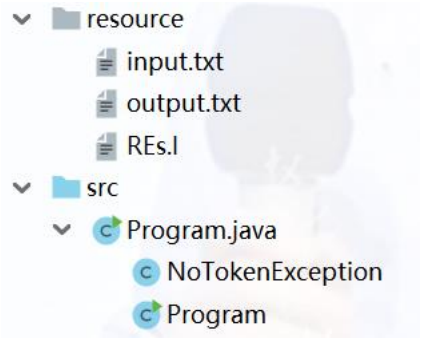
# Lex 实验报告模板

学号：171250013 姓名：常卓

## 源文件目录截图：




## REs.l 资源文件截图:

```
PUBLIC public
CLASS class
STATIC static
VOID void
STRING String
INT int
RELOP/LT <
RELOP/GT >
RELOP/ADD +
NUM [0-9]+[.[0-9]+]
ID _?[a-zA-Z]+[0-9a-zA-Z]*
LITERAL 两个"之间除"以外的任何字符
ASSIGN_OP =
BRACKET/LRO (
BRACKET/RRO )
BRACKET/LSQ [
BRACKET/RSQ ]
BRACKET/LCU {
BRACKET/RCU }
DOT .
SEMICOLON ;
```

## 输入文件/流内容截图：

```
public class Input{
    public static void main_(String[] args){
        int n=100;
        for(int i=0;i<n;i=i+1) {
            System.out.println("Hello World!");
        }
    }
}
```

输出 TOKEN 的截图：

```
<PUBLIC>
<CLASS>
<ID, Input>
<BRACKET, LCU>
<PUBLIC>
<STATIC>
<VOID>
<ID, main>
<BRACKET, LRO>
<STRING>
<BRACKET, LSQ>
<BRACKET, RSQ>
<ID, args>
<BRACKET, RRO>
<BRACKET, LCU>
<INT>
<ID, n>
<ASSIGN_OP>
<NUM, 100>
<SEMICOLON>
<ID, for>
<BRACKET, LRO>
<INT>
<ID, i>
```

```
<ASSIGN_OP>
<NUM, 0>
<SEMICOLON>
<ID, i>
<RELOP, LT>
<ID, n>
<SEMICOLON>
<ID, i>
<ASSIGN_OP>
<ID, i>
<RELOP, ADD>
<NUM, 1>
<BRACKET, RRO>
<BRACKET, LCU>
<ID, System>
<DOT>
<ID, out>
<DOT>
<ID, println>
<BRACKET, LRO>
<LITERAL, Hello World!>
<BRACKET, RRO>
<SEMICOLON>
```

```
<BRACKET, RCU>
<BRACKET, RCU>
<BRACKET, RCU>
```

---

*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？

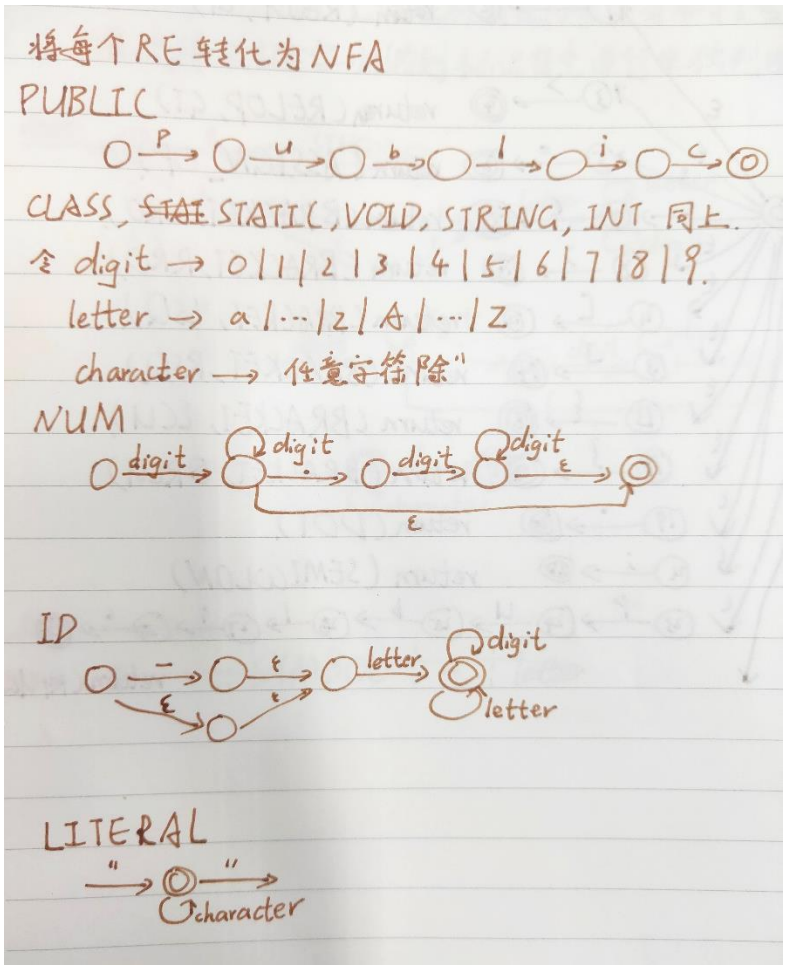
首先定义出 TOKEN，然后 1.用正则表达式表示，2.转为分别的 NFA，3.构造整体 NFA，4.将 NFA 转化为 DFA，5.将 DFA 转化为 DFA°，6.画出 FA 转换图，7.根据 FA 转换图进行编码，读取 input.txt 并输出 output.txt，即 TOKEN 序列

## 2.具体说明其中重要的转化步骤怎么实现

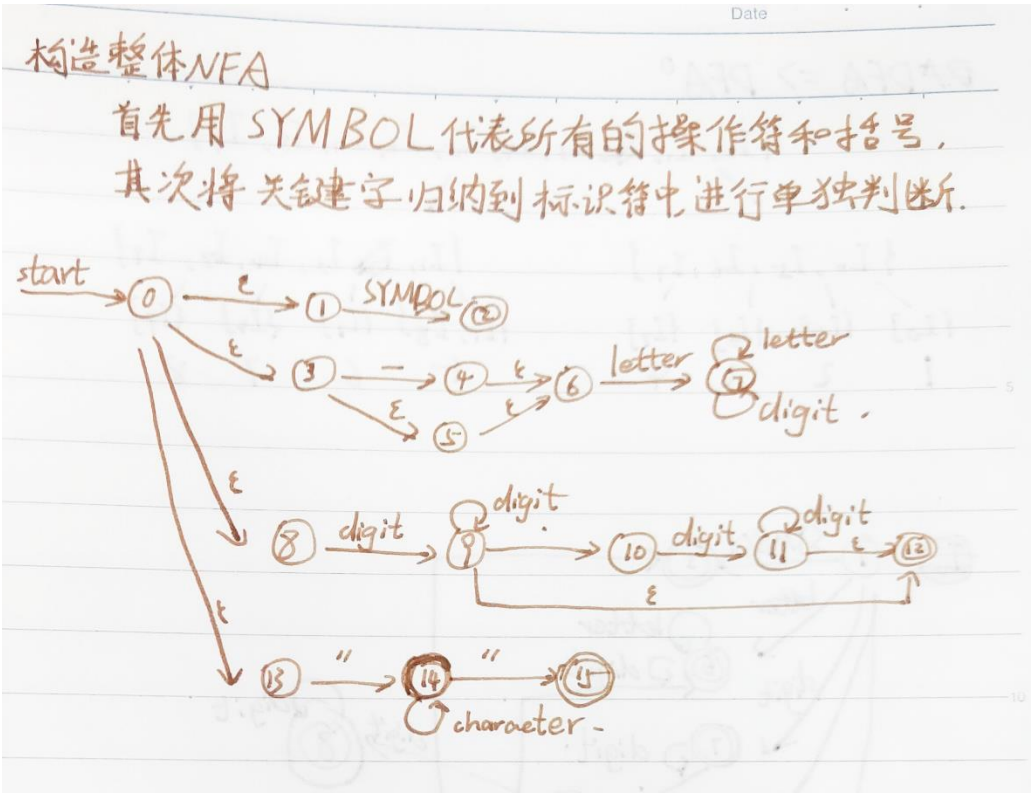
1. 定义 TOKEN 并规范 RE 表达式，LITERAL 由于比较特殊所以没有在此步给出正则表达式

```
PUBLIC public
CLASS class
STATIC static
VOID void
STRING String
INT int
RELOP/LT <
RELOP/GT >
RELOP/ADD +
NUM [0-9]+[.[0-9]+]
ID _?[a-zA-Z]+[0-9a-zA-Z]*
LITERAL 两个"之间除"以外的任何字符
ASSIGN_OP =
BRACKET/LRO (
BRACKET/RRO )
BRACKET/LSQ [
BRACKET/RSQ ]
BRACKET/LCU {
BRACKET/RCU }
DOT .
SEMICOLON ;
```

2. 将较为复杂的几个 RE 转换为 NFA


将每个RE转化为NFA
PUBLIC
P
u
b
l
i
c
CLASS, ~~STAT~~ STATIC, VOID, STRING, INT 同上.
digit → 0|1|2|3|4|5|6|7|8|9.
letter → a|…|z|A|…|Z
character → 任意字符除"
NUM
digit
digit
digit
digit
ε
ε
ID
letter
digit
letter
LITERAL
"
"
character

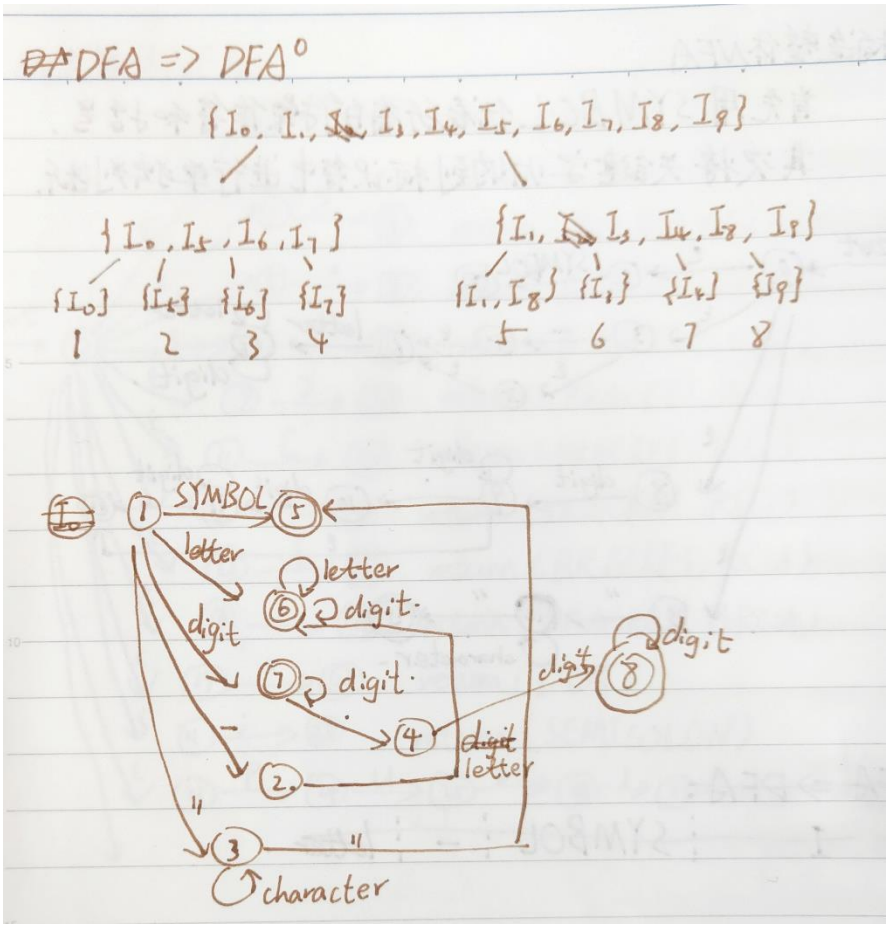
3. 构造整体 NFA




4. 将 NFA 转化为 DFA

| I | SYMBOL | letter | digit | character | _ | . | " |
|---|---|---|---|---|---|---|---|
| I0={0,1,3,5,6,8,13} | I1={2} | I3={7} | I4={9,12} | | I5={4,6} | | I6={14} |
| I1={2} | | | | | | | |
| I3={7} | | I3={7} | I3={7} | | | | |
| I4={9,12} | | | I4={9,12} | | | I7={10} | |
| I5={4,6} | | I3={7} | | | | | |
| I6={14} | | | | I6={14} | | | I8={15} |
| I7={10} | | | I9={11,12} | | | | |
| I8={15} | | | | | | | |
| I9={11,12} | | | I9={11,12} | | | | |

5. 将 DFA 转化为 DFA°并画出 FA 图




6. 进行编程

## 3.是否有 ERROR HANDLING，具体如何实现的？

有

对无法处理的字符自定义了异常 NoTokenException，在出现意外字符时（查表结果为空或不在表中）会报错并输出错误字符串。

如图，main_格式错误

```
public static void main_(String[] args){
```

报错信息如下


## 4.实验中出现的问题和相应的解决办法。

实验中出现的问题主要是对于 LITERAL（两个“之间的字符串）的处理，由于 LITERAL 是两个”之间但又不包含“的字符串，所以不能用 RE 表示，在转换为 NFA 的时候也无法画出合适的转换图，最后通过编码的时候进行特殊处理得到了解决。

编码时通过检测到“时把前面的指针后移一位，再次检测到“时把后面的指针前移一位来对“进行处理。

## 5.对实验的评价和感觉。（简短一些就好）

由于对“的特殊处理所以选择了硬编码的处理方式，使得需要在每种状态下都需要进行错误处理，比较费时费力。对于字符的判断也比较粗暴，没有很好的灵活性。

在进行 DFA 优化的过程中只得到了很小的优化，并且优化之后引入了一定的逻辑复杂度（多种结束状态变成一种，需要对不同的结束状态进行分辨），感觉在 DFA 规模比较小的情况下并不是必要的举措。