|  |  |
| --- | --- |
| **编译原理综合实训 实验报告** | |
|  | |
| 图片包含 标牌  已生成极高可信度的说明 | |
|  | |
|  | |
|  | |
| **姓名** | 张健浩 |
| **班级** | 软件72班 |
| **学号** | 2174110113 |
| **电话** | 13289392873 |
| **Email** | 1439347899@qq.com |
| **日期** | 2020-09-17 |

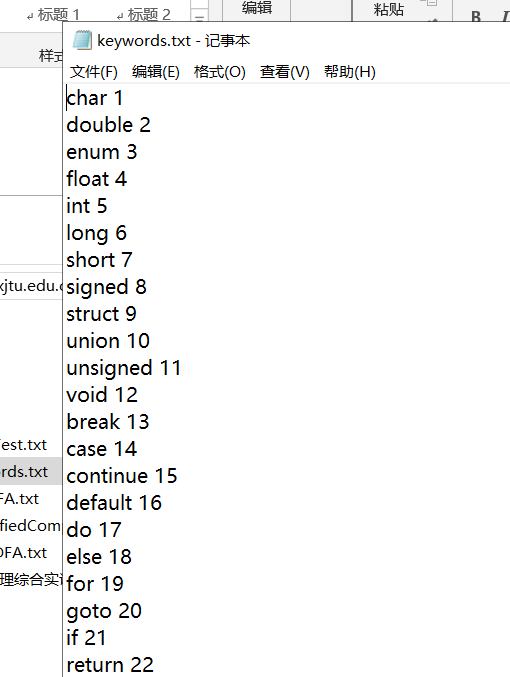
基于DFA的词法分析器设计

**词法分析器的功能和输出形式：**

根据词法分析器通用设计样例，本词法分析器是基于Java编写的对简单C语言源程序的词法分析，讲输入源程序转换为单词符号输出。其中单词符号分为五种。

1. 关键字，此部分为C语言中保留字，即具有特殊意义的标识符。在本词法分析器中每一个关键字都占有一个单词种别，标号为1~37。
2. 标识符，此部分为用户在程序中自定义的名称，包含变量名、方法名等。在此词法分析器中所有标识符占有一个单词种别，标号为38。
3. 常数，一般有整型、实型、布尔型等，本词法分析器识别整型与实型，每种数据类型占有一个单词种别，整型为39，实型为40。
4. 运算符，包含+、-、\*、/，每种占有一个单词种别，分别为42、43、44、45。
5. 界符，括号、分号、逗号等，全体占有一个单词种别，标号为41。

对此，对程序中所有的关键词等做出了表示，其包含了C语言中基本的保留词。

程序保留关键字以及其对应标号：

具体程序设计

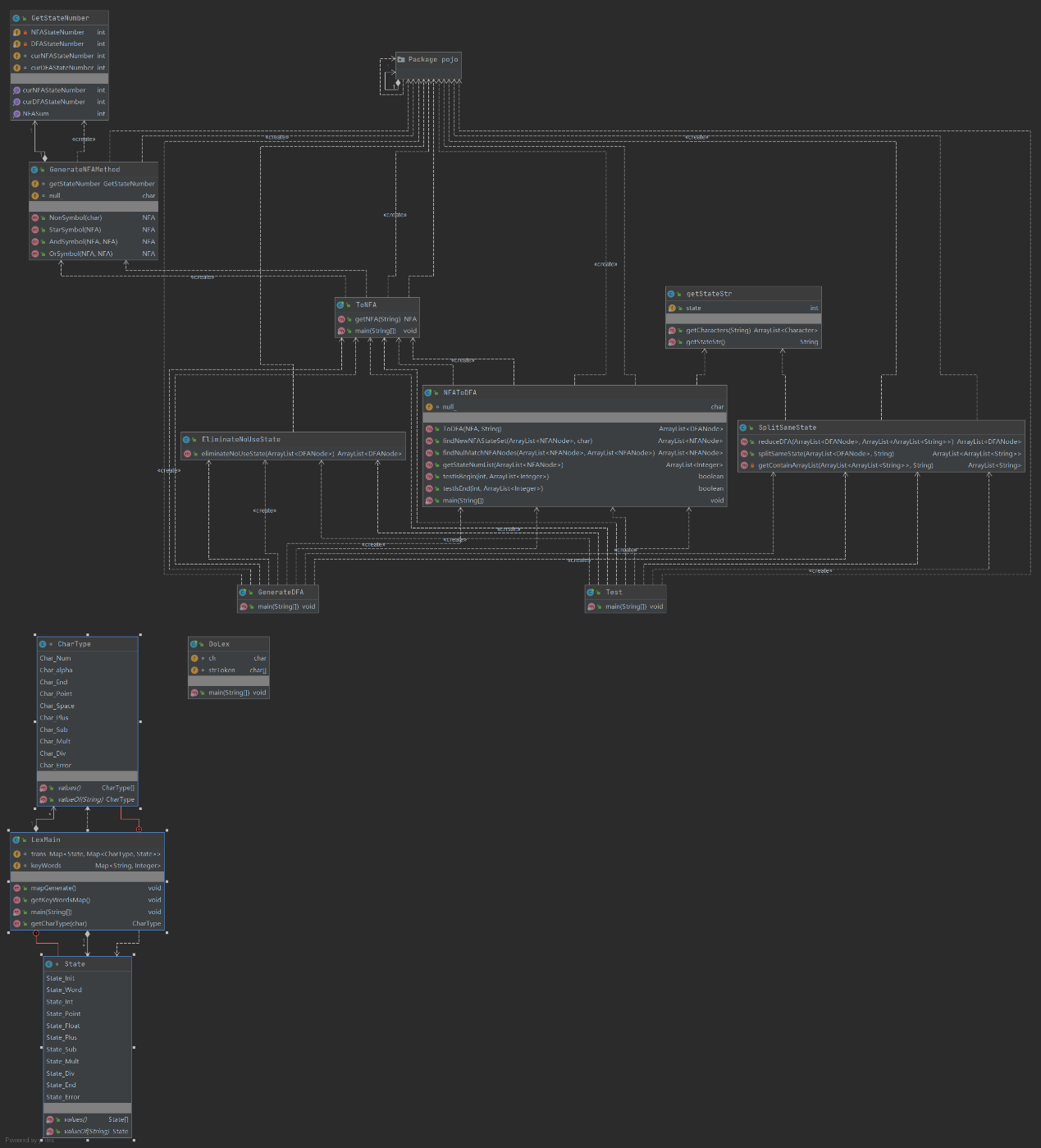
本程序中主要包含：

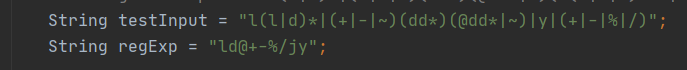
正规式到NFA的转换

NFA到DFA的转换

DFA的无效状态删除以及等价状态化简

源程序文件的读取以及处理

主要类结构图如下：

正规式设计：本程序中识别输入的正规式为：

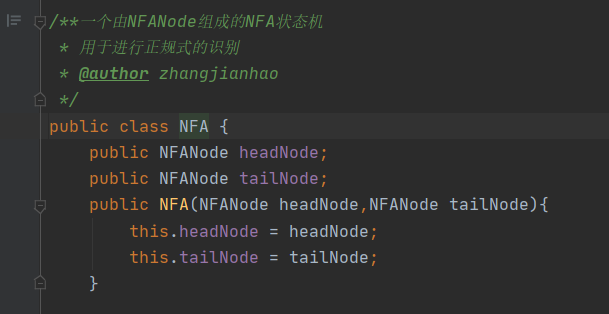
testInput为所识别的正规式，而regExp为只所有识别符号的集合，其中：

l为字母以及下划线，d为0到9的数字，@为程序中的小数点（为处理方便），+-\*/y为运算符，j为界符。

NFA的设计，对读入的正规式识别为NFA栈以及符号栈，算法根据符号栈中元素对NFA栈进行操作，在一个正确的正规式读入处理完成之后，NFA栈中会只剩余一个NFA，而符号栈会转换为空。

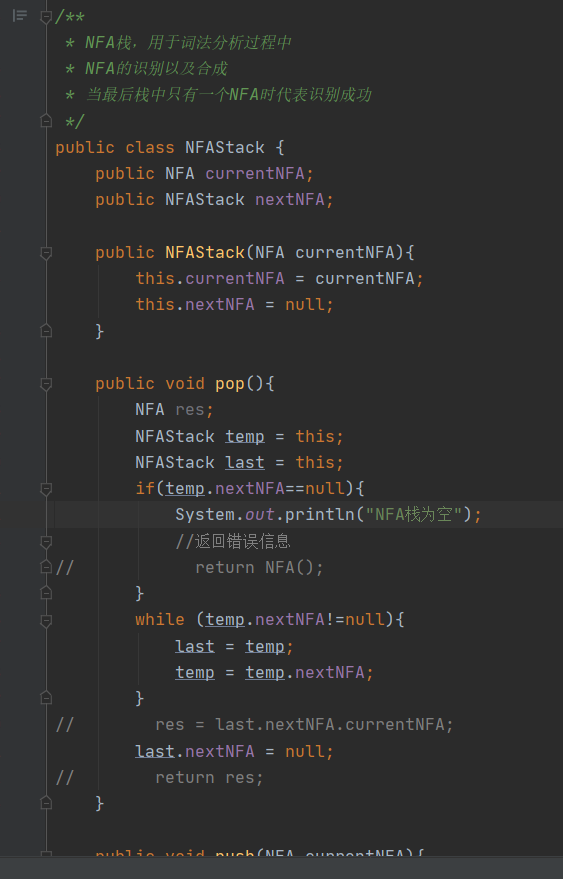
一个NFA有两个NFANode构成，分别为头结点以及尾结点，在这之中还存在着其他的NFANode，每个NFANode都表明了自己的状态标志，对于下一个输入所跳转的NFANode。

NFANode代码：



NFA代码：

NFAStack代码：



对于不同的符号读入，回对NFAStack中的NFA进行不同的连接操作：

1、 当遇到左括号’(‘时：直接压入栈中即可。

2、 当遇到右括号')'时：依次弹出符号栈中的符号直到遇到'('为止。依次弹出NFA栈顶的两个元素，进行相应的符号操作后合成一个新的NFA并压入栈。

3、当遇到或操作'|'时：此操作符的优先级最低，在压入栈时需要对符号栈中'('以上的符号进行判断，对于优先级高于或操作的连接操作需要将其先弹出后进行连接操作，直到栈中不存在连接操作后再将'|'压入符号栈中。

4、当遇到闭包操作'\*'时：此操作符的优先级最高，无须将其压入符号栈中，直接将NFA栈中的栈顶NFA弹出栈后进行闭包操作后再将新的NFA压入NFA栈。

5、当遇到隐含的连接操作'.'时：该操作符是隐含在正规式中的 ，如：ab，a(b|c)\*。因此在扫描过程中，需要对是否添加连接符进行判断。其有以下三种情况：当遇到非运算符时，需要对其后面的符号进行判断，若遇到左括号或非运算符时，则需要往符号栈中添加连接符'.'；当遇到闭包运算符'\*'时，需要判断其右边的符号，若非'|'和')'则需要在符号栈中天年假连接符'\*'；当遇到右括号')'时需要对其右边的符号进行判断，若遇到'('或非运算字符时需要加入连接符'.'。

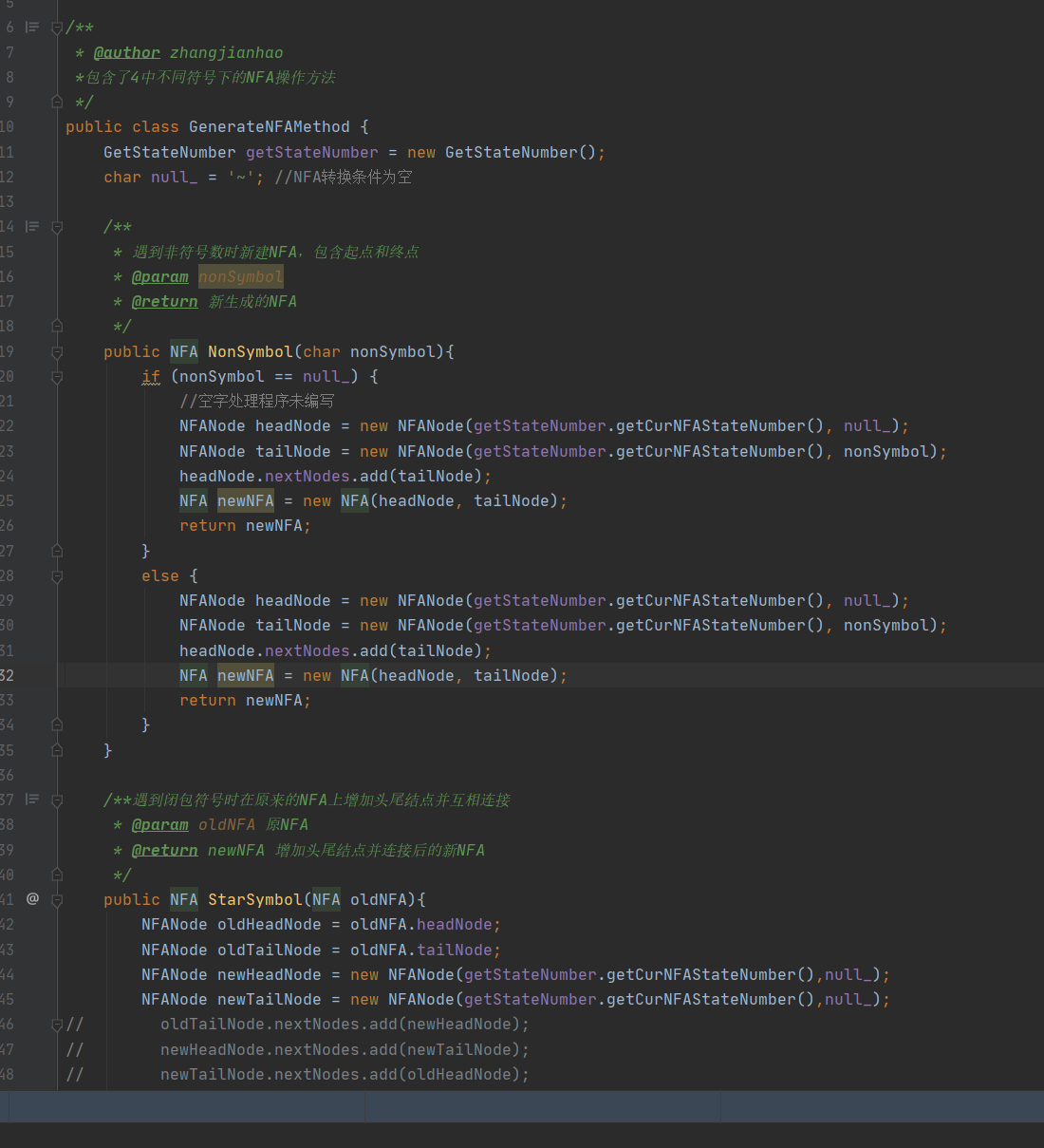
对于操作符，有以下操作：

当遇到非符号数时新建一个NFA，其中包含起点和终点。

当遇到符号数'\*'时增加头尾节点并连接。

当遇到符号数为'.'即表示连接操作时，前一个NFA的尾节点与后一个NFA的头节点相连,需要增加头尾节点重新组成一个NFA。

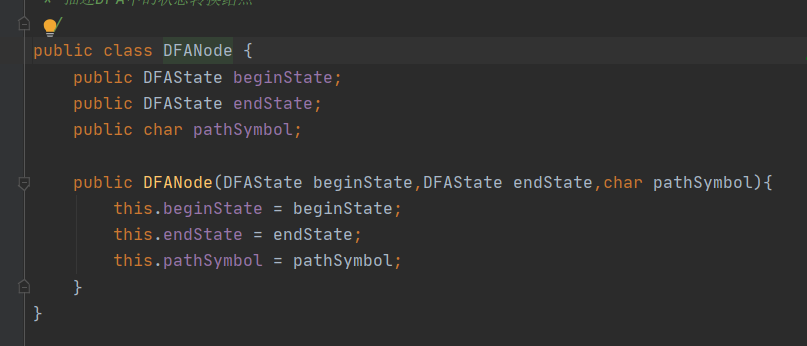
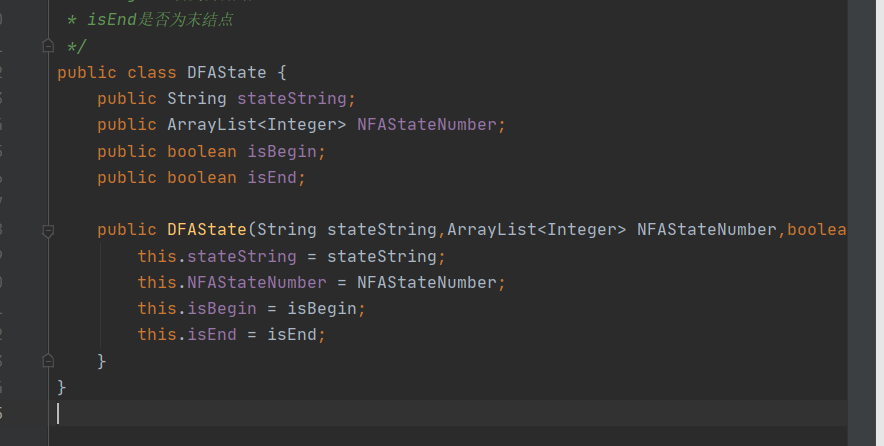
当遇到符号数为'|'时添加头尾节点进行或操作。

生成NFA类：

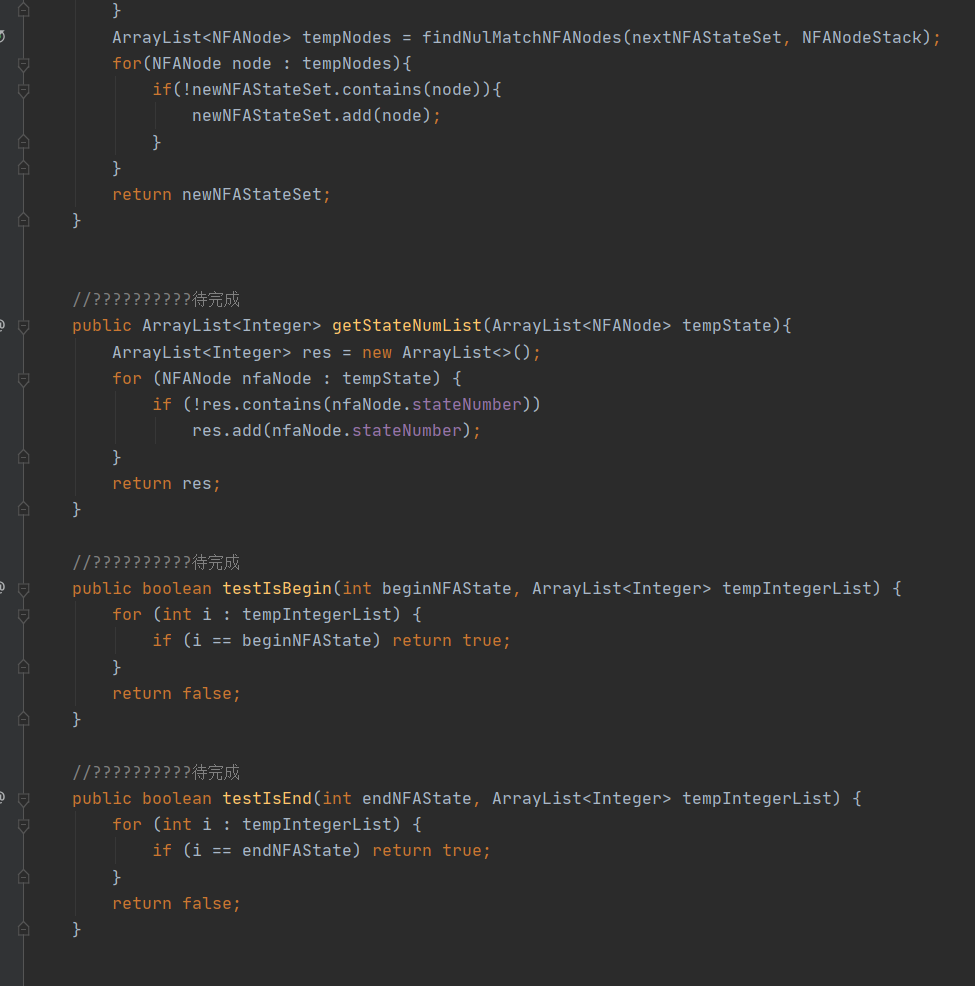
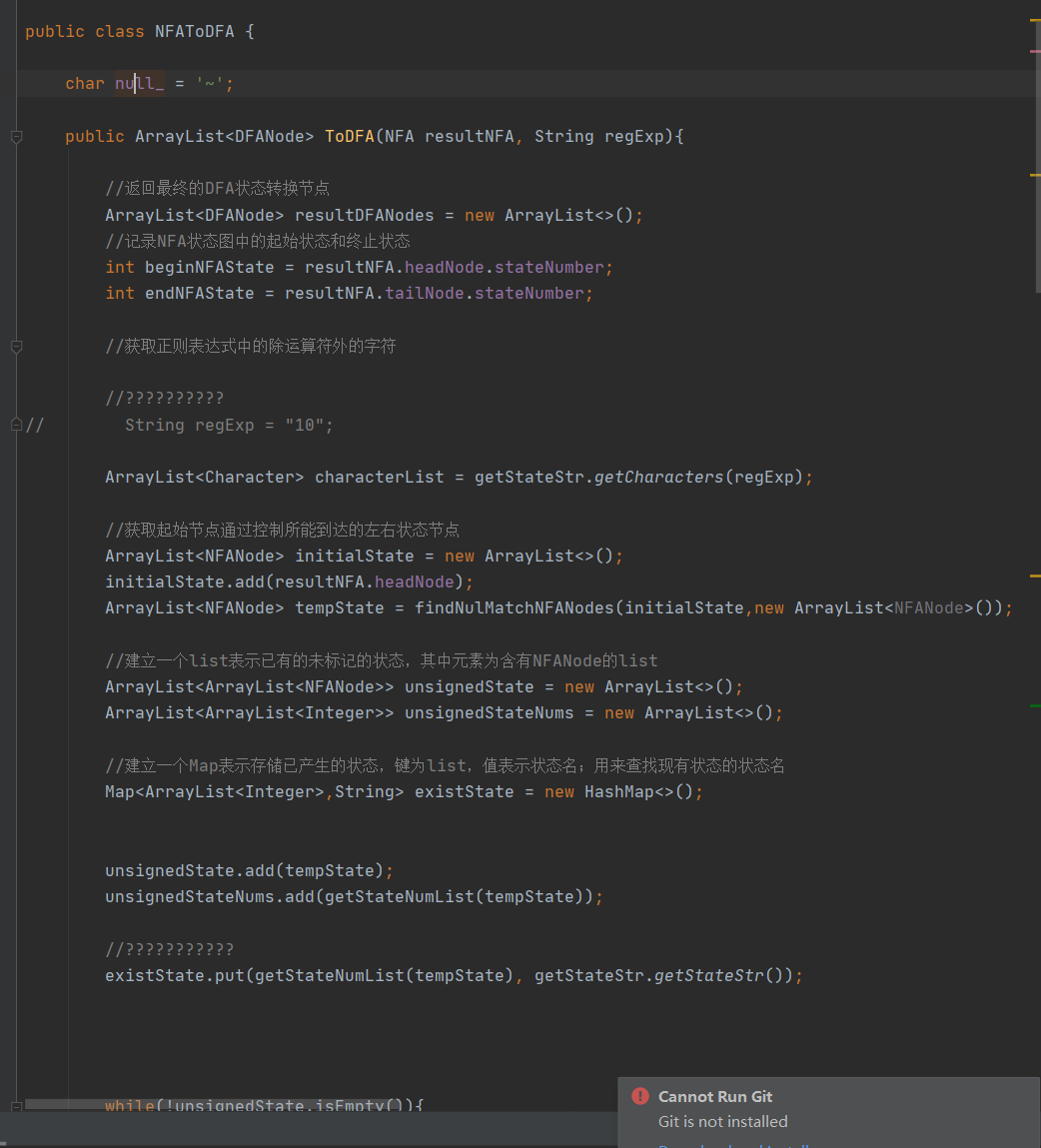
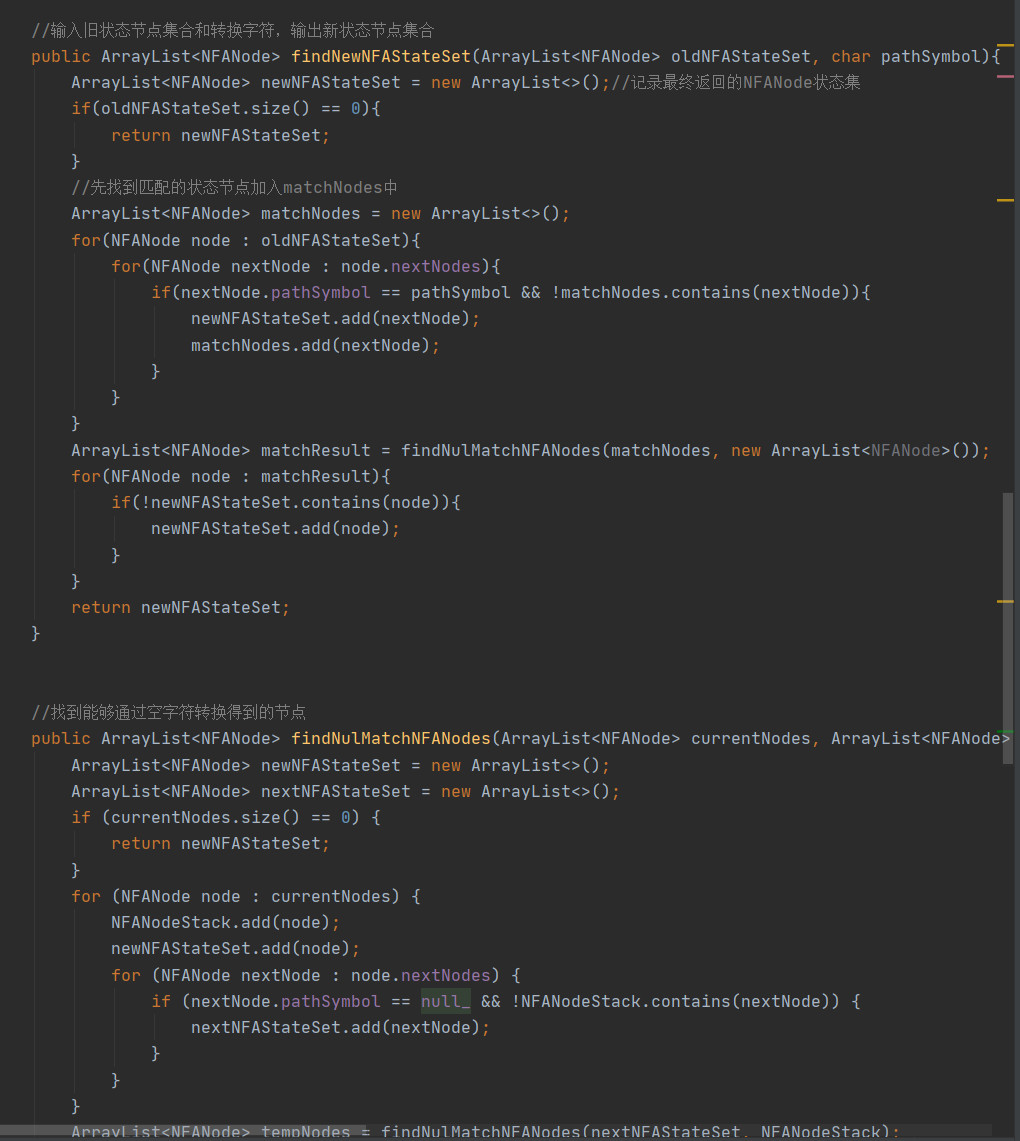
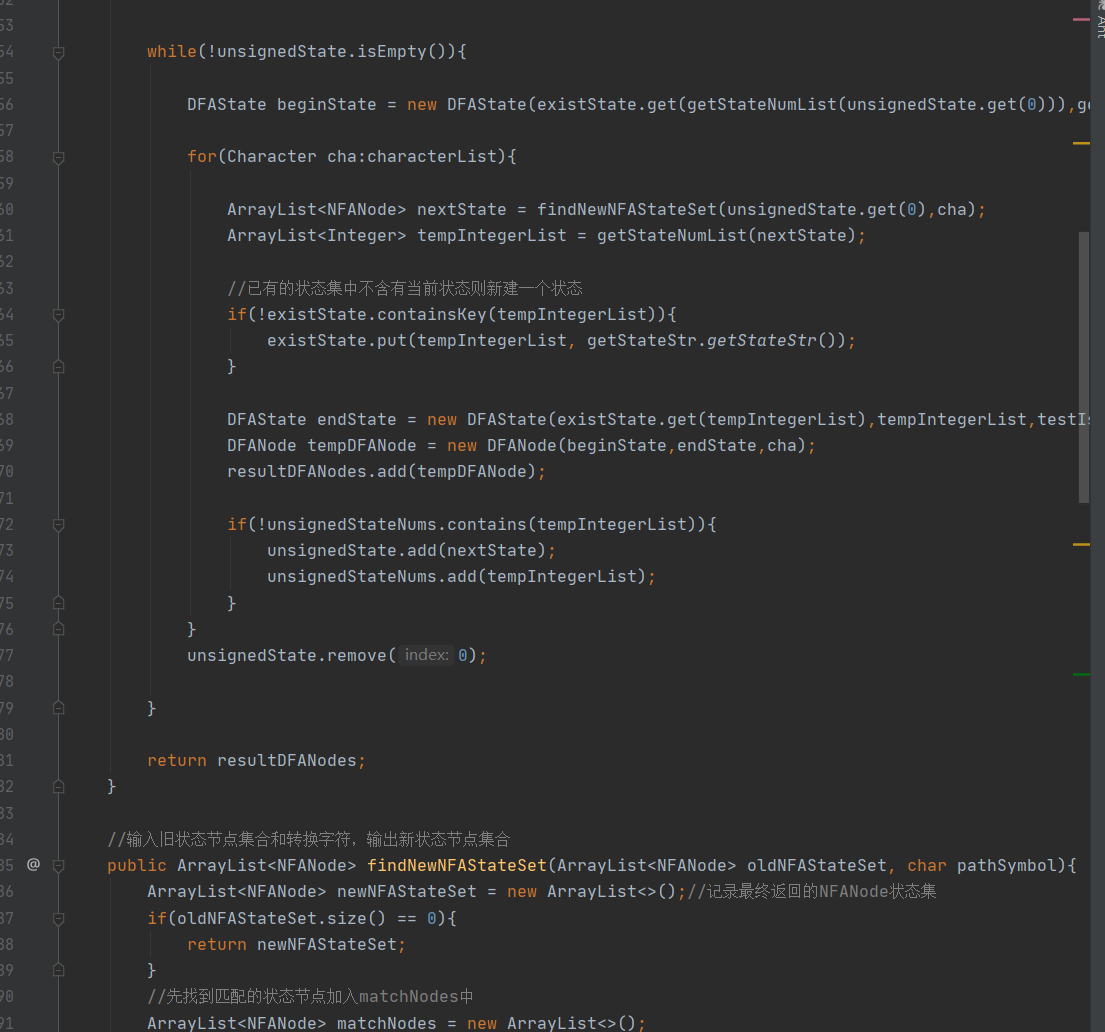
NFA转为DFA

以上程序在输入正确的正规式后会对正规式进行处理，得到等价的NFA，接下来对生成的NFA进行处理转换为等价的DFA。

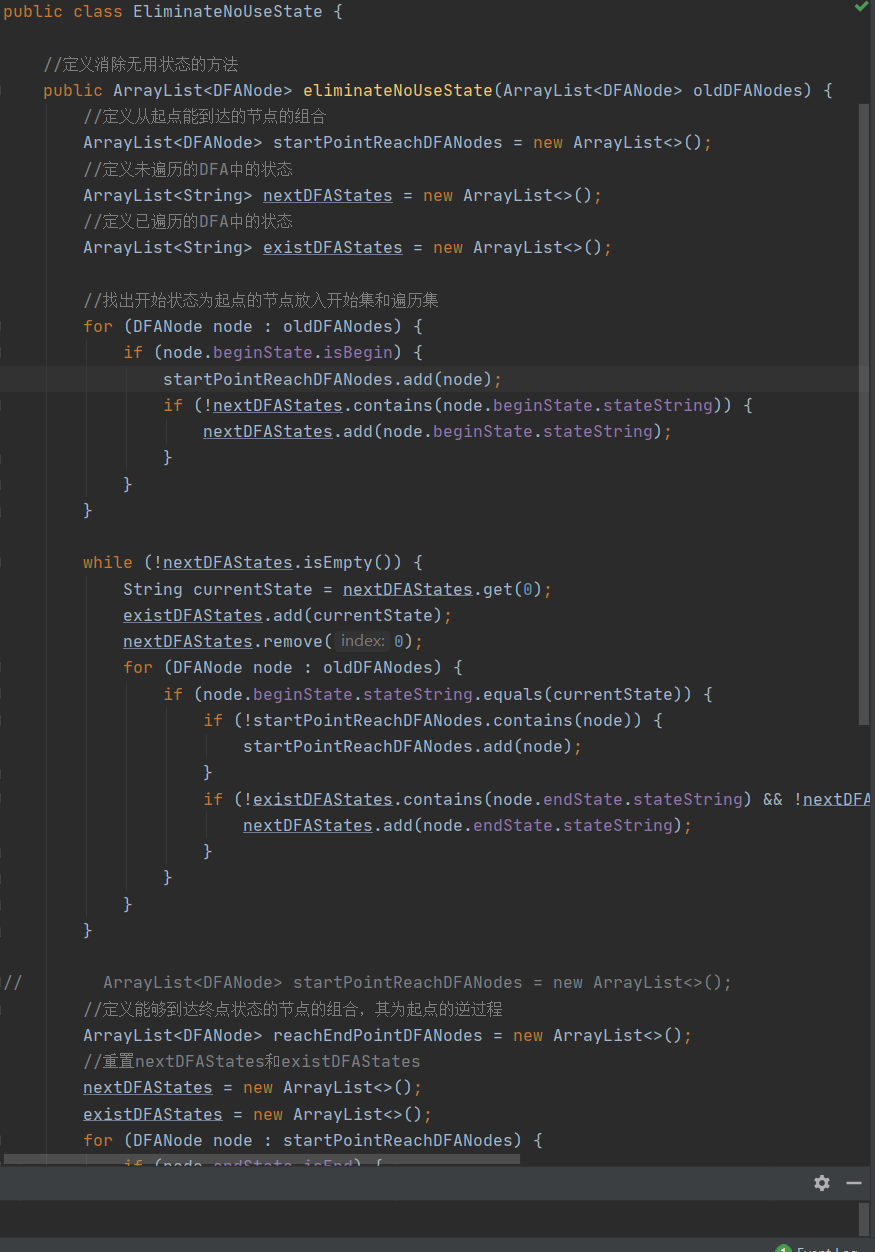
DFA的设计，DFA由DFANode组成，以下是其两者的数据结构：



通过子集法化简NFA为DFA，子集法具体算法如编译原理课程中讲述，具体如下，讲NFA中结点上空字相连的进行归并，归并成为一个新的DFA状态，通过添加新的状态到DFA中直到其规模不在变化，即生成了等价的DFA。



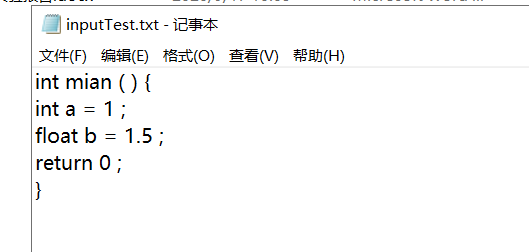
在通过NFA转换得到DFA之后，对DFA进行相应的化简，主要为消除无效状态以及合并等价状态。

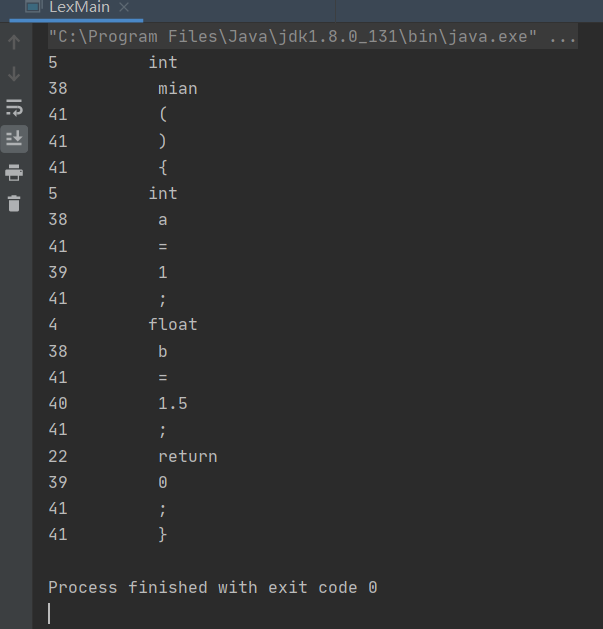
 以下为对应部分源码展示：

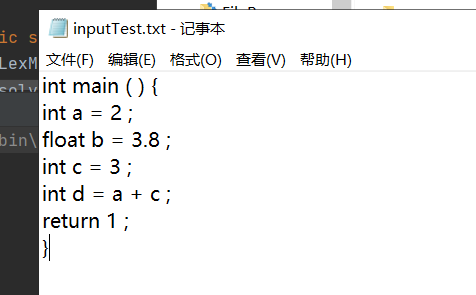


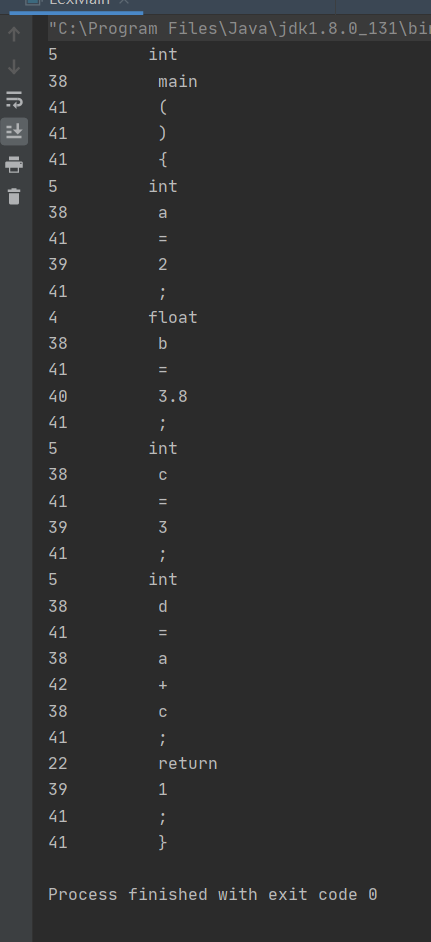
通过消除无效状态和合并等价状态后得到我们对应正规式的DFA，通过DFA构建状态转换机，对读入的源程序代码段进行识别，输出相应的单词符号。

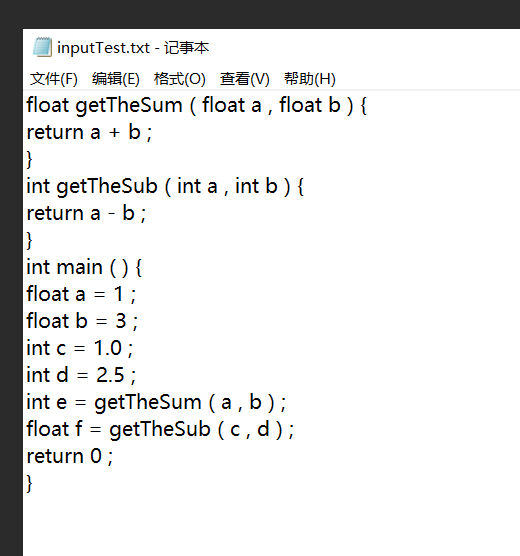
在对源码进行扫描识别是在读取完成一个单词后判断其所在状态是否为结束状态，若为结束状态则打印出相关信息，若不为结束状态则代表出现错误，停止扫描，打印错误信息。

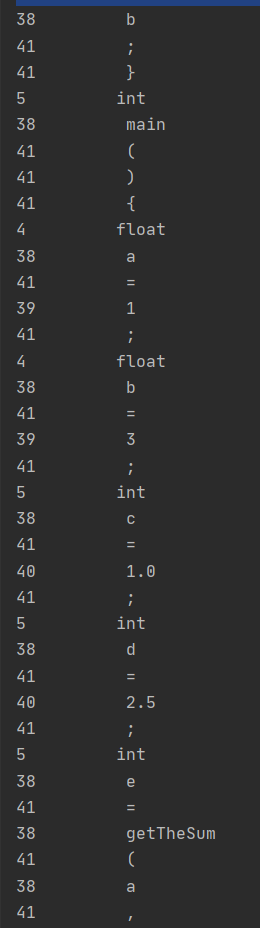
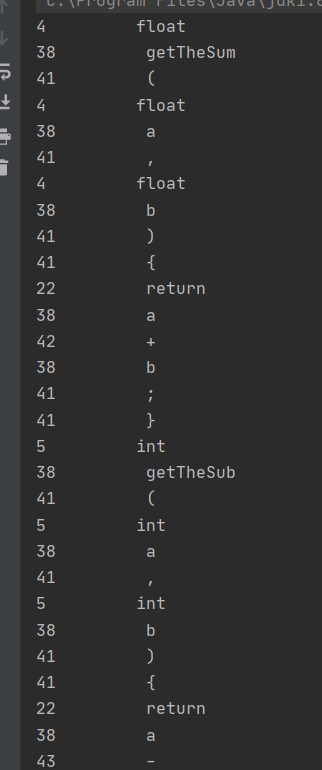
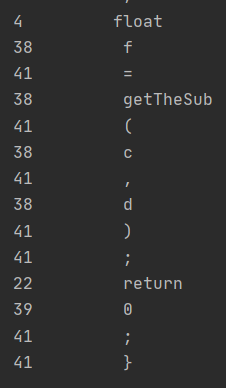
输入测试：

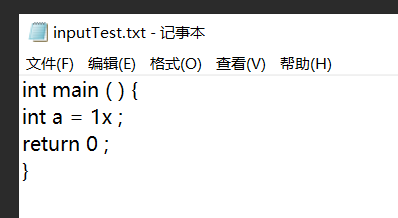
允许程序，得到如下结果：

输入用例：

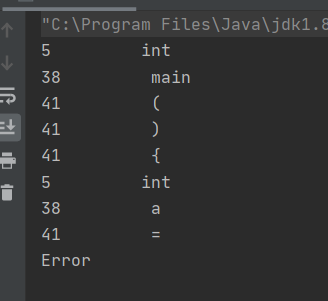
运行结果：

运行代码：

运行结果如下：

运行有错误的源代码：

程序报告错误，停止运行：



**总结**

通过该课程设计，我全面系统的理解了编译原理程序构造的一般原理和基本实现方法。把理论的课本知识变成实践，激发了我学习的积极性。把学过的计算机编译原理的知识强化，能够把课堂上学的知识通过自己设计的程序表示出来，加深了理论知识的理解，也增强了我对于相关算法实例化的能力。