**软件学院实践报告（编译原理）**

**课程编号:** 3152100131 **实践课程名称：编译原理 学年：2018-2019 学期：秋**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **学生姓名** | | 田宇 | **学号** | 2016212011 |
| **指导教师姓名** | | 修佳鹏 | 完成时间 | 2018年12月20日 |
| **实验名称** | | 语法分析器 | | |
| **实验目的** | 掌握语法分析的基础知识，对语法分析过程能够实现具体实现方法 | | | |
| **实验要求** | 编写语法分析程序，实现对算法表达式的语法分析。要求所分析算数表达式由如下的文法产生。  E -> E+T | E-T | T  T -> T\*F | T/F | F  F -> id | (E) | num | | | |
| **实验环境** | Dev C++  使用C语言编写 | | | |
| **设计方案** | 1. 根据LR分析方法设计了如下功能的变量： 2. 移入或规约时压入分析栈的符号、移入或规约操作后转而进入的状态 3. 定义分析栈 4. 设计了如下具有功能的函数： 5. void TRANS();//将读入的buffer数组内容按字符转换为相应代号存入code数组中 6. void GetFromCode();//取得当前输入符号串的元素 7. void PUSH(int A, int S);//入栈操作 8. void POP();//出栈操作 9. void SHITF();//移入操作 10. void REDUCE();//规约操作 11. void ACC();//接受操作 12. void E1();//错误处理 13. 定义了一个结构体便于进行操作： 14. typedef struct 15. { 16. /\*定义文法产生式的结构\*/ 17. int Vn;/\*文法产生式的左部非终结符的代码\*/ 18. int Str[4];/\*文法产生式右部的代码串\*/ 19. int size;/\*文法产生式右部的长度\*/ 20. }G; 21. 设计了三个表： 22. 文法产生式表： 23. G production[11] = {/\*该文法的所有产生式\*/ 24. {0}, 25. {101,{0,102},1},/\*S -> E\*/ 26. {102,{0,102,3,103},3},/\*E -> E+T\*/ 27. {102,{0,102,4,103},3},/\*E -> E-T\*/ 28. {102,{0,103},1},/\*E -> T\*/ 29. {103,{0,103,5,104},3},/\*T -> T\*F\*/ 30. {103,{0,103,6,104},3},/\*T -> T/F\*/ 31. {103,{0,104},1},/\*T -> F\*/ 32. {104,{0,7},1},/\*F -> id\*/ 33. {104,{0,1,101,2},3},/\*F -> (E)\*/ 34. {104,{0,8},1},/\*F -> num\*/ 35. }; 36. LR分析表goto： 37. int GOTO[17][5] 38. LR分析表action： 39. int ACTION[17][9] | | | |
| **成源程序** | 1. /\* 2. \*语法分析器 3. \* 4. \* 5. \*By Tian Yu 6. \*2018 12 20 7. \*/ 8. #include<stdlib.h> 9. #include<stdio.h> 10. #include<string.h> 11. /\*移入或规约时压入分析栈的符号、移入或规约操作后转而进入的状态\*/ 12. int symbol,status; 13. /\*定义分析栈\*/ 14. int analysis\_stack [50]; 15. /\*ip为栈顶元素下标\*/ 16. int ip = -1; 17. /\*将输入符号串w保存于数组中\*/ 18. char buffer[30]; 19. /\*将读入的字符串转换为代号形式\*/ 20. int code[30]; 21. /\*position为code数组的下标\*/ 22. int position = 0; 23. /\*X为当前获取的输入符号的代号\*/ 24. int X; 25. /\*循环标志\*/ 26. int flag = 1; 27. typedef struct{/\*定义文法产生式的结构\*/ 28. int Vn;/\*文法产生式的左部非终结符的代码\*/ 29. int Str[4];/\*文法产生式右部的代码串\*/ 30. int size;/\*文法产生式右部的长度\*/ 31. }G; 32. G production[11] = {/\*该文法的所有产生式\*/ 33. {0}, 34. {101,{0,102},1},/\*S -> E\*/ 35. {102,{0,102,3,103},3},/\*E -> E+T\*/ 36. {102,{0,102,4,103},3},/\*E -> E-T\*/ 37. {102,{0,103},1},/\*E -> T\*/ 38. {103,{0,103,5,104},3},/\*T -> T\*F\*/ 39. {103,{0,103,6,104},3},/\*T -> T/F\*/ 40. {103,{0,104},1},/\*T -> F\*/ 41. {104,{0,7},1},/\*F -> id\*/ 42. {104,{0,1,101,2},3},/\*F -> (E)\*/ 43. {104,{0,8},1},/\*F -> num\*/ 44. }; 45. int GOTO[17][5] = {/\*LR分析表goto\*/ 46. {0,0,1,2,3},/\*0\*/ 47. {0,0,0,0,0},/\*1\*/ 48. {0,0,0,0,0},/\*2\*/ 49. {0,0,0,0,0},/\*3\*/ 50. {0,0,0,0,0},/\*4\*/ 51. {0,0,11,2,3},/\*5\*/ 52. {0,0,0,0,0},/\*6\*/ 53. {0,0,0,12,3},/\*7\*/ 54. {0,0,0,13,3},/\*8\*/ 55. {0,0,0,0,14},/\*9\*/ 56. {0,0,0,0,15},/\*10\*/ 57. {0,0,0,0,0},/\*11\*/ 58. {0,0,0,0,0},/\*12\*/ 59. {0,0,0,0,0},/\*13\*/ 60. {0,0,0,0,0},/\*14\*/ 61. {0,0,0,0,0},/\*15\*/ 62. {0,0,0,0,0}/\*16\*/ 63. }; 64. int ACTION[17][9] = {/\*LR分析表action\*/ 65. {51,5,52,51,51,51,51,4,6},//0 66. {999,53,52,7,8,0,0,53,53},//1 67. {-4,53,-4,-4,-4,9,10,53,53},//2 68. {-7,0,-7,-7,-7,-7,-7,0,0},//3 69. {-8,0,-8,-8,-8,-8,-8,0,0},//4 70. {0,5,52,51,51,51,51,4,6},//5 71. {-10,0,-10,-10,-10,-10,-10,0,0},//6 72. {51,5,52,51,51,51,51,4,6},//7 73. {51,5,52,51,51,51,51,4,6},//8 74. {51,5,52,51,51,51,51,4,6},//9 75. {51,5,52,51,51,51,51,4,6},//10 76. {0,53,16,7,8,0,0,53,53},//11 77. {-2,53,-2,-2,-2,9,10,53,53},//12 78. {-3,53,-3,-3,-3,9,10,53,53},//13 79. {-5,0,-5,-5,-5,-5,-5,0,0},//14 80. {-6,0,-6,-6,-6,-6,-6,0,0},//15 81. {-9,0,-9,-9,-9,-9,-9,0,0}//16 82. }; 83. void TRANS();/\*将读入的buffer数组内容按字符转换为相应代号存入code数组中\*/ 84. void GetFromCode();/\*取得当前输入符号串的元素\*/ 85. void PUSH(int A,int S);/\*入栈操作\*/ 86. void POP();/\*出栈操作\*/ 87. void SHITF();/\*移入操作\*/ 88. void REDUCE();/\*规约操作\*/ 89. void ACC();/\*接受操作\*/ 90. void E1();/\*错误处理\*/ 91. void E2(); 92. void E3(); 93. main() 94. { 95. int c=1; 96. printf("请输入表达式,并以$结尾。例如：'a+b$'\n"); 97. scanf("%s",buffer);/\*读入算术表达式，保存至buffer中\*/ 98. printf("\n"); 99. TRANS();/\*将字符型的buffer数组内容以字符为单位，转换为相应代号，保存至整型数组w中\*/ 101. PUSH(0,0);/\*状态S0入栈\*/ 102. GetFromCode();//从输入缓冲区读取一个字符 104. while(flag){ 105. if( ACTION[analysis\_stack[ip]][X] > 0 && ACTION[analysis\_stack[ip]][X] <50 ){/\*进行移入操作\*/ 106. SHITF(); 107. GetFromCode(); 108. } 109. else if( ACTION[analysis\_stack [ip]][X] < 0 ){/\*进行规约操作\*/ 110. REDUCE(); 111. } 112. else if( ACTION[analysis\_stack[ip]][X] == 999 ){/\*分析成功\*/ 113. ACC(); 114. } 115. else 116. { 117. switch(ACTION[analysis\_stack[ip]][X]) 118. { 119. case 51: E1();break; 120. case 52: E2();GetFromCode();break; 121. case 53: E3();GetFromCode();break; 122. } 123. } 124. } 126. system("pause"); 127. return 0; 128. } 129. void TRANS(){/\*将读入的buffer数组内容按字符转换为相应代号存入w数组中\*/ 130. int x=0,y=0;/\*x为指向buffer数组的下标 \*/ 131. while(buffer[x]!='$'){ 132. if( buffer[x]>=48 && buffer[x]<=57 ){/\*当前字符为数字，即num\*/ 133. while( buffer[x]>=48 && buffer[x]<=57 ) x++; 134. code[y++] = 8;/\*在buffer的替身数组w中写入代表num的8,并且y后移\*/ 135. } 136. else if( (buffer[x]>=97 && buffer[x]<=122) || (buffer[x]>=65 && buffer[x]<=90) ){/\*当前字符为字母，即id\*/ 137. while( (buffer[x]>=97 && buffer[x]<=122) || (buffer[x]>=65 && buffer[x]<=90) ) x++; 138. code[y++] = 7;/\*在buffer的替身数组w中写入代表id的7，并且y后移\*/ 139. } 140. else{ 141. switch(buffer[x++]){/\*当buffer中的字符为一下任意一种时，自动转换为相应的代号存入w中\*/ 142. case '&': code[y++] = 0;break; 143. case '(': code[y++] = 1;break; 144. case ')': code[y++] = 2;break; 145. case '+': code[y++] = 3;break; 146. case '-': code[y++] = 4;break; 147. case '\*': code[y++] = 5;break; 148. case '/': code[y++] = 6;break; 149. } 150. } 151. } 152. code[y] = 0; 153. } 154. void GetFromCode(){/\*取得当前输入符号串的元素\*/ 155. X = code[position++]; 156. } 157. void PUSH(int A,int S){/\*入栈操作\*/ 158. ip++;/\*将栈指针ip指向最新元素 \*/ 159. analysis\_stack[ip] = A;/\*将符号A移入栈顶\*/ 160. ip++; 161. analysis\_stack[ip] = S;/\*将状态S移入栈顶\*/ 162. }; 163. void POP(){/\*出栈操作\*/ 164. ip--;/\*弹出一个栈顶元素，栈指针ip下移，指向新的栈顶\*/ 165. ip--; 166. }; 167. void SHITF(){/\*移入操作\*/ 168. int x; 169. x = ACTION[analysis\_stack[ip]][X];/\*查找goto表以确定当前移入操作所需移入的状态\*/ 170. PUSH(X,x);/\*将当前读入的字符与状态压入栈\*/ 171. printf("s%d\tShift in.\n",x); 172. } 173. void REDUCE(){/\*规约操作\*/ 174. int x,y,z; 175. x = -ACTION[analysis\_stack[ip]][X];/\*查找action表以确定当前规约操作，即用哪个产生式规约\*/ 176. printf("r%d\tReduce by\t",x); 177. switch(production[x].Vn){ 178. case 101: printf("S -> ");break; 179. case 102: printf("E -> ");break; 180. case 103: printf("T -> ");break; 181. case 104: printf("F -> ");break; 182. } 183. for(int i = 1; i <= production[x].size; i++){ 184. POP();/\*将栈中元素，按规约产生式的右部长度进行弹出操作\*/ 185. switch(production[x].Str[i]){ 186. case 0: printf("$");break; 187. case 1: printf("(");break; 188. case 2: printf(")");break; 189. case 3: printf("+");break; 190. case 4: printf("-");break; 191. case 5: printf("\*");break; 192. case 6: printf("/");break; 193. case 7: printf("id");break; 194. case 8: printf("num");break; 195. case 101: printf("S");break; 196. case 102: printf("E");break; 197. case 103: printf("T");break; 198. case 104: printf("F");break; 199. } 200. } 201. printf("\n"); 202. y = GOTO[analysis\_stack[ip]][production[x].Vn-100];/\*查找goto表以确定规约完成后需要转换的状态\*/ 203. PUSH(production[x].Vn,y);/\*将规约产生式的左部压入栈中，并将相应的转移状态压入栈中\*/ 204. } 205. void ACC(){ 206. flag = 0; 207. printf("Acc!\n"); 208. } 209. void E1() 210. {//期待(或运算对象首字符，但出现运算符或者$ 211. printf("e1\t\t\t缺少运算对象,id压入栈\n"); 212. PUSH(7,4); 213. } 214. void E2() 215. {//括号不匹配，删掉右括号 216. printf("e2\t\t\t括号不匹配,删除右括号\n"); 218. } 219. void E3() 220. {//期待运算符号，但出现(或运算对象 221. int pointer = position; 222. switch(analysis\_stack[ip]) 223. { 224. case 1: PUSH(3,7);printf("e3\t\t\t缺少运算符,添加运算符至栈");break; 225. case 2: case 12: case 13: PUSH(5,9);printf("e3\t\t\t缺少运算符,添加运算符至栈");break; 226. case 11: 227. if( code[pointer] >=3 && code[pointer] <=6 )//期待运算符号 228. {PUSH(2,16);printf("e3\t\t\t缺少右括号,添加右括号至栈");} 229. else if( code[pointer] >= 7 && code[pointer]<=8 ) 230. {PUSH(3,7);printf("e3\t\t\t缺少运算符,添加运算符至栈");} 231. break; 232. } 233. } | | | |
| **实验结果** |  | | | |
| **实验分析** | 此次的分析，中其实就是将之前学习到的相关知识，作为一个具体实现，只要将每一个步骤每一种状态进行分析。并将其实现为一个小方法，最后整合到一起即可。  在一开始的编写过程中， | | | |
| **心得体会** |  | | | |

程序附录：