信 息 工 程 学 院

《编译原理》实验报告（三）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **学院**:信息工程学院 | **班级**:软工2001班 | **姓名**:张宇晨 | **学号**:2020012249 | **成绩**: |

# 实习目标

1. 掌握自顶向下语义分析中语义子程序的添加过程；

2. 掌握“拉链”、“回填”操作的原理及实现；

3. 根据 MiniC 的上下文无关文法，对赋值语句、算术表达式、关系表达式、 if-else 语句、while 语句、布尔表达式等语法结构添加语义子程序；

4. 针对测试代码，输出四元式序列。

# 实验过程

## 词法定义与语法定义

词法定义与语法定义按照规则和需求，正确定义，由于前两次实习已经详细介绍了我的词法定义与语法定义过程，本次实习报告中不再赘述。

## 辅助类的定义

为了完成语义分析工作，需要定义辅助JAVA类来帮助实现功能。本次实习中定义了6个辅助类，如图1所示。类名即此类的功能：ArrayInfo.java中存储数组的相关信息，包括行列数和数组名；ArrayMap.java中维护了一个HashMap,用于唯一标识一个数组，便于检错；ConditionValue.java中维护了两个列表，真链和假链，同时提供了拉链和回填方法，用于条件表达式和布尔表达式的语义分析；QTInfo.java和QTList.java用于四元式的生成与存储；VariableNameGenerator.java用于生成不同的变量名。具体实现见代码2.2.1-2.2.6

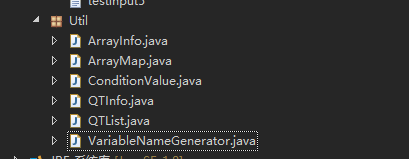


图1 辅助类

### ArrayInfo.java

1. **package** Util;
3. **public** **class** ArrayInfo {
4. **private** String col=""+0;
5. **private** String raw=""+0;
7. **public** ArrayInfo(String raw, String col) {
9. **this**.col = col;
10. **this**.raw = raw;
11. }
13. **public** ArrayInfo(String raw) {
15. **this**.col = 1+"";
16. **this**.raw = raw;
17. }
19. **public** ArrayInfo() {
21. **this**.col = 1+"";
22. **this**.raw = 1+"";
23. }
25. **public** **void** setCol(String col) {
26. **this**.col = col;
27. }
29. **public** **void** setRaw(String raw) {
30. **this**.raw = raw;
31. }
33. **public** String getCol() {
34. **return** col;
35. }
37. @Override
38. **public** String toString() {
39. **return** "ArrayInfo{"  + '\'' +
40. ", col=" + col +
41. ", raw=" + raw +
42. '}';
43. }
45. **public** String getRaw() {
46. **return** raw;
47. }

50. }

### ArrayMap.java

1. **package** Util;
3. **import** java.util.HashMap;
5. **public** **class** ArrayMap {
6. **private** HashMap<String, ArrayInfo> map = **new** HashMap<>();
8. **public** **boolean** createArray(String name,String raw, String col)
9. {
10. **if**(map.containsKey(name))
11. {
12. **return** **false**;
13. }
14. ArrayInfo aif = **new** ArrayInfo(raw,col);
15. map.put(name,aif);
16. **return** **true**;
17. }
18. **public** **boolean** containsKey(String key)
19. {
20. **return** map.containsKey(key);
21. }
22. **public** String getcol(String name)
23. {
24. **return** map.get(name).getCol();
25. }
27. **public** String getraw(String name)
28. {
29. **return** map.get(name).getRaw();
30. }
31. **public** **boolean** Check(String name, String col, String raw)
32. {
33. **if**((!map.containsKey(name)) ||
34. (Integer.parseInt(raw)>Integer.parseInt(**this**.getraw(name))) ||
35. (Integer.parseInt(col)>Integer.parseInt(**this**.getcol(name))))
36. {
37. **return** **false**;
38. }
39. **return** **true**;
40. }
42. }

### ConditionValue.java

1. **package** Util;
3. **import** java.util.ArrayList;
4. **import** java.util.Iterator;
5. /\*真假条件值，属性里面有 真链和假链条(TrueChain,FalseChain)\*/
6. **public** **class** ConditionValue {
7. **private** ArrayList<QTInfo> trueChain = **new** ArrayList<>();
8. **private** ArrayList<QTInfo> falseChain = **new** ArrayList<>();
10. **public** **void** setTrueChain(ConditionValue cValue) {
11. trueChain = cValue.trueChain;
12. }
14. **public** **void** setFalseChain(ConditionValue cValue) {
15. falseChain = cValue.falseChain;
16. }
18. **public** **void** mergeTrue(QTInfo qtTrue) {
19. trueChain.add(qtTrue);
21. }
23. **public** **void** FEImerge(ConditionValue cValue)
24. {
25. trueChain.addAll(cValue.falseChain);
26. falseChain.addAll(cValue.trueChain);
27. }
29. **public** **void** mergeTrue(ConditionValue cValue) {
30. trueChain.addAll(cValue.trueChain);
31. }

34. **public** **void** mergeFalse(QTInfo qtFalse) {
35. falseChain.add(qtFalse);
36. }
38. **public** **void** mergeFalse(ConditionValue cValue) {
39. falseChain.addAll(cValue.falseChain);
40. }
42. **public** **void** backpatchTrueChain(**int** result) {
43. Iterator<QTInfo> itr = trueChain.iterator();
44. **while**(itr.hasNext()) {
45. itr.next().setResult(result);
46. }
47. }
49. **public** **void** backpatchFalseChain(**int** result) {
50. Iterator<QTInfo> itr = falseChain.iterator();
51. **while**(itr.hasNext()) {
52. itr.next().setResult(result);
53. }
54. }
55. }

### QTInfo.java

1. **package** Util;
2. //四元式的形态
3. **public** **class** QTInfo {
4. **public** **static** **int** size = 0; // 四元式全局个数，这个Static变量是用来记录QTInfo实例化对象的个数的
5. **public** **int**    innerId;      // 当前四元式ID
6. **public** String operator;
7. **public** String arg1;
8. **public** String arg2;
9. **public** String result;
10. **public** QTInfo(String operator, String arg1, String arg2, String result) {
11. **super**();
12. **this**.innerId   = ++size;
13. **this**.operator   = operator;
14. **this**.arg1 = arg1;
15. **this**.arg2 = arg2;
16. **this**.result = result;
17. }
19. **public** QTInfo(String operator, String arg1, String arg2, **int** result) {
20. **this**(operator, arg1, arg2, ""+result);
21. }

24. **public** String getOperator() {
25. **return** **this**.operator;
26. }
28. **public** **void** setResult(String result) {
29. **this**.result = result;
30. }
32. **public** **void** setResult(**int** result) {
33. **this**.result = ""+result;
34. }
36. **public** String getResult() {
37. **return** **this**.result;
38. }
40. **public** **void** setInnerId(**int** innerId) {
41. **this**.innerId = innerId;
42. }
44. **public** **int** getInnerIdSeqen() {
45. **return** size;
46. }
48. **public** String toString() {
49. **return** "(" + innerId + ")(" + operator + ", " + arg1 + ", " + arg2 + ", " + result + ")";
50. }
51. }

### QTList.java

1. **package** Util;
3. **import** java.util.ArrayList;
4. **import** java.util.Iterator;
5. //将四元式保存到这里
6. **public** **class** QTList {
7. ArrayList<QTInfo> QTList =  **new** ArrayList<QTInfo>();
8. **public** **void** addQTInfo(QTInfo info) {
9. QTList.add(info);
10. }
12. **public** **void** addQTInfo(**int** index, QTInfo info) {
13. QTList.add(index, info);
14. }
16. **public** QTInfo get(**int** index) {
17. **return** (QTInfo) QTList.get(index);
18. }
19. **public** **void** setResultbyIndex(**int** index, **int** Result)
20. {
21. QTInfo temp = QTList.get(index-1);
22. temp.setResult(Result);
23. QTList.set(index-1, temp);
24. }
26. **public** QTInfo remove(**int** index) {
27. **return** QTList.remove(index - 1);
28. }
30. **public** **void** clear() {
31. QTList.clear();
32. QTInfo.size = 0;
33. }
35. **public** String printQTTable() {
36. String res = "\n";
37. Iterator<QTInfo> it = QTList.iterator();
38. **try** {
39. **while** (it.hasNext()) {
40. QTInfo tmp = (QTInfo) it.next();
41. res += tmp.toString()+"\n";
42. }
43. } **catch** (Exception e) {
44. e.printStackTrace();
45. }
46. **return** res;
48. }
49. }

### VariableNameGenerator.java

1. **package** Util;
2. //用于产生不同的T1,T2,T3...
3. //不同名字生成器
4. **public** **class** VariableNameGenerator {
5. **static** **final** String VAR\_PREFIX\_STRING = "T";
6. **static** **int** sequenceId = 0;
8. **public** **static** String genVariableName() {
9. ++sequenceId;
10. **return** VAR\_PREFIX\_STRING + sequenceId;
11. }
13. **public** **void** clear() {
14. sequenceId = 0;
15. }
16. }

## 编写基本语义程序

按照老师的《超详细的javaCC语义分析》中的详细流程，定义了声明语句（2.3.1），赋值语句（2.3.2），if条件语句（2.3.3），while循环（2.3.4），条件表达式（2.3.5），Expression（2.3.6）的语义语法程序。基本代码与视频中的代码出入不大，但是我另外实现了数组、布尔表达式和for循环的翻译，所以在条件表达式（2.3.5），声明语句（2.3.1），赋值语句（2.3.2）部分与视频中不同。具体代码如2.3.1-2.3.6所示。

### 声明语句语义分析

1. **void** shengming() :
2. {
3. String arg1 = **null**;
4. String result = **null**;
5. String raw = ""+ 1;
6. String col = ""+ 1;
7. }
8. {
9. (
10. "int"
11. | "double"
12. | "float"
13. | "char"
14. )
15. result = Identifier() //int a=12;
16. (
17. (
18. "=" arg1 = biaodashi()
19. {
20. QTInfo qt0 = **new** QTInfo("=", arg1, "\_", result);
21. qtlist.addQTInfo(qt0);
22. }
23. )
24. |
25. (
26. "["
27. (
28. col = Integer()
29. | col = Identifier()
30. )
31. "]"
32. (
33. "["
34. (
35. raw = Integer()
36. | raw = Identifier()
37. )
38. "]"
39. )?
40. {
41. arrmap.createArray(result, raw, col);
42. }
43. )
44. )?
45. (
46. "," result = Identifier()
47. (
48. (
49. "=" arg1 = biaodashi()
50. {
51. QTInfo qt = **new** QTInfo("=", arg1, "\_", result);
52. qtlist.addQTInfo(qt);
53. }
54. )
55. |
56. (
57. "["
58. (
59. col = Integer()
60. | col = Identifier()
61. )
62. "]"
63. (
64. "["
65. (
66. raw = Integer()
67. | raw = Identifier()
68. )
69. "]"
70. )?
71. {
72. arrmap.createArray(result, raw, col);
73. }
74. )
75. )?
76. )\*
77. }

### 赋值语句语义分析

1. **void** fuzhi() :
2. {
3. String arg1 = **null**;
4. String result = **null**;
5. String Temp = **null**;
6. String arrname = **null**;
7. String T1 = **null**;
8. String T2 = **null**;
9. String T3 = **null**;
10. String acol = ""+ 1;
11. String araw = ""+ 1;
12. String col = ""+ 1;
13. String raw = ""+ 1;
14. }
15. {
16. //(=,biaodashi,\_,biaoshifu)
17. //QTInfo(String operator, String arg1, String arg2, String result)
18. result = Identifier()
19. (
20. "["
21. (
22. col = Integer()
23. | col = Identifier()
24. )
25. "]"
26. (
27. "["
28. (
29. raw = Integer()
30. | raw = Identifier()
31. )
32. "]"
33. )?
34. {
35. System.out.println("到此！"+result);
36. }
37. "=" arg1 = biaodashi()
38. {
39. **if** (!arrmap.containsKey(result))
40. {
41. System.out.println("\u6570\u7ec4\u672a\u5b9a\u4e49");
42. **throw** **new** NoSuchFieldException("\u6570\u7ec4\u672a\u5b9a\u4e49\uff01");
43. }
44. acol = arrmap.getcol(result);
45. araw = arrmap.getraw(result); //获取数组的大小（这里是横向存储的数组）
46. T1 = VariableNameGenerator.genVariableName();
47. T2 = VariableNameGenerator.genVariableName();
48. T3 = VariableNameGenerator.genVariableName();
49. QTInfo qt1 = **new** QTInfo("\*", col, acol, T1);
50. QTInfo qt2 = **new** QTInfo("+", raw, T1, T1);
51. Temp = String.valueOf(Integer.parseInt(acol) + 1);
52. QTInfo qt3 = **new** QTInfo("-", result, Temp, T2);
53. QTInfo qt4 = **new** QTInfo("[]=", arg1, "-", T2 + "[" + T1 + "]");
54. qtlist.addQTInfo(qt1);
55. qtlist.addQTInfo(qt2);
56. qtlist.addQTInfo(qt3);
57. qtlist.addQTInfo(qt4);
58. }
59. )?
60. (
61. (
62. "="
63. (
64. LOOKAHEAD(2)
65. ( //将数组的值赋给标识符“ a = b[1][2]
66. arrname = Identifier()
67. "["
68. (
69. col = Integer()
70. | col = Identifier()
71. )
72. "]"
73. (
74. "["
75. (
76. raw = Integer()
77. | raw = Identifier()
78. )
79. "]"
80. )?
81. {
82. **if** (!arrmap.containsKey(arrname))
83. {
84. System.out.println("\u6570\u7ec4\u672a\u5b9a\u4e49");
85. **throw** **new** NoSuchFieldException("\u6570\u7ec4\u672a\u5b9a\u4e49\uff01");
86. }
87. System.out.println("case 1");
88. T1 = VariableNameGenerator.genVariableName();
89. acol = arrmap.getcol(arrname);
90. araw = arrmap.getraw(arrname); //获取数组的大小（这里是横向存储的数组）
91. QTInfo qt1 = **new** QTInfo("\*", col, acol, T1);
92. T2 = VariableNameGenerator.genVariableName();
93. T3 = VariableNameGenerator.genVariableName();
94. QTInfo qt2 = **new** QTInfo("+", raw, T1, T1);
95. Temp = String.valueOf(Integer.parseInt(acol) + 1);
96. QTInfo qt3 = **new** QTInfo("-", arrname, Temp, T2);
97. QTInfo qt4 = **new** QTInfo("=[]", T2 + "[" + T1 + "]", "-", T3);
98. QTInfo qt5 = **new** QTInfo("=", T3, "-", result); //
99. qtlist.addQTInfo(qt1);
100. qtlist.addQTInfo(qt2);
101. qtlist.addQTInfo(qt3);
102. qtlist.addQTInfo(qt4);
103. qtlist.addQTInfo(qt5);
104. }
105. )
106. |
107. (
108. arg1 = biaodashi()
109. {
110. System.out.println("case 1");
111. QTInfo qt = **new** QTInfo("=", arg1, "\_", result);
112. qtlist.addQTInfo(qt);
113. }
114. )
115. )
116. )
117. |
118. (
119. "++"
120. {
121. /\*
122. ("+",i,1,T1)
123. ("=",T1,\_,i)
124. \*/
125. Temp = VariableNameGenerator.genVariableName();
126. QTInfo qt01 = **new** QTInfo("+", result, "1", Temp);
127. QTInfo qt02 = **new** QTInfo("=", Temp, "\_", result);
128. qtlist.addQTInfo(qt01);
129. qtlist.addQTInfo(qt02);
130. }
131. )
132. |
133. (
134. "--"
135. {
136. /\*
137. ("-",i,1,T1)
138. ("=",T1,\_,i)
139. \*/
140. Temp = VariableNameGenerator.genVariableName();
141. QTInfo qt00 = **new** QTInfo("-", result, "1", Temp);
142. QTInfo qt11 = **new** QTInfo("=", Temp, "\_", result);
143. qtlist.addQTInfo(qt00);
144. qtlist.addQTInfo(qt11);
145. }
146. )
147. )?
148. }

### if条件语句语义分析

1. **void** tiaojianyuju() :
2. /\*条件语句    if (E) S1 else S2  或者 if (E) S1
3. 扫描完右括号之后就可以确定真出口；
4. 扫描完可以暂时确定假出口
5. else会产生一个无条件跳转语句
6. \*/
7. {
8. ConditionValue cv = **null**;
9. Token t = **null**;
10. **int** index = 0;
11. }
12. {
13. "if" "(" cv = Boolean() ")"
14. {
15. cv.backpatchTrueChain(QTInfo.size + 1);
16. }
17. SentenceBlock()
18. (
19. LOOKAHEAD(1) /\*防止if else 嵌套导致的冲突\*/
20. t = < ELSE >
21. {
22. QTInfo qt = **new** QTInfo("J", "-", "-", "-");
23. qtlist.addQTInfo(qt);
24. index = QTInfo.size;
25. cv.backpatchFalseChain(index + 1);
26. }
27. (
28. SentenceBlock()
29. {
30. qtlist.setResultbyIndex(index, QTInfo.size);
31. }
32. )
33. )?
34. {
35. **if** (t == **null**)
36. {
37. cv.backpatchFalseChain(QTInfo.size + 1);
38. }
39. }
40. }

### While循环语义分析

1. **void** xunhuan() :
2. {
3. **int** index = 0;
4. ConditionValue cValue = **new** ConditionValue();
5. String result = **null**;
6. String arg1 = **null**;
7. **int** indexE2 = 0;
8. **int** indexE3 = 0;
9. }
10. {
11. /\*while循环只产生一个四元式即无条件转移四元式，至于循环体内的四元式，不由while产生\*/
12. /\*
13. while (E) S1;
14. 真出口：S1; 即扫描完右括号后便知道真出口的位置。
15. 假出口：无条件转移之后
16. \*/
17. "while" "("
18. {
19. index = QTInfo.size + 1; //记录进入while循环的四元式位置
20. }
21. cValue = Boolean()
22. ")" /\*扫描完右括号，下一个就是循环体了，循环体的第一句也就是while语句的真出口，
23. 此时利用Boolean()语句返回的cvalue,回填真出口的位置，也就是QTInfo.size+1\*/
24. {
25. cValue.backpatchTrueChain(QTInfo.size + 1);
26. }
27. SentenceBlock()
28. /\*扫描完循环体语句即SentenceBlock()之后就该生成while的四元式，返回while语句开始的地方，即index四元式\*/
29. {
30. QTInfo qt = **new** QTInfo("j", "\_", "\_", index);
31. qtlist.addQTInfo(qt);
32. /\*假出口在此，while条件为假后就应该跳出while循环了，所以，此时QTInfo.size+1就是假出口的四元式编号了\*/
33. cValue.backpatchFalseChain(QTInfo.size + 1);
34. }
35. }

### 条件表达式语义分析

1. ConditionValue tiaojian() : /\*条件表达式：a<b-- >(<j<,a,b,T) (j<,-,-,F) 真假出口四元式\*/
2. {
3. String e1 = **null**;
4. String e2 = **null**;
5. String r = **null**;
6. ConditionValue cValue = **new** ConditionValue();
7. }
8. {
9. e1 = biaodashi()
10. (
11. r = guanxifu() e2 = biaodashi()
12. )?
13. {
14. **if** (r == **null**) /\*只有表达式(jnz,e1,-,T)\*/
15. {
16. QTInfo qt1 = **new** QTInfo("jnz", e1, "\_", "T");
17. qtlist.addQTInfo(qt1);
18. cValue.mergeTrue(qt1);
19. }
20. **else** /\*(j+r,e1,e2,T)\*/
21. {
22. QTInfo qt = **new** QTInfo("j" + r, e1, e2, "T");
23. qtlist.addQTInfo(qt);
24. cValue.mergeTrue(qt);
25. }
26. /\*无论如何都有一个假四元式(j,-,-,F)\*/
27. QTInfo qt = **new** QTInfo("j", "\_", "\_", "F");
28. qtlist.addQTInfo(qt);
29. cValue.mergeFalse(qt);
30. }
31. {
32. **return** cValue; /\*条件语句要返回一个条件语句的值ConditionValue\*/
33. }
34. }

### Expression语义程序

1. String Expression() :
2. {
3. String s = **null**;
4. }
5. {
6. s = AdditiveExpression()
7. {
8. **return** s;
9. }
10. }
12. String AdditiveExpression() : //附加表达式
13. {
14. String a = **null**;
15. String b = **null**;
16. String res = **null**;
17. Token t = **null**;
18. }
19. {
20. a = MultiplicativeExpression() /\*加法表达式    a+b+c -- >(+,a,b,T1) (+,T1,c,T2)\*/
21. {
22. res = a;
23. }
24. (
25. (
26. t = "+"
27. | t = "-"
28. )
29. b = MultiplicativeExpression()
30. {
31. res = VariableNameGenerator.genVariableName();
32. QTInfo qt = **new** QTInfo(t.image, a, b, res);
33. qtlist.addQTInfo(qt);
34. a = res;
35. }
36. )\*
37. {
38. **return** res;
39. }
40. }
42. String MultiplicativeExpression() : //乘法表达式 a\*b\*c-- >(\*,a,b,T1)(\*,T1,c,T2)
43. //QTInfo(String operator, String arg1, String arg2, String result)
44. {
45. String a = **null**;
46. String b = **null**;
47. String res = **null**;
48. Token t = **null**;
49. }
50. {
51. a = UnaryExpression()
52. {
53. res = a; /\*如果只有UnaryExpression()而没有后面的()\*,这时候返回的String值是单目运算表达式的返回String\*/
54. }
55. (
56. (
57. t = "\*"
58. | t = "/"
59. | t = "%"
60. )
61. b = UnaryExpression()
62. {
63. res = VariableNameGenerator.genVariableName();
64. QTInfo qt = **new** QTInfo(t.image, a, b, res);
65. qtlist.addQTInfo(qt);
66. a = res; /\*将res 赋给a,满足a\*b\*c的情况，返回两个四元式，即让下一次调用时，a=T1\*/
67. }
68. )\*
69. {
70. **return** res;
71. }
72. }
74. String UnaryExpression() : //一元表达式
75. {
76. String s = **null**;
77. }
78. {
79. "(" s = Expression() ")"
80. {
81. **return** s;
82. }
83. | s = Identifier()
84. {
85. **return** s;
86. }
87. | s = Integer()
88. {
89. **return** s;
90. }
91. | s = Float()
92. {
93. **return** s;
94. }
95. }

## 基本语义程序的测试

按照实习指导书中的四个测试样例，对本次实习的代码进行测试。测试结果如图2-图5所示。

### 测试样例1

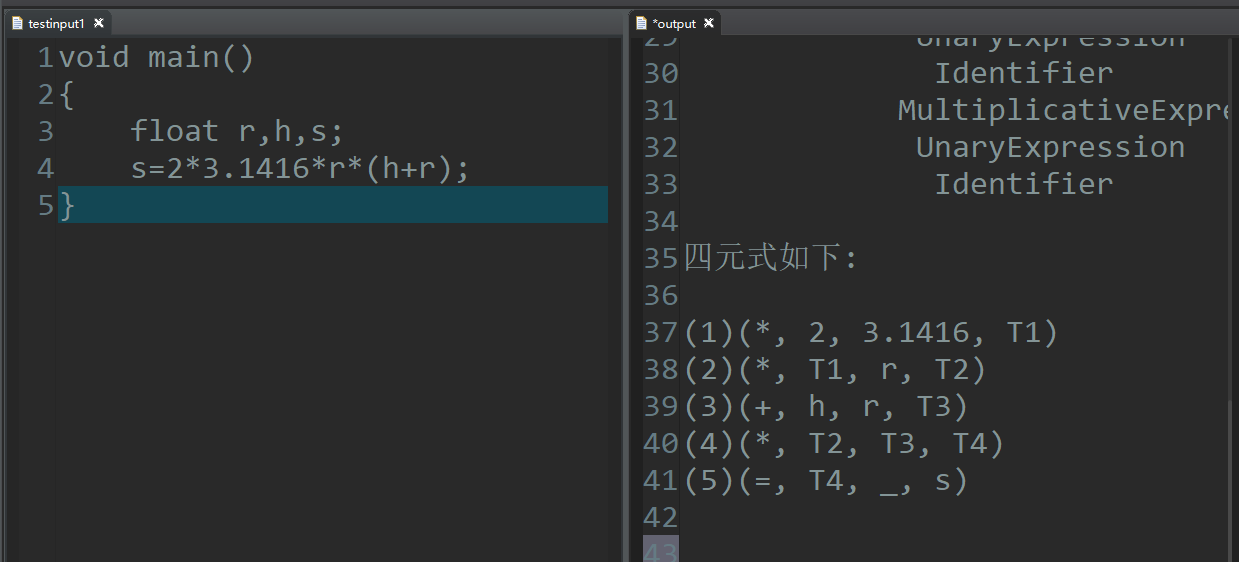


图2 测试样例1

（1）：2与3.1416的乘积存入T1

（2）：T1与r的乘积存入T2

（3）：h与r的和存入T3

（4）：T2与T3的和存入T4

（5）：将T4的值赋给s

### 测试样例2

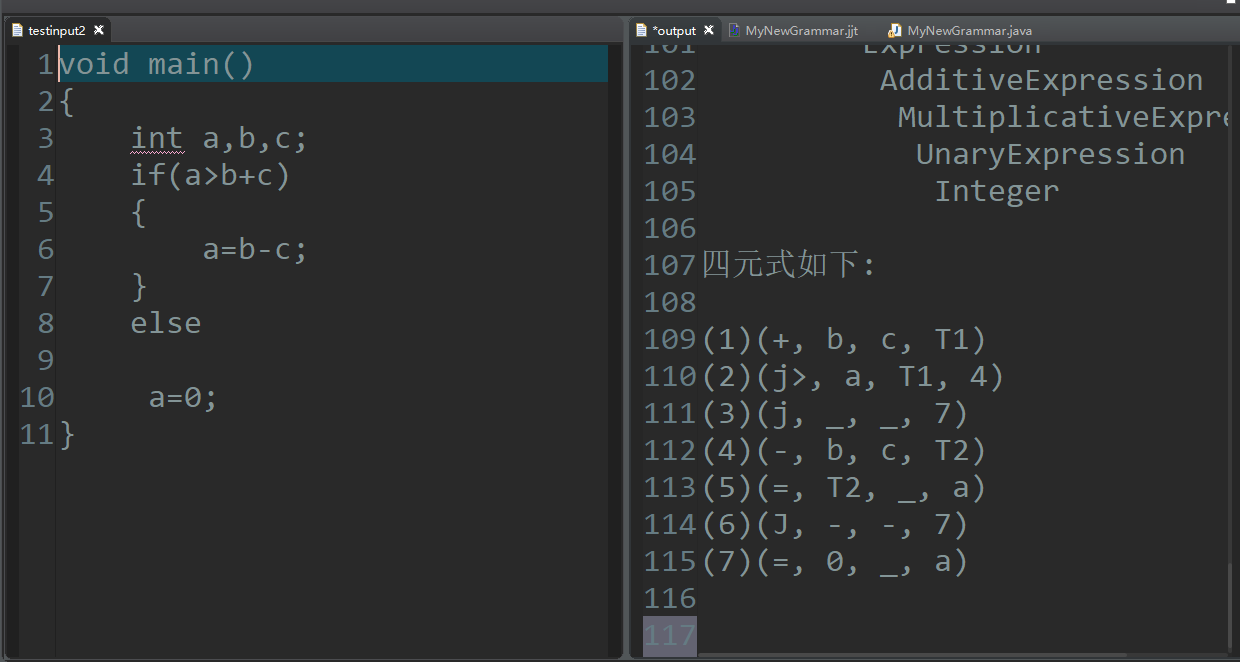


图3 测试样例2

（1）：b与c的和存入T1

（2）：如果a>T1就跳转到（4）号四元式

（3）：无条件跳转到（7）四元式

（4）：b与c的差存入T2

（5）：T2的值赋给a

（6）：无条件跳转到7

（7）：将0赋给a

### 测试样例3

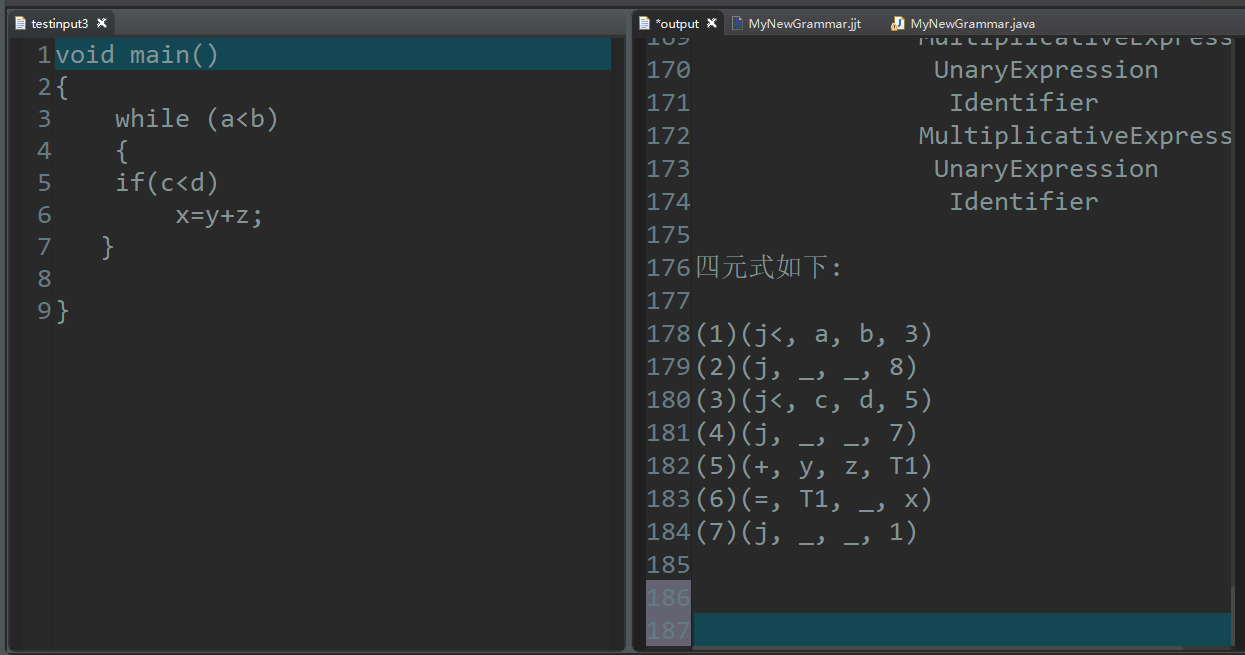


图4 测试样例3

（1）：如果a<b跳转到（3）号四元式

（2）：无条件跳转到（8）号四元式

（3）：如果c<d跳转到（5）号四元式

（4）：无条件跳转到（7）号四元式

（5）：y与z的和放入T1

（6）：将T1的值赋给x

（7）：无条件跳转回（1）号四元式

### 测试样例4

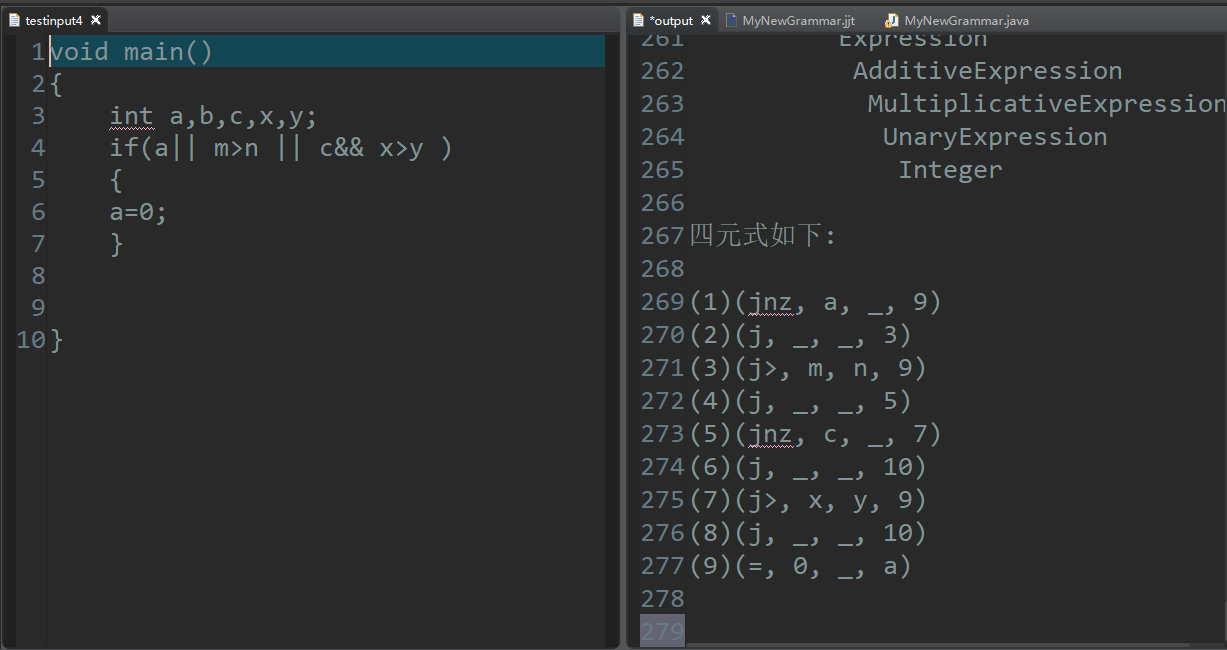


图5 测试样例4

（1）：如果a为真则跳转到（9）号四元式

（2）：无条件跳转到（3）号四元式

（3）：如果m>n则跳转到（9）号四元式

（4）：无条件跳转到（5）号四元式

（5）：如果c为真，则跳转到（7）号四元式

（6）：无条件跳转到（10）号四元式

（7）：如果x>y则跳转到（9）号四元式

（8）：无条件跳转到（10）号四元式

（9）：赋值0给a

## 扩展功能

除了实现了上述的基本的语法功能外，还实现了for循环，布尔表达式数组以及数组越界的纠错功能。详细信息见2.5.1-2.5.5.

### For循环的实现

For循环语句结构

1. **for**(E1;E2;E3){
2. S1
3. }

其中花括号中的循环体是可有可无，当没有花括号的时候，要在小括号后加分号。关于执行顺序，E2的真出口是S1假出口是S1+1， S1执行完后执行E3， E3执行后无条件跳转到E2。

循环语义分析程序中在while循环后增加如下程序语句

1. |
2. (
3. /\*
4. For循环的实现
5. for(E1;E2;E3)
6. {
7. S1
8. } \*/
9. < FOR >
10. < LRBRACKET >
11. /\*E1\*/
12. (
13. (
14. ("int")?
15. result = Identifier()
16. "=" arg1 = biaodashi()
17. {
18. QTInfo qt0 = **new** QTInfo("=", arg1, "\_", result);
19. qtlist.addQTInfo(qt0);
20. }
21. )?
22. ";"
23. )
24. /\*E2\*/
25. {
26. indexE2 = QTInfo.size + 1;
27. }
28. cValue = Boolean()
29. ";"
30. /\*E3
31. 执行完此语句之后要进行一个无条件跳转，跳转到E2判断for循环是否为真
32. (j,\_,\_,indexE2)
33. \*/
34. {
35. indexE3 = QTInfo.size + 1; //记录E3的位置
36. }
37. (
38. zizeng()
39. | fuzhi()
40. )
41. {
42. //扫描完E3要跳回E2
43. QTInfo qtE2 = **new** QTInfo("j", "\_", "\_", indexE2);
44. qtlist.addQTInfo(qtE2);
45. }
46. < RRBRACKET >
47. {
48. index = QTInfo.size + 1;
49. }
50. /\*S1\*/
51. (
52. (
53. "{"
54. { /\*扫描完 { 就知道for循环的真出口在下一句了，回填biaodashi的cValue的真出口\*/
55. cValue.backpatchTrueChain(QTInfo.size + 1);
56. }
57. (
58. SentenceBlock()
59. )?
60. { /\*扫描完Senten之后要产生无条件跳转语句，跳转到E3处\*/
61. QTInfo qtE3 = **new** QTInfo("j", "\_", "\_", indexE3);
62. qtlist.addQTInfo(qtE3);
63. }
64. "}"
65. { /\*扫描完 }就知道for循环的假出口在下一句了，回填biaodashi的cValue的假出口\*/
66. cValue.backpatchFalseChain(QTInfo.size + 1);
67. }
68. )
69. |
70. (
71. ";"
72. { /\*扫描完 ； 就知道for循环的真出口在E3了，回填biaodashi的cValue的真出口\*/
73. cValue.backpatchTrueChain(indexE3);
74. cValue.backpatchFalseChain(QTInfo.size + 1);
75. }
76. )
77. )
78. )

### 布尔表表达式的实现

布尔表达式中有四种运算符，关系符、与、或、非（> < >= <= != == && || !），为了实现上述复杂的布尔表达式的翻译，定义了三个语法语义程序：

BaiscBoolean()，最简单的布尔表达式即包含非（！）操作的条件表达式

YuBoolean()，与（&&）结合起来的布尔表达式，由BasicBoolen()+&&组成

Boolean()，或（||）结合起来的布尔表达式，由YuBoolean()+||组成

因为运算优先级 rop >! >&& >|| ，所以Boolean()即是最后可以匹配布尔表达式语法。另外，由于此处定义的布尔表达式包含了条件表达式，要想让if(),while()等语句支持布尔表达式，所以要将相应的语法程序中的tiaojian()替换为Boolean()。BasicBoolean(),YuBoolean与Boolean()的实现如下：

1. ConditionValue BasicBoolean() :
2. {
3. ConditionValue cv = **new** ConditionValue();
4. ConditionValue cvt = **null**;
5. Token test = **null**;
6. }
7. {
8. (
9. (
10. test = < FEI >
11. (
12. < LRBRACKET >
13. cvt = tiaojian()
14. < RRBRACKET >
15. )
16. )
17. |
18. (
19. cvt = tiaojian()
20. )
21. )
22. {
23. **if** (test == **null**)
24. {
25. cv.mergeFalse(cvt);
26. cv.mergeTrue(cvt);
27. }
28. **else** //有 ！ 号，要采用不同的操作merge真假串
29. {
30. cv.FEImerge(cvt);
31. }
32. }
33. {
34. **return** cv;
35. }
36. }
38. ConditionValue YuBoolean() :
39. {
40. ConditionValue cv3 = **new** ConditionValue();
41. ConditionValue cv2 = **null**;
42. ConditionValue cv1 = **null**;
43. Token test = **null**;
44. }
45. {
46. cv1 = BasicBoolean()
47. {
48. cv3.mergeFalse(cv1); //cv1的假出口即cv3的假出口
49. }
50. (
51. test = < YU >
52. {
53. cv1.backpatchTrueChain(QTInfo.size + 1); //E&&M 当E为真时，要哦继续计算M的真假，所以E的真出口在M处
54. }
55. cv2 = YuBoolean()
56. {
57. cv3.mergeFalse(cv2);
58. }
59. )?
60. {
61. **if** (cv2 == **null**) //如果没有YU
62. {
63. cv3 = cv1; //直接返回CV1即可
64. }
65. **else**
66. {
67. cv3.mergeTrue(cv2); //cv2的真链merge到cv3中
68. }
69. }
70. {
71. **return** cv3;
72. }
73. }
75. ConditionValue Boolean() :
76. {
77. ConditionValue cv1 = **null**; //第一个简单布尔表达式
78. ConditionValue cv2 = **null**; //&&号后面的布尔表达式
79. ConditionValue cv3 = **new** ConditionValue(); //整个YuBoolean表达式的condationvalue的值
80. Token test = **null**;
81. }
82. {
83. cv1 = YuBoolean()
84. {
85. cv3.mergeTrue(cv1);
86. }
87. (
88. test = < HUO >
89. {
90. cv1.backpatchFalseChain(QTInfo.size + 1);
91. }
92. cv2 = Boolean()
93. {
94. cv3.mergeTrue(cv2);
95. }
96. )?
97. {
98. **if** (test == **null**)
99. {
100. cv3 = cv1;
101. }
102. **else**
103. {
104. cv3.mergeFalse(cv2);
105. }
106. }
107. {
108. **return** cv3;
109. }
110. }

### 数组赋值与定义的实现

数组的实现比较简单，用基址+变址来表示数组元素的内存地址，本实习中按照“按行存放”的规则计算数组元素的内存地址。定义数组的四元式为：变址存数，( []=, X, \_ , T[T1] );变址取数，( =[] , T[T1] , \_ , X ).

数组定义程序如下：

1. **void** shengming() :
2. {
3. String arg1 = **null**;
4. String result = **null**;
5. String raw = ""+ 1;
6. String col = ""+ 1;
7. }
8. {
9. (
10. "int"
11. | "double"
12. | "float"
13. | "char"
14. )
15. result = Identifier() //int a=12;
16. (
17. (
18. "=" arg1 = biaodashi()
19. {
20. QTInfo qt0 = **new** QTInfo("=", arg1, "\_", result);
21. qtlist.addQTInfo(qt0);
22. }
23. )
24. |
25. (
26. "["
27. (
28. col = Integer()
29. | col = Identifier()
30. )
31. "]"
32. (
33. "["
34. (
35. raw = Integer()
36. | raw = Identifier()
37. )
38. "]"
39. )?
40. {
41. arrmap.createArray(result, raw, col);
42. }
43. )
44. )?
45. (
46. "," result = Identifier()
47. (
48. (
49. "=" arg1 = biaodashi()
50. {
51. QTInfo qt = **new** QTInfo("=", arg1, "\_", result);
52. qtlist.addQTInfo(qt);
53. }
54. )
55. |
56. (
57. "["
58. (
59. col = Integer()
60. | col = Identifier()
61. )
62. "]"
63. (
64. "["
65. (
66. raw = Integer()
67. | raw = Identifier()
68. )
69. "]"
70. )?
71. {
72. //将新创建的数组信息加入map中，并加入纠错提示
73. **if**(!arrmap.createArray(result, raw, col))
74. {
75. System.out.println("重复定义");
76. **throw** **new** Exception();
77. }
78. }
79. )
80. )?
81. )\*
82. }

赋值定义程序如下：

1. **void** fuzhi() :
2. {
3. String arg1 = **null**;
4. String result = **null**;
5. String Temp = **null**;
6. String arrname = **null**;
7. String T1 = **null**;
8. String T2 = **null**;
9. String T3 = **null**;
10. String acol = ""+ 1;
11. String araw = ""+ 1;
12. String col = ""+ 1;
13. String raw = ""+ 1;
14. }
15. {
16. //(=,biaodashi,\_,biaoshifu)
17. //QTInfo(String operator, String arg1, String arg2, String result)
18. result = Identifier()
19. (
20. "["
21. (
22. col = Integer()
23. | col = Identifier()
24. )
25. "]"
26. (
27. "["
28. (
29. raw = Integer()
30. | raw = Identifier()
31. )
32. "]"
33. )?
34. {
35. System.out.println("到此！"+result);
36. }
37. "=" arg1 = biaodashi()
38. {
39. **if** (!arrmap.containsKey(result))
40. {
41. System.out.println("\u6570\u7ec4\u672a\u5b9a\u4e49");
42. **throw** **new** NoSuchFieldException("\u6570\u7ec4\u672a\u5b9a\u4e49\uff01");
43. }
44. acol = arrmap.getcol(result);
45. araw = arrmap.getraw(result); //获取数组的大小（这里是横向存储的数组）
46. T1 = VariableNameGenerator.genVariableName();
47. T2 = VariableNameGenerator.genVariableName();
48. T3 = VariableNameGenerator.genVariableName();
49. QTInfo qt1 = **new** QTInfo("\*", col, acol, T1);
50. QTInfo qt2 = **new** QTInfo("+", raw, T1, T1);
51. Temp = String.valueOf(Integer.parseInt(acol) + 1);
52. QTInfo qt3 = **new** QTInfo("-", result, Temp, T2);
53. QTInfo qt4 = **new** QTInfo("[]=", arg1, "-", T2 + "[" + T1 + "]");
54. qtlist.addQTInfo(qt1);
55. qtlist.addQTInfo(qt2);
56. qtlist.addQTInfo(qt3);
57. qtlist.addQTInfo(qt4);
58. }
59. )?
60. (
61. (
62. "="
63. (
64. LOOKAHEAD(2)
65. ( //将数组的值赋给标识符“ a = b[1][2]
66. arrname = Identifier()
67. "["
68. (
69. col = Integer()
70. | col = Identifier()
71. )
72. "]"
73. (
74. "["
75. (
76. raw = Integer()
77. | raw = Identifier()
78. )
79. "]"
80. )?
81. {
82. **if** (!arrmap.containsKey(arrname))
83. {
84. System.out.println("\u6570\u7ec4\u672a\u5b9a\u4e49");
85. **throw** **new** NoSuchFieldException("\u6570\u7ec4\u672a\u5b9a\u4e49\uff01");
86. }
87. System.out.println("case 1");
88. T1 = VariableNameGenerator.genVariableName();
89. acol = arrmap.getcol(arrname);
90. araw = arrmap.getraw(arrname); //获取数组的大小（这里是横向存储的数组）
91. QTInfo qt1 = **new** QTInfo("\*", col, acol, T1);
92. T2 = VariableNameGenerator.genVariableName();
93. T3 = VariableNameGenerator.genVariableName();
94. QTInfo qt2 = **new** QTInfo("+", raw, T1, T1);
95. Temp = String.valueOf(Integer.parseInt(acol) + 1);
96. QTInfo qt3 = **new** QTInfo("-", arrname, Temp, T2);
97. QTInfo qt4 = **new** QTInfo("=[]", T2 + "[" + T1 + "]", "-", T3);
98. QTInfo qt5 = **new** QTInfo("=", T3, "-", result); //
99. qtlist.addQTInfo(qt1);
100. qtlist.addQTInfo(qt2);
101. qtlist.addQTInfo(qt3);
102. qtlist.addQTInfo(qt4);
103. qtlist.addQTInfo(qt5);
104. }
105. )
106. |
107. (
108. arg1 = biaodashi()
109. {
110. System.out.println("case 1");
111. QTInfo qt = **new** QTInfo("=", arg1, "\_", result);
112. qtlist.addQTInfo(qt);
113. }
114. )
115. )
116. )
117. |
118. (
119. "++"
120. {
121. /\*
122. ("+",i,1,T1)
123. ("=",T1,\_,i)
124. \*/
125. Temp = VariableNameGenerator.genVariableName();
126. QTInfo qt01 = **new** QTInfo("+", result, "1", Temp);
127. QTInfo qt02 = **new** QTInfo("=", Temp, "\_", result);
128. qtlist.addQTInfo(qt01);
129. qtlist.addQTInfo(qt02);
130. }
131. )
132. |
133. (
134. "--"
135. {
136. /\*
137. ("-",i,1,T1)
138. ("=",T1,\_,i)
139. \*/
140. Temp = VariableNameGenerator.genVariableName();
141. QTInfo qt00 = **new** QTInfo("-", result, "1", Temp);
142. QTInfo qt11 = **new** QTInfo("=", Temp, "\_", result);
143. qtlist.addQTInfo(qt00);
144. qtlist.addQTInfo(qt11);
145. }
146. )
147. )?
148. }

### 扩展测试1

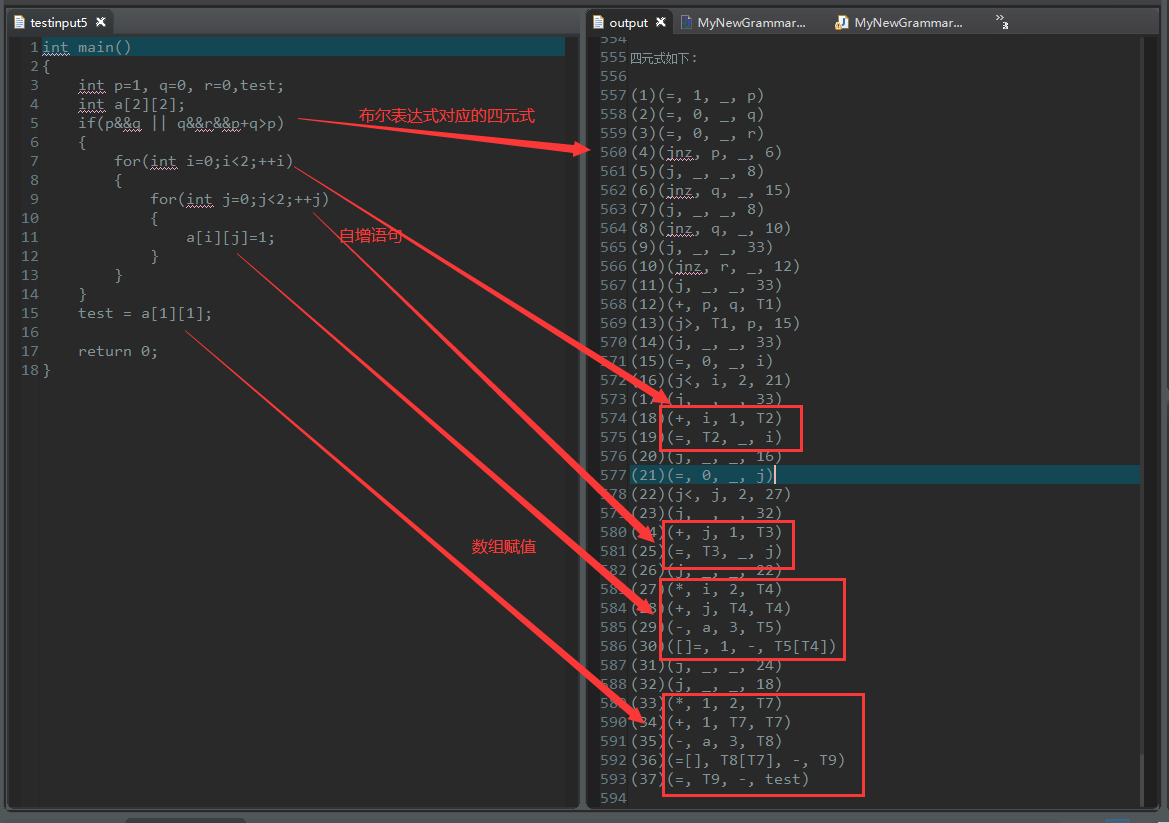


图6 扩展测试1

### 扩展测试2

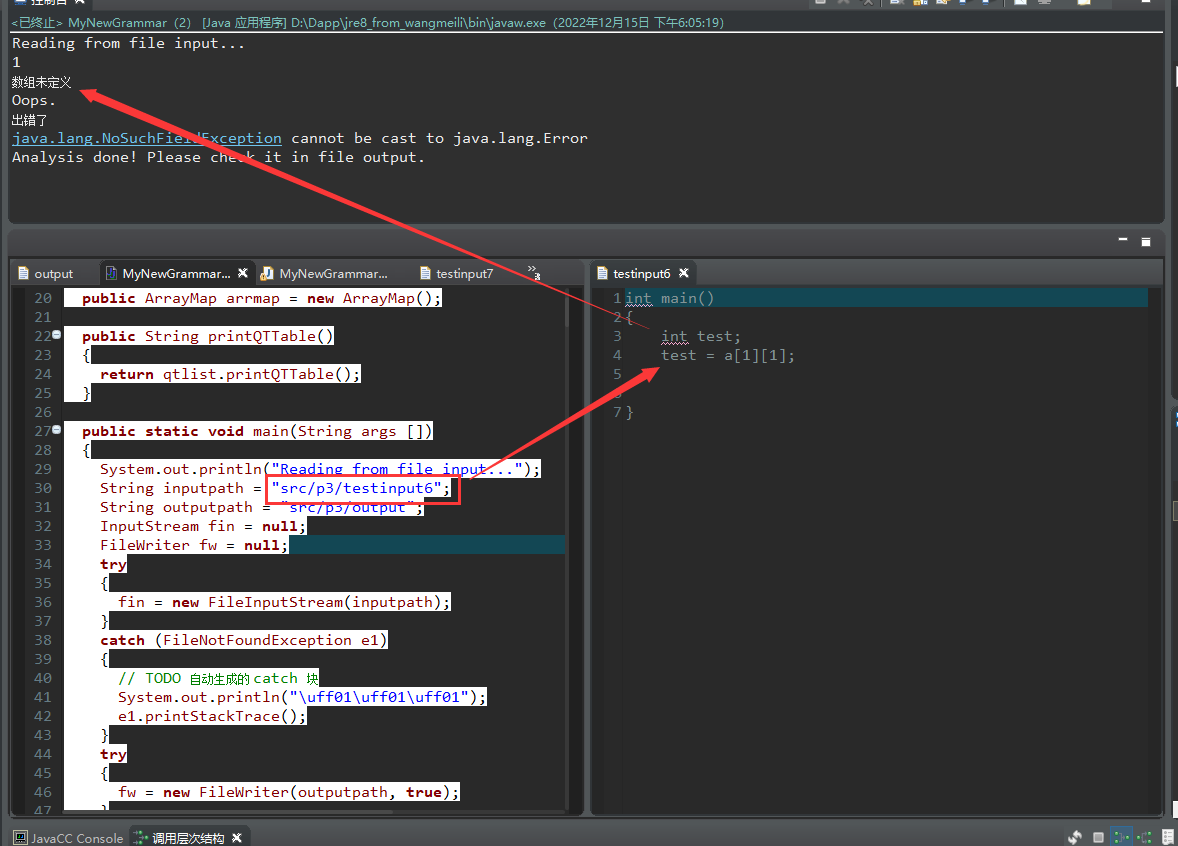


图7 扩展测试2-数组未定义

# 疑问和建议

## 疑问

老师视频中原理与操作讲解的十分详细，没有疑问的地方。但是在实现数组的纠错时， 我只实现了数组是否定义的纠错算法。至于在访问存取数组时是否越界，却没有实现，如果访问时给出的地址是确定的Integer()则很容易判断是否越界，但是，如果给出的是一个变量Identifier()，则需要在运行时才能判断变量的值具体是多少，那这样在运行时才能判断的纠错机制是可以实现的吗？

## 建议

老师上课生动有趣，知识讲解清楚到位，且喜欢与学生交流，及时发现问题所在并解决问题。建议就是希望老师能够保持这样的优良传统，永远年轻，永远活泼！

最后感谢老师在本学期的编译原理学习过程中的悉心指导与帮助！