**词法分析实验**

**一、实验目的**

通过设计、开发一个高级语言的词法分析程序，加深对课堂教学内容（包括正规文法、正规表达式、有限自动机、NFA到DFA的转换、DFA的最小化）的理解，提高词法分析方法的实践能力。

**二、实验要求**

(1)深入理解、掌握有限自动机及其应用；

(2)掌握根据语言的词法规则构造识别其单词的有限自动机的方法；

(3)掌握NFA到DFA的等价变换方法、DFA最小化的方法；

(4)掌握设计、编码、调试词法分析程序的技术与方法，具体实现S语言（见附录A）的词法分析程序。

**三、实验原理**

词法分析是编译过程的第一个阶段。它的任务是对输入的字符串形式的源程序按顺序进行扫描，根据源程序的词法规则识别具有独立意义的单词，并输出单词的序列。有限自动机是描述程序设计语言单词结构的工具，而状态转换图是有限自动机的比较直观的描述方法。根据程序设计语言的词法规则构造描述该语言单词结构的有限自动机，获取识别各类单词的形式模型，进而通过编程模拟该形式模型的运行，可实现词法分析程序。

1. **实验报告**

**1.单词种类**

<字符>： := <字母>|<数字>|<特定符号>|<无效字符>

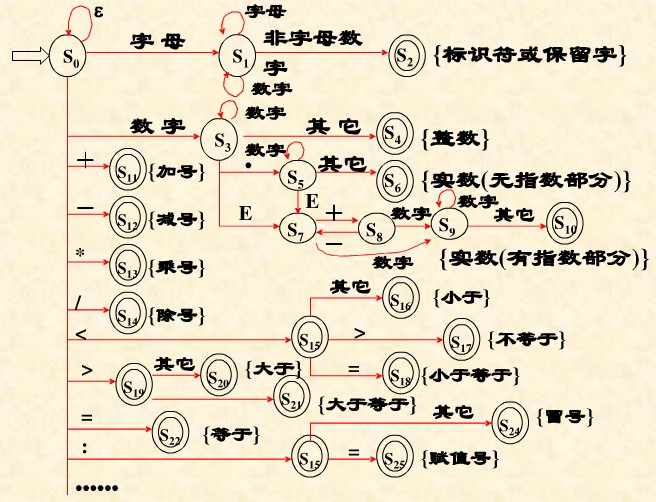
<字母>： := A|B|C|…|Z|a|b|c|…|z

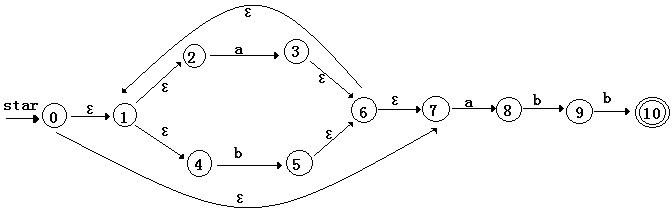
<数字>： := 0|1|2|…|9

<特定符号>： := +|-|\*|/|;=|<|<=|…|·

<无效字符>： := <回车符>|<换行符>|<制表符>

**2.单词状态转换图**

**3.NFA转化为等价的DFA**



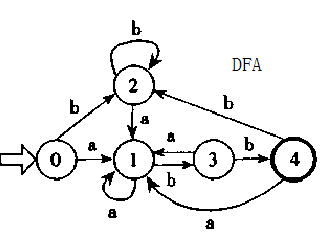
先将上图, 运用子集矩阵法，通过运算得到表1。其中S表示状态，即算法描

述中的自己族C；a，b 表示输入字符。该阶段，可能存在没有参加运算的状态，这些状态就是不可到达的状态。不可到达过程的运算也可通过图论中的可达性进行检查。

**表1.状态转换矩阵表**

|  |  |  |
| --- | --- | --- |
| **S** | **a** | **b** |
| 0,1,2,4,7 | 3,8,6,7,1,4,2 | 5,6,7,1,2,4 |
| 3,8,6,7,1,2,4 | 3,8,6,7,1,2,4 | 5,9,6,7,1,2,4 |
| 5,6,7,1,2,4 | 3,8,6,7,1,2,4 | 5,6,7,1,2,4 |
| 5,9,6,7,1,2,4 | 8,3,6,7,1,2,4 | 10,5,6,7,1,2,4 |
| 10,5,6,7,1,2,4 | 8,3,6,7,1,2,4 | 5,6,7,1,2,4 |

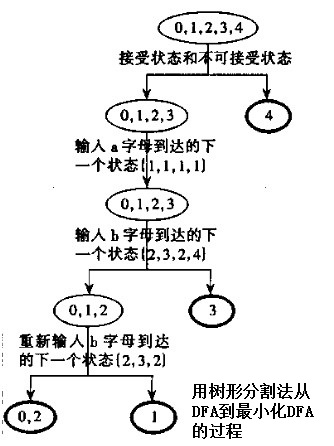
将表1转换矩阵中的所有子集重新命名得到表2矩阵表示形式。

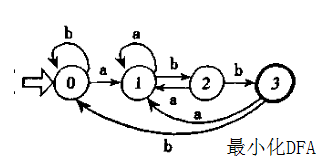
****

**表2.重命名的状态转换矩阵表**

|  |  |  |
| --- | --- | --- |
| **S** | **a** | **b** |
| 0 | 1 | 2 |
| 1 | 1 | 3 |
| 2 | 1 | 2 |
| 3 | 1 | 4 |
| 4 | 1 | 2 |

1. **对DFA进行最小化操作**

这阶段是采用树型分割法。该方法的主要思想是：将DFA分割成一些不相交的子集，使得任何不同两子集的状态是可区别的，而同一子集中任何两个状态是等价的。一般先将DFA的所有状态分割成两个状态的集合，即可接受的状态和不可接受的状态集合；再根据等价原则，求出它的最大集。最大等价状态集是某一状态集中某一状态与任何状态不等价。



1. **词法分析程序代码**
2. #include<iostream>
3. #include<stdio.h>
4. #include<string.h>
5. #include<stdlib.h>
6. using namespace std;
7. #define ID 15
8. #define NUM 16
9. //存放处理后的字符串
10. char tempstr[255] = {};
11. //空格标志
12. bool temp = false;
13. //临时数组
14. char word[255] = {};
15. //keyword关键字
16. string keyword[9] = {
17. "Const", "Var", "if", "else", "then",
18. "while", "do", "begin", "end"
19. };
20. int keyword\_num[9] = {
21. 1,2,3,4,5,
22. 6,7,8,9
23. };
24. //部分运算符，定界符等
25. char symbol[9] = { '+','-','\*','/',';','(',')','{','}'};
26. //对应的种码值
27. int symbol\_num[9] = { 21,22,23,24,25,26,27,28,29 };
28. //判断是否为字母
29. bool IsLetter(char ch)
30. {
31. if ((ch >= 'a'&&ch <= 'z') || (ch >= 'A'&&ch <= 'Z'))
32. return true;
33. return false;
34. }
35. //判断是否为数字
36. bool IsDigit(char ch)
37. {
38. if (ch >= '0'&&ch <= '9')
39. return true;
40. return false;
41. }
42. //判断是否为定界符等
43. int IsSymbol(char ch)
44. {
45. for (int i = 0; i<9; i++)
46. {
47. if (ch == symbol[i])
48. return i;
49. }
50. return -1;
51. }
52. //判断是否为关键字
53. int IsKeyword(string str)
54. {
55. for (int i = 0; i<26; i++)
56. {
57. if (str == keyword[i])
58. {
59. return i;
60. }
61. }
62. //不是关键字即为ID
63. return ID;
64. }
65. //空格处理
66. void HandleSpace(char a[])
67. {
68. int j = 0;
69. memset(word, 0, 255);//需要清空，不然可能残留上次的字符串
70. temp = false;
71. for (int i = 0; i < strlen(a); i++)
72. {
73. if (a[i] != ' ' && a[i] != '\t')
74. {
75. word[j++] = a[i];
76. temp = false;
77. }
78. else
79. {
80. if (!temp && a[i] != '\t')
81. {
82. word[j++] = a[i];
83. temp = true;
84. }
85. }
86. }
87. }
88. //处理"//"注释
89. void prePro()
90. {
91. int j = 0;
92. memset(tempstr, 0, 255);
93. for (int i = 0; i < strlen(word); i++)
94. {
95. if (word[i] == '/' && word[i + 1] == '/')
96. {
97. while (i < strlen(word))
98. {
99. i++;
100. }
101. }
102. else {
103. tempstr[j++] = word[i];
104. }
105. }
106. }
107. void Scanner(char \*str);
108. int main()
109. {
110. char instr[255] = {}; //接收输入字符串
111. bool flag = false; //多行注释标志,false为未处于注释区域
112. string Token;//存放字符串
113. char \*str = NULL;//存放每行的字符串
114. char delims[] = " ";//分割标志
115. freopen("test.cpp", "r", stdin);
116. freopen("result.txt", "w", stdout);
117. printf("词法分析结果如下：\n");
118. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
119. while ((gets\_s(instr)) != NULL)
120. {
121. HandleSpace(instr);
122. prePro();
123. str = strtok(tempstr, delims);//分割字符串
125. while (str != NULL)
126. {
127. //头文件，宏定义
128. if (\*(str) == '#')
129. {
130. printf("#\n");
131. break;
132. }
134. Scanner(str);
135. str = strtok(NULL, delims);
136. }
137. }
138. return 0;
139. }
140. void Scanner(char \*str) {
141. bool flag = false; //多行注释标志,false为未处于注释区域
142. string Token;//存放字符串
143. int type = 0;
144. for (int i = 0; i<strlen(str); i++)
145. {
146. if (\*(str + i) == '/')
147. {
148. if (\*(str + i + 1) == '\*')
149. {
150. flag = true;
151. break;
152. }
153. }
154. //注释处理: \*/,注释区域结束
155. if (\*(str + i) == '\*'&&flag)
156. {
157. if (\*(str + i + 1) == '/')
158. {
159. flag = false;
160. i++;
161. break;
162. }
163. }
164. //标识符，关键词
165. if (IsLetter(\*(str + i)) && (!flag))
166. {
167. while (IsLetter(\*(str + i)) || IsDigit(\*(str + i)) || \*(str + i) == '\_')
168. {
169. Token += \*(str + i);
170. i++;
171. }
172. if (IsKeyword(Token) != ID)
173. {
174. int syb = keyword\_num[IsKeyword(Token)];
175. type = syb;
176. }
177. else
178. {
179. type = ID;
180. }
181. printf("< %s , %d >\n", Token.c\_str(), type);
182. Token = "";
183. }
184. if (IsDigit(\*(str + i)) && (!flag))
185. {
186. while (IsDigit(\*(str + i)))
187. {
188. Token += \*(str + i);
189. i++;
190. }
191. type = NUM;
192. printf("< %s , %d >\n", Token.c\_str(), type);
193. Token = "";
194. }
195. //<,<=,<>
196. if (\*(str + i) == '<' && (!flag))
197. {
198. if (\*(str + i + 1) == '=')
199. {
200. type = 32;
201. printf("< %s , %d >\n", "<=", type);
202. }
203. if (\*(str + i + 1) == '>')
204. {
205. type = 31;
206. printf("< %s , %d >\n", "<>", type);
207. }
208. else
209. {
210. type = 30;
211. printf("< %s , %d >\n", "<", type);
212. }
213. }
214. //>,>=
215. else if (\*(str + i) == '>' && (!flag))
216. {
217. if (\*(str + i + 1) == '=')
218. {
219. type = 34;
220. printf("< %s , %d >\n", ">=", type);
221. }
222. else
223. {
224. type = 33;
225. printf("< %s , %d >\n", ">", type);
226. }
227. }
228. //=,==
229. else if (\*(str + i) == '=' && (!flag))
230. {
231. if (\*(str + i + 1) == '=')
232. {
233. type = 36;
234. printf("< %s , %d >\n", "==", type);
235. }
236. else
237. {
238. type = 35;
239. printf("< %s , %d >\n", "=", type);
240. }
241. }
242. //余下定界符等
243. else if (IsSymbol(\*(str + i)) != -1 && (!flag))
244. {
245. int syb = symbol\_num[IsSymbol(\*(str + i))];
246. type = syb;
247. printf("< %c , %d >\n", \*(str + i), syb);
248. }
249. }
250. }

测试输入：

#include<stdio.h>

int main()

{

//test

Const const\_a = 10;

Var var\_b = 15;

if (const\_a > 5){

var\_b = var\_b + 1;

}

else

var\_b = var\_b - 4;

do {

/\* test \*/

} while(var\_b > 5);

a, b, c = 0;

abc = 0;

return 0;

}

输出结果为：

词法分析结果如下：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

< int , 15 >

< main , 15 >

< ( , 26 >

< ) , 27 >

< { , 28 >

< Const , 1 >

< const\_a , 15 >

< = , 35 >

< 10 , 16 >

< ; , 25 >

< Var , 2 >

< var\_b , 15 >

< = , 35 >

< 15 , 16 >

< ; , 25 >

< if , 3 >

< ( , 26 >

< const\_a , 15 >

< > , 33 >

< 5 , 16 >

< ) , 27 >

< { , 28 >

< var\_b , 15 >

< = , 35 >

< var\_b , 15 >

< + , 21 >

< 1 , 16 >

< ; , 25 >

< } , 29 >

< else , 4 >

< var\_b , 15 >

< = , 35 >

< var\_b , 15 >

< - , 22 >

< 4 , 16 >

< ; , 25 >

< do , 7 >

< { , 28 >

< test , 15 >

< \* , 23 >

< / , 24 >

< } , 29 >

< while , 6 >

< ( , 26 >

< var\_b , 15 >

< > , 33 >

< 5 , 16 >

< ) , 27 >

< ; , 25 >

< a , 15 >

< b , 15 >

< c , 15 >

< = , 35 >

< 0 , 16 >

< ; , 25 >

< abc , 15 >

< = , 35 >

< 0 , 16 >

< ; , 25 >

< return , 15 >

< 0 , 16 >

< ; , 25 >

< } , 29 >