编译原理实验报告

--基于有限自动机的词法分析器构造

南京大学 软件学院 韦祖策 131250148

1. 目标

本次实验的主要目的是对自定义的程序语言的词法分析器程序构造，我从 C 语言当中选择了部分具有代表性的子集，实现词法分析器，主要是对编译原理课程中学习的从正则达式转化为 NFA，再从 NFA 转化为 DFA 以及后续的代码生成的过程有更深刻的认识。同时，也希望对于在编译原理课程中所体现出的计算机科学当中的一些朴素而优美的思想有更多的体会。

2. 内容概述

本报告主要描述了一个简单的词法分析器构造过程，包括最终成品的功能概要，实现过程中的理论推导，具体的核心算法和数据结构的描述，以及个人的收获和体会。

3 实验环境

操作系统 Win8.1

实验的编译器 eclipse

编码格式 Utf-8

3.1 语言定义

3.1.1 记号定义

3.1.1.1 保留字

我选择了 C 语言当中一部分具有代表性的保留字，同时为了保持整个编译原理课程实验的延续性和后续语义分析的实验内容，选择的保留字能够涵盖程序设计中的顺序、分支、和循环结构，这样也可以很好的对正则表达式当中的三种基本运算连接、或、闭包运算形成很好的对应，除此之外，选择了能够形成对函数调用，变量声明进行支持的部分保留字。具体选择的保留字有如下 几个：

else, if, int,double,char,do,continue,break,float, return, void, while,for

3.1.1.2 特殊符号

算术运算符：+、-、*、/

比较运算符：<、<=、>、>=、==、!=

分割符号：;、,

括号：()、[]、{}

数字：支持整数 int，浮点数 doule、float、char

4. 思路和方法

首先，依据复杂性，把上文定义的 lexeme patterns 进行分类，我一共把它们分为三种类别，一种是简单的 lexeme patterns，包括所有单个字符且不成为任意其他任何 pattern 的前缀的 pattern；第二种是其正则表达式是有限长度的子序列的 pattern，也即其自动机不包含回路的情况；第三种是正则表达式子序列的长度可以是无限的 pattern，也就是其有限自动机包含回路，这种分类可以便于基于不同复杂度进行分别处理。

对于第一种类型的 pattern，为其定义对应的 toekn 枚举类型为

符号　Token type 枚举变量

+	PLUS
-	MINES
*	MUTIPLY
/	DIVISION
(	LPAREN
)	RPAREN
;	SEMI
,	COMMA
[	LSQUARE
]	RSQUARE
{	LBRACE
}	RBRACE
!=	NEQ
<	LT
<=	LTE
>	GT
>=	GTE
==	EQ
整数	INT
整数	DOUBLE
保留字	KEY_WORD

程序的大致流程是，从文件当中读入源文件，一次读取一行进行解析，忽略空白，然后依次对一行的缓存当中的每一个字符进行扫描，由状态机进行匹配，返回识别的 token 以及具体信息或者报出错误提示。

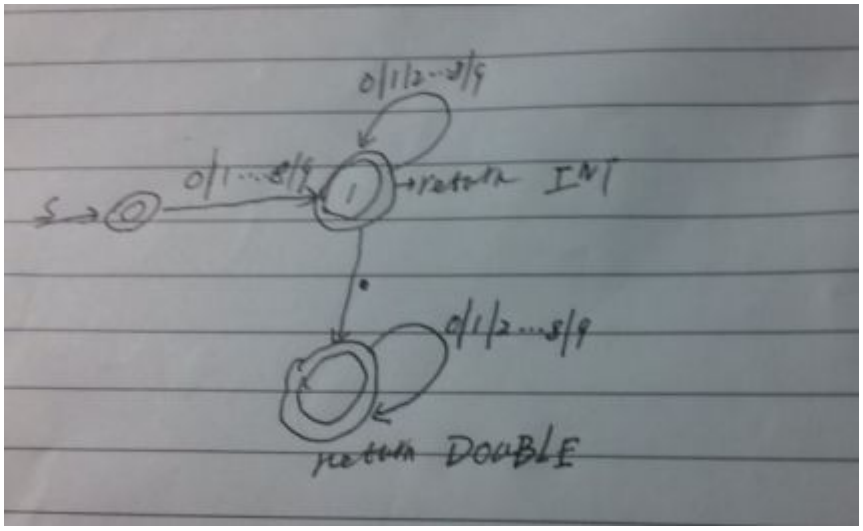
5. 相关有限自动机描述

对于数字，num--->INT|DOUBLE

INT-->digit dight*

DOUBLE-->dight*digit*.dightdigit*

digit-->0|1|2|3|4|5|6|7|8|9



在实现时，一旦检测到 digit，就进入 INNUM 状态，自此之后，程序将不对新扫描的 digit 进行存储和输出，直到遇到一个非 digit 为止，此时 INNUM 状态转变为 DONE 状态，对缓存的数字进行输出，并把 token 类型定义为 NUM

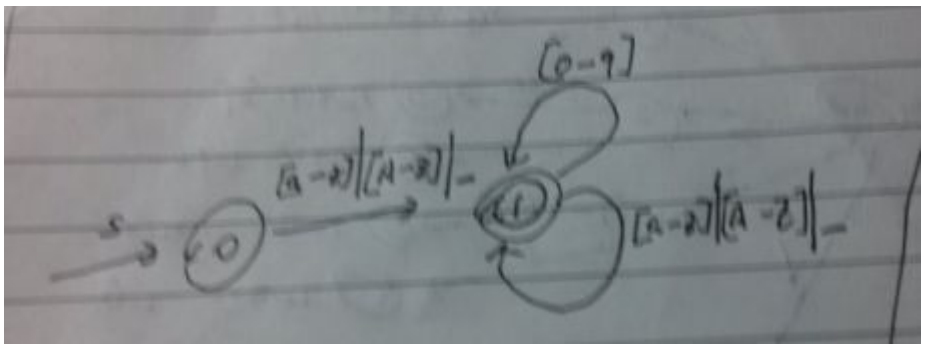
对于保留字和变量：

本程序包含以下划线或者字母开头，包含数字或者字母的变量名，其正则表达式如下：

digit-->0|1|2|3|4|5|6|7|8|9

Leter-->[a-z]|[A-Z]

Identifier-->(letter|_)(digit|leter|_)*

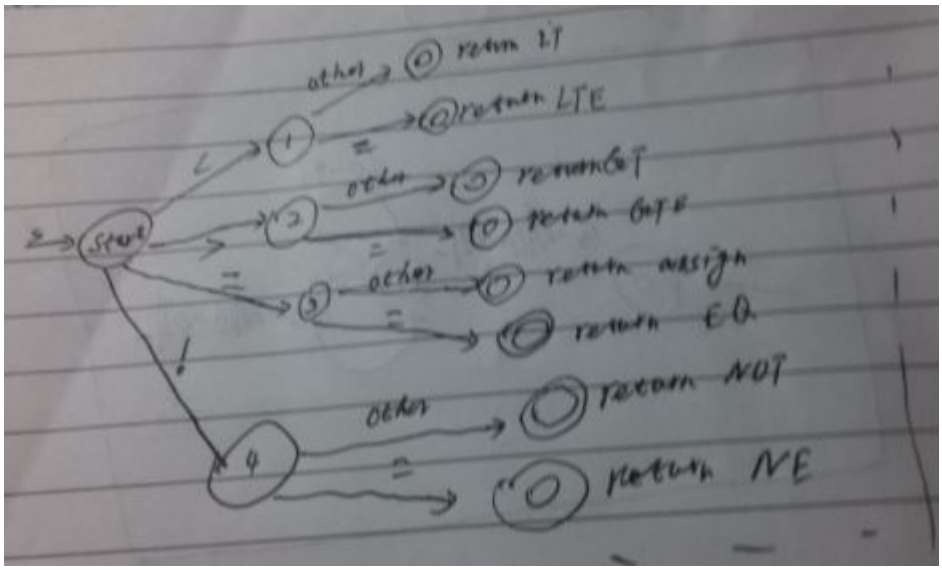


一旦扫描到字母就进入 INID 状态，此后不保存当前字符，知道遇到第一个非数字和字母的字符为止。对于任意一个识别出的字母序列，都要在保留字列表里面进行查找，用于发现是否在表保留字当中，不是就返回 ID，否则就是 keyWord。

逻辑表达式符号：

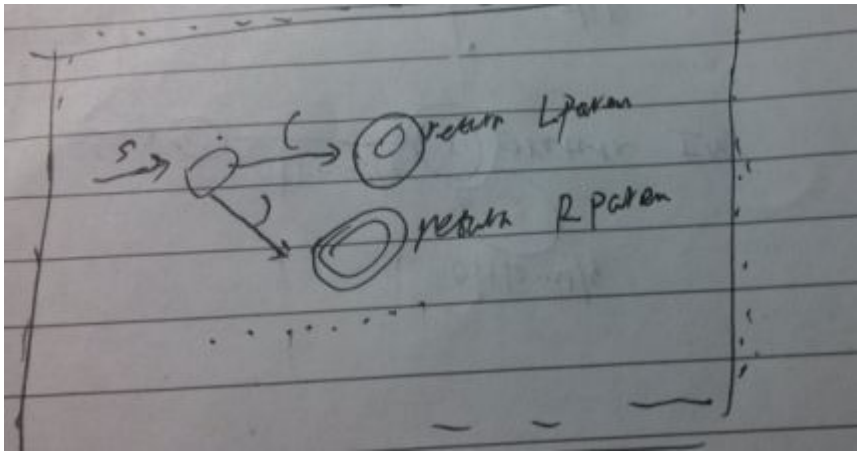
以>和>=为例子,正则表达式：GE → >，GTE→>=

对这些转化为 DFA，状态机如下



而对于其他字符：

以(和）为列子，Lparen→(,Rparen→)　　其他的符号类似。



6. 核心算法描述

这个词法分析器程序的核心算法就是 getToken()这个函数，它消耗字符并根据 DFA 返回下一个被识别的记号。具体来说，getToken 实现用 if 语句对当前输入的 ch 进行下一次状态判断，在每个状态当中，再用 Case 语句对当前字符的特性进行判断，这个双重嵌套的情况分析就是对 DFA 的代码实现，识别了每个状态之后，在 case 语句下面进行是否存储当前字符，是否输出，是否回退等操。

7. 测试用例

7.1 测试输入

测试代码输入在项目中的 program1.txt 文件中。

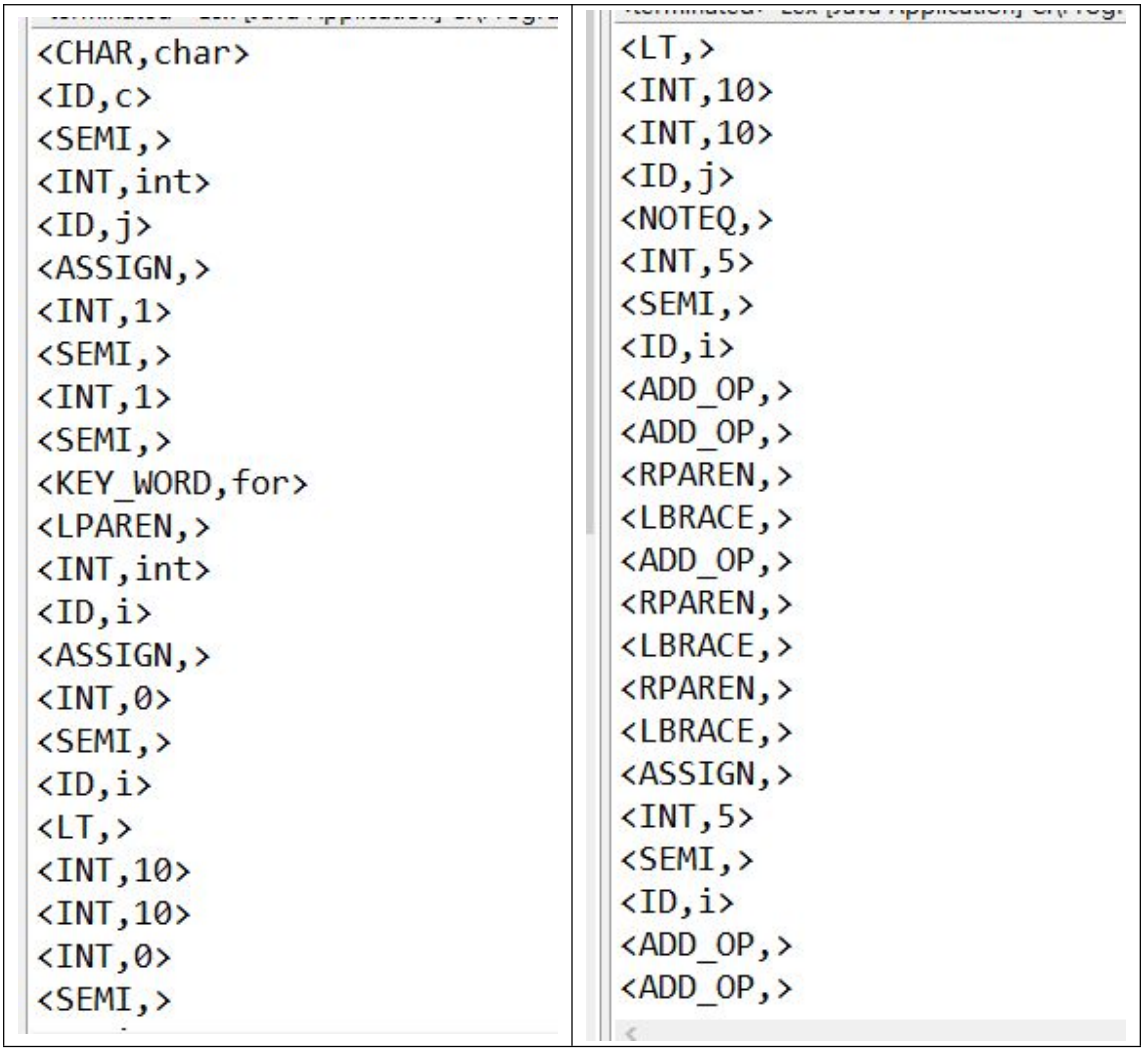
在这个测试用例当中，包括了多行注释，单行注释，有返回值为 int 的函数声明，有选择结构，循环结构，整形常量，标识符等，基本可以验证所有的特性。

```
char c;
```

```
int j=1;
for(int i=0;i<10 && j!=5;i++){
   j=j-1.123;
   double x=2.6;
   int a[10]
   x++;
   if(x>=6){
      break;
   }
}
int lenth=10;
while(length==0){
   length--;
}
```

7.2 测试输出

测试输出如下截图所示，同时打印出了行号，用于后续的错误跟踪信息的处理。如果是标识符就打印为 ID，输出标识符的内容，如果是保留字，就用 KEY_WORD 标志，整形数字用 INT，浮点数字用 DOUBLE，其余符号正常输出，也对应其标识符，可以看出，可以实现上文所述的特性。在测试时会在控制台打印出来，也会在项目中的 result.txt 文件中生成一份。



8. 困难与解决方案

在实现过程中遇到的问题主要有系统的整体设计思路、保留字的处理、 对于需要回退情况的处理

8.1 整体思路：

首先是将系统划分为输入，输出，得到 token，三个部分，这是一种非常经典的划分方式，也很好用。然后对和性的 getToken 进行处理。把 token 的识别放在一个函数当中，单独的函数处理比如读入字符，维护字符缓冲区，放回字符等内容，来隔离复杂度。每次处理一个 char 以后不是舍弃掉，因为是存在数组中，只是移动数组的下标来获取新的字符 ch。

8.2 保留字的处理

关于保留字和标识符的冲突问题，这一点在龙书上就有叙述，在正则表达式的描述当中，保留字和标识符可以被不同的正则表达式匹配，比如 if 和 while 的串既可以是标识符也可以是保留字，而正则表达式无法给出约束，所以要在程序语言定义中规定解释方式，而且是一个无二义性的规则，来处理所有可能出现二义性的情况。有两种典型的规则，一种是当串既可以是标识符也可以是关键字时，这种情况下我采取的是和龙书上所提把保留字预先存储在符号表当中（此处我用的是一个保留字数组进行存储，也即是一个字符串的数组），当新的标识符插入符号表的过程中发现此序列在符号表当中已经存在时，就认定此序列为保留字，改变其 token 的类型。

8.3 回退处理

对于判断一个数字的结尾，一个标识符的结尾，一个<符号而非<=的情况下，需要读入下一个字符，才能判断当前 tokenString 的类型和状态已经到达了结束，但是这个提前读入的字符必须不能被消耗。

我采取的处理方法是，我进行处理的方法是将字符放置于一个数组当中，设置一个表示处理当前字符的指向器 p（就是数组的下标），然后当读出的字符不符合，不处理，下标不变，使用递归的方法调用当前的方法，传入将要处理的字符位置 p。

9.总结与收获

通过这次实验，首先是对词法分析的过程有了更深刻的认识，其次是为后续进行词法分析器生成器的实践奠定了一定的基础，有了一定的感性认识。另一方面来说，自己写词法分析器也是一件很有意思的事情，尽管只是 C 语言词法的一个子集，但是还是很令人开心。一开始一头雾水，不知道要干什么，到自己写出了尽管不是功能很强大，但基本成型的词法分析器的时候，对自己信心的增加还是大有帮助的。