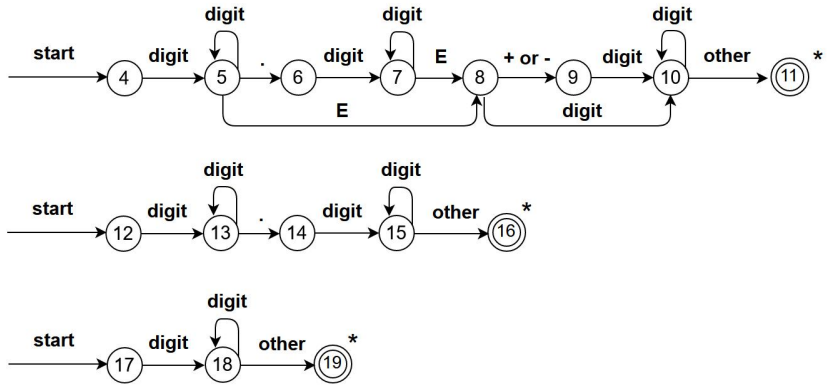
Transition Diagram for Identifier and Keywords：



3) 整常数：整数、有符号整数、无符号整数

0-9,.....

Transition Diagram for Unsigned Numbers

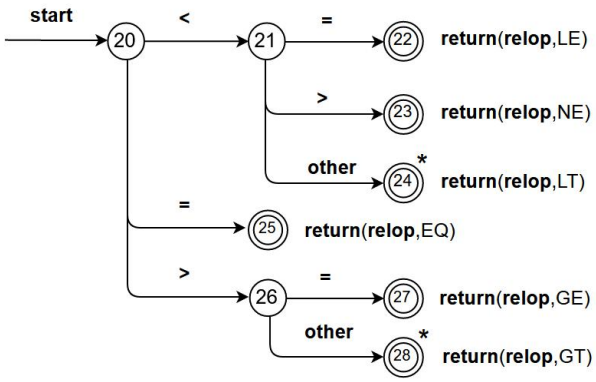


而 C++语言中不支持科学计数法的常数输入；

4) 运算符：

"+"，"-"，"*"，"/"，"%"，"++"，"--"，"+="，"-="，"*="，"/="，"%="，"&&"，"||"，"<="，"=="，">="，"<<"，"="，">>"，"!"，"&"，"|"，"^"，"!="，"&="，"|="，"^="......

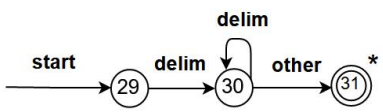
Transition Diagram for Relational Operators



5) 分隔符："("，"["，")"，"]"，","，"{"，";"，"}".....

其中空格、制表符和换行符这三者在词法分析阶段通常被忽略。

Transition Diagram for Separators



由此，相关单词类号与内码建表如下：

类型 type	类号 classNumber	内码 internalCode
关键词 keywords	1	keywords 在关键词表中的位置
标识符 id	2	id 在符号表中的位置
整常数 constantNumber	3	num 在常数表中的位置
运算符 op	4	op 运算符表中的位置
分割符 delim	5	分隔符在分隔符表中的位置

关键词 keywords：

单词符号 word	内码	单词符号 word	内码
asm	1	namespace	32
auto	2	new	33
bool	3	operator	34
break	4	private	35
case	5	protected	36
catch	6	public	37
char	7	register	38
class	8	reinterpret_cast	39
const	9	return	40
const_cast	10	short	41
continue	11	signed	42
default	12	sizeof	43
delete	13	static	44
do	14	static_cast	45
double	15	struct	46
dynamic	16	switch	47
else	17	template	48
enum	18	this	49
explicit	19	throw	50
export	20	true	51
extern	21	try	52
false	22	typedef	53
float	23	typeid	54
for	24	typename	55
friend	25	union	56
goto	26	unsigned	57
if	27	using	58
inline	28	virtual	59
int	29	void	60
long	30	volatile	61

mutable	31	while	62

标识符 id：

单词符号 word	内码	单词符号 word	内码
a	1	main	27
b	2	fun	28
c	3	number	29
d	4	value	30
e	5	state	31
f	6	width	32
g	7	height	33
h	8	max	34
i	9	min	35
g	10	ptr	36
k	11	str	37
l	12	token	38
m	13	words	39
n	14	constant	40
o	15	pre	41
p	16	next	42
q	17	count	43
r	18	example	44
s	19	comment	45
t	20	keyword	46
u	21	set	47
v	22	fp	48
w	23	name	49
x	24	type	50
y	25	num	51
z	26	letter	52

整常数 constantNumber：

单词符号 word	内码	单词符号 word	内码
0	0	5	5
1	1	6	6
2	2	7	7
3	3	8	8
4	4	9	9

运算符 op：

单词符号 word(Op)	内码	单词符号 word(relop)	内码

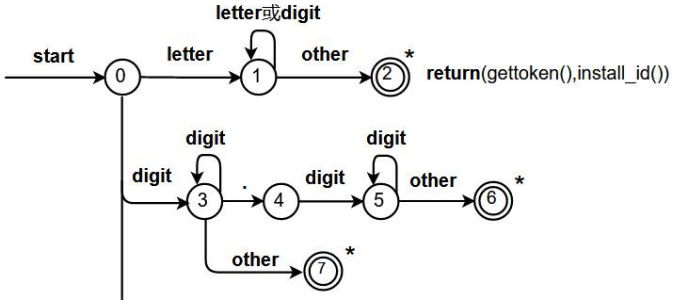
+	1	!	22
++	2	!=	23
+=	3	&&	24
-	4	||	25
--	5	~	26
-=	6	<	27
*	7	<=	28
*=	8	<>	29
/	9	>	30
/=	10	>=	31
%	11	==	32
%=	12		
|	13		
|=	14		
&	15		
&=	16		
^	17		
^=	18		
<<	19		
>>	20		
=	21		

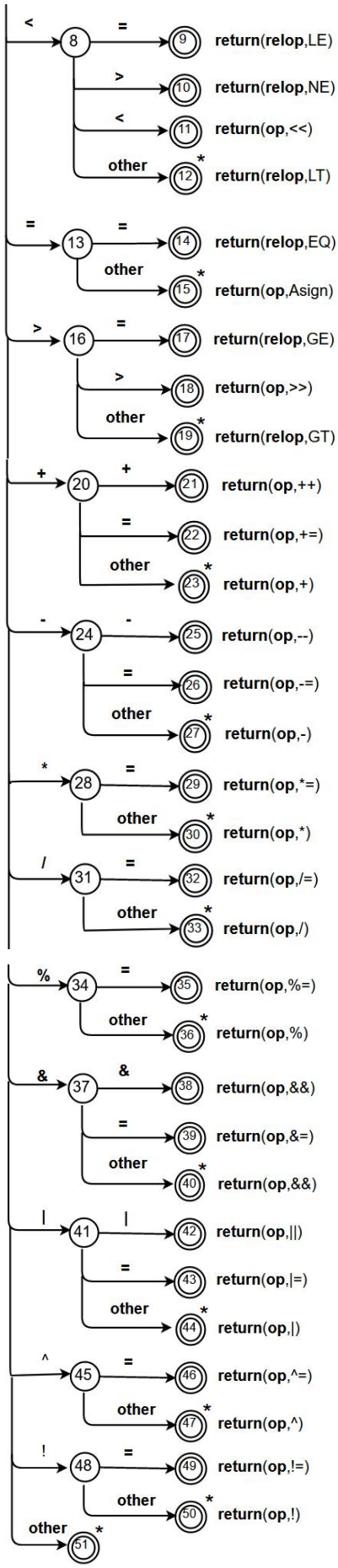
分割符 delim：

单词符号 word	内码	单词符号 word	内码
(	1	[	5
)	2	]	6
,	3	{	7
;	4	}	8

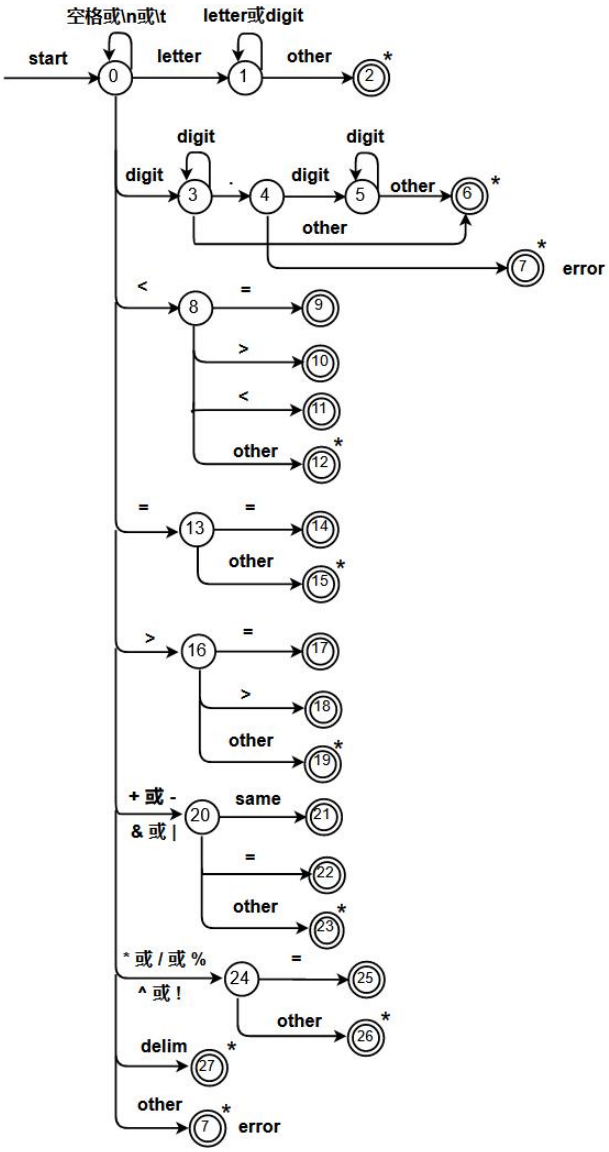
五．相关自动机描述 Related FA descriptions

将各个 NFA 合并为单个 NFA：




< 8 = 9 return(relop,LE)
> 10 return(relop,NE)
< 11 return(op,<<)
other * 12 return(relop,LT)
= 13 = 14 return(relop,EQ)
other * 15 return(op,Asign)
> 16 = 17 return(relop,GE)
> 18 return(op,>>)
other * 19 return(relop,GT)
+ 20 + 21 return(op,++)
= 22 return(op,+=)
other * 23 return(op,+)
- 24 - 25 return(op,--)
= 26 return(op,-=)
other * 27 return(op,-)
* 28 = 29 return(op,*=)
other * 30 return(op,*)
/ 31 = 32 return(op,/=)
other * 33 return(op,/)
% 34 = 35 return(op,%=)
other * 36 return(op,%)
& 37 & 38 return(op,&&)
= 39 return(op,&=)
other * 40 return(op,&&)
| 41 | 42 return(op,||)
= 43 return(op,|=)
other * 44 return(op,|)
^ 45 = 46 return(op,^=)
other * 47 return(op,^)
! 48 = 49 return(op,!=)
other * 50 return(op,!)
other * 51

将该 NFA 转换为具有最小状态的 DFAo：



六．核心数据结构描述 Description of important Data Structures

1) 生成的 token 序列由 word、type、classNumber、internalCode 组成，利用 struct 构建，由于大小未知，采用 vector 保存。

```
struct token {
    string word;    //单词
    string type;    //类型
    int classNumber; //类号
    int internalCode; //内码
};
vector<token>Token;
```

2) 单词的类号、内码表的建立，全部用 map 保存，这样便于查找，提高效率。

```
map<string, int>Keywords;
map<string, int>Identifier;
```

```
map<string, int>constantNumber;
map<string, int>Op;
map<string, int>Delim;
```

七．核心算法描述 Description of core Algorithms

1) void lexical_analyzer()

词法分析器，从测试文件中按字符读入，利用 switch...case...结构将 DFAo 状态转移刻画出来，从状态 0 开始处理，如果是回车、换行、制表符、空格则维持状态 0，如果第一个字符为字母，则转为状态 1；若第一个字符为数字，则转为状态 2，详见状态转换图以及打包代码（太长就不粘贴进报告了）。而每次分析出来一个 token 都要利用 addToken()函数将其添加到 Token 表中并回到状态 0，继续往下处理。

2) void addToken(string word, int classNumber)

将分析出来的 token（string word）添加到 Token 序列表中，如果类型（int classNumber）是 1，查看关键词表，若未找到再查标识符表，类型是 2 查标识符表，找不到但满足标识符特征的也可添加进表中附以内码，类型 3 同理类型 2，类型 4 查运算符表，类型 5 查分隔符表，其他类型就记作词法错误，将错误信息记录下来。

```
void addToken(string word, int classNumber)
{
    map<string, int>::iterator iter;
    token t;
    switch (classNumber) {
    case 1:    //查看关键词表
        iter = Keywords.find(word);
        if (iter != Keywords.end()) {
            t = { word,"keywords",1,iter->second };
            Token.push_back(t);
        }
        else {      //查找是否为 id
            iter = Identifier.find(word);
            if (iter == Identifier.end()) {   //未在 id 表中找到
                Identifier[word] = idNum;     //添加到 id 表中
                t = { word,"identifier",2,idNum++ };
                Token.push_back(t);
            }
            else {
                t = { word,"identifier",2,iter->second };
                Token.push_back(t);
            }
        }
        break;
    case 2:
        iter = Identifier.find(word);
```

```
            if (iter != Identifier.end()) {
                t = { word,"identifier",2,iter->second };
                Token.push_back(t);
            }
            else {
                Identifier[word] = idNum;     //添加到 id 表中
                t = { word,"identifier",2,idNum++ };
                Token.push_back(t);
            }
            break;
        case 3:
            iter = constantNumber.find(word);
            if (iter != constantNumber.end()) {
                t = { word,"constant_number",3,iter->second };
                Token.push_back(t);
            }
            else {
                constantNumber[word] = nNum;
                t = { word,"constant_number",3,nNum++ };
                Token.push_back(t);
            }
            break;
        case 4:
            iter = Op.find(word);
            if (iter != Op.end()) {
                t = { word,"operator",4,iter->second };
                Token.push_back(t);
            }
            break;
        case 5:
            iter = Delim.find(word);
            if (iter != Delim.end()) {
                t = { word,"delimeter",5,iter->second };
                Token.push_back(t);
            }
            break;
        default:   //error
            t = { word,"lexical error",-1,-1 };
            Token.push_back(t);
            break;
        }
    }
```

八．测试用例 Use cases on running

待测字符串放在 TestCases.txt 文件中，

写了一个简单的 C++语言用例如下：

TestCases.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

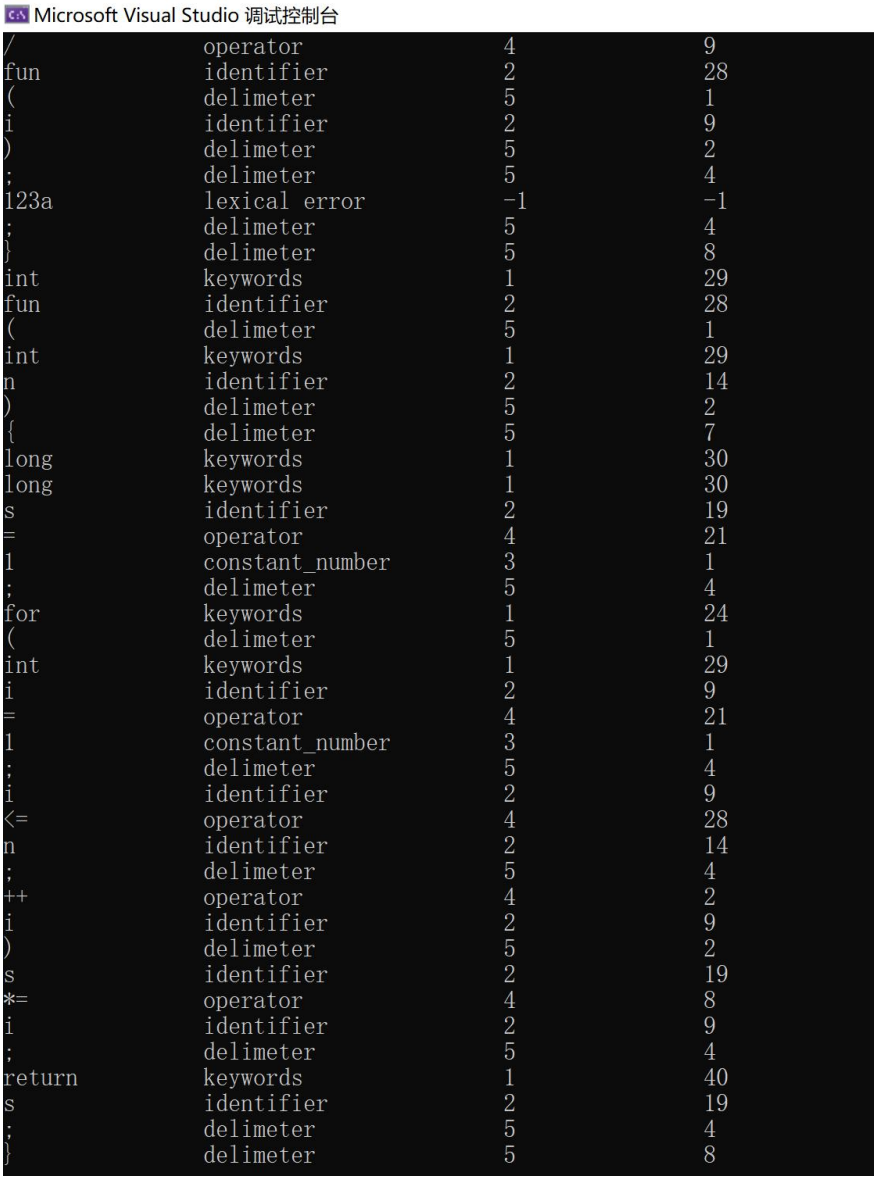
```
void main()
{
	int a=3,m=5;
	double e=1;
	for(int i=1;i<=m-1;++i)   e+=double(1)/fun(i);
	123a;
}
int fun (int n)
{
	long long s=1;
	for (int i=1;i<=n;++i)
		s*=i;
	return s;
}
```

输出打印结果如下：

Microsoft Visual Studio 调试控制台

```
token sequence:

RE              type            classNumber             internalCode
void            keywords                1               60
main            identifier              2               27
(               delimeter               5               1
)               delimeter               5               2
{               delimeter               5               7
int             keywords                1               29
a               identifier              2               1
=               operator                4               21
3               constant_number         3               3
,               delimeter               5               3
m               identifier              2               13
=               operator                4               21
5               constant_number         3               5
;               delimeter               5               4
double          keywords                1               15
e               identifier              2               5
=               operator                4               21
1               constant_number         3               1
;               delimeter               5               4
for             keywords                1               24
(               delimeter               5               1
int             keywords                1               29
i               identifier              2               9
=               operator                4               21
1               constant_number         3               1
;               delimeter               5               4
i               identifier              2               9
<=              operator                4               28
m               identifier              2               13
-               operator                4               4
1               constant_number         3               1
;               delimeter               5               4
++              operator                4               2
i               identifier              2               9
)               delimeter               5               2
e               identifier              2               5
+=              operator                4               3
double          keywords                1               15
(               delimeter               5               1
1               constant_number         3               1
)               delimeter               5               2
```



可见，例如 123a;这样的词法错误也检测出来了。

九．出现的问题与解决方案 Problems occurred and related solutions

出现的问题有：

1. 对待分析的简单的词法（关键词/标识符/整常数/运算符/界限符等）先分别建立自己的 FA 时，判断终态上是否要带星号*表示将输入回退一个位置的设计；
2. 将多个 FA 合并并化简为一个具有最小状态的 DFA，不明确哪些状态可以合并；
3. 代码中出现死循环没有输出；

解决方案为：

1. 通过实际生活中的 C++语言去设想如果某类词法输入 other 后是否需要回退，回退后判断的单词二元式是否合理等；
2. 有些状态在化简时不确定是否可以合并，可以保留进行代码编写，在代码层面可能会发现某两个状态(case state)后的代码是（相同或）等价的，从而会发现这两个状态是可以合并，并进一步化简 DFA 的；

3. 原因是测试文件放错位置，导致读取错误，一直输出换行符，更换到正确目录下即可。

十．心得体会 Your feelings and comments

通过动手实践，使我对构造词法分析器的基本原理有更为深入的理解和掌握，将词法分析过程的理论运用到实际，更加熟练地构造与化简 FA。其中，实验中有一处设计算是比较满意的，对于 C+语言的关键词可以建表存储，但是能识别的常量与识别符是无限的，光靠建表存储是不可能囊括的，我们会通过其特征识别并将其分配特有的内码，存储在相应表中，而不是一味的在表中查询内码。不足的是，算法采用的是基于 FA 的词法分析方法，存在大两 switch...case...语句，可能并不高效，对于 Lex 并没有尝试。不过总的来说，这次实验还是达到了实验初衷，实现了一定的词法分析功能，得到了锻炼。