|  |
| --- |
| 南京大学软件学院 |
| 编译原理实验报告 | |
|  |

|  |
| --- |
| 张云飞 141250197  2016-10-29 |

## 实验题目

词法分析器的代码实现

## 实验内容

Generating programs based on your own Lex

1. Design your own Lex

(ⅰ)Define .l file

(ⅱ)Programming your own Lex

1. Generating programs based on your own Lex

## 实验步骤

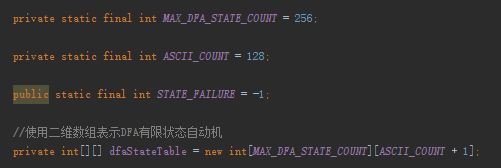
1. 定义 .l文件

新建rule.l文件，文件内容分为两部分，用%%%分隔。第一部分为保留字，运算符和分界符的定义，第二部分定义要识别的正则表达式，名称和值用：分开，这意味着正则表示式中不支持使用： 。目前正则表达式只支持[],\*和字符的定义，不支持转义字符以及+,?等其他正则表达式符号。关于rule.l文件的修改在后面补充说明中会介绍。

1. 用代码实现Lex
2. 正则表达式转为NFA
3. 处理单个正则表达式。建立REHander类负责读取rule.l文件，获得正则表达式的字符串后，使用Thompson构造法转换正则表达式，每个状态由一个NFANode节点表示，并在内部保存到达下一个状态的边和引用。读取正则表达式字符时，如果为普通字符就直接新建两个状态和一条连接边，将边赋值为该字符值，如果为[A-Za-z]这样的形式，就一直读取到]结束，将两个状态的连接边设置为字符集，如果遇到（,就先对括号内的字符串进行构造，构造完成后得到构造好的首尾状态继续处理下一个字符。
4. 处理多个正则表达式。处理完第一个正则表达式后，再处理第二个正则表达式，然后将两个正则表达式用一个开始状态和两条ε边连接起来，合并为一个NFA，之后继续处理新的正则表达式然后再和之前的NFA合并，以此类推得到所有正则表达式合并后的NFA，NFAConstructor类是正则表达式构造NFA的主要处理部分。
5. NFA转DFA

主要思想：通过ε闭包构造NFA集合，得到DFA。

为了表示DFA状态之间的转换，建立一张二维表：



首先得到初始状态下的ε-closure，然后计算从不同字符边可以到达的NFA的ε-closure，如果该闭包不存在，就新建一个DFA状态，如果通过某个字符边不能得到NFA状态，就将该位置的值设置为STATE\_FAILURE = -1。这样经过有限次的循环，就可以得到DFA的所有状态和转换关系了。

1. 处理输入字符串生成Token序列

将要处理的字符串放在source.txt文件中，由InputStr类读入后，先由DFA进行匹配，如果匹配成功，再判断该字符串是否为关键字字，如果是则新建一个关键字类型的Token保存，否则按照正则表达式的名称新建一个Token保存。如果DFA匹配不成功，再进行分界符和运算符的匹配，匹配成功后保存对应类型的Token，否则忽略该字符继续处理下一个字符。

## 补充说明

1. 没有对DFA进行最小化操作，主要是在当前的rule.l文件下定义的id和num两个正则表达式生成的DFA如果最小化后只剩一个接受状态，然后就不能判断匹配成功的字符串是id还是num了。这里也可能是我对题目的理解有问题，但由于时间比较紧迫，就没有再做处理了。
2. 关于输入和输出

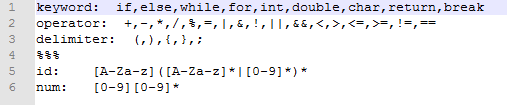
目前对输入异常的处理有所欠缺，处理的输入字符串是source.txt文件中的内容，按照rule.l中设定好的规则输入一般不会出现问题。rule.l文件中已经定义好了当前的规则，具体内容可以查看rule.l文件。如果要修改rule.l文件中的内容，不能随意添加新行和： ，否则输入可能不能正常处理，主要还是时间紧迫来不及完善，请谅解。前面三行可以在后面添加新的内容或者删除一些内容，后边正则表达式的定义可以添加新的正则表达式，格式为

(名称：正则表达式),但是不能添加空行。

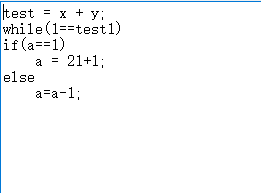
输出在控制台，首先会输出定义的正则表达式，然后是由正则表达式构造的NFA，接下来输出的是NFA转化为的DFA状态连接表：Dfa:0{15,1,21}表示DFA的0状态，里面包含NFA的状态有15,1,21。-0->(1)表示从字符0到达DFA的1状态。最后输出的是根据输入得到的Token序列。

Token格式为：序列中如果匹配的是正则表达式，显示格式为 <正则表达式名称：匹配成功的字符串>，如果匹配为关键字或者运算符的直接将对应关键字或运算符放在<>中。

初步规则如下：



样例输入如下：



输出的Token序列如下：

