# 《编译原理》课程实验1报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 文天鑫 | 学号 | 0221120281 |
| 专业 | 软件工程 | 年级 | 22级 |
| 实验题目 | 扩展PL/0编译器 | 实验学时 | 32 |

**一、实验目的**

根据所学的编译原理理论知识，在符合 PL/0 语言基本词法、语法规则的前提下，以原PL/0 编译程序C语言版本代码为基础，对 PL/0 语言编译器中的此法分析功能进行实现。

**二、实验环境**

C++、VSCODE

1. **词法分析程序的设计与实现**

***（说明：参考软件工程、系统设计类课程中对程序的设计思路、流程和主要的特点进行阐述）***

设计思路如下:

通过读取源文件的代码，将字符(串)存入一个数组中。通过对该数组的扫描与识别。从而将不同类型的单词与其属性值输出到输出文件中，并具有一定的错误处理能力。

设计流程：

创建两个数组，一个用来存储关键字、变量等字符。另一个用来存源文件的代码。

再创建几个常量用来遍历与记录。

再创建两个数组用来存关键字的类型。

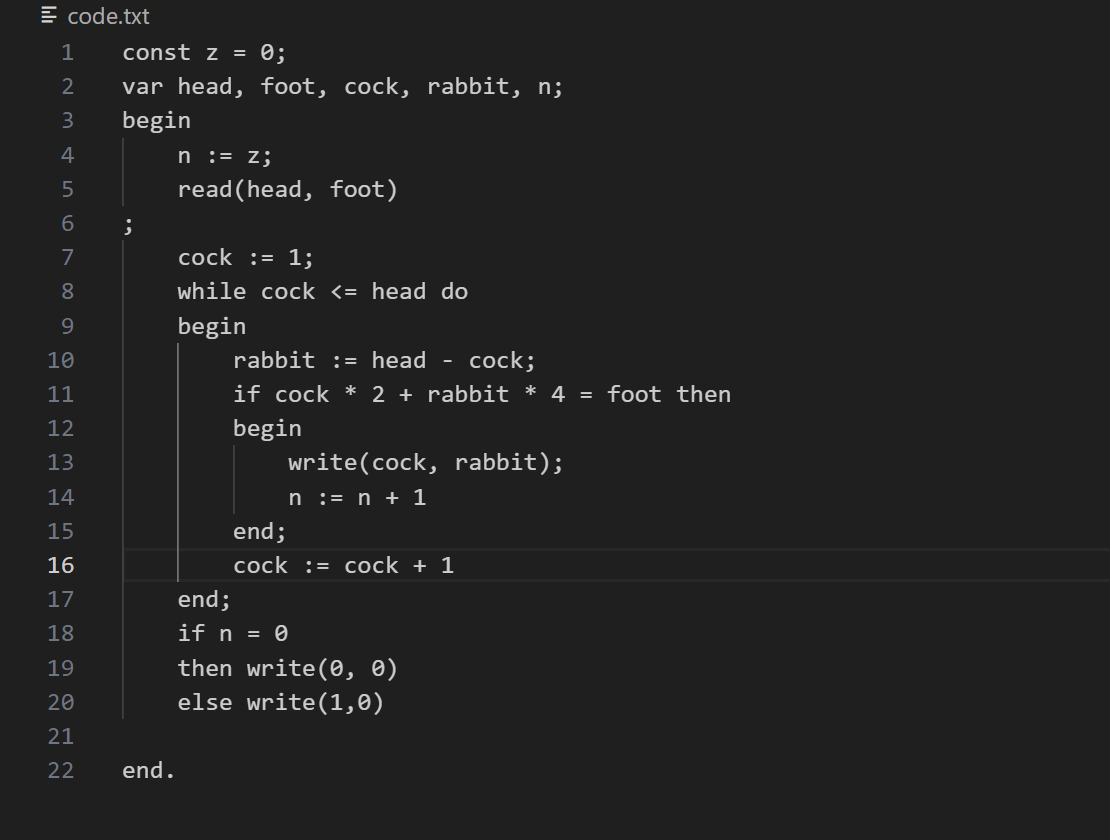
依次读入源程序符号，对源程序进行单词的分解与识别，直到其结束

正确，则输出种别码与值到输出文件

否则，进行错误处理

1. **输入样例测试**

***（自行选择3组程序测试，将测试程序和结果列出，并分析测试结果的正确性）***



**屏幕上写着字

描述已自动生成**

**屏幕上写着字

描述已自动生成**

**文本

描述已自动生成**

**屏幕上写着字

描述已自动生成**

**屏幕上写着字

描述已自动生成**

**文本

描述已自动生成**

**屏幕上写着字

描述已自动生成**

**屏幕上写着字

描述已自动生成**

1. **程序中的关键代码**

*/\*\**

  \* util.h: This header file contains the function declarations and global variables used in the lexical analysis program.

1. \*
2. \* @author CheeseSilly
3. \* @date 2024-10-16
4. \*/
5. # pragma once
6. *//test code file*
7. #define CODE "code.txt" *//code、code1、code2*
8. *//result file*
9. #define RESULT "result.txt"
10. #include <iostream>
11. #include <fstream>
12. #include <cstring>
13. #include <windows.h>
14. */\*\**
15. \* The token array is used to receive keywords, variables, operators, and delimiters.
16. \* Here, the length of the variable name is limited to a maximum of 10.
17. \* The prog array stores the source code string length.
18. \* @param token: The token array used to store keywords, variables, operators, and delimiters.
19. \* @param prog: The prog array stores the source code string length.
20. \* @param ch: The current character being processed.
21. \* @param syn: The syn variable stores the number corresponding to each word or symbol.
22. \* @param p: The p variable is used to track the index of the prog array.
23. \* @param m: The m variable is used to track the index of the token array.
24. \* @param n: The n variable is used to track the index of the rwtab1 and rwtab2 arrays.
25. \* @param line: The line variable is used to track the current line number.
26. \* @param sum: The sum variable is used to store the number.
27. \* @param rwtab1: The rwtab1 array stores the reserved words and delimiters.
28. \* @param rwtab2: The rwtab2 array stores the reserved words and delimiters.
29. \*/
30. char token[10];
31. char \*prog;
32. char ch;
33. int syn, p, m = 0, n, line, sum = 0;
34. std::string rwtab1[10] = { "begin","if","then","else","while","do","end" };
35. std::string rwtab2[4] = { "const","var","procedure","call" };
36. */\*\**
37. \* scaner(): This function is used to scan the source code and identify the keywords, variables, operators, and delimiters.
38. \*
39. \*/
40. void scaner();
41. *//read the code from the file*
42. void read();
43. */\*\**
44. \* @file LexicalAnalyzer.cpp
45. \* @brief This file contains the implementation of a lexical analyzer for PL/0.
46. \*
47. \* @details
48. \* The lexical analyzer reads the source code character by character, groups characters into tokens, and assigns a type to each token.
49. \* It handles different types of tokens such as keywords, identifiers, numbers, and operators. It also manages errors for unrecognized characters
50. \* and handles special cases like comments and whitespace.
51. \*
52. \* The main functions in this file are:
53. \* void scaner(): This function performs the lexical analysis by reading characters from the source code, grouping them into tokens, and assigning types.
54. \* void read(): This function reads the entire source code from a file into a buffer.
55. \* int main(): The main function that drives the lexical analysis process, reads the source code, calls the scanner, and outputs the results.
56. \*
57. \* The results of the lexical analysis are written to an output file specified by the `RESULT` macro.
58. \*
59. \* @author CheeseSilly
60. \* @date 2024-10-16
61. \*/
62. #include"util.h"
63. void scaner()
64. {
66. *//initialize the token array*
67. for (n = 0; n<10; n++)
68. token[n] = '\0';
69. ch = prog[p++];
70. *//skip the space*
71. while (ch == ' ')
72. {
73. ch = prog[p];
74. p++;
75. }
76. *//*
77. if ((ch >= 'a'&&ch <= 'z') || (ch >= 'A'&&ch <= 'Z'))
78. {
79. *//Initialize m to 0,which is used to track the index of the token array*
80. m = 0;
81. while ((ch >= '0'&&ch <= '9') || (ch >= 'a'&&ch <= 'z') || (ch >= 'A'&&ch <= 'Z'))
82. {
83. token[m++] = ch;
84. ch = prog[p++];
85. }
86. *//end the token array*
87. token[m++] = '\0';
88. p--;
89. syn = 10;
90. *//check if the token is a reserved word,all reserved words are lowercase*
91. for (n = 0; n<6; n++)
92. if (token==rwtab1[n])
93. {
94. syn = n + 1;
95. break;
96. }
97. for (n = 0; n < 4;n++)
98. {
99. if (token==rwtab2[n])
100. {
101. syn = n + 31;
102. break;
103. }
104. }
105. }
106. else if ((ch >= '0'&&ch <= '9'))  *//store the number with the sum*
107. {
108. sum = 0;
109. while ((ch >= '0'&&ch <= '9'))
110. {
111. *//only support the integer*
112. sum = sum \* 10 + ch - '0';
113. ch = prog[p++];
114. }
115. p--;
116. syn = 11;
117. *//max=32767*
118. if (sum>32767)
119. syn = -1;
120. }
121. *//check the operator and delimiter*
122. else switch (ch)
123. {
124. case '<':
125. m = 0;
126. token[m++] = ch;
127. ch = prog[p++];
128. if (ch == '>')
129. {
130. *//for <>*
131. syn = 21;
132. token[m++] = ch;
133. }
134. else if (ch == '=')
135. {
136. *//for <=*
137. syn = 22;
138. token[m++] = ch;
139. }
140. else
141. {
142. *//for <*
143. syn = 23;
144. *//back to the front of <*
145. p--;
146. }
147. break;
148. case '>':
149. m = 0;
150. token[m++] = ch;
151. ch = prog[p++];
152. if (ch == '=')
153. {
154. syn = 24;
155. token[m++] = ch;
156. }
157. else
158. {
159. syn = 20;
160. p--;
161. }
162. break;
163. case ':':
164. m = 0;
165. token[m++] = ch;
166. ch = prog[p++];
167. if (ch == '=')
168. {
169. *//:=*
170. syn = 18;
171. token[m++] = ch;
172. }
173. else
174. {
175. *//:*
176. syn = 17;
177. p--;
178. }
179. break;
180. case '\*':
181. syn = 13;
182. token[0] = ch;
183. break;
184. case '/':
185. syn = 14;
186. token[0] = ch;
187. break;
188. case '+':
189. syn = 15;
190. token[0] = ch;
191. break;
192. case '-':
193. syn = 16;
194. token[0] = ch;
195. break;
196. case '=':
197. syn = 25;
198. token[0] = ch;
199. break;
200. case ';':
201. syn = 26;
202. token[0] = ch;
203. break;
204. case '(':
205. syn = 27;
206. token[0] = ch;
207. break;
208. case ')':
209. syn = 28;
210. token[0] = ch;
211. break;
212. case ',':
213. syn = 29;
214. token[0] = ch;
215. break;
216. case '!':
217. syn = 30;
218. token[0] = ch;
219. break;
220. case '{':
221. m=0;
223. break;
224. case '.':
225. *//end.*
226. syn = 0;
227. token[0] = ch;
228. break;
229. case '\n':
230. *//change the line,line++*
231. syn = -2;
232. break;
233. default:
234. *//undefined character,return error*
235. syn = -1;
236. break;
237. }
238. }
239. void read()
240. {
241. FILE \*fp;
242. fp = fopen(CODE, "r");
243. *//get the size of the file*
244. fseek(fp, 0, SEEK\_END);
245. int file\_size;
246. file\_size = ftell(fp);
247. fseek(fp, 0, SEEK\_SET);
248. prog = new char[file\_size\*sizeof(char)];
249. *//read the file*
250. fread(prog, file\_size, sizeof(char), fp);
251. fclose(fp);
252. }
253. int main()
254. {
255. int p = 0;
256. int line = 1;
257. std::ofstream outfile(RESULT);
258. read();
259. p = 0;
260. do
261. {
262. scaner();
263. switch (syn)
264. {
265. *//output number*
266. case 11:
267. outfile << "(" << syn << "," << sum << ")" << std::endl;
268. break;
269. *//output error*
270. case -1:
271. outfile << "Error in line" << line << "!" << std::endl;
272. break;
273. *//handle the line change*
274. case -2:
275. line = line++;
276. break;
277. *//output the reserved word*
278. default:
279. outfile << "(" << syn << "," << token << ")" << std::endl;
280. break;
281. }
282. } while (syn != 0);
283. outfile.close();
284. std::cout << "complete!check it in the result.txt" << std::endl;
285. system("pause");
286. return 0;
287. }
288. **实验体会与收获**

学会了如何写一个词法分析程序