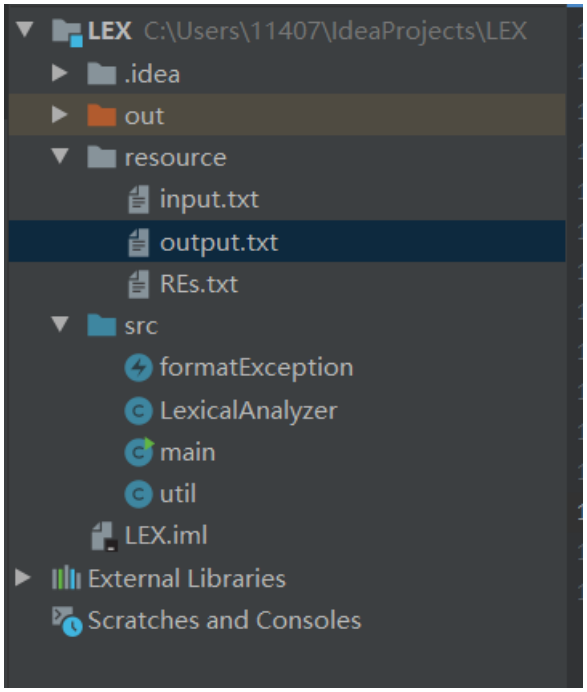
Lex 实验报告

学号：171250535 姓名：蔡明卫

1-源文件目录截图：



2-REs.txt 资源文件截图:

```
THEN then
ELSE else
PUBLIC public
STATIC static
IF if
MAIN main
WHILE while
DO do

ID letter(letter|digit)*
NUMBER digitdigit*(\.digitdigit*|ε)
MYREG1 digitletter(digitletter)*digit
MYREG2 digit(letterdigitdigit*)*letter
digit 0|1|2|3|4|5|6|7|8|9
letter a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

RELOP/LT <
RELOP/GT >
RELOP/GE >=
RELOP/NE !=
RELOP/LE <=
RELOP/EQ ==
RELOPE/E =
```

3-输入文件/流内容截图：(input.txt)

```
main if 15.2123 432
6f8d8d8d8
    7d78986698r7989d
    do <
  >
  =
    gfdjdasgsd
    as785hgq3sag
   gdusgijtruhagyiew7898574
 7asdag
 ASDAS&
```

4-输出 token 的截图：(ouput.txt)

```
MAIN null main
IF null if
NUMBER 15.2123 15.2123
NUMBER 432 432
MYREG1 6f8d8d8d8 6f8d8d8d8
MYREG2 7d78986698r7989d 7d78986698r7989d
DO null do
RELOPE LT <
RELOPE GT >
RELOPE E =
ID gfdjdasgsd gfdjdasgsd
ID as785hgq3sag as785hgq3sag
ID gdusgijtruhagyiew7898574 gdusgijtruhagyiew7898574
```

5-回答以下问题：

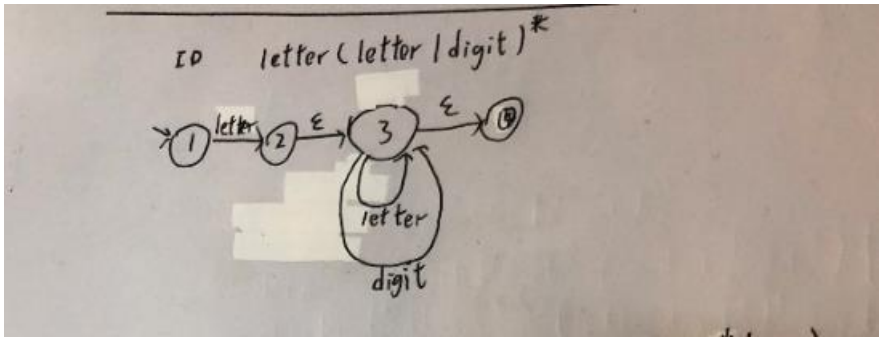
1.简要说明在实验中实现词法分析需要哪几个具体的步骤?

1-设计并规范 RE 表达式；2-画出各个表达式的 NFA 图；3-将各个 RE 表达式的 NFA 图合并；4-将合并后的 NFA 转化成 DFA；5-将 DFA 转成 DFA0；6-本词法分析器可以从 input.txt 读出各个单词（输入中以空格或 tab 分割单词），判断每个单词对应的正确 token 并输出（同时能判断部分保留字和关键字），同时 token 序列会被输出到 ouput.txt；不符合 REs 的单词会在控制台输出报错信息。

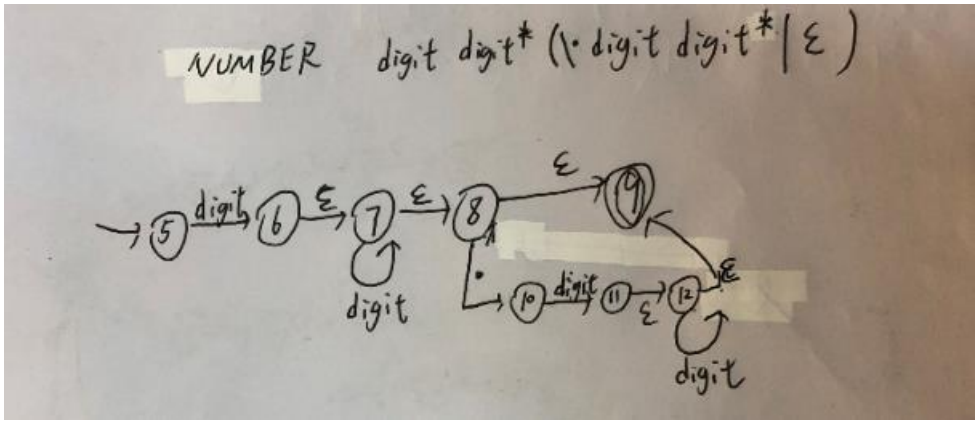
2.具体说明其中重要的转化步骤怎么实现

1-REs 转换成 NFA

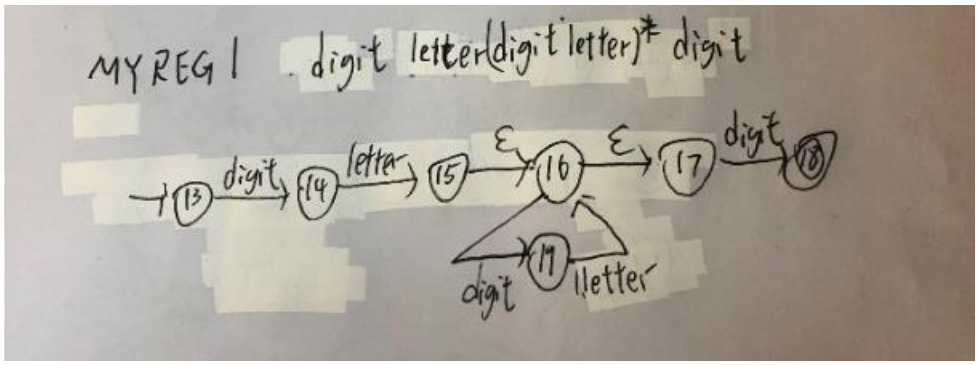
ID:



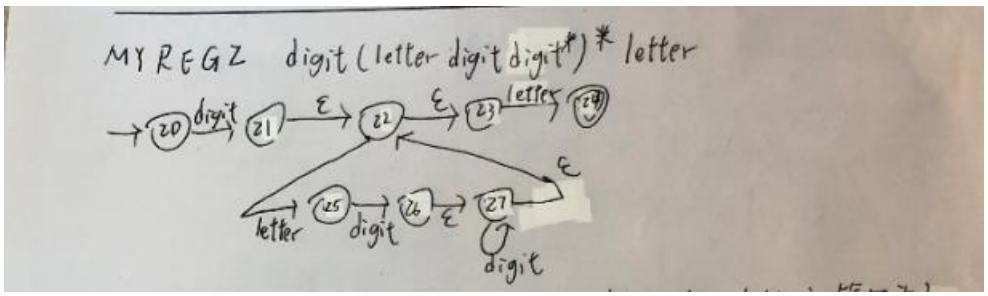
NUMBER:



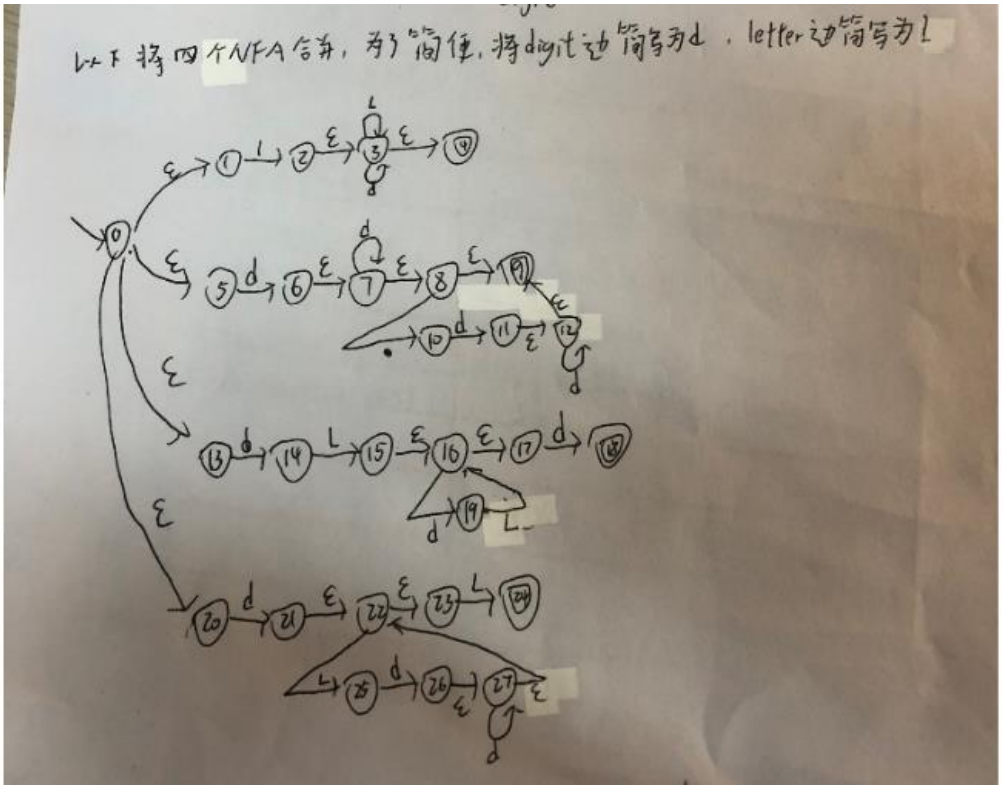
MYREG1:



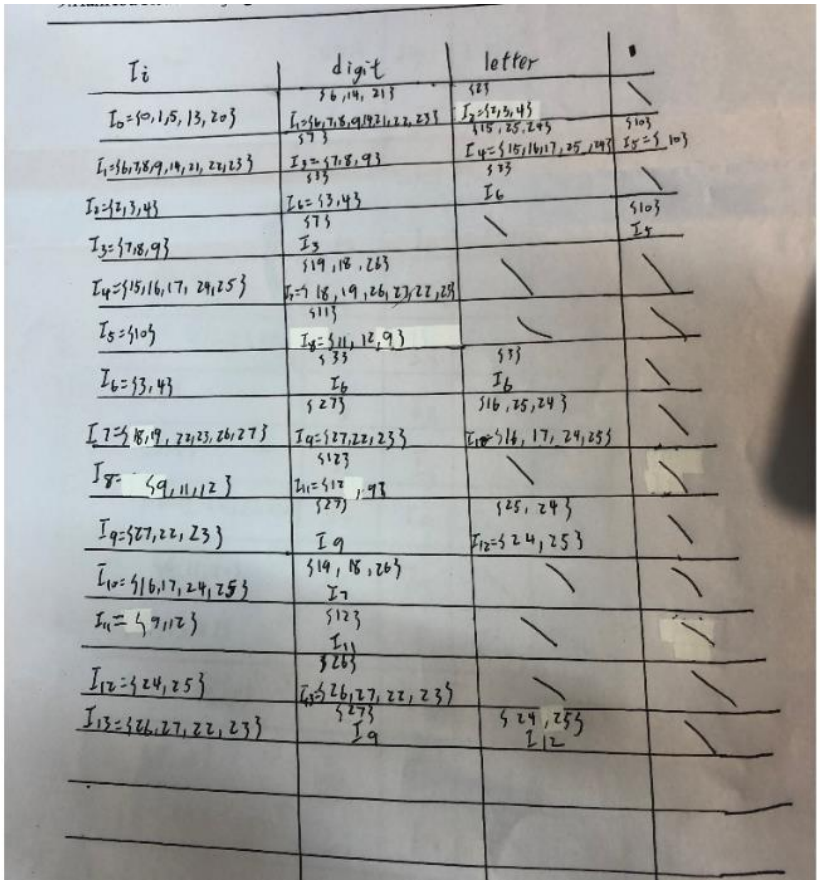
MYREG2:

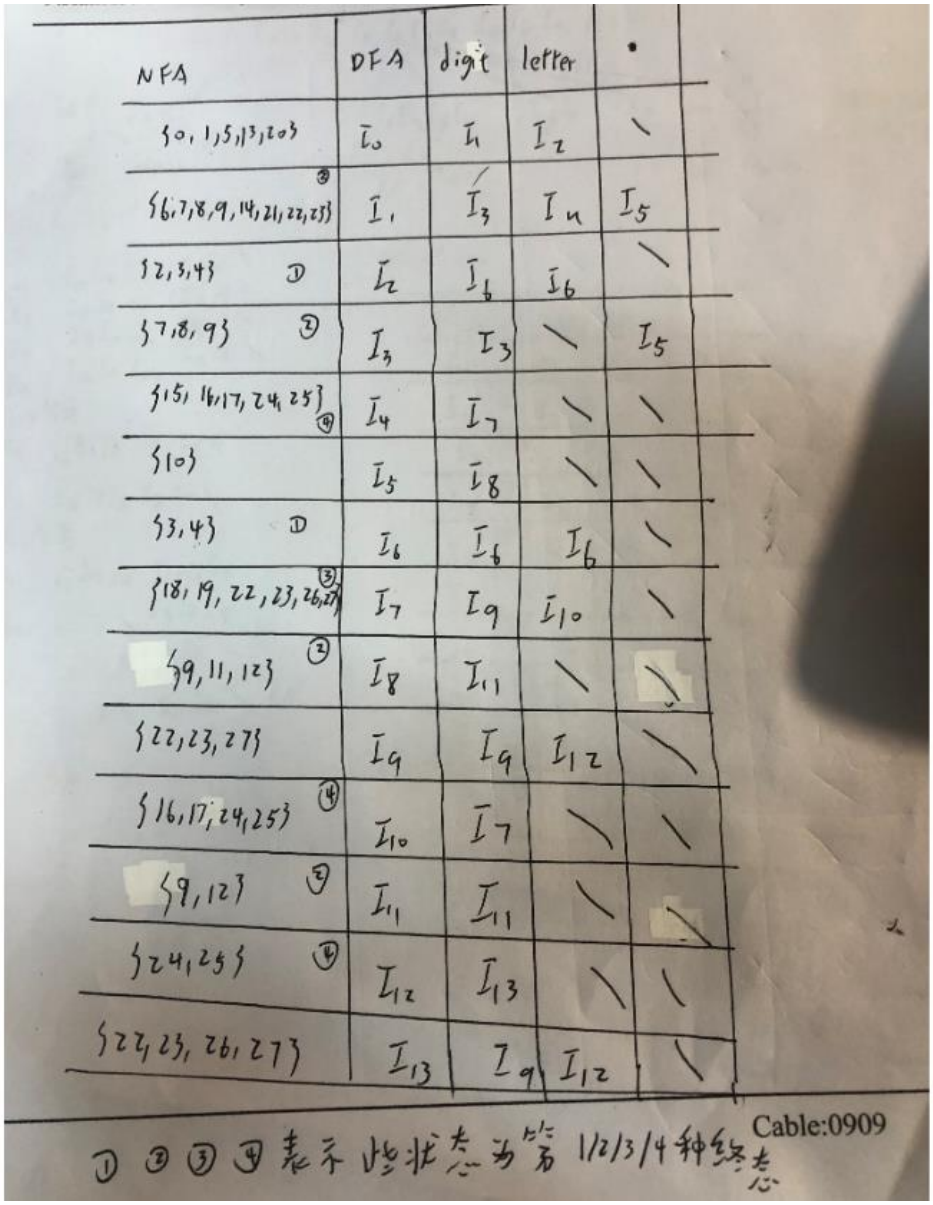


2-合并 NFA


以下将四个NFA合并，为了简便，将digit边简写为d，letter边简写为l

3-NFA 转成 DFA


Ii
digit
letter
.
I0={0,1,5,13,20}
{6,14,21}
I1={6,7,8,9,14,15,21,22,23}
{7}
I2={2,3,4}
{15,25,24}
{10}
I4={15,16,17,25,24}
I5={10}
I1={6,7,8,9,14,15,21,22,23}
{7,8,9}
{15,16,17,25,24}
{10}
I2={2,3,4}
{3,4}
I6
I3={7,8,9}
{7}
{10}
I5
{19,18,26}
I4={15,16,17,24,25}
I7={18,19,26,23,22,27}
{11}
I5={10}
I8={11,12,9}
{33}
{33}
I6={3,4}
I6
I6
{27}
{16,25,24}
I7={18,19,22,23,26,27}
I9={27,22,23}
{16,17,24,25}
{123}
I8={9,11,12}
I11
{27}
{25,24}
I9={27,22,23}
I9
I12={24,25}
{19,18,26}
I10={16,17,24,25}
I7
{123}
I11={9,11,12}
I11
{26}
I12={24,25}
I13={26,27,22,23}
{27}
{24,25}
I9
I12

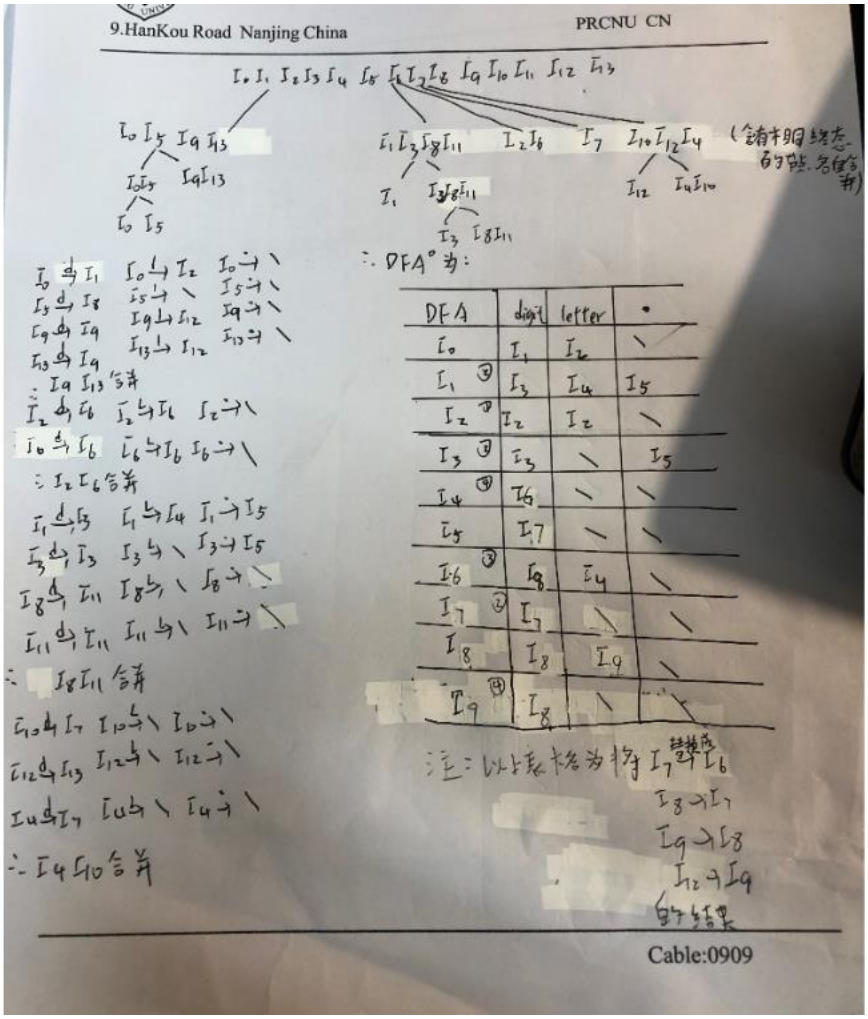


NFA	DFA	digit	letter	.
{0,1,5,13,20}	I_0	I_1	I_2	\
{6,7,8,9,14,21,22,23} ③	I_1	I_3	I_4	I_5
{2,3,4} ①	I_2	I_6	I_6	\
{7,8,9} ③	I_3	I_3	\	I_5
{15,16,17,24,25} ④	I_4	I_7	\	\
{10}	I_5	I_8	\	\
{3,4} ①	I_6	I_6	I_6	\
{18,19,22,23,26,27} ③	I_7	I_9	I_{10}	\
{9,11,12} ②	I_8	I_{11}	\	\
{22,23,27}	I_9	I_9	I_{12}	\
{16,17,24,25} ④	I_{10}	I_7	\	\
{9,12} ②	I_{11}	I_{11}	\	\
{24,25} ④	I_{12}	I_{13}	\	\
{22,23,26,27}	I_{13}	I_9	I_{12}	\

①②③④表示终状态为第1/2/3/4种终态

Cable:0909

4-DFA 转成 DFA0



5-编码：

1-从文件读取输入

```java
    private static List<String> readFile(String path) throws IOException {
        List<String> list = new ArrayList<~>();
        FileInputStream fis = new FileInputStream(path);
        InputStreamReader isr = new InputStreamReader(fis);
        BufferedReader br = new BufferedReader(isr);
        String line = "";
        while ((line = br.readLine()) != null) {
            if(line.length()==0){
                continue;
            }
            line=line.replaceAll( regex: "\\t", replacement: " ");
            line=line.trim();
            String test=line.replaceAll( regex: "\\s{1,}", replacement: " " );
            String[] list1=test.split( regex: " ");

            for(int i=0;i<list1.length;i++){
                list.add(list1[i]);
            }

        }
        br.close();
        isr.close();
        fis.close();
        return list;
    }
}
```

2-读取下一个字符的 getChar（）方法

```java
    private char getChar(){
        if(ptr<len){
            ptr+=1;
            return word.charAt(ptr-1);
        }else{
            return '\0';
        }

    }
```

3-判断读取的字符的种类的代码

```java
public class util {
    public static  boolean isdigit(char achar){
        if(achar>=48&&achar<=57){
            return true;
        }else{
            return false;
        }
    }
    public static boolean isletter(char achar){
        if(achar>=97&&achar<=122){
            return true;
        }else {
            return false;
        }
    }
    public static boolean isdot(char achar){
        if(achar=='.'){
            return true;
        }else{
            return false;
        }
    }
    public static int chartype(char achar){
        if(isdigit(achar)){
            return 0;
        }else if(isletter(achar)){
            return 1;
        }else if(isdot(achar)){
            return 2;
        }else{
            return 3;
        }
    }
}
```

util › isdigit()

4-硬编码实现的状态转移的核心代码

```
while(true){
    aChar=getChar();
    if(aChar=='\0'){
        switch (state){
            case 1:
            case 3:
            case 7:throw new formatException("NUMBER");
            case 2:throw new formatException("ID");
            case 6:throw new formatException("MYREG1");
            case 4:
            case 9:throw new formatException("MYREG2");
            default:throw new formatException("ERROR");
        }
    }
    charstate=util.chartype(aChar);
    switch (state){
        case 0:
            switch (charstate){
                case 0:state=1;break;
                case 1:state=2;break;
                case 2:
                    throw new formatException("ILLEGAL_GRAMMAR");
                case 3:
                    throw new formatException("ILLEGAL_CHAR");
            }
            break;
        case 1:
            switch (charstate){
                case 0:state=3;break;
                case 1:state=4;break;
```

5-错误处理及格式化输出

```
for(int i=0;i<strlist.size();i++){
    lexicalAnalyzer=new LexicalAnalyzer(strlist.get(i));
    try{
        lexicalAnalyzer.isSpecial();
        lexicalAnalyzer.start();
    }catch (formatException e){
        String s="";
        if(e.getMsg().split( regex: "/").length==2){
            System.out.println(e.getMsg().split( regex: "/")[0]+" "+e.getMsg().split( regex: "/")[1]+" "+strlist.get(i));
            s+=(e.getMsg().split( regex: "/")[0]+" "+e.getMsg().split( regex: "/")[1]+" "+strlist.get(i));
            res.add(s);
        }else{

            if(e.getMsg().indexOf("ILLEGAL")==-1){
                s+=e.getMsg()+" "+strlist.get(i)+" "+strlist.get(i);
                res.add(s);
                System.out.println(e.getMsg()+" "+strlist.get(i)+" "+strlist.get(i));
            }else{
                System.out.println(e.getMsg()+" "+strlist.get(i)+" "+strlist.get(i));

            }

        }
    }
}
```

6-将 token 序列写入输出文件

```java
private static void writeFile(String s,String path) {
    FileWriter fw = null;
    try {
        File f=new File(path);
        fw = new FileWriter(f, append: true);

    } catch (IOException e) {
        e.printStackTrace();
    }
    PrintWriter pw = new PrintWriter(fw);
    pw.println(s);
    pw.flush();
    try {
        fw.flush();
        pw.close();
        fw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

3.是否有 Error Handling，具体如何实现的?

使用 java 中的异常机制,对含有未识别字符的单词,不符合 REs 的单词抛出异常报错,由 main 方法输出错误信息，错误的单词不会被写入 ouput.txt 文件。

```
ILLEGAL_GRAMMAR 7asdag 7asdag
ILLEGAL_CHAR ASDAS& ASDAS&
```

4.实验中出现的问题和相应的解决办法。

（1）原本使用控制台读取输入，可是发现这样实现输入多个回车符很麻烦，而且需要通过设定标识符来判断输入是否结束，这样设定标识符就不能作为识别的输入了。所以换成从文件中读取输入。

（2）从文件中读取输入一开始使用一次性全部读取进一个 char 数组的方式，后续删除回车符，合并，去除空格很麻烦，改用逐行读取的方式。

（3）一开始未以空格分割，读到当读取下一个字符没有对应相应的转移状态时候就输出，但是这样存在二义性，public134 可能被识别成 PUBLIC 和 NUMBER，所以在输入文件中做处理，强制要求将要识别的单词间用空格分割。可是这样无法识别 a=123123 等本意就

是要拆开的成 3 个 token 的却未用空格分割的字符串。

5.对实验的评价和感觉。

本次实验通过编写调试一个词法分析程序，让我更加深入了解了从 RE 到 NFA 到 DFA 到 DFA0 的过程是如何执行的，这次作业的转换过程也是学习过程中遇到过的最复杂的一次，增长了经验和能力，遗憾的是因为编程伊始就存在的想法的偏差，后期为了避免 public134 可能被识别成 PUBLIC 和 NUMBER 的错误，我限制输入的单词间必须以空格或者 tab 分割，不能识别连在一起的 token 序列，有些遗憾，后面如果有时间我想进一步完善我的词法分析程序。