|  |
| --- |
| 编译原理课程设计  实验报告  **指导教师：肖荣**  **年 级： 2020级**  **班 级：8班**  **小组编号：**  **组长学号姓名：55200808 张煜坤**  **组员学号姓名：55200809 刘兆雄**  **组员学号姓名：55200810 樊昱隆**  **2023 年 5 月 9 日**  **计算机科学与技术学院** |

|  |  |  |
| --- | --- | --- |
| **完成实验内容** | | |
| 设计并实现SNL语言的编译系统模块，包括词法分析部分和语法分析部分。语法分析部分使用递归子程序方法实现。 | | |
| **小组成员任务完成情况** | | |
| 姓名 | 具体完成任务 | 工作量  百分比 |
| 张煜坤 | 完成了编译系统的整体设计，完成和语法分析递归子程序方法的设计实现。完成了实验报告的编写。 | 50% |
| 刘兆雄 | 参与了词法分析模块的设计和词法分析状态机的实现。 | 25% |
| 樊昱隆 | 参与了词法分析模块的设计和词法分析状态机的实现。 | 25% |
| **小组成员协作情况** | | |
| 本次实验由组长张煜坤结合课堂原理进行编译系统的总体设计和语法分析部分的实现，两位组员在组长的指导下合作完成了词法分析部分的设计和实现。 | | |
| **实验平台与编程语言** | | |
| **操作系统：Windows11**  **编程语言：C**  **IDE：Microsoft Visual Studio 2022** | | |
| **实验方案设计** | | |
| 一．词法分析程序  1.总体设计  考虑词法分析的过程：首先输入源程序，并使用一个缓存（字符数组）分多次读入。进行词法分析时，每次读入缓存中一个字符，并通过状态机的方式确定单词结尾，得到相应的单词类型和语义信息并存入链表；当读到程序结尾时，将整个链表存入文件，该文件就是词法分析的输出。  分析以上过程可知，词法分析的关键是要设计状态机，结合SNL语言语法，实现状态机大概如下图：    根据上图，使用switch-case语句，在每个状态下根据读入的下一个字符进行状态转换，当状态转换为done时，将当前读取的所有字符拼成单词，并整理成Token。将多个Token结点链成链表，即为词法分析的结果。为了方便编译系统的模块化，此处采用词法分析程序单独为一遍的方式，因此会先将输出写至文件中。  2.数据结构和类型设计  （1）关键字表：采用字符串数组常量保存。    （2）Token结点：使用struct类型自定义，包括lineno（行号）、lex（单词词法类型）、sem（单词语义信息）三个属性域和一个用于形成链表的指针域。    （3）SNL字符类型：使用枚举类型自定义，除数字、字母、错误类型外，将其他符号根据是否需要特别处理（一般需要特殊处理的是双分隔符），分为other（无需特别处理），和colon（：）、dot（.）、lcomment（{）、squote（‘）等。    （4）SNL词法类型：使用枚举类型自定义，包括标识符、保留字、数字、分隔符、注释等类型。    （5）状态类型：用于模拟状态机的状态，使用枚举类型实现，包括初始状态（start）、标识符状态（inid）、终止状态（done）等。    3.主要函数设计  （1）：getTokenList()，包括核心的状态机部分，生成Token链表，代码如下：   1. token\* getTokenList() 2. { 3. */\*打开源文件\*/* 4. infp = fopen(inPath, "r"); 5. state s = start; 6. lexType lex = errortoken; 7. while (1) { 8. int endIndex = 0; 9. for (int i = 0; i < 256 && !feof(infp); ++i) { *//注意feof判断为真后会再读一次，但不会成功读取* 10. fscanf(infp, "%c", &lineBuf[i]); 11. endIndex = i; 12. } 13. if (feof(infp)) *//若在填满缓冲区前文件读取就结束，需要为缓冲区添加终止符* 14. lineBuf[endIndex] = '\0'; 15. curChar = 0; 16. while (curChar != 256) { 17. char ch = lineBuf[curChar]; 18. if (s == incomment || inAlphabet(ch)) { *//字符属于字符表中字符或处于注释状态，进行处理* 19. switch (s) { 20. case start: *//开始状态的状态转换* 21. switch (getType(ch)) { 22. case letter: s = inid; break; 23. case num: s = innum; break; 24. case other: 25. s = done; 26. if (isBlank(ch)) { 27. lex = blank; 28. if (ch == '\n') 29. ++lineCount; 30. } 31. else { 32. lex = sseperator; 33. } 34. break; 35. case colon: s = inassign; break; 36. case lcomment: s = incomment; break; 37. case dot: s = inrange; break; 38. case squote: s = inchar; break; 39. case error: s = errorstate; break; 40. } 41. break; 42. case inid: *//标识符状态的状态转换* 43. switch (getType(ch)) { 44. case letter: 45. case num: s = inid; break; 46. default: 47. s = done; 48. lex = id; 49. ungetNextChar(); 50. break; 51. } 52. break; 53. case innum: *//数字状态的状态转换* 54. switch (getType(ch)) { 55. case num: s = innum; break; 56. default: 57. s = done; 58. lex = unsignedint; 59. ungetNextChar(); 60. break; 61. } 62. break; 63. case inassign: *//赋值状态的状态转换* 64. switch (ch) { 65. case '=': 66. s = done; 67. lex = dseperator; 68. break; 69. default: s = errorstate; break; 70. } 71. break; 72. case incomment: *//注释状态的状态转换* 73. switch (ch) { 74. case '}': 75. s = done; 76. lex = comment; 77. break; 78. case '\n': ++lineCount; 79. default: s = incomment; break; 80. } 81. break; 82. case inrange: *//下标状态的状态转换* 83. switch (ch) { 84. case '.': 85. s = done; 86. lex = index; 87. break; 88. default: 89. s = done; 90. lex = end; 91. ungetNextChar(); 92. break; *//程序结束标志* 93. } 94. break; 95. case inchar: *//字符状态的状态转换* 96. switch (getType(ch)) { 97. case letter: 98. case num: 99. s = endchar; 100. break; 101. default: 102. s = errorstate; 103. break; 104. } 105. break; 106. case endchar: *//字符终止状态的状态转换* 107. switch (ch) { 108. case '\'': 109. s = done; 110. lex = character; 111. break; 112. default: 113. s = errorstate; 114. break; 115. } 116. break; 117. } 118. } 119. else { *//出现不在字符表中的字符，出错* 120. s = errorstate; 121. } 122. getNextChar(); 123. if (s == done) { *//保存token并重置至开始状态* 124. if (lex != comment && lex != blank) { *//注释不处理，空白不处理* 125. token\* t = (token\*)malloc(sizeof(token)); 126. t->lineshow = lineCount; 127. tokenString[strLen] = '\0'; 128. if (lex == id) { *//判断标识符是否是关键字* 129. for (int i = 0; i < 21; ++i) { 130. if (strcmp(RESERVERD\_WORD[i], tokenString) == 0) 131. lex = reserved; 132. } 133. } 134. *//根据类型进行lex和sem的生成* 135. if (lex == id) { 136. strcpy(t->lex, "id"); 137. strcpy(t->sem, tokenString); 138. } 139. else if (lex == unsignedint) { 140. strcpy(t->lex, "intc"); 141. strcpy(t->sem, tokenString); 142. } 143. else if (lex == character) { 144. strcpy(t->lex, "charc"); 145. strcpy(t->sem, tokenString); 146. } 147. else if (lex == errortoken) { 148. strcpy(t->lex, "error"); 149. strcpy(t->sem, tokenString); 150. } 151. else if (lex == reserved || lex == sseperator || lex == dseperator || lex == index || lex == end) { 152. strcpy(t->lex, tokenString); 153. strcpy(t->sem, ""); 154. } 155. lex = errortoken; *//重置状态为error* 156. */\*保存新生成的token\*/* 157. if (pList == NULL) { 158. pList = t; 159. pTail = pList; 160. pTail->pNext = NULL; 161. } 162. else { 163. pTail->pNext = t; 164. pTail = pTail->pNext; 165. pTail->pNext = NULL; 166. } 167. } 168. */\*重置状态和词法分析字符串长度\*/* 169. s = start; 170. strLen = 0; 171. } 172. if (s == errorstate) { 173. *//错误处理* 174. fclose(infp); 175. return pList; 176. } 177. if (ch == '\0') { *//文件尾，处理结束* 178. token\* t = (token\*)malloc(sizeof(token)); 179. t->lineshow = lineCount; 180. strcpy(t->lex, "EOF"); 181. strcpy(t->sem, ""); 182. pTail->pNext = t; 183. pTail = pTail->pNext; 184. pTail->pNext = NULL; 185. fclose(infp); 186. return pList; 187. } 188. }*//while* 189. lineBuf[0] = '\0'; *//模拟清空缓冲区* 190. } 191. }   （2）listToFile()：将Token链表保存至文件中，核心代码如下：   1. char\* listToFile() 2. { 3. listfp = fopen(listPath, "w"); 4. token\* p = pList; 5. while (p) { 6. fprintf(listfp, "%d %s %s\n", p->lineshow, p->lex, p->sem); 7. p = p->pNext; 8. } 9. fclose(listfp); 10. return listPath; 11. }   （3）其余函数包括单词类型的判断函数等辅助函数，实现简单，不作列举。  二．语法分析程序（递归子程序方法）  1.总体设计  考虑递归向下子程序语法分析的过程，首先需要根据生成式实现多个语法分析子程序。然后需要从文件中读取Token，根据Token的类型在当前的语法分析子程序中选择Predict集包含当前Token的生成式对应的分支执行，如此递归执行至Token列表全部处理完毕，就可获得对应的语法树。由以上过程可知，语法分析的关键就是实现各个产生式对应的语法分析子程序。  因此首先总结SNL语言的上下文无关生成式。（见代码中文件：SNL上下文无关文法.docx）  然后便可根据生成式编写子程序，理论上每个生成式可以对应一个子程序，但实际过程中较简单的子程序可以不实现为单独的函数。子程序采用两种结构，一种是在程序内部生成结点并返回给上级，另一种是接收上级传递的结点并修改其属性值。  子程序编写完成后就可以获得完整的语法树，语法分析结束后一般不产生中间文件，因此没有为该模块提供输出至文件的功能。  2.数据结构和类型设计  （1）类型枚举：语法树结点类型和结点属性较为复杂，使用枚举类型定义如下：    （2）语法树结点：使用struct类型自定义。Child指针域维护语法的上下级关系，sibling指针域维护同一语法成分的多个结点间的同级关系，lineno记录该语法成分所处的行数，nodekind和kind共同实现结点类型的具体区分，idnum和name用于保存标识符，type\_name用于保存类型定义中的类型名。Attr中保存具体内容结点的属性。    3.主要函数设计：  （1）readToken()：读取token文件中下一个token，保存到全局变量curToken中。   1. void readToken() { 2. if (!feof(listfp)) { 3. scanf("%d%s", &curToken.lineno, curToken.lex); 4. if (strcmp(curToken.lex, "id") == 0 || strcmp(curToken.lex, "intc") == 0 || strcmp(curToken.lex, "charc") == 0) { 5. scanf("%s", curToken.sem); 6. } 7. else { 8. strcpy(curToken.sem, NULL); 9. } 10. } 11. }   （2）Match()：匹配非终极符，若匹配成功则根据类型决定是否读入token的语义信息sem部分，并返回1；若匹配失败则返回0。   1. int Match(const char\* key) 2. { 3. if (strcmp(curToken.sem, key) == 0) { *//匹配成功，读入下一个token* 4. readToken(); 5. strcpy(tmp, curToken.sem); 6. return 1; 7. } 8. else 9. return 0; 10. }   （3）递归子程序举例：格式一，创建并返回结点。   1. TreeNode\* TypeDecList() 2. { 3. TreeNode\* p0 = (TreeNode\*)malloc(sizeof(TreeNode)); 4. p0->lineno = curToken.lineno; 5. p0->nodekind = DecK; 6. TypeId(p0); 7. Match("="); 8. TypeDef(p0); 9. Match(";"); 10. TreeNode\* p1 = TypeDecMore(); 11. if (p1) { 12. p0->sibling = p1; 13. } 14. return p0; 15. }   （4）递归子程序举例：格式二，修改上一级传递的结点属性。   1. void TypeDef(TreeNode\* t) 2. { 3. if (t) { 4. if (strcmp(curToken.lex, "integer") == 0 || strcmp(curToken.lex, "char") == 0) { 5. BaseType(t); 6. } 7. else if (strcmp(curToken.lex, "array") == 0 || strcmp(curToken.lex, "record") == 0) { 8. StructureType(t); 9. } 10. else if (strcmp(curToken.lex, "id") == 0) { 11. Match("id"); 12. t->kind.dec = DecKind(IdK); 13. strcpy(t->type\_name, tmp); 14. } 15. else 16. return; 17. } 18. else 19. return; 20. } | | |
| **程序界面及运行截图** | | |
| 界面的主函数代码如下：    运行效果如下：    输入文件如下：    输出Token文件如下：    语法分析不产生中间结果文件。 | | |