实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2019级计算机科学与技术专业4班** | | **姓名** | **谭勇** |
| **实验题目** | 词法分析程序的设计与实现 | | | | |
| **实验时间** | **2022.4** | | **实验地点** | **DS3401** | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 ■综合性** | |
| 教师评价：  □算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □报告规范；  实验代码得分（满分7分）：  实验报告得分（满分3分）：  评价教师签名： | | | | | |
| 一、实验目的  🡪了解编译器中词法分析环节的作用和意义  🡪掌握词法分析基础，实现词法分析器 | | | | | |
| 二、实验项目内容   1. 实验要求：   请根据给定的文法设计并实现词法分析程序，从源程序中识别出单词，记录其单词类别和单词值，输入输出及处理要求如下：     （1）数据结构和与语法分析程序的接口请自行定义；类别码需按下表格式统一定义；     （2）为了方便进行自动评测，输入的被编译源文件统一命名为testfile.txt（注意不要写错文件名）；输出的结果文件统一命名为output.txt（注意不要写错文件名），结果文件中每行按如下方式组织：  单词类别码 单词的字符/字符串形式(中间仅用一个空格间隔)  单词的类别码请统一按如下形式定义：   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 单词名称 | 类别码 | 单词名称 | 类别码 | 单词名称 | 类别码 | 单词名称 | 类别码 | | 标识符 | IDENFR | else | ELSETK | - | MINU | = | ASSIGN | | 整形常量 | INTCON | switch | SWITCHTK | \* | MULT | ; | SEMICN | | 字符常量 | CHARCON | case | CASETK | / | DIV | , | COMMA | | 字符串 | STRCON | default | DEFAULTTK | < | LSS | ( | LPARENT | | const | CONSTTK | while | WHILETK | <= | LEQ | ) | RPARENT | | int | INTTK | for | FORTK | > | GRE | [ | LBRACK | | char | CHARTK | scanf | SCANFTK | >= | GEQ | ] | RBRACK | | void | VOIDTK | printf | PRINTFTK | == | EQL | { | LBRACE | | main | MAINTK | return | RETURNTK | != | NEQ | } | RBRACE | | if | IFTK | + | PLUS | ： | COLON |  |  |   【输入形式】testfile.txt中的符合文法要求的测试程序。 【输出形式】要求将词法分析结果输出至output.txt中。  【特别提醒】（1）读取的字符串要原样保留着便于输出，特别是数字，这里输出的并不是真正的单词值，其实是读入的字符串，单词值需另行记录。                      （2）本次作业只考核对正确程序的处理，但需要为今后可能出现的错误情况预留接口。                      （3）在今后的错误处理作业中，需要输出错误的行号，在词法分析的时候注意记录该信息。                      （4）单词的类别和单词值以及其他关注的信息，在词法分析阶段获取后，后续的分析阶段会使用，请注意记录；当前要求的输出只是为了便于评测，完成编译器中无需出现这些信息，请设计为方便打开/关闭这些输出的方案。  【文法定义】请到“2020年文法定义(用于查看文法，不计入成绩）”作业中查看文法   1. 实验要求：   1、【开发语言及环境】用C/C++实现，平台支持C++11标准，源代码文件必须使用UTF-8编码，才能够输出评测平台能够识别的中文(如果不确定源文件的编码，推荐使用notepad++查看)  2、提交实验报告及源代码。实验报告需严格遵循学校文档规范，内容包含对应文法、词类编码表、词法分析测试用例。 | | | | | |
| 三、实验过程或算法 | | | | | |
| 首先, 根据单词的类别定义,初始化一个map用来保存单词名称与类别码间的映射关系.  然后设计有限状态机,主要可以分为对KEYWORD和IDENFR, INTCON, CHARCON, STRCON间的状态判断.  需要设置一个状态变量来保存对当前词素的类别判断.    然后从初始状态到目标状态,直到结束词素才重置状态.  其中的关键代码大致如下:            实验过程中遇到主要问题有:   1. 各种类别判断的先后顺序 2. 判断条件不完整 3. 特殊条件,边界条件考虑不周 4. 文件输入与输出语句使用不当 5. …   举一个典型的例子,比如字符串中”hello world\\\””的判断,不能单纯看见“就结束词素.需要多加一个条件判断,类似的还有\t这样的情况.  实验代码如下:   1. #include <iostream> 2. #include <cstdio> 3. #include <cstring> 4. #include <map> 6. **using** **namespace** std; 8. map<string, string> mapping; 10. // 初始化 11. **void** init() 12. { 13. mapping["const"] = "CONSTTK"; 14. mapping["int"] = "INTTK"; 15. mapping["char"] = "CHARTK"; 16. mapping["void"] = "VOIDTK"; 17. mapping["main"] = "MAINTK"; 18. mapping["if"] = "IFTK"; 19. mapping["else"] = "ELSETK"; 20. mapping["switch"] = "SWITCHTK"; 21. mapping["case"] = "CASETK"; 22. mapping["default"] = "DEFAULTTK"; 23. mapping["while"] = "WHILETK"; 24. mapping["for"] = "FORTK"; 25. mapping["scanf"] = "SCANFTK"; 26. mapping["printf"] = "PRINTFTK"; 27. mapping["return"] = "RETURNTK"; 28. // ---------- 这是分割线，以上与 IDENFR 一起判断， 一下独立判断 29. mapping["+"] = "PLUS"; 30. mapping["-"] = "MINU"; 31. mapping["\*"] = "MULT"; 32. mapping["/"] = "DIV"; 33. mapping["<"] = "LSS"; 34. mapping["<="] = "LEQ"; 35. mapping[">"] = "GRE"; 36. mapping[">="] = "GEQ"; 37. mapping["=="] = "EQL"; 38. mapping["!="] = "NEQ"; 39. mapping[":"] = "COLON"; 40. mapping["="] = "ASSIGN"; 41. mapping[";"] = "SEMICN"; 42. mapping[","] = "COMMA"; 43. mapping["("] = "LPARENT"; 44. mapping[")"] = "RPARENT"; 45. mapping["["] = "LBRACK"; 46. mapping["]"] = "RBRACK"; 47. mapping["{"] = "LBRACE"; 48. mapping["}"] = "RBRACE"; 49. } 51. // 判断是否是关键字 52. **void** isKeyword(string str) 53. { 54. string s = str; 55. // lower string 56. **for** (**int** i = 0; i < str.length(); i++) 57. { 58. **if** (s[i] <= 'Z' && s[i] >= 'A') 59. { 60. s[i] = s[i] - 'A' + 'a'; 61. } 62. } 63. // 如果是关键字 64. **if** (mapping.count(s)) 65. { 66. cout << mapping[s] << " " << str << endl; 67. } 68. // 不是关键字就是标识符 69. **else** 70. cout << "IDENFR " << str << endl; 71. } 73. // 是否是字符 74. **int** isletter(**char** a) 75. { 76. **if** ((a >= 'a' && a <= 'z') || (a >= 'A' & a <= 'Z') || a == '\_') 77. **return** 1; 78. **return** 0; 79. } 81. // 是否是数字 82. **int** isdigit(**char** a) 83. { 84. **if** (a >= '0' && a <= '9') 85. **return** 1; 86. **return** 0; 87. } 89. // 初始 state 为 0, 根据类别变化 90. **int** state = 0; 92. // 单词 (词素) 93. string word = ""; 95. // 有限状态机函数 96. **void** FSM(string str) 97. { 98. **for** (**int** i = 0; i < str.length(); i++) 99. { 100. **switch** (state) 101. { 102. // 初始判断 103. **case** 0: 104. // KEYWORD 和 IDENFR, 分割线以上部分 105. **if** (isletter(str[i])) 106. { 107. state = 1; 108. word += str[i]; 109. **break**; 110. } 111. // INTCON 112. **if** (isdigit(str[i])) 113. { 114. state = 2; 115. word += str[i]; 116. **break**; 117. } 118. // CHARCON 119. **if** (str[i] == '\'') 120. { 121. state = 3; 122. **break**; 123. } 124. // STRCON 125. **if** (str[i] == '"') 126. { 127. state = 4; 128. **break**; 129. } 130. // KEYWORD, 分割线以下部分 131. **else** 132. { 133. **if** (mapping.count(str.substr(i, 2))) 134. { 135. cout << mapping[str.substr(i, 2)] << " " << str.substr(i, 2) << endl; 136. // 一次性取了俩，手动多跳一次 i 137. i++; 138. **break**; 139. } 140. **else** **if** (mapping.count(str.substr(i, 1))) 141. { 142. cout << mapping[str.substr(i, 1)] << " " << str[i] << endl; 143. **break**; 144. } 145. **break**; 146. } 147. // KEYWORD 148. **case** 1: 149. **if** (isletter(str[i]) || isdigit(str[i])) 150. { 151. word += str[i]; 152. **break**; 153. } 154. **else** 155. { 156. // 不是关键字则是 IDENFR 157. isKeyword(word); 158. word = ""; 159. // 返回左边索引，避免漏掉判断类似 main (中的 (这样的情况 160. i--; 161. state = 0; 162. **break**; 163. } 164. // INTCON 165. **case** 2: 166. **if** (isdigit(str[i])) 167. { 168. word += str[i]; 169. **break**; 170. } 171. **else** 172. { 173. cout << "INTCON " << word << endl; 174. word = ""; 175. i--; 176. state = 0; 177. **break**; 178. } 179. // CHARCON 180. **case** 3: 181. // 不严谨的判断右单引号 182. **if** (str[i] == '\'') 183. { 184. **if** (word.length() == 1) 185. { 186. cout << "CHARCON " << word << endl; 187. word = ""; 188. state = 0; 189. **break**; 190. } 191. **else** 192. { 193. cout << "format error: " << word << " is not char" << endl; 194. word = ""; 195. state = 0; 196. **break**; 197. } 198. } 199. **else** **if** (str[i] == '\\') 200. { 201. **if** (str[i + 1] == '\\') 202. { 203. word += '\\'; 204. i++; 205. **break**; 206. } 207. **else** **if** (str[i + 1] == '\'') 208. { 209. word += '\''; 210. i++; 211. **break**; 212. } 213. } 214. **else** 215. { 216. word += str[i]; 217. **break**; 218. } 219. // STRCON 220. **case** 4: 221. // 不严谨的判断右双引号 222. **if** (str[i] == '"') 223. { 224. cout << "STRCON " << word << endl; 225. word = ""; 226. state = 0; 227. **break**; 228. } 229. **else** **if** (str[i] == '\\') 230. { 231. word += str.substr(i, 2); 232. i++; 233. **break**; 234. } 235. **else** 236. { 237. word += str[i]; 238. **break**; 239. } 240. **default**: 241. **break**; 242. } 243. } 244. } 246. **int** main() 247. { 248. freopen("testfile.txt", "r", stdin); 249. freopen("output.txt", "w", stdout); 250. init(); 251. string str; 252. **while** (getline(cin, str)) 253. { 254. FSM(str); 255. } 256. fclose(stdin); 257. fclose(stdout); 258. **return** 0; 259. } | | | | | |
| 四、实验测试  平台测试:    样例输入:   1. coNst **int** cONst1 = 001, const2 = -100; 2. **const** **char** const3 = '\_'; 3. **int** change1; 4. **char** change3; 5. **int** gets1(**int** var1,**int** var2){ 6. change1 = var1 + var2; 7. **return** (change1); 8. } 9. **void** main(){ 10. printf("Hello World"); 11. printf(gets1(10, 20)); 12. }   样例输出:   1. CONSTTK coNst 2. INTTK **int** 3. IDENFR cONst1 4. ASSIGN = 5. INTCON 001 6. COMMA , 7. IDENFR const2 8. ASSIGN = 9. MINU - 10. INTCON 100 11. SEMICN ; 12. CONSTTK **const** 13. CHARTK **char** 14. IDENFR const3 15. ASSIGN = 16. CHARCON \_ 17. SEMICN ; 18. INTTK **int** 19. IDENFR change1 20. SEMICN ; 21. CHARTK **char** 22. IDENFR change3 23. SEMICN ; 24. INTTK **int** 25. IDENFR gets1 26. LPARENT ( 27. INTTK **int** 28. IDENFR var1 29. COMMA , 30. INTTK **int** 31. IDENFR var2 32. RPARENT ) 33. LBRACE { 34. IDENFR change1 35. ASSIGN = 36. IDENFR var1 37. PLUS + 38. IDENFR var2 39. SEMICN ; 40. RETURNTK **return** 41. LPARENT ( 42. IDENFR change1 43. RPARENT ) 44. SEMICN ; 45. RBRACE } 46. VOIDTK **void** 47. MAINTK main 48. LPARENT ( 49. RPARENT ) 50. LBRACE { 51. PRINTFTK printf 52. LPARENT ( 53. STRCON Hello World 54. RPARENT ) 55. SEMICN ; 56. PRINTFTK printf 57. LPARENT ( 58. IDENFR gets1 59. LPARENT ( 60. INTCON 10 61. COMMA , 62. INTCON 20 63. RPARENT ) 64. RPARENT ) 65. SEMICN ; 66. RBRACE } | | | | | |
| 五、实验总结  通过本次实验,我了解了如何使用工具(比如flex)简化工作,进行词法分析, 以及自己编写一个简单的,但不灵活的词法分析器. 了解到了编译器第一阶段的工作原理, 加深了对编译原理的理解.  其中,我遇到的主要问题有   1. 各种类别判断的先后顺序 2. 判断条件不完整 3. 特殊条件,边界条件考虑不周 4. 文件输入与输出语句使用不当 5. …   对于问题一,比如<=和<,解决方案时先判断<=再判断<.  对于问题二,比如对IDENFR 的判断, 要求第一个char不能是数字, 补全判断条件即可.  对于问题三, 比如什么时候不应该有输出, 除了空格之外还有换行符等,应该留意. | | | | | |