中山大学数据科学与计算机学院本科生实验报告

课程名称：编译器构造实验 任课教师：陈炬桦

|  |  |  |  |
| --- | --- | --- | --- |
| 学年度 | 2018-2019 | 学期 | 第二学期 |
| 年级 | 16级 | 专业（方向） | 计算机科学与技术 |
| 学号 | 16337341 | 姓名 | 朱志儒 |
| 电话 | 15989184223 | Email | 739741104@qq.com |
| 开始日期 | 2019/5/17 | 完成日期 | 2019/5/17 |

1. 实验题目

**LR(K) 语法分析程序**

* 1. Description

输入开始符号，非终结符，终结符，产生式  
输出LR(k)优先分析过程  
以拓广算术表达式G[A]: 为例  
A→E  
E→E+T | T  
T→T\*F | F  
F→(E) | a

* 1. Input

非终结符个数，非终结符，空格符分隔；  
终结符个数，终结符，空格符分隔；  
产生式的个数,各产生式的序号,产生式的左边和右边符号,空格符分隔；  
状态数，ACTION列数，GOTO列数，空格符分隔；  
状态，ACTION矩阵(k 0 表示空 A 0 表示接收)，GOTO矩阵(0表示 空)，空格符分隔；  
输入分析字符串，#结束

* 1. Output

用“ & ”分隔，左边表示栈底到栈顶符号；右边表示尚未分析的字符串。

1. 算法描述（介绍程序模块功能，流程图）

**算法：**

分析程序的构型由两部分组成：栈中的符号串和尚待扫描的输入串。分析程序的工作过程就是从一种构型到另一种构型的转换过程。分析程序的下一次移动就是有栈顶状态s和当前输入符号a去查看ACTION表并执行ACTION[s, a]规定的动作所唯一确定。具体来说，共有4种不同的动作，如下所示：

(1) 若移进，则分析程序执行一个移进动作，将输入符号和下一状态移进栈，变成新的当前输入符号，变成栈顶状态。

(2) 若归约，则分析程序执行一个归约操作，从栈中弹出2r个符号，使得成为一个暂时的栈顶状态，再将产生式左部符号A和状态下推进栈，s变成栈顶状态，对输入串不做任何改变。

(3) 若接收，则表示分析工作已经正常完成，应停止分析程序的工作。

(4) 若，则表示分析程序已发现输入串有错，需要进行出错处理。

1. 测试数据（2组）

|  |  |  |
| --- | --- | --- |
|  | 输入 | 输出 |
| 第一组 | A  4 E T F A  6 a + \* ( ) #  7  0 A E  1 E E+T  2 E T  3 T T\*F  4 T F  5 F (E)  6 F a  12 6 3  0 s 5 k 0 k 0 s 4 k 0 k 0 1 2 3  1 k 0 s 6 k 0 k 0 k 0 A 0 0 0 0  2 k 0 r 2 s 7 k 0 r 2 r 2 0 0 0  3 k 0 r 4 r 4 k 0 r 4 r 4 0 0 0  4 s 5 k 0 k 0 s 4 k 0 k 0 8 2 3  5 k 0 r 6 r 6 k 0 r 6 r 6 0 0 0  6 s 5 k 0 k 0 s 4 k 0 k 0 0 9 3  7 s 5 k 0 k 0 s 4 k 0 k 0 0 0 10  8 k 0 s 6 k 0 k 0 s 11 k 0 0 0 0  9 k 0 r 1 s 7 k 0 r 1 r 1 0 0 0  10 k 0 r 3 r 3 k 0 r 3 r 3 0 0 0  11 k 0 r 5 r 5 k 0 r 5 r 5 0 0 0  (a+a)\*a# | #0 & (a+a)\*a#  #0(4 & a+a)\*a#  #0(4a5 & +a)\*a#  #0(4F3 & +a)\*a#  #0(4T2 & +a)\*a#  #0(4E8 & +a)\*a#  #0(4E8+6 & a)\*a#  #0(4E8+6a5 & )\*a#  #0(4E8+6F3 & )\*a#  #0(4E8+6T9 & )\*a#  #0(4E8 & )\*a#  #0(4E8)11 & \*a#  #0F3 & \*a#  #0T2 & \*a#  #0T2\*7 & a#  #0T2\*7a5 & #  #0T2\*7F10 & #  #0T2 & #  #0E1 & # |
| 第二组 | A  4 E T F A  6 a + \* ( ) #  7  0 A E  1 E E+T  2 E T  3 T T\*F  4 T F  5 F (E)  6 F a  12 6 3  0 s 5 k 0 k 0 s 4 k 0 k 0 1 2 3  1 k 0 s 6 k 0 k 0 k 0 A 0 0 0 0  2 k 0 r 2 s 7 k 0 r 2 r 2 0 0 0  3 k 0 r 4 r 4 k 0 r 4 r 4 0 0 0  4 s 5 k 0 k 0 s 4 k 0 k 0 8 2 3  5 k 0 r 6 r 6 k 0 r 6 r 6 0 0 0  6 s 5 k 0 k 0 s 4 k 0 k 0 0 9 3  7 s 5 k 0 k 0 s 4 k 0 k 0 0 0 10  8 k 0 s 6 k 0 k 0 s 11 k 0 0 0 0  9 k 0 r 1 s 7 k 0 r 1 r 1 0 0 0  10 k 0 r 3 r 3 k 0 r 3 r 3 0 0 0  11 k 0 r 5 r 5 k 0 r 5 r 5 0 0 0  (a+a)\*(a+a)# | #0 & (a+a)\*(a+a)#  #0(4 & a+a)\*(a+a)#  #0(4a5 & +a)\*(a+a)#  #0(4F3 & +a)\*(a+a)#  #0(4T2 & +a)\*(a+a)#  #0(4E8 & +a)\*(a+a)#  #0(4E8+6 & a)\*(a+a)#  #0(4E8+6a5 & )\*(a+a)#  #0(4E8+6F3 & )\*(a+a)#  #0(4E8+6T9 & )\*(a+a)#  #0(4E8 & )\*(a+a)#  #0(4E8)11 & \*(a+a)#  #0F3 & \*(a+a)#  #0T2 & \*(a+a)#  #0T2\*7 & (a+a)#  #0T2\*7(4 & a+a)#  #0T2\*7(4a5 & +a)#  #0T2\*7(4F3 & +a)#  #0T2\*7(4T2 & +a)#  #0T2\*7(4E8 & +a)#  #0T2\*7(4E8+6 & a)#  #0T2\*7(4E8+6a5 & )#  #0T2\*7(4E8+6F3 & )#  #0T2\*7(4E8+6T9 & )#  #0T2\*7(4E8 & )#  #0T2\*7(4E8)11 & #  #0T2\*7F10 & #  #0T2 & #  #0E1 & # |

1. 程序清单
2. #include <iostream>
3. #include <string>
4. **using** **namespace** std;
6. **char** s;
7. **struct** VN {
8. **char** vns[100];
9. **int** num;
10. } vn;
11. **struct** VT {
12. **char** vts[100];
13. **int** num;
14. } vt;
15. **struct** PS {
16. **int** num;
17. **char** left[100];
18. string right[100];
19. } ps;
20. **struct** TABLE {
21. **int** rows;
22. **int** action\_column;
23. **int** goto\_column;
24. **char** action[100][100];
25. **int** action\_num[100][100];
26. **int** goto\_num[100][100];
27. } table;
28. string expresssion;
30. **int** find\_vt\_index(**char** x) {
31. **for** (**int** i = 0; i < vt.num; ++i)
32. **if** (vt.vts[i] == x)
33. **return** i;
34. **return** -1;
35. }
37. **int** find\_vn\_index(**char** x) {
38. **for** (**int** i = 0; i < vn.num; ++i)
39. **if** (vn.vns[i] == x)
40. **return** i;
41. **return** -1;
42. }
44. **int** main() {
45. cin >> s;
46. cin >> vn.num;
47. **for** (**int** i = 0; i < vn.num; ++i)
48. cin >> vn.vns[i];
49. cin >> vt.num;
50. **for** (**int** i = 0; i < vt.num; ++i)
51. cin >> vt.vts[i];
52. cin >> ps.num;
53. **for** (**int** i = 0; i < ps.num; ++i)
54. cin >> ps.left[i] >> ps.left[i] >> ps.right[i];
55. cin >> table.rows >> table.action\_column >> table.goto\_column;
56. **for** (**int** i = 0; i < table.rows; ++i) {
57. cin >> table.action\_num[i][0];
58. **for** (**int** j = 0; j < table.action\_column + table.goto\_column; ++j) {
59. **if** (j < table.action\_column)
60. cin >> table.action[i][j] >> table.action\_num[i][j];
61. **else**
62. cin >> table.goto\_num[i][j - table.action\_column];
63. }
64. }
65. cin >> expresssion;
66. string stack = "#0";
67. **while** (**true**) {
68. cout << stack.substr(0, stack.size() - 1) << stack[stack.size() - 1] - '0' << " & " << expresssion << endl;
69. **int** state = stack[stack.size() - 1] - '0';
70. **int** index = find\_vt\_index(expresssion[0]);
71. **char** action = table.action[state][index];
72. **int** no = table.action\_num[state][index];
73. **switch** (action) {
74. **case** 'A':
75. **return** 0;
76. **case** 's':
77. stack.push\_back(expresssion[0]);
78. stack.push\_back('0' + no);
79. expresssion = expresssion.substr(1);
80. **break**;
81. **case** 'r':
82. **int** i = stack.size() - 2;
83. string pop = string(1, stack[i]);
84. **while** (ps.right[no] != pop) {
85. i -= 2;
86. pop.insert(pop.begin(), stack[i]);
87. }
88. stack = stack.substr(0, i);
89. stack.push\_back(ps.left[no]);
90. stack.push\_back('0' + table.goto\_num[stack[i - 1] - '0'][find\_vn\_index(ps.left[no])]);
91. **break**;
92. }
93. }
94. }