编译系统课程实验报告

实验 1：词法分析

姓名	余涛	院系	计算机科学与技术学院	学号	1180300829
任课教师	陈鄞		指导教师	陈鄞	
实验地点	格物 213		实验时间	2021.4.17	
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

（1）能识别以下几类单词：

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）

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）

运算符（①算术运算符；②关系运算符；③逻辑运算）

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“.”）

常数（无符号整数（含八进制和十六进制数）、浮点数（含科学计数法）、字符串常数等）

注释。识别两种风格的注释：一种是使用双斜线“//”进行单行注释，在这种情况下，该行在“//”符号之后的所有字符都将作为注释内容而直接被词法分析程序丢弃掉；另一种是使用“/*”以及“*/”进行多行注释，在这种情况下，在“/*”与之后最先遇到的“*/”之间的所有字符都被视作注释内容而直接被词法分析程序丢弃掉。

（2）能够进行简单的错误处理，即识别出测试用例中的非法字符。程序在输出错误提示信息时，需要输出具体的错误类型（即词法错误）、出错的位置（源程序行号）以及相关的说明文字，其格式为：Lexical error at Line [行号]: [说明文字].说明文字的内容没有具体要求（例如：非法字符），但是错误类型和出错的行号一定要正确，因为这是判断输出错误提示信息是否正确的唯一标准。

（3）系统的输入形式：要求能够通过文件导入测试用例。测试用例要涵盖“实验内容”中列出的各类单词。

（4）系统的输出形式：打印输出测试用例对应的token序列，格式如下图所示：

- 输入
  - while(num!=100){num++;}
- 输出

```
while    <WHILE,   _   >
(        <  SLP ,   _   >
num      <  IDN ,  num  >
!=       <   NE ,   _   >
100      <CONST,  100  >
)        <  SRP ,   _   >
{        <   LP ,   _   >
num      <  IDN ,  num  >
++       <  INC ,   _   >
;        < SEMI ,   _   >
}        <   RP ,   _   >
```

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）

令 digit—>[0-9]

letter_ —> [A-Za-z_]

（1）标识符：

letter_(letter_|digit)*

（2）关键字：

char | long | short | float | double | const | Boolean | void | null | false | true | enum | int | do | while | if | else | for | then | break | continue | class | static | final | extends | new | return | signed | struct | union | unsigned | goto | switch | case | default | auto | extern | register | sizeof | typedef | volatile | String

（3）运算符：

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

（4）界符：

(|) | { | } | ; | [|] | , | =

（5）常数：

无符号数(包含科学计数法)：digit digit* (\.digit digit*)? ([Ee][+-]? digit digit*)?

无符号八进制整数：0[0-7][0-7]*

无符号十六进制整数：0x[0-9a-fA-F] [0-9a-fA-F]*

字符串常数：\"(.|\\)*\"

字符常数：\'(.)\'

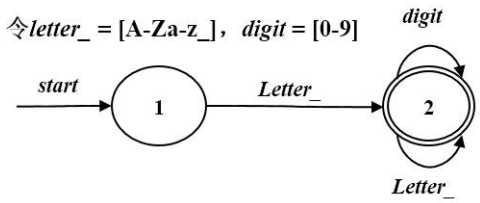
（6）注释：

/* */型注释：/*(.|\r\n)**/

//型注释：//[\s\S]*?\n

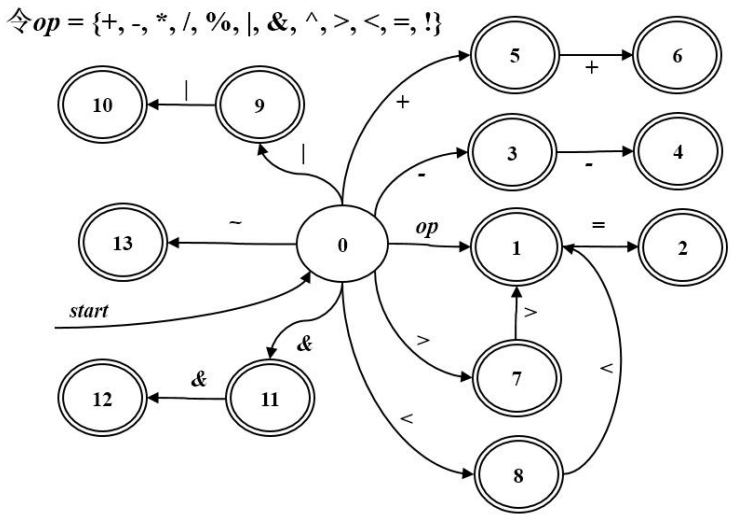
2. 各类单词的转换图

（1）标识符：



（2）关键字：只需要判断是否属于关键字集合即可

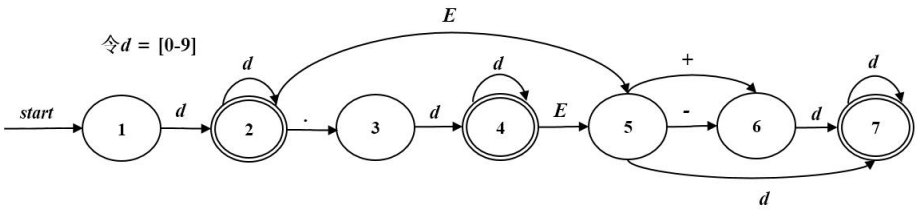
（3）运算符：



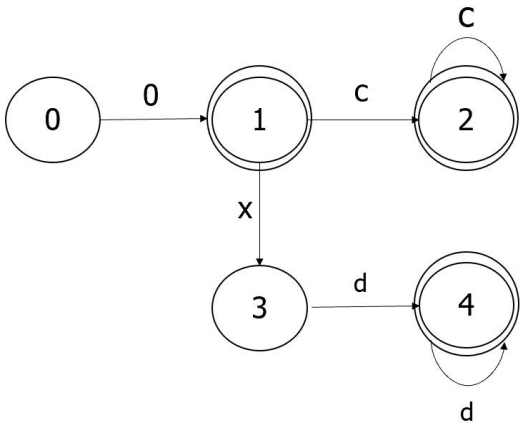
（4）界符：只需要判断是否属于界符集合即可

（5）常数：

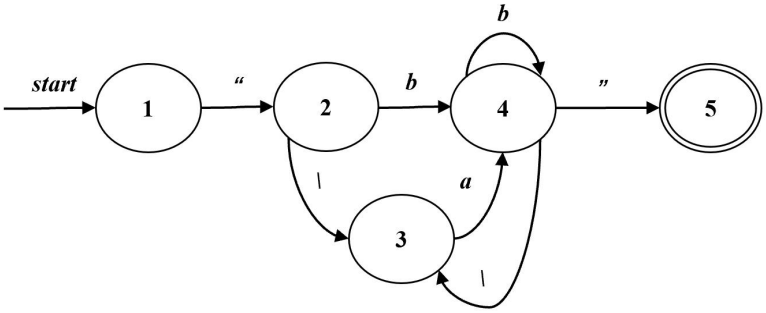
无符号常数：



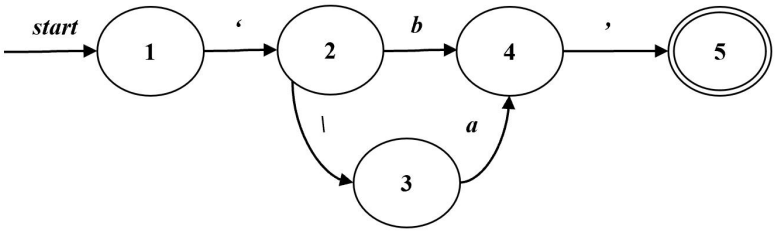
八进制或者十六进制常数：c = [0-7]，d = [0-9a-fA-F]



字符串常量：a 为能转义的字符，b 为除了\和”的任意字符

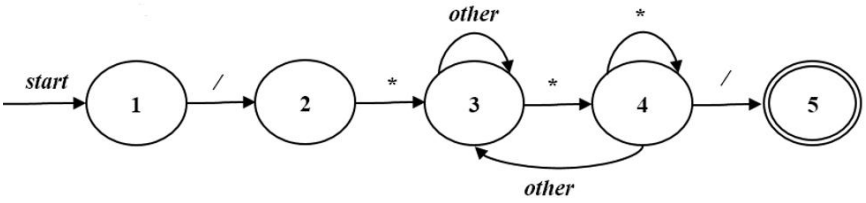


字符常量：a 为能转义的字符，b 为除了\和”的任意字符

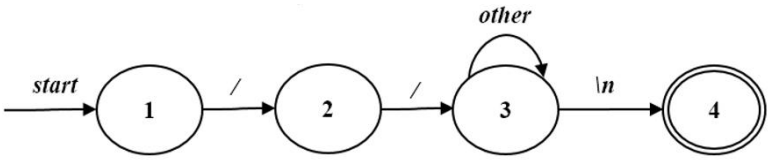


（6）注释：

/* */型注释：other = [.|\r|\n]



//型注释：other = [\s|\S]



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。

1.1 系统框架图：

整个系统包括三个层次：外观表示层、常量数据层、处理层。外观表示层负责 Token 序列与错误信息的输出显示。常量数据层中包含两个部分：其中 words 类包含种别码表和 DFA 转换表，还包含测试用例的 txt 文件。处理层为 Lexical 类，是词法分析程序，用来对给定输入进行词法分析。

外观表示层	ui类：GUI程序界面	
常量数据层	words类：常量及其种别码表，DFA状态转换表	测试用例txt文件
处理层	Lexical类：词法分析程序	

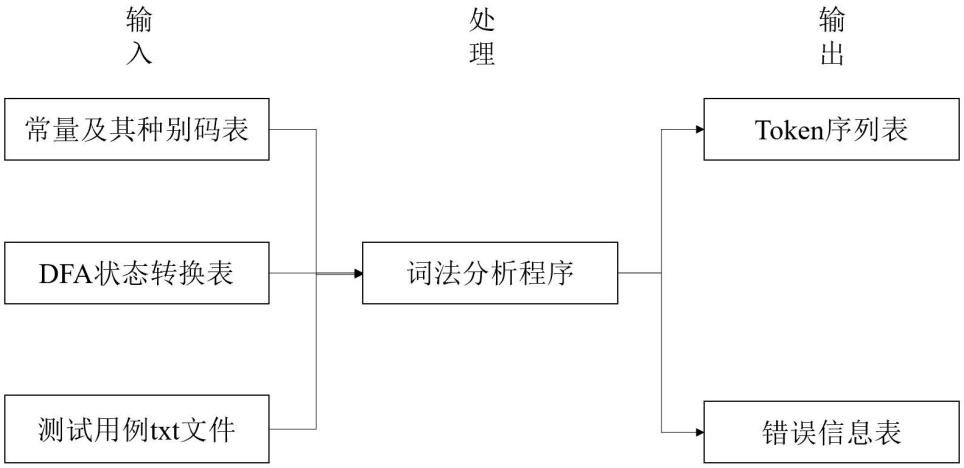
1.2 数据流图

数据流图包括三个部分：输入、处理、输出。

输入：待分析文件为测试用例的 txt 文件，然后需要输入 DFA 状态转换表和种别码表，用于后来的词法分析程序进行词法分析。

处理：通过调用相关的 DFA 状态转换表完成词法分析，并将 Token 序列和错误报告储存在两个 List 里面，用于后来的输出。

输出：对词法分析后的 Token 序列和错误报告进行 GUI 界面的输出。



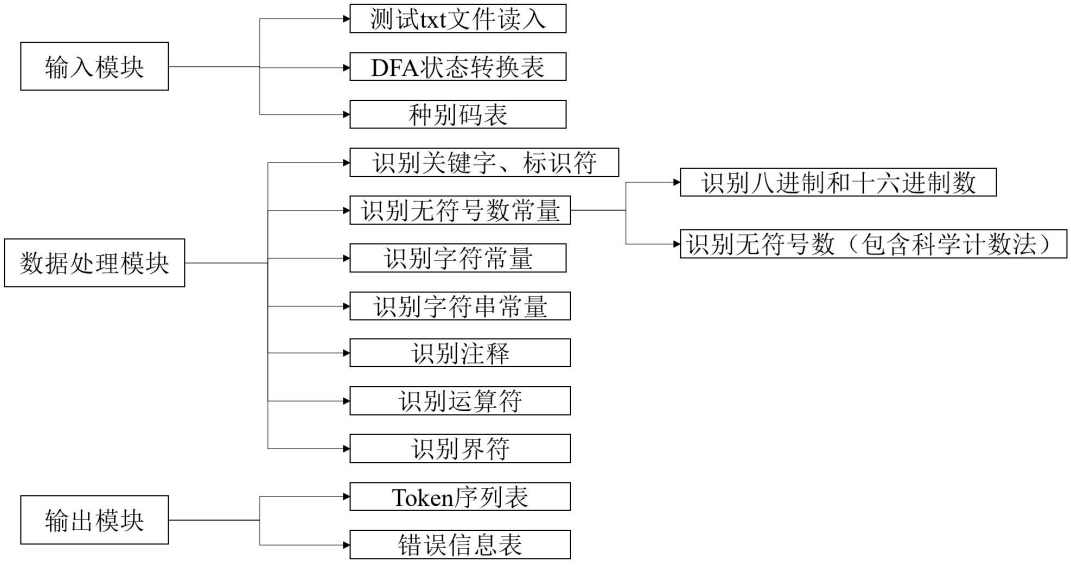
1.3 功能模块图

整个系统一共包括三个模块：输入模块、数据处理模块、输出模块

输入模块：负责通过 GUI 读入 txt 文件中的测试用例，并且读入 DFA 状态转换表和种别码表用于数据处理。

数据处理模块：按照每次读入的字符来分别识别各种词法。

输出模块：负责通过 GUI 输出词法分析后的 Token 序列表和错误信息表。



（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. Token 类：用于储存词法分析后的一个 Token

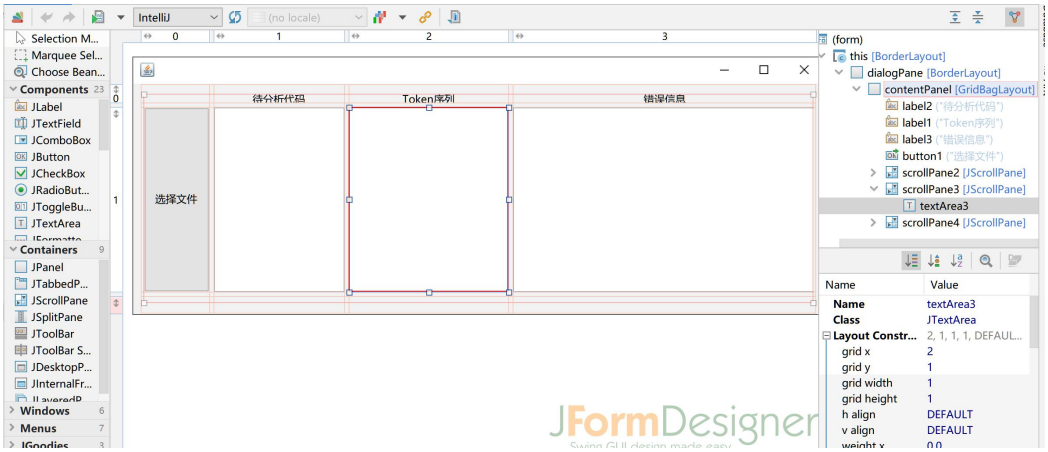
```
public class Token {
    public final int tag;  //种别码
    public final String value;  // 单词
    public final int line;  // 行数

    public Token(int tag, String value, int line) {
        this.tag = tag;
        this.value = value;
        this.line = line;
    }
```

包括三个字段：种别码，单词，行数

2. 错误集合 allmistake 和 Token 集合 allTokenClass：用于储存词法分析后的所有的错误信息和 Token 集合，以便在 GUI 输出：

```
public List<String> allmistake = new ArrayList<>();  // 错误集合
public List<Token> allTokenClass = new ArrayList<>();  // Token集合
```

3. ui 类：使用 idea 的 JformDesigner 插件绘制 GUI，并生成相应的代码：



✓ 主要功能函数说明

1. toString()函数：

为 Token 类中重写的 toString()方法，用于返回一个 Token 序列，对两种 Token 序列进行判断，其中标识符、无符号整型（包括八进制和十六进制）、浮点型常量、科学计数法、字符常量、字符串常量这些需要输出 value 值。

：

```
// 返回Token序列
@Override
public String toString() {
    if (tag == 1 || tag == 2 || tag == 3 || tag == 4 || tag == 5 || tag == 6){
        return line + "  " + value + "  < " + tag + ", " + value + " > ";
    }
    else{
        return line + "  " + value + "  < " + tag + ",  - > ";
    }
}
```

2. 各种 DFA 转换匹配函数：使用一个字符串数组来完成每个状态可识别字符的标注，然后配合 DFA 转换匹配函数进行 DFA 状态转换。

```
// String DFA : a为能转义的字符，b代表除\和"之外的字符
public static String stringDFA[] =
        {
                "#\\b#",
                "##a#",
                "#\\b\"",
                "####"
        };
```

```
// char DFA :a为能转义的字符，b代表除\和'之外的字符
public static String charDFA[] =
        {
                "#\\b#",
                "##a#",
                "###\'",
                "####"
        };

 // 八进制和十六进制无符号整数DFA
 // c为[0-7]，d为[0-9a-fA-F]
 public static String eightsixDFA[] =
        {
                "#cx#",
                "#c##",
                "###d",
                "###d",
        };
// 十进制无符号数DFA，包括小数
public static String digitDFA[] =
        {
                "#d#####",
                "#d.#e##",
                "###d###",
                "###de##",
                "####+-d",
                "######d",
                "######d"
        };

 // 注释DFA：c表示[.|\r|\n]
 public static String noteDFA[] =
        {
                "#####",
                "##*##",
                "##c*#",
                "##c*/",
                "#####"
        };
```

这些字符串数组所对应的 DFA 转换匹配函数如下：

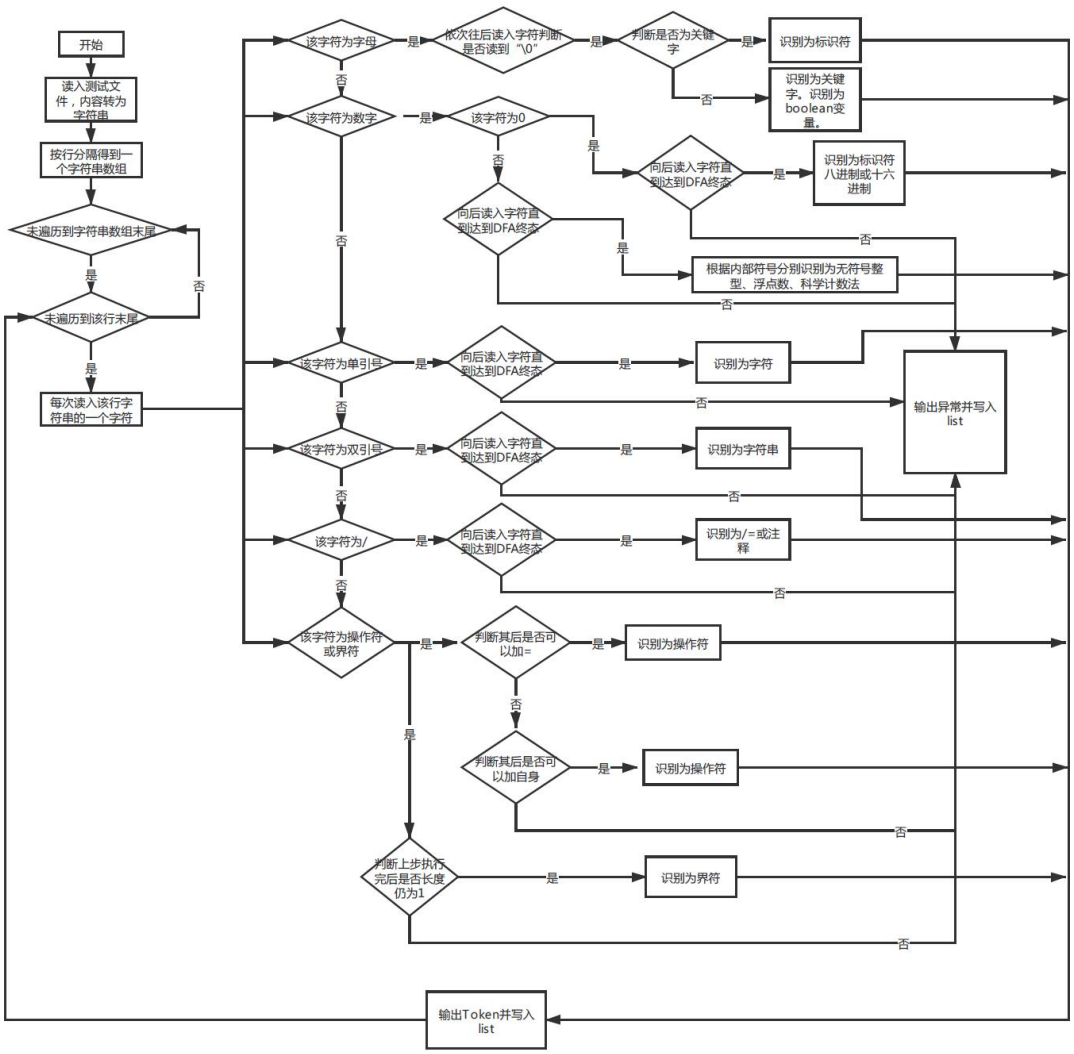
```
/**
 * 字符串DFA状态匹配函数
 * @param ch 当前字符
 * @param key 状态表中的字符
 * @return 匹配成功返回true，否则返回false
 */
public static Boolean is_string_state(char ch, char key)
```

```
/**
 * 字符DFA状态匹配函数
 * @param ch 当前字符
 * @param key 状态表中的字符
 * @return 匹配成功返回true，否则返回false
 */
public static Boolean is_char_state(char ch, char key)

/**
 * 八进制和十六进制无符号整数DFA状态匹配函数
 * @param ch 当前字符
 * @param test 状态表中的字符
 * @return 匹配成功返回true，否则返回false
 */
public static Boolean is_eightsix_state(char ch, char test){

 /**
  * 数字DFA状态匹配函数
  * @param ch 当前字符
  * @param test 状态表中的字符
  * @return 匹配成功返回1，否则返回0
  */
 public static int is_digit_state(char ch, char test)
 /**
  * 注释DFA状态匹配函数
  * @param ch 当前字符
  * @param nD 状态表中的字符
  * @param s 状态
  * @return 匹配成功返回true，否则返回false
  */
 public static Boolean is_note_state(char ch, char nD, int s)
```

3. init()函数：负责种别码的初始化：

```
/*
 * 种别码的初始化函数
 */
public static void init(){
    for (int i = 0; i < keywords.length; i++) {  // 关键字初始化
        keywords_node.put(keywords[i], 101 + i);
    }
    for (int i = 0; i < operators.length; i++) {  // 操作符初始化
        operators_node.put(operators[i], 201 + i);
    }
    for (int i = 0; i < delimiters.length; i++) {  // 界符初始化
        delimiters_node.put(delimiters[i], 301 + i);
    }
}
```

✓ 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；

（1.1）DFA 设计与实现：

为了简化的实现 DFA 自动机，采用了一个二维数组来描述对应的自动机转化：以识别字符串的自动机为例：

```
// String DFA : a为能转义的字符，b代表除\和"之外的字符
public static String stringDFA[] =
        {
                "#\\b#",
                "##a#",
                "#\\b\"",
                "####"
        };
```

这个数组每一行代表一个状态，所以该自动机一共有 0、1、2、3 四个状态，以第一行也就是 0 状态为例，用第一行的每一列作为当前 0 状态可以转向的状态，接受#表示不能完成转换，所以第 0 行表示的就是在 0 状态可以接收\转向 1 状态，也可以接收 b 转向 2 状态。其他的状态同理。

（1.2）种别码的设置（如附录）：

为了后续语法分析的方便，选择对标识符、无符号整型（包括八进制和十六进制）、浮点型常量、科学计数法、字符常量、字符串常量、布尔型常量分别设置表示符为 1-7。对于关键字需要一字一码，从 101 开始每次加一作为种别码。对于运算符也需要一符一码，从 201 开始每次加一作为种别码。对于界符也需要一符一码，从 301 开始每次加一作为种别码。

（1.3）无符号数的识别：

由于无符号数包括无符号整数和浮点数（含科学计数法），无符号整数中包括十进制、八进制和十六进制数，我构建了两个自动机，第一个用来识别八进制和十六进制，当识别 0 字符后，如果后续字符为 x 或者[0-7]，这样才能进入这个自动机。而其他的情况就会进入另外一个识别浮点数的自动机。这样就完成了无符号整数和浮点数的区别开。并且在识别科学技术法的时候对于 E 和 e 进行合并，也就是说 E 的大小写不影响科学计数法的识别。

（1.4）字符串或者字符中识别到转义字符的处理问题：

当识别到字符串或者字符时，若遇到了转义字符\，且其后的元素不满足可转义（可转义的字符包括 a,b,f,n,r,t,v,?,0,”），则不再进行自动机的转换，直接一直往后读新的字符，直到读到一个新的”结束，进行字符串或者字符的输出，若没有”进行结束，则报错。

（1.5）如何储存 Token 序列和错误报告：

创建了一个 Token 类，用于储存 Token 序列中的任意一个 Token：

```
public class Token {
    public final int tag;  //种别码
    public final String value;  // 单词
    public final int line;  // 行数
```

然后创建两个 List，分别用于储存 Token 序列和错误报告，在识别到一个词后或者错误时，将信息加入到 List 中即可。

```
public List<String> allmistake = new ArrayList<>();  // 错误集合
public List<Token> allTokenClass = new ArrayList<>();  // Token集合
```

（2） 针对某测试程序输出其词法分析结果；

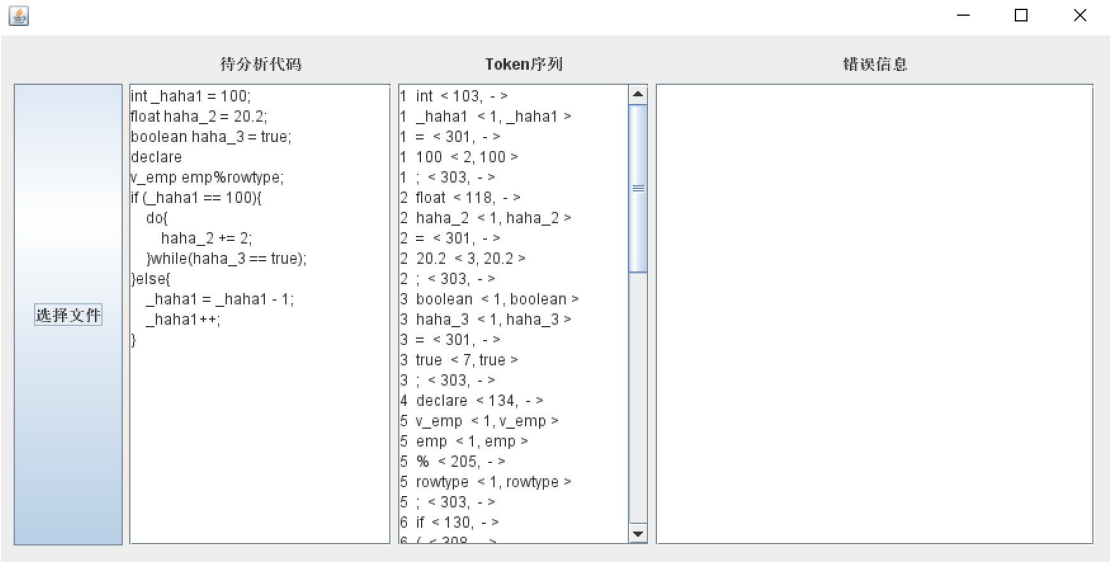
创建了多个测试用例文件进行测试：

（2.1）测试标识符和关键字：identify_keyword.txt

```
1. int _haha1 = 100;
2. float haha_2 = 20.2;
3. boolean haha_3 = true;
```

```
declare
v_emp emp%rowtype;
if (_haha1 == 100){
    do{
        haha_2 += 2;
    }while(haha_3 == true);
}else{
    _haha1 = _haha1 - 1;
    _haha1++;
}
```

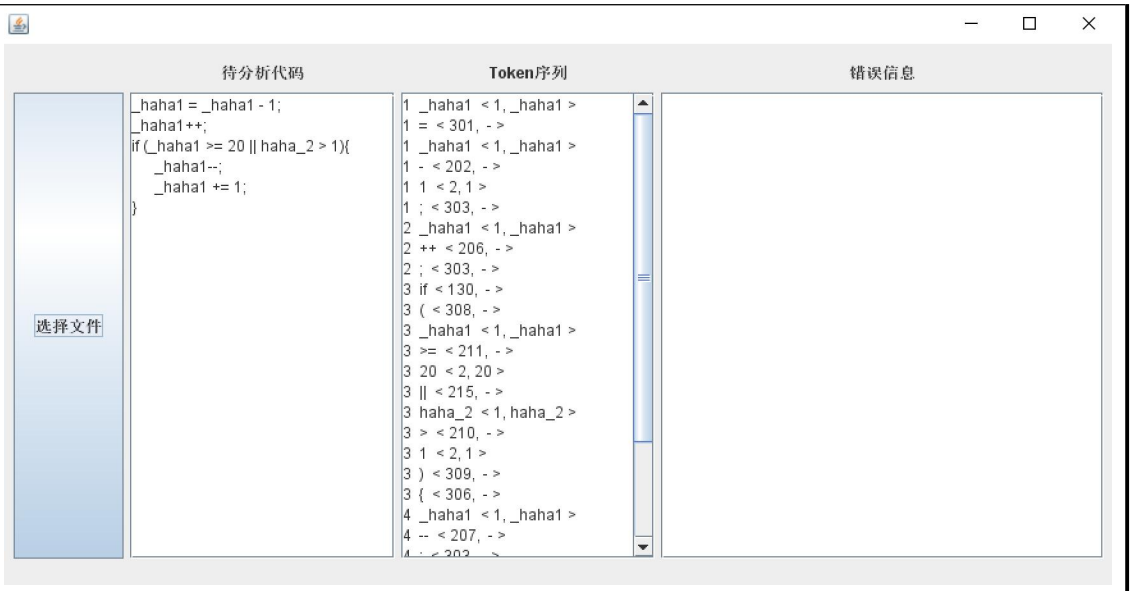
测试结果如下：



（2.2）测试运算符：operator.txt

```
_haha1 = _haha1 - 1;
_haha1++;
if (_haha1 >= 20 || haha_2 > 1){
    _haha1--;
    _haha1 += 1;
}
```

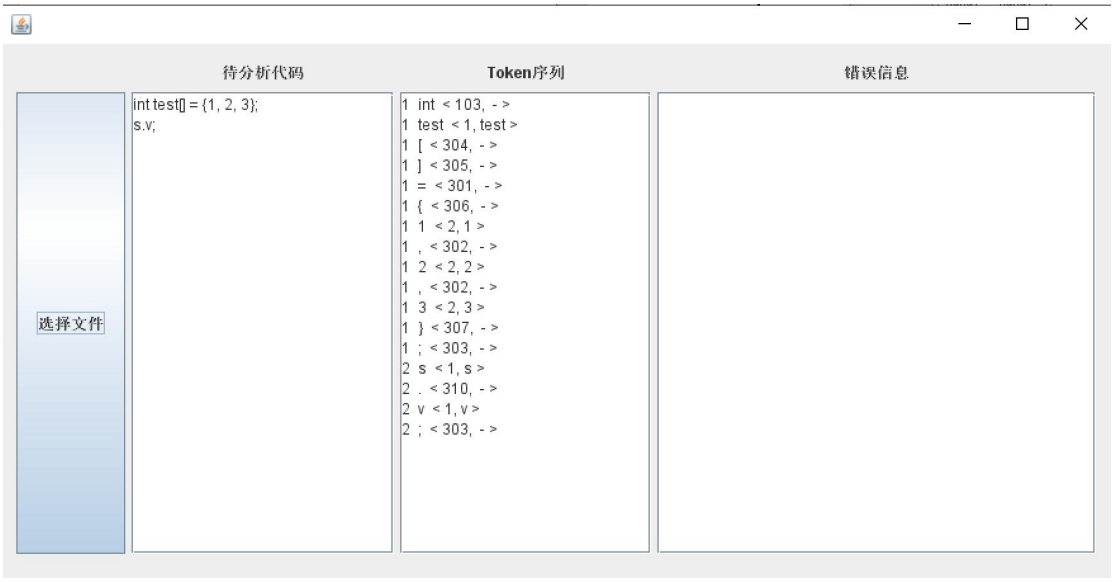
测试结果如下：



（2.3）测试界符：delimiter.txt

```
int test[] = {1, 2, 3};
s.v;
```

测试结果如下：

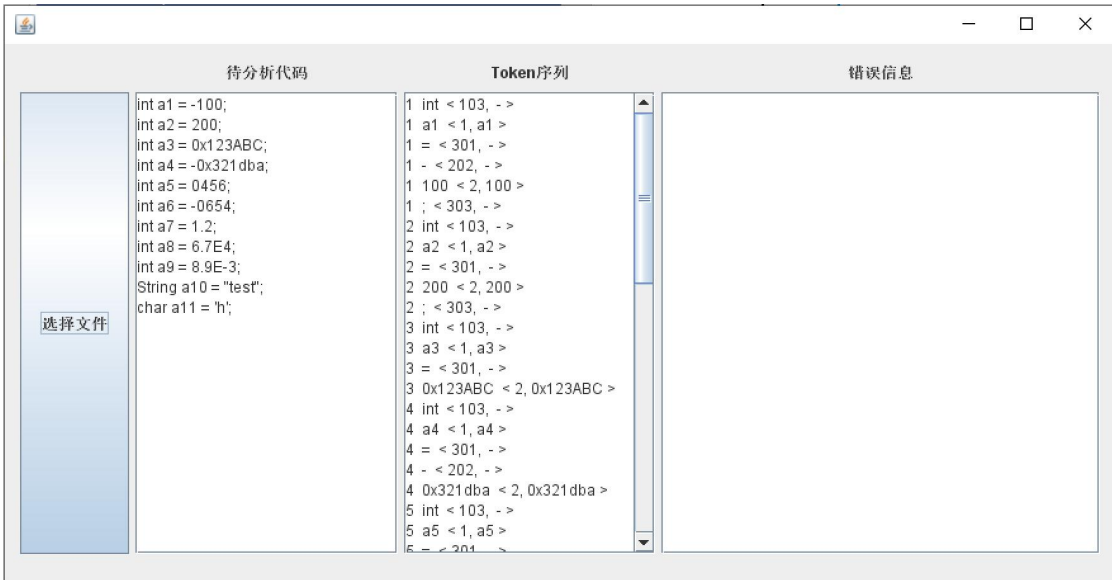


（2.4）测试常数：constant.txt

```
int a1 = -100;
int a2 = 200;
int a3 = 0x123ABC;
int a4 = -0x321dba;
int a5 = 0456;
int a6 = -0654;
int a7 = 1.2;
int a8 = 6.7E4;
```

```
9.  int a9 = 8.9E-3;
10. String a10 = "test";
11. char a11 = 'h';
```

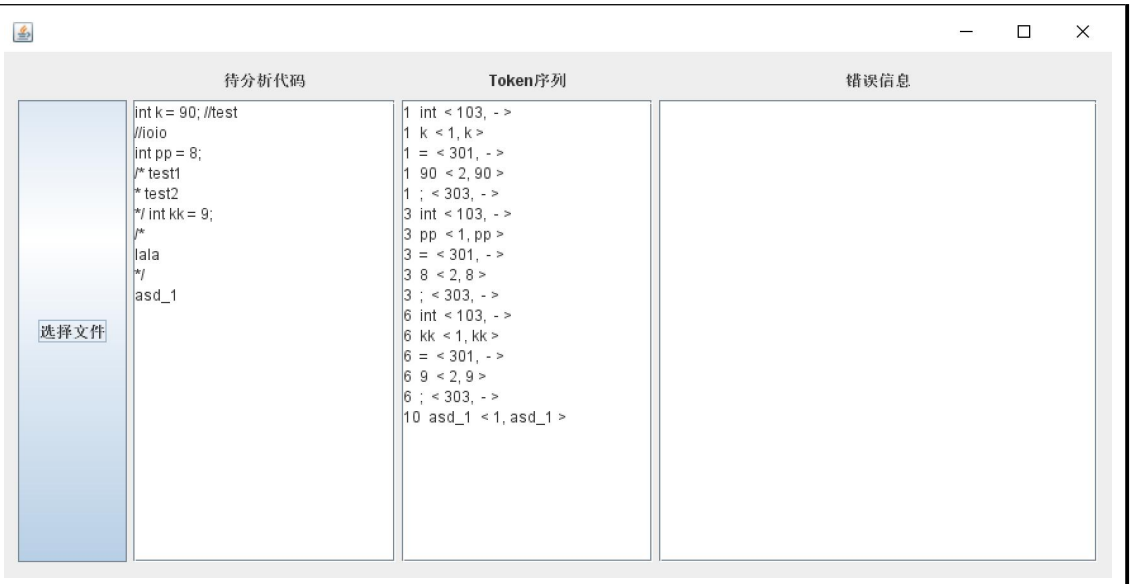
测试结果如下：



（2.5）测试注释：annotation.txt

```
1.  int k = 90; //test
2.  //ioio
3.  int pp = 8;
4.  /* test1
5.  * test2
6.  */ int kk = 9;
7.  /*
8.  lala
9.  */
10. asd_1
```

测试结果如下：

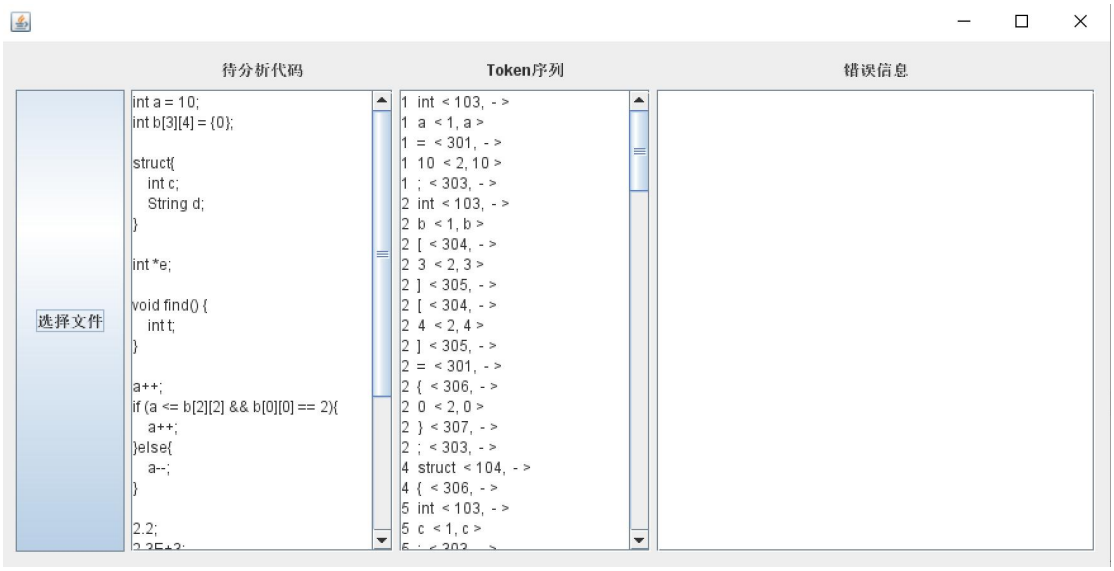


（2.6）测试实验课堂验收要求：test.txt

```
int a = 10;
int b[3][4] = {0};

struct{
    int c;
    String d;
}

int *e;

void find() {
    int t;
}

a++;
if (a <= b[2][2] && b[0][0] == 2){
    a++;
}else{
    a--;
}

2.2;
2.3E+3;
075;
0x123abc;

int f = b[0][1];
```

```
28. b[2][1] =20;
29.
30. while(a > 0){
31.     a += 1;
32. }
```

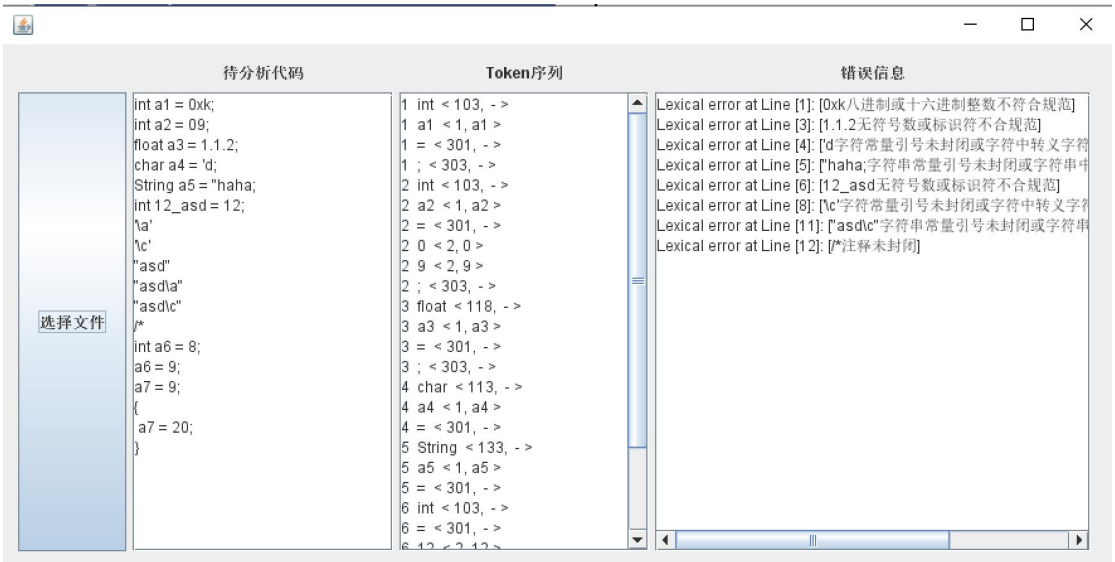
测试结果如下：



（3） 输出针对此测试程序对应的词法错误报告；
测试错误信息：errortest.txt

```
1.  int a1 = 0xk;
2.  int a2 = 09;
3.  float a3 = 1.1.2;
4.  char a4 = 'd;
5.  String a5 = "haha;
6.  int 12_asd = 12;
7.  '\a'
8.  '\c'
9.  "asd"
10. "asd\a"
11. "asd\c"
12. /*
13. int a6 = 8;
14. a6 = 9;
15. a7 = 9;
16. {
17.  a7 = 20;
18. }
```

测试结果如下：



（4） 对实验结果进行分析。

首先分析各测试文件，对于标识符、关键字、运算符、界符、常数、注释的正确词法分析：可以看到，识别了标识符、整型、浮点型、布尔型、if、else、do、while、过程声明和调用的关键字、算术运算符、关系运算符、逻辑运算、赋值界符、句子结尾界符、数组表示界符、浮点数表示的界符、无符号整数（包含八进制和十六进制）、浮点数（含科学计数法）、字符串常数、字符常数、两种风格注释（直接跳过，不输出到 Token 序列中），并进行输出。

然后分析错误测试文件：对于八进制或十六进制整数不符合规范、无符号数或标识符不合规范、字符常量引号未封闭或字符中转义字符后的字符不合法、字符串常量引号未封闭或字符串中转义字符后的字符不合法、注释未封闭等错误都进行了符合规范 Lexical error at Line [行号]: [说明文字].的错误信息输出。

指导教师评语：

日期：

附录：

种别码对照表 1：

标识符	1
整型常量（包括八进制，十六进制）	2
浮点型常量	3
科学计数法	4
字符常量	5
字符串常量	6
布尔型常量	7

种别码对照表 2（关键字）：

auto	101
double	102
int	103
struct	104
break	105
else	106
long	107
switch	108
case	109
enum	110
register	111
typedef	112
char	113
extern	114
return	115
union	116
const	117
float	118
short	119
unsigned	120
continue	121
for	122
signed	123
void	124
default	125
goto	126
sizeof	127
volatile	128
do	129
if	130
while	131
static	132

String	133

种别码对照表 3（运算符）：

+	201
-	202
*	203
/	204
%	205
++	206
--	207
<	208
<=	209
>	210
>=	211
==	212
!=	213
&&	214
||	215
!	216
~	217
&	218
|	219
^	220
>>	221
<<	222
+=	223
-=	224
*=	225
/=	226
%=	227
&=	228
^=	229
|=	230
<<=	231
>>=	232

种别码对照表 4（界符）

,	301
;	302
[	303
]	304
{	305
}	306

(	307
)	308
=	309