编译原理课程实验报告

实验 1：词法分析

姓名	杨军瑗		院系	计算机		学号	1160300427	
任课教师	陈鄞			指导教师	陈鄞			
实验地点	格物 214			实验时间	4 月 14 日 3、4 节			
实验课表现	出勤、表现得分			实验报告得分		实验总分		
	操作结果得分							

一、需求分析

得分

要求：阐述词法分析系统所要完成的功能

词法分析时整个程序编译分析的第一部分，需要完成识别以下几类单词：
1、标识符：由大小写字母、数字以及下划线组成，但必须以字母或者下划线开头；
2、关键字：
（1）类型关键字：整型、浮点型、布尔型、记录型；
（2）分支结构中的 if 和 else；
（3）循环结构中的 do 和 while；
（4）过程声明和调用中的关键字
3、运算符：
（1）算术运算符；+、-、*、/、++、--
（2）关系运算符；<、>、<=、>=、==、!=
（3）逻辑运算：!、&&、||
4、界符
（1）用于赋值语句的界符，如“=”；
（2）用于句子结尾的界符，如“;”、“,”；
（3）用于数组表示的界符，如“[”和“]”；
（4）用于浮点数表示的界符“.”
（5）其他界符：″(″, ″)″, ″{″, ″}″,” ” ”,” ‘ ”
5、常数：无符号整数和浮点数，包括科学计数法，字符串常数（“xxxxx”）等）
6、注释（/*……*/形式）

除此之外，可以实现一些额外功能，如：
1、识别字符常数 ‘x’
2、八进制和十六进制数。

需要识别出以上这些单词，并且 给出相应的 Token 序列，识别过程中的 DFA 转化关系，如果识别出错需要给出报错信息。

二、文法设计	得分	

要求：对如下内容展开描述

1、给出各类单词的词法规则描述（正则文法或正则表达式）

（1）识别标识符

digit → 0|1|2|…|9

letter_ → A|B|…|Z|a|b|…|z|_

id → letter_(letter_|digit)*

（2）识别关键字

程序中预先存储了一个常用关键字集合：

```
public static String keywords[] = { "auto", "double", "int", "struct",
"break", "else", "long", "switch", "case", "enum", "register",  "typedef",
"char", "extern", "return", "union", "const", "float",  "short", "unsigned",
"continue","for","signed","void",  "default","goto","sizeof","volatile",
"do", "if", "while",
```

当判断得到当前单词是标识符时，比对当前单词是否是关键字，如果是，则判断为关键字，否则为一般标识符。

（3）识别十进制无符号整型，并在此基础上识别浮点数和科学计数法

digit → 0|1|2|…|9

digits → digit digit*

optionalFraction → .digits| ε

optionalExponent → (E(+|-| ε)digits)| ε

number → digits optionalFraction optionalExponent

（4）识别八进制数、十六进制数

八进制：OCT -> 0(1|2|3|4|5|6|7)（0|1|2|3|4|5|6|7）*

十六进制：HEX-> 0x(1|„|9|a|„|f) (0|„|9|a|„|f)*

（5）识别注释

NOTE -> /*other*/　　Other 指代任意字符，可有多个

（6）识别字符串常量

STRING -> “other”　　Other 指代任意字符，可有多个

（7）识别字符常量

CHAR -> ‘one_other’　one_other 指代任意字符,只可有一个

（8）识别数组

ARRAR -> id[digits]

id 为标识符，digits 为数字

（9）识别算数运算符

SUMOP -> + | - | ++ | -- | * | /

（10）识别关系运算符

RELOP -> < | > | <= | >= | != | ==

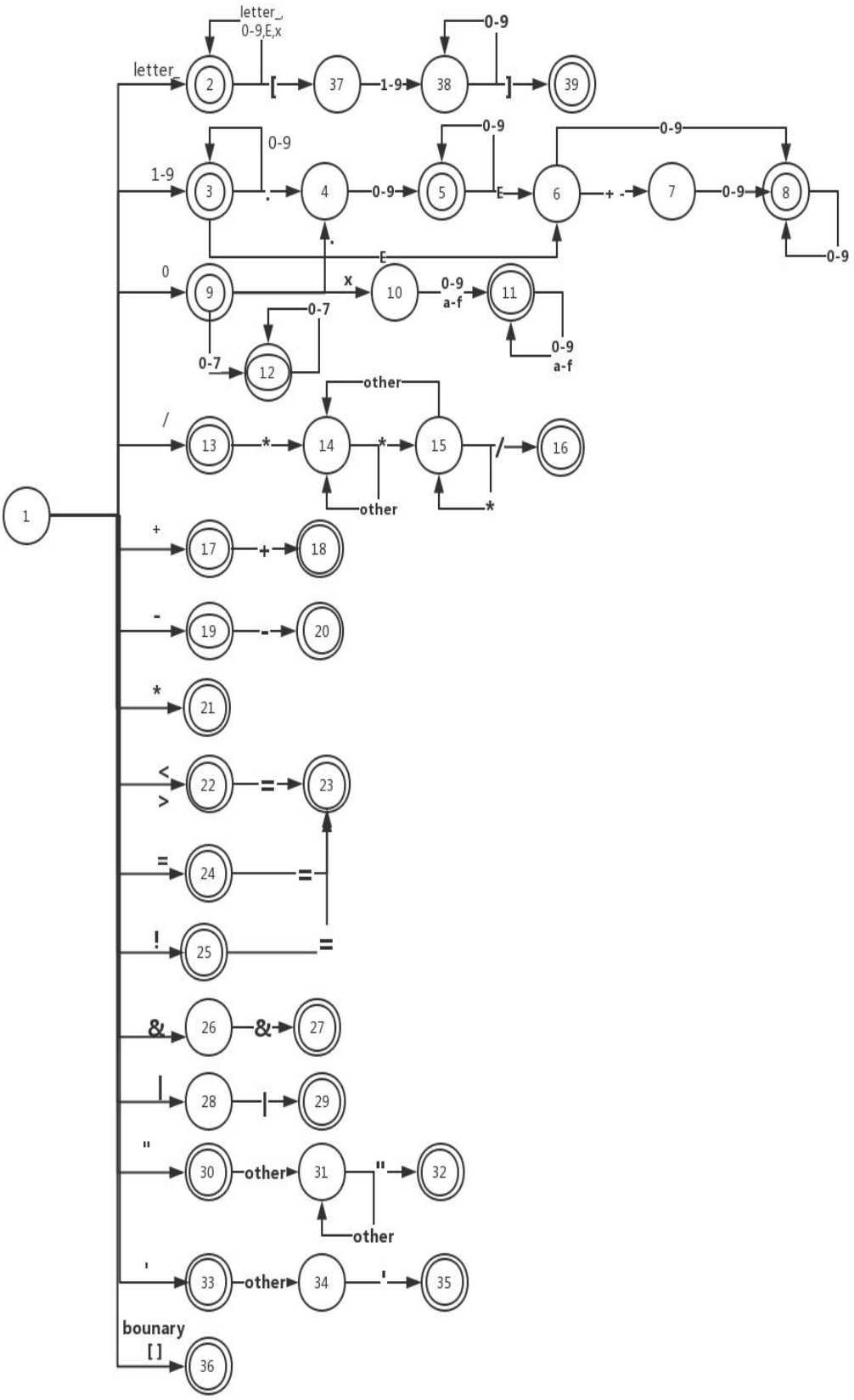
（11）识别逻辑运算符

LOGICOP -> && | || | !

（12）识别界符

Bounary -> (|) | { | }| ,| ; | ”| ’ | [|] | . | =

3、各类单词的转换图


letter_
0-9,E,x
letter_
2
[
37
1-9
38
0-9
]
39
0-9
1-9
3
.
4
0-9
5
E
6
+ -
7
0-9
8
0-9
.
E
0
9
x
10
0-9
a-f
11
0-9
a-f
0-7
0-7
12
other
/
13
*
14
*
15
/
16
other
*
1
+
17
+
18
-
19
-
20
*
21
<
>
22
=
23
=
24
=
!
25
=
&
26
&
27
|
28
|
29
"
30
other
31
"
32
other
'
33
other
34
'
35
bounary
[ ]
36

1 -> 2 是标识符的识别过程；
1 、3 -> 8、9 是十进制无符号常数的识别过程，其中 3、9 表示十进制无符号整型，5 表示浮点数，8 表示科学计数法；
1、9 ->12 是八进制和十六进制数的识别过程，其中 11 表示十六进制，12 表示 8 进制；
1、13 ->16 是注释的识别过程，13 表示注释；
1、13、17->21 是算数运算符的识别过程；
1、22、23 是关系运算符的识别过程；
1、25->29 是逻辑运算符的识别过程
1、30->32 是字符串常量的识别过程
1、33->35 是字符常量的识别过程
24、30、33、36 均是界符的识别过程
1->2、37->39 是数组的识别过程

FA 表：

FA1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

	f-w	1——7	8——9	bounary	a-f	0	E	x	+	-	*	/	<	>	=	!	&	|	.
1	2	3	3	36	2	9	-1	-1	17	19	21	13	22	22	24	25	26	28	-1
2*	2	2	2	-1	2	2	2	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3*	-1	3	3	-1	-1	3	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	4
4	-1	5	5	-1	-1	5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
5*	-1	5	5	-1	-1	5	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
6	-1	8	8	-1	-1	8	-1	-1	7	7	-1	-1	-1	-1	-1	-1	-1	-1	-1
7	-1	8	8	-1	-1	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8*	-1	8	8	-1	-1	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	4
9*	-1	12	-1	-1	-1	12	-1	10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
10	-1	11	11	-1	11	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11*	-1	11	11	-1	11	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
12*	-1	12	-1	-1	-1	12	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
13*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	14	-1	-1	-1	-1	-1	-1	-1	-1
14	14	14	14	14	14	14	14	14	14	14	15	14	14	14	14	14	14	14	14
15	14	14	14	14	14	14	14	14	14	14	15	16	14	14	14	14	14	14	14
16*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
17*	-1	-1	-1	-1	-1	-1	-1	-1	-1	18	-1	-1	-1	-1	-1	-1	-1	-1	-1
18*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
19*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	20	-1	-1	-1	-1	-1	-1	-1	-1
20*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
21*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
22*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	23	-1	-1	-1	-1
23*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
24*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	23	-1	-1	-1	-1
25*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	23	-1	-1	-1	-1
26	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	27	-1	-1	-1
27*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
28	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	29	-1	-1
29*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
30*	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
32*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
33*	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34
34	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
35*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
36*	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
37	-1	38	38	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

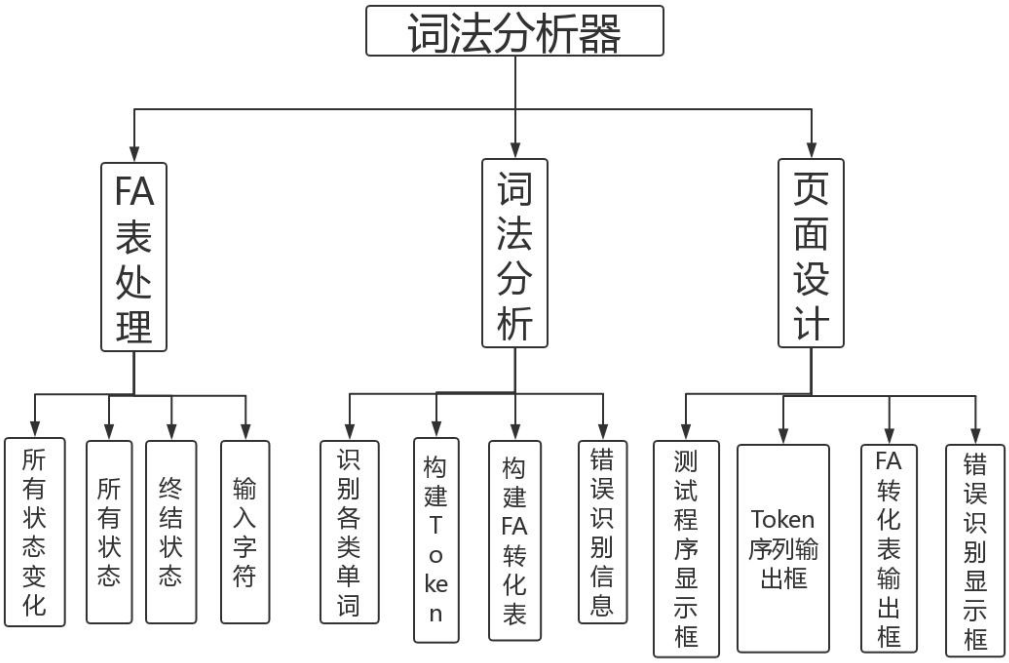
三、系统设计

得分

要求：分为系统概要设计和系统详细设计。

1、系统概要设计：给出必要的系统宏观层面设计图，如系统框架图、数据流图、功能模块图等以及相应的文字说明。

（1）系统框架图

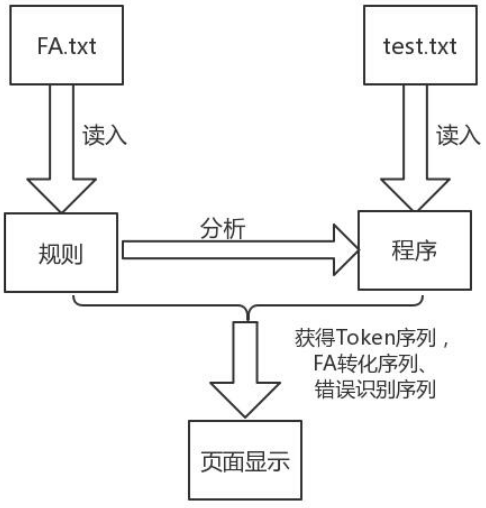


FA 表的处理：通过读入 FA.txt 获得词法分析的规则，这个过程中需要对 FA 表进行处理，除了直观读入的 FA 转化二维数组以外，还需要从中分离出所有状态，终结状态和输入字符集合。

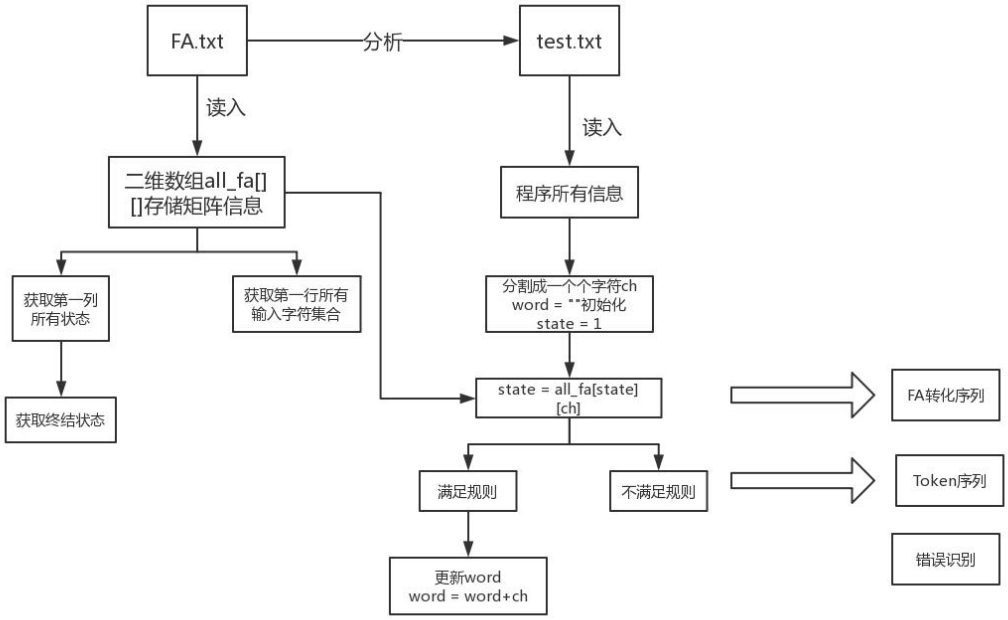
词法分析：读入测试程序，通过从 FA 表得到的分析规则分析程序，将程序分析后得到的 token 序列，FA 转化关系，错误识别信息存储。

页面设计：将词法分析中得到的结果输入显示到页面上。

（2）数据流图



（3）功能模块图



2、系统详细设计：对如下工作进行展开描述

✓ 核心数据结构的设计

（1）FA 相关

读入的 FA 二维数组：

```
public String[][] all_fa = new String[50][30];
```

根据 FA 得到的输入符号集合：

```
public ArrayList<String> inop_fa = new ArrayList<String>();
```

根据 dfa 得到的状态集合集合：

```
public ArrayList<String> state_fa_s = new ArrayList<String>();
```

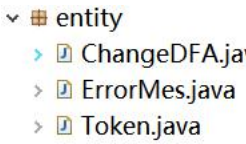
将状态集合转化为 Int 形式：

```
public ArrayList<Integer> state_fa = new ArrayList<Integer>();
```

得到终止状态集合 ：

```
public ArrayList<Integer> finalstate_fa = new ArrayList<Integer>();
```

（2）自定义的几个数据类



Token.java:与 Token 序列相关的值，包括单词名称，种别码，属性值，单词类型；

ErrorMes.java:与错误信息序列有关的值，包括行号，识别的字符以及字符序列号（方便定义错误地址）、错误类型，当前错误的单词名；

ChangeDFA.java:与 FA 转换有关的值，包括当前状态，当前输入字符，下一个状态

（3）分析过程相关

读入文本：

```
public ArrayList<String> test_txt = new ArrayList<String>();
```

记录 Token:

```
public ArrayList<Token>  tokenlist = new ArrayList<Token>();
```
记录 DFA 的转换过程：
```
public ArrayList<ChangeDFA> chdfalist = new ArrayList<ChangeDFA>();
```
记录错误信息：
```
public ArrayList<ErrorMes> errorlist = new ArrayList<ErrorMes>();
```

✓ 主要功能函数说明

（1）InputFA.java——与 FA 相关信息处理

`public static String[][] readFile()`：读入 FA.txt 并得到二维数组

`public static ArrayList<String> GetInOps(String[][] rows,int length)`：获取合法符号集合

`public static ArrayList<String> GetState(String[][] rows,int length)`：获取所有状态集合；

`public static ArrayList<Integer> GetFinalState(ArrayList<String> state_fa_s)` 获取终结状态集合；

`public static ArrayList<Integer> GetInOp(ArrayList<String> state_fa_s)` ：将终结状态集合转化为 int 格式，方便处理

（2）WordWindow.java:分析过程的主要逻辑实现以及页面设计

`private void Analyze(ArrayList<String> test_txt2,int state)`：分析过程主程序

`private void DealError(int j, int i, char ch, int state,String word)`：错误处理分析；

`private boolean isKeyWord(String word)`：判断是否是关键字

`private boolean isFinalState(int state)` ：判断当前字符是否是终结符

`private int JudgeKindByCh(char ch)` ：判断当前字符是哪一种类型的输入，返回列号

`private boolean isOp(String[] col, String word)` ：判断当前字符是否属于某一集合，比如边界符；

`public  void readTestFile()`：输入测试程序

`private void ShowGUI1()` ,`private void ShowGUI2()`,`private void ShowGUI3()`：更新页面，输出 token 序列，FA 转化序列，错误信息处理

`private void cleartokenlist()`、`private void clearerrorlist()`、

`private void cleardfachangelist()`处理相应序列中的空值信息。

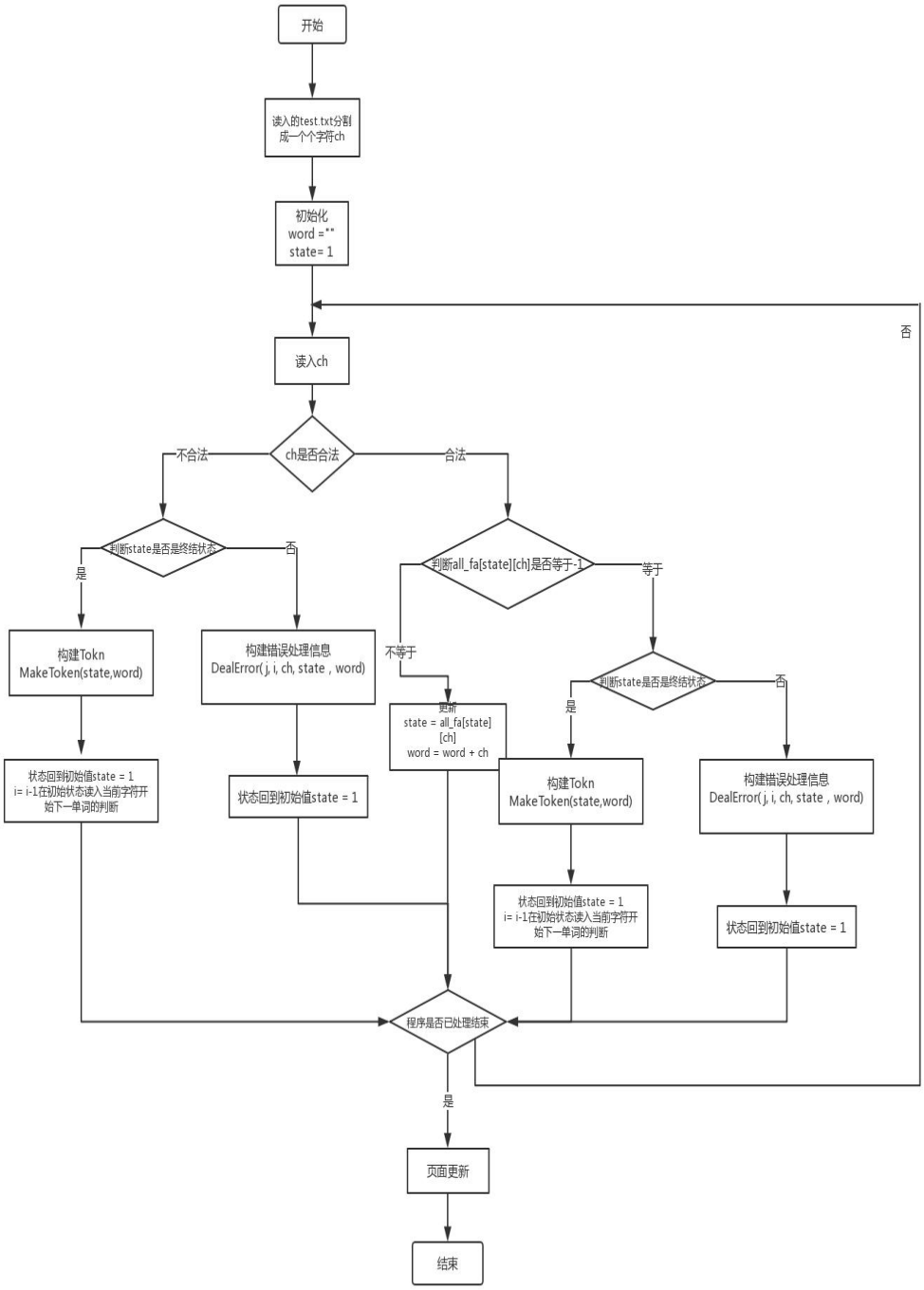
`private void MakeToken(int state, String word)`：构建 Token 序列

Value 是当前识别到的单词的值

终止状态	类型	Token
2	表示符	<300,value>
2	关键字	<value,->
3、9	十进制无符号整型	<301,value>
5	浮点数	<302,value>
8	科学计数法	<303,value>
11	十六进制无符号整型	<304,value>
12	八进制无符号整型	<305,value>
13、17、18、19、20、21	算数运算符	<307,value>
24、30、33、36	界符	<value,->
25、27、29	逻辑运算符	<309,value>

32	字符串常量	<310,value>
35	字符常量	<311,value>
39	数组	<312,value>
22、23	关系运算符	<313,value>

✓ 程序核心部分的程序流程图



四、系统实现及结果分析	得分	

要求：对如下内容展开描述。

1、系统实现过程中遇到的问题；

（1）FA 表的设计

FA 表中的所有 null 项应该使用-1 填充，方便进行计算；

FA 表中的输入字符应该进行过分类集合，否则 abcd...ABDC...1234...每一个都做单独字符输入时，FA 表过大也不利于修改维护，所以需要对输入字符集合分类，分类的标准由 FA 设计产生，比如因为要识别八进制和十六进制，那么[0,9]就被分成了 0、[1,7],[8,9]；

FA 中的终止状态应该有所标识，比如 7*，带有*号的状态即为终止状态，通过读入 FA 表获取规则时，可以凭此得到终止状态集合。

（2）如何根据读入的程序来进行词法分析

程序读入后进行词法分析的单位是一个个字符，但是输出 token 时却是以一个个单词输出的，所以这个设计到一个单词的构建过程。读入程序后，将程序分割成一个个字符，初始化识别状态 state=1，识别单词 word = “”,根据得到的 DFA 规则，依次更新 state 和 word 并跳转入相关程序进行处理。详情可参见上述程序流程图。

这个过程中还有一个需要注意的点，我们需要根据输入的字符来得到 all_fa[][]的相应列数，而在 FA 表中的某些输入字符是以集合的形式存在的，所以需要对该字符具体对应某一列做一个判断。

（3）如何构建 Token 序列

Token 的构建过程需要有单词值<种别码，属性值>，具体的规定报告前面有给出。

构建 Token 主要是使用一个 switch()语句来判断当前状态属于哪一个终结状态，不同的终结状态对应着不同的 token 类型。

（4）如何进行错误处理

同样使用 switch()语句来比对状态，当当前状态时终结状态，但是当前输入并不是此状态的合法输入时，发生错误。

为了方便插卡，错误分析中给出错误定位。

（5）如何构建 dfa 转化表

每一次读入一个字符时，都有当前状态，当前输入，下一个状态 all_fa[state][ch]，写入 dfalist 中。

（6）如何更新页面

更新页面依据分析过程中的三个 list——tokenlist , chdfalist , errorlist 来将相关信息显示到页面上。

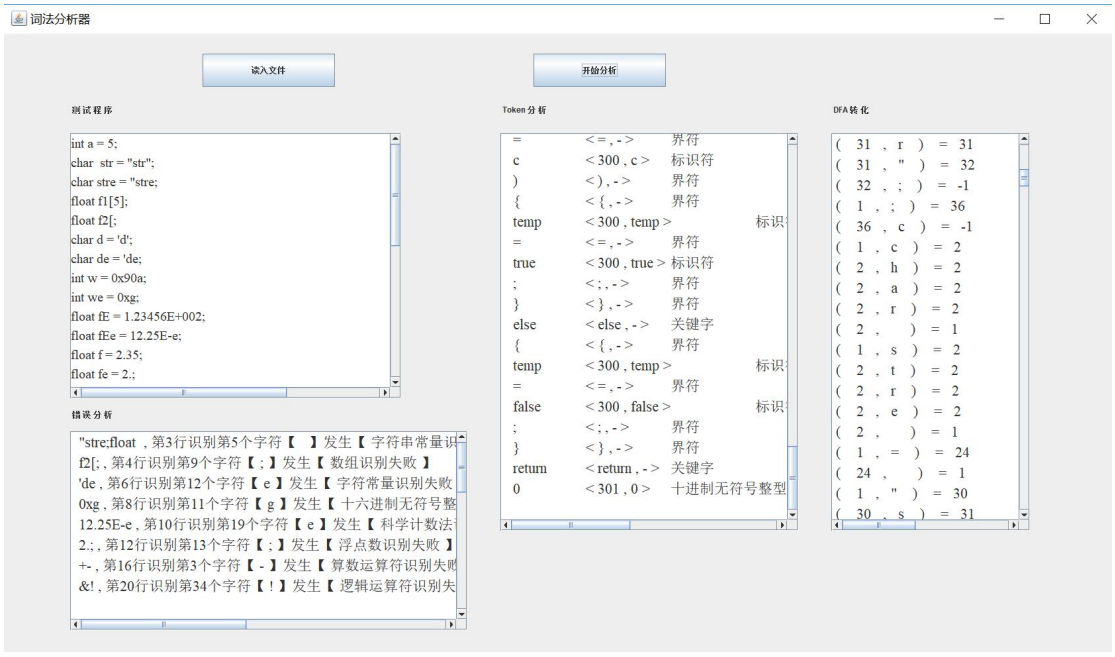
有时，需要对 tokenlist , chdfalist , errorlist 的空值做相关处理。

2、针对某测试程序输出其词法分析结果；

test.txt

```
1   int a = 5;
2   char  str = "str";
3   char stre = "stre;
4   float f1[5];
5   float f2[;
6   char d = 'd';
7   char de = 'de;
8   int w = 0x90a;
9   int we = 0xg;
10  float fE = 1.23456E+002;
11  float fEe = 12.25E-e;
12  float f = 2.35;
13  float fe = 2.;
14  int i = 0;
15  do{
16      i++;
17      i+-;
18  }while(i < 4);
19  /*test*/
20  boolean temp;
21  if(a  *  b  ==  20  &&  a  <  b  &! a = c )
22  {
23      temp = true;
24  }
25  else
26  {
27      temp = false;
28  }
29  return  0;
```

整体界面：



Token 序列

```
int         < int , - >       关键字
a           < 300 , a >       标识符
=           < = , - >         界符
5           < 301 , 5 >       十进制无符号整型
;           < ; , - >         界符
char        < char , - >      关键字
str         < 300 , str >     标识符
=           < = , - >         界符
"str"       < 310 , "str" >   字符串常量
;           < ; , - >         界符
char        < char , - >      关键字
stre        < 300 , stre >    标识符
=           < = , - >         界符
f1[5]       < 312 , f1[5] >   数组
;           < ; , - >         界符
float       < float , - >     关键字
char        < char , - >      关键字
d           < 300 , d >       标识符
=           < = , - >         界符
```

```
=           < = , - >         界符
'd'         < 311 , 'd' >     字符常量
;           < ; , - >         界符
char        < char , - >      关键字
de          < 300 , de >      标识符
=           < = , - >         界符
;           < ; , - >         界符
int         < int , - >       关键字
w           < 300 , w >       标识符
=           < = , - >         界符
0x90a       < 304 , 0x90a >   十六
;           < ; , - >         界符
int         < int , - >       关键字
we          < 300 , we >      标识符
=           < = , - >         界符
;           < ; , - >         界符
float       < float , - >     关键字
fE          < 300 , fE >      标识符
=           < = , - >         界符
```

```
1.23456E+002        < 303 , 1.23456E+0
;           < ; , - >         界符
float       < float , - >     关键字
fEe         < 300 , fEe >     标识符
=           < = , - >         界符
;           < ; , - >         界符
float       < float , - >     关键字
f           < 300 , f >       标识符
=           < = , - >         界符
2.35        < 302 , 2.35 >    浮点数
;           < ; , - >         界符
float       < float , - >     关键字
fe          < 300 , fe >      标识符
=           < = , - >         界符
int         < int , - >       关键字
i           < 300 , i >       标识符
=           < = , - >         界符
0           < 301 , 0 >       十进制无符号整型
;           < ; , - >         界符
```

```
;           < ; , - >         界符
do          < do , - >        关键字
{           < { , - >         界符
i           < 300 , i >       标识符
++          < 307 , ++ >      算数运算符
;           < ; , - >         界符
i           < 300 , i >       标识符
+           < 307 , + >       算数运算符
-           < 307 , - >       算数运算符
;           < ; , - >         界符
}           < } , - >         界符
while       < while , - >     关键字
(           < ( , - >         界符
i           < 300 , i >       标识符
<           < 313 , < >       关系运算符
4           < 301 , 4 >       十进制无符号整型
)           < ) , - >         界符
;           < ; , - >         界符
boolean     < boolean , - >   关键
```

boolean	< boolean , - >	关键
temp	< 300 , temp >	标识
;	< ; , - >	界符
if	< if , - >	关键字
(	< (, - >	界符
a	< 300 , a >	标识符
*	< 307 , * >	算数运算符
b	< 300 , b >	标识符
==	< 313 , == >	关系运算符
20	< 301 , 20 >	十进制无符号整型
&&	< 309 , && >	逻辑运算符
a	< 300 , a >	标识符
<	< 313 , < >	关系运算符
b	< 300 , b >	标识符
a	< 300 , a >	标识符
=	< = , - >	界符
c	< 300 , c >	标识符
)	<) , - >	界符
{	< { , - >	界符

temp	< 300 , temp >	标识
=	< = , - >	界符
true	< 300 , true >	标识符
;	< ; , - >	界符
}	< } , - >	界符
else	< else , - >	关键字
{	< { , - >	界符
temp	< 300 , temp >	标识
=	< = , - >	界符
false	< 300 , false >	标识
;	< ; , - >	界符
}	< } , - >	界符
return	< return , - >	关键字
0	< 301 , 0 >	十进制无符号整型

顺序为从左到右，从上到下

DFA 转化序列

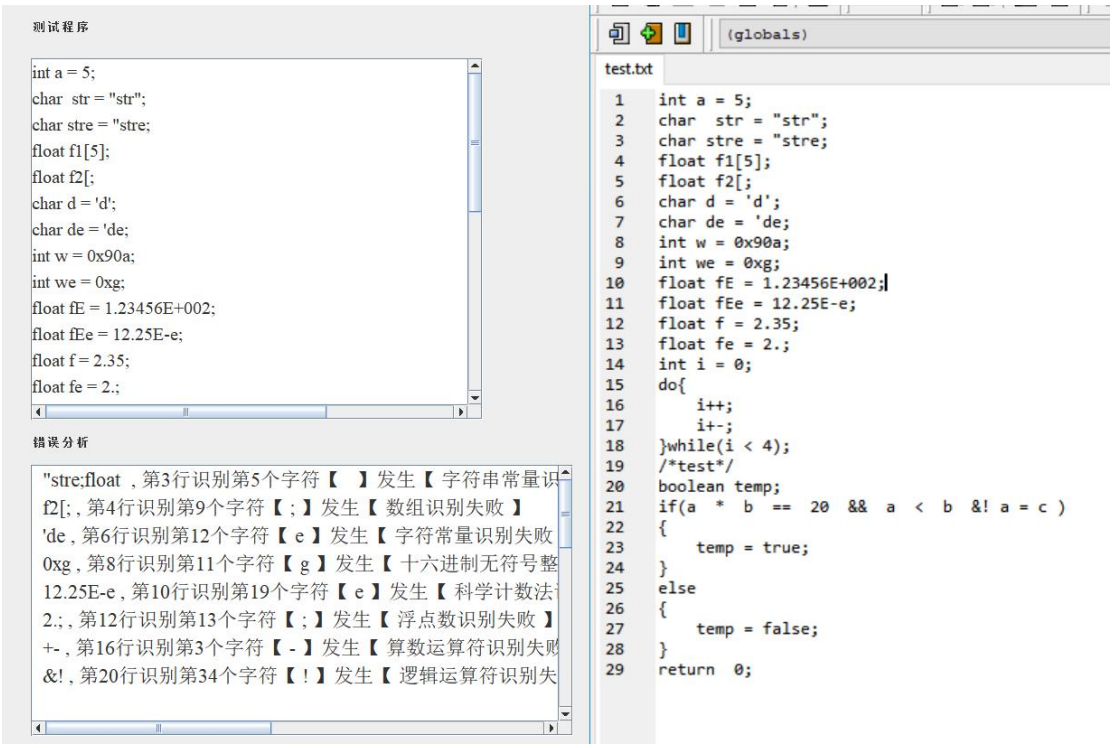
```
( 1 , i ) = 2
( 2 , n ) = 2
( 2 , t ) = 2
( 2 , ) = 1
( 1 , a ) = 2
( 2 , ) = 1
( 1 , = ) = 24
( 24 , ) = 1
( 1 , 5 ) = 3
( 3 , ; ) = -1
( 1 , ; ) = 36
( 36 , ) = 1
( 1 , c ) = 2
( 2 , h ) = 2
( 2 , a ) = 2
( 2 , r ) = 2
( 2 , ) = 1
( 1 , ) = 1
( 1 , s ) = 2
```

```
( 2 , r ) = 2
( 2 , ) = 1
( 1 , ) = 1
( 1 , s ) = 2
( 2 , t ) = 2
( 2 , r ) = 2
( 2 , ) = 1
( 1 , = ) = 24
( 24 , ) = 1
( 1 , " ) = 30
( 30 , s ) = 31
( 31 , t ) = 31
( 31 , r ) = 31
( 31 , " ) = 32
( 32 , ; ) = -1
( 1 , ; ) = 36
( 36 , c ) = -1
( 1 , c ) = 2
( 2 , h ) = 2
```

```
( 31 , r ) = 31
( 31 , " ) = 32
( 32 , ; ) = -1
( 1 , ; ) = 36
( 36 , c ) = -1
( 1 , c ) = 2
( 2 , h ) = 2
( 2 , a ) = 2
( 2 , r ) = 2
( 2 , ) = 1
( 1 , s ) = 2
( 2 , t ) = 2
( 2 , r ) = 2
( 2 , e ) = 2
( 2 , ) = 1
( 1 , = ) = 24
( 24 , ) = 1
( 1 , " ) = 30
( 30 , s ) = 31
```

数量过多，不一一贴出

3、输出针对此测试程序对应的词法错误报告；



4、对实验结果进行分析。

本词法分析器输入程序后，能够对大多数单词进行识别，成功构建 token 序列和 dfa 转化过程并且输出在界面上。

对于绝大多数单词拼写错误，可以正确检测并且给出错误报告，少部分错误无法识别或者识别有一些错误，这部分应该可以在后期语法语义分析中给出正确的错误分析。

注：其中的测试样例自行产生。

指导教师评语：

日期：