 

**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **编译技术** |
| **学生姓名：** | **何博韬** |
| **学生学号：** | **201630676706** |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2017-2018第一学期** |

**软件学院**

**2017年11月**

# XXXX2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **地 点：** | | | B7 楼 | | 233 房； | | **实验台号：** | | 29 | |
| **实验日期与时间：** | | | 2017年11月1日星期三 | | | | **评 分：** | |  | |
| **预习检查纪录：** | | |  | | | | **实验教师：** | | 徐扬 | |
| **电子文档存放位置：** | | |  | | | | | | | |
| **电子文档文件名：** | | |  | | | | | | | |
| **批改意见：** | | |  | | | | | | | |
| 出勤情况 | 课堂态度 | | 实验报告质量 | | | | | |
| 任务完成情况 | | 源代码注释 | | 报告内容 | |
| □ 出勤  □ 缺勤  □ 早退  □ 请假 | □ 认真  □ 不认真 | | □ 完成  □ 未完成 | | □ 优秀  □ 良好  □ 通过  □ 未通过 | | □ 优秀  □ 良好  □ 通过  □ 未通过 | |

# 报告内容

1、实验目标

实现decaf 语言的词法分析程序

2、实验环境

Window10 JVM Eclipse IDE

3、关键数据结构和核心算法

3.1 数据结构

哈希表

3. 2 核心算法

核心算法是Lexer.java文件中的public Token scan()函数。scan()函数用来扫描输入的文本，并输出Token序列。

scan()函数的伪代码如下

for( ；；readch) 循环读取输入序列

｛

if(是空格或\t)

continue;

else if(是\n)

整形变量line加一；

else

break;

｝

switch(peek)

｛

case 此语言的关键字或者操作符：

return 关键字或操作符对应的Word对象；

｝

if(peek是数字)

｛

do{将连续的多个数字转换为一个十进制数}

while（peek不是数字）;

if（不出现小数点）

return 一个Num类的对象；

else

for（一个无条件的for循环）

｛if（下一个字符不是数字） break;

else 将连续的多个数字转换为十进制小数｝

｝

if（peek是字母）

｛

new一个缓冲区来储存输入的串

do｛将输入的连续的字符储存到缓冲区｝

while（peek是字母或数字）

将缓冲区中的内容赋值给一个String 对象，再将String对象映射的哈希表中的值取出并强制转换为Word 对象。

if（此Word对象值不为null）

return 此Word对象；

else

返回一个tag 为ID的Word对象

｝

3. 3 错误处理

1. **default**:
2. System.out.println("(SYM , "+token.toString()+")");

3.4 运行结果



图 1

4、总结

通过对代码的标注，我对词法分析的理解又加深了。

#### 附 标记后的代码

Main.java

1. **package** main;
3. **import** java.io.IOException;
4. **import** java.util.Hashtable;
6. **import** lexer.Lexer;
7. **import** lexer.Token;
9. **public** **class** Main {
10. **public** **static** **void** main(String[] args) **throws** IOException {
11. Lexer lexer = **new** Lexer();
13. **do** {
14. Token token=lexer.scan();
15. **switch** (token.tag) {
16. **case** 270:
17. **case** 272:
18. System.out.println("(NUM , "+token.toString()+")");
19. **break**;
20. **case** 264:
21. System.out.println("(ID , "+token.toString()+")");
22. **break**;
23. **case** 256:
24. **case** 257:
25. **case** 258:
26. **case** 259:
27. **case** 260:
28. **case** 265:
29. **case** 274:
30. **case** 275:
31. System.out.println("(KEY , "+token.toString()+")");
32. **break**;
33. **case** 13:  //case \r 键，确保程序正常运行
34. **break**;
35. **default**:
36. System.out.println("(SYM , "+token.toString()+")");
37. **break**;
38. }
40. } **while** (lexer.getPeek()!='\n');
41. }
42. }

Word.java

1. **package** lexer;
2. //管理保留字，标识符
3. **public** **class** Word **extends** Token {
4. **public** String lexeme = "";
6. **public** Word(String s, **int** tag) {
7. **super**(tag);  //调用父类Token的构造函数
8. lexeme = s;
9. }
11. **public** String toString() {
12. **return** lexeme;
13. }
15. **public** **static** **final** Word
16. and = **new** Word("&&", Tag.AND),
17. or = **new** Word("||",Tag.OR),
18. eq = **new** Word("==", Tag.EQ),
19. ne = **new** Word("!=", Tag.NE),
20. le = **new** Word("<=", Tag.LE),
21. ge = **new** Word(">=", Tag.GE),
22. minus = **new** Word("minus", Tag.MINUS),
23. True = **new** Word("true",Tag.TRUE),
24. False = **new** Word("false", Tag.FALSE),
25. temp = **new** Word("t", Tag.TEMP);
26. }

Lexer.java

1. **package** lexer;
3. **import** java.io.IOException;
4. **import** java.util.Hashtable;
6. **public** **class** Lexer {
8. **public** **static** **int** line = 1;
9. **char** peek = ' ';   //一个空字符，用来储存输入的文本
10. Hashtable words = **new** Hashtable();  //用默认的初始容量 (11) 和加载因子 (0.75) 构造一个新的空哈希表。
11. **void** reserve(Word w) {  // 哈希表保留函数  w.lexeme 为键  w 为值
12. words.put(w.lexeme, w);
13. }
15. **public** Lexer() {  //哈希表初始化
16. reserve(**new** Word("if", Tag.IF));  // 键 if
17. reserve(**new** Word("else", Tag.ELSE));  // 键 else
18. reserve(**new** Word("while", Tag.WHILE));  //键 while
19. reserve(**new** Word("do", Tag.DO));  //键 do
20. reserve(**new** Word("break", Tag.BREAK));  //键 break
21. reserve(Word.True); //键 true
22. reserve(Word.False);  //键 false
23. }
24. //读入用户输入字符，储存在字符变量 peek 中
25. **public** **void** readch() **throws** IOException {
26. peek = (**char**) System.in.read();
28. }
29. //实现 最长子串匹配原则 而需要的 readch 函数
30. **boolean** readch(**char** c) **throws** IOException {
31. readch();
32. **if** (peek != c) {  //若输入不是连续的c 则返回 false
33. **return** **false**;
34. }
35. peek = ' ';  //若是连续的c 则返回true 并将peek 置为空
36. **return** **true**;
37. }
39. **public** Token scan() **throws** IOException {  //扫描用户输入的字符串
40. **for** (;; readch()) {  //扫描输入中的空格 制表符 换行符 。忽略空格，制表符。换行符使行数增加。
41. **if** (peek == ' ' || peek == '\t')  //遇到空格或制表符 重新开始for循环
42. **continue**;
43. **else** **if** (peek == '\n') {  //遇到换行符，行数加一
44. line += 1;
45. } **else** {
46. **break**;
47. }
48. }
49. //空的内容排除后，提取真正的字符，并与语言保留字匹配
50. **switch** (peek) {  //字符型peek被默认转换为整形
51. **case** '&':   //若有连续两个&&符号，则与保留字and匹配
52. **if** (readch('&'))
53. **return** Word.and;
54. **else**  //只有一个&符号
55. **return** **new** Token('&');
56. **case** '|':  // 与&同理
57. **if** (readch('|'))
58. **return** Word.or;
59. **else**
60. **return** **new** Token('|');
61. **case** '=':  // 与&同理
62. **if** (readch('='))
63. **return** Word.eq;
64. **else**
65. **return** **new** Token('=');
66. **case** '!':  // 与&同理
67. **if** (readch('='))
68. **return** Word.ne;
69. **else**
70. **return** **new** Token('!');
71. **case** '<':  // 与&同理
72. **if** (readch('='))
73. **return** Word.le;
74. **else**
75. **return** **new** Token('<');
76. **case** '>':  // 与&同理
77. **if** (readch('='))
78. **return** Word.ge;
79. **else**
80. **return** **new** Token('>');
81. }
82. **if** (Character.isDigit(peek)) {  //调用Character类中的isDigit()方法    判断输入是否为数字0~9
83. **int** v = 0;
84. **do** {
85. v = 10 \* v + Character.digit(peek, 10);  // 把peek的值转换为十进制，依照十进制规则，越左边数值的幂次是其相邻右边数值幂次的十倍
86. readch();
87. } **while** (Character.isDigit(peek));  //处理连续输入的十进制数字
88. **if** (peek != '.')  //若 连续的数字中没有小数点，则返回 一个 Num类型的对象，扫描结束。Num为Token的子类。
89. **return** **new** Num(v);
90. //若数字中包含小数点，表明输入数字为小数。 一下为对小数部分的扫描
91. **float** x = v;
92. **float** d = 10;
93. **for** (;;) {
94. readch();
95. **if** (!Character.isDigit(peek))  //输入序列中下一个字符不是数字，则跳出循环
96. **break**;
97. x = x + Character.digit(peek, 10) / d;  // 十进制小数的处理。
98. d = d \* 10;
99. }
100. **return** **new** Real(x);  //返回一个Real类型的对象。Real类为Token类的子类。扫描结束。
101. }
102. //以字母开头的连续的串，如果后面包含数字，数字不会被抽取出来
103. **if** (Character.isLetter(peek)) {  //判断输入字符是否为字母
104. StringBuffer b = **new** StringBuffer();  //构造一个空字符缓冲区，初始容量为16个字符
105. **do** {
106. b.append(peek);  //将peek加进缓冲区
107. readch();
108. } **while** (Character.isLetterOrDigit(peek));  //将连续输入的字符加进一个缓冲区
109. String s = b.toString();  // 将缓冲区b中的内容赋给一个String 变量s
110. Word w = (Word) words.get(s);  //返回制定键s所映射的值 并强制类型转换 转换为Word类型 若 此映射不包含此键的映射 则返回null
111. **if** (w != **null**)  // w不为null 表示 输入为一个关键字或保留值
112. **return** w;  //结束扫描
113. w = **new** Word(s, Tag.ID);  // 否则 输入为一个普通字符 标记为 ID
114. words.put(s, w);  // 将输入存入哈希表
115. **return** w;  //结束扫描
116. }
117. Token tok = **new** Token(peek);
118. peek = ' ';
119. **return** tok;
120. }
122. **public** **void** out() {
123. System.out.println(words.size());  // 返回哈希表中键的数量
125. }
127. **public** **char** getPeek() {
128. **return** peek;
129. }
131. **public** **void** setPeek(**char** peek) {
132. **this**.peek = peek;
133. }
135. }