编译原理实验报告

LAB1 Lexical Analyzer

学号：141250031 姓名：杜天蛟 日期：2016.10.22

目录

目的

设计并编写一个词法分析器，加深对词法分析的理解。

输入输出

Input: 字符串，自己定义的正则表达式
Output: token 序列

实现思路

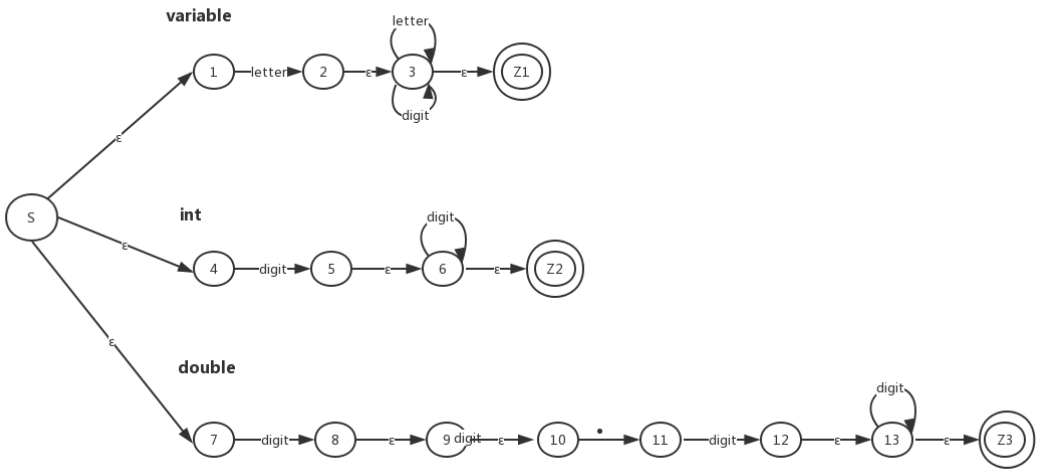
词法分析程序的功能是：读入字符串形式的源程序，识别出具有意义的最小语法单位：单词符号。本程序定义能够识别出的单词符号有：保留字、运算符、标识符、格式符。

大概的设计过程是：

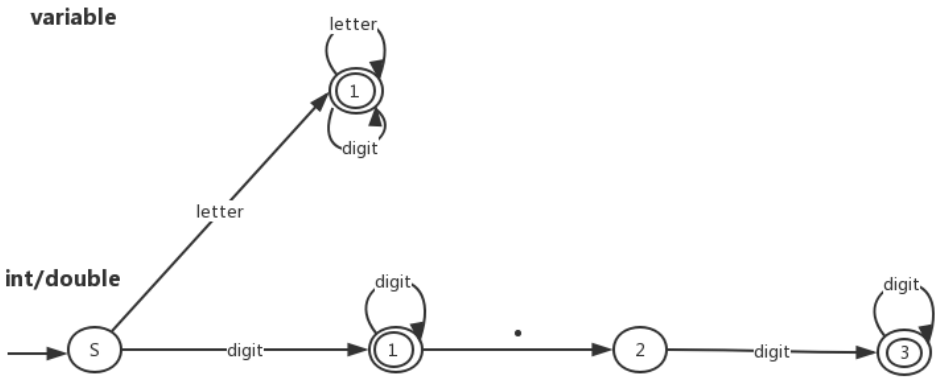
1. 根据要求写出词法分析的正则表达式 RE
2. 根据 RE，画出各自的 NFA，再合并为一个 NFA

3. 将 NFA 转化成 DFA
4. 将 DFA 最小化
5. 根据最小化的 DFA 编写程序

NFA 如下图所示：



最小化的 DFA 如下图所示：



支持的词法

支持的关键字：

```
if 60
else 61
while 62
do 63
case 64
switch 65
for 66
```

```
try 67
catch 68
throws 69
true 70
false 71
int 72
char 73
abstract 74
assert 75
boolean 76
break 77
byte 78
catch 79
class 80
const 81
continue 82
default 83
do 84
double 85
enum 86
extends 87
final 88
finally 89
float 90
implements 91
import 92
instanceof 93
interface 94
long 95
native 96
new 97
package 98
private 99
public 100
return 101
short 102
static 103
strictfp 104
super 105
switch 106
synchronized 107
this 108
transient 109
```

```
void 110
volatile 111
```

支持的运算符和格式符：

```
+ 10
- 11
* 12
/ 13
% 14
> 15
< 16
= 17
( 18
) 19
{ 20
} 21
| 22
& 23
! 24
? 25
~ 26
^ 27
[ 28
] 29
: 30
. 31
; 32
, 33
' 34
" 35
+= 36
-= 37
>= 38
<= 39
== 40
!= 41
|| 42
&& 43
```

支持的标识符：

```
variable->letter(letter|digit)*
```

支持的基本数据类型：

```
Int: int->digit digit*
Double: double->digit digit* . digit digit*
digit->[0-9]
letter->[a-zA-Z]
```

其他：

空格、制表符、换行符都不作处理。

重要的数据结构描述

LexicalData 类中保存了保留字和操作符的键值对

```
private Map<String, String> reservedWordMap;
private Map<String, String> operatorMap;
```

封装了识别保留字和操作符的方法

```
public int isReservedWord(String str) {
    return getId(str, reservedWordMap);
}

public int isOperator(String str) {
    return getId(str, operatorMap);
}
```

LexicalAnalyzer 类中将读取的文本文件转换成 char 数组

```
private List<Character> programme;
```

主要的算法在 scan 里

核心算法描述

算法的基本任务是从字符串表示的源程序中识别出具有独立意义的单词符号。

根据 DFA 来看，从一个状态到另一个状态的转换都可以用分支语句来编写，每个自环都可以用循环来表示。

开始扫描后，首先跳过无用的空白字符。当遇到的是字母时，则有可能是关键字或者一般的标识符。把关键字作为特殊标识符处理，把它们预先安排在一张表格中，当扫描程序识别出标识符时，查关键字表。如能查到匹配的单词，则该单词为关键字，否则为一般标识符。运算符和格式符同理。当遇到的是数字时，则根据 DFA 来判断是否为正确的 int 或者 double 型。

运行实例

输入的 text.txt：

```
package lex;

import lex.algorithm.*;

public class Main {
```

```
   public static void main(String[] args){
       double x = 3.2;
      Lexlogic lexLogic=new Lexlogic();
      lexLogic.entrence();
      if(x<=0){
          return 0;
      }
   }
}
```

输出的 output.txt

```
keyword98: package
ID: lex
operator32: ;
keyword92: import
ID: lex
operator31: .
ID: algorithm
operator31: .
operator12: *
operator32: ;
keyword100: public
keyword80: class
ID: Main
operator20: {
keyword100: public
keyword103: static
keyword110: void
ID: main
operator18: (
ID: String
operator28: [
operator29: ]
ID: args
operator19: )
operator20: {
keyword85: double
ID: x
operator17: =
double: 3.2
operator32: ;
ID: Lexlogic
ID: lexLogic
```

```
operator17: =
keyword97: new
ID: Lexlogic
operator18: (
operator19: )
operator32: ;
ID: lexLogic
operator31: .
ID: entrence
operator18: (
operator19: )
operator32: ;
keyword60: if
operator18: (
ID: x
operator39: <=
operator17: =
int: 0
operator19: )
operator20: {
keyword101: return
int: 0
operator32: ;
operator21: }
operator21: }
operator21: }
```

发生的问题和解决方案

最一开始我是用 map 保存提取出来的 token 的，后来打印文件时发现少了很多很多的 token，而且是乱序，才想到 map 是无序而且 key 不能重复，所以才会出现这种情况。于是我就用 list 直接存储了字符串。

个人感想

主要学习和体会了基于编译器构造技术中的由正规表达式到最小化 DFA 的算法设计和实现技术：主要包括由正规表达式构造 NFA 所用到的 Thompson 构造法、 把 NFA 转化为与其等价的 DFA 所使用的子集构造算法以及把 DFA 最小化的算法，最后实现词法分析。