# **实验一 词法分析器实验：基于自动机的词法分析器**

## 实验目的

通过本实验的实现，可以深入理解词法分析的原理，掌握词法分析器的设计与实现方法。通过构建一个基于自动机的词法分析器，能够更好地理解词法分析中各类标记的识别方法以及如何将源代码中的关键字、标识符、运算符、分隔符等识别为特定的词法单元（Token）。

## 问题描述

### 实验内容

设计一个词法分析器，功能包括：

输入：读取源程序的字符串内容。源程序存储在编码格式为ANSI的文本文件中，文件名通过命令行参数传入。

输出：解析输入的源程序字符串，输出Token序列到标准输出。

### **实验功能**

1. 实现预处理功能

源程序中可能包含有对程序执行无意义的符号，要求将其剔除。首先编制一个源程序的输入过程，从键盘、文件或文本框输入若干行语句，依次存入输入缓冲区（字符型数据）；然后编制一个预处理子程序，去掉输入串中的回车符、换行符和跳格符等编辑性文字；把多个空白符合并为一个；去掉注释。

1. 实现词法分析功能

输入：所给文法的源程序字符串。

输出：二元组（syn,token 或 sum）构成的序列。其中，

syn 为单词种别码。

Token 为存放的单词自身字符串。

Sum 为整型常量。

具体实现时，可以将单词的二元组用结构进行处理。

1. 待分析的 C 语言子集的词法(可以自行扩充，也可以按照 C 语言的词法定义)
2. 关键字

main

if then else

while do

repeat until

for from to step

switch of case default

return

nteger real char bool

and or not mod

read write

所有关键字都是小写。

1. 运算符和界符

+ - \* / : := < <> <= > >= = ; ( ) #

1. 其他标记 ID 和 NUM

通过以下正规式定义其他标记：

ID→letter(letter|digit)\*

NUM→digit digit\*

letter→a|…|z|A|…|Z

digit→0|…|9…

1. 空格由空白、制表符和换行符组成

空格一般用来分隔 ID、NUM、专用符号和关键字，词法分析阶段通常被忽

略。

### **实现原理与方法**

基于自动机的原理，设计状态转换来识别源代码中的Token类型，如关键字、标识符、常量、运算符、分隔符等。通过状态转换的方式区分不同类型的Token并存储，以便输出Token序列。

## 软件设计方法的选择

在设计时采用了面向对象的方法，将词法分析的不同功能模块化，以便于维护和扩展。本程序主要由以下几个类和模块组成：

**Token类：**代表词法单元，包括种别码、字符串表示和数值表示，用于存储和传递分析结果。

**Lexer类：**实现词法分析的主要功能，包括对源代码的预处理、Token识别和生成。

**Table模块：**保存关键字和符号的查找表，以便快速查找每个词法单元对应的种别码。

开发环境为C++语言，开发工具为Visual Studio

## 分析模型

**对象模型：**

Token类:表示一个词法单元，包含种别码（syn）、字符串表示（token）、数值表示（sum）。

Lexer:类负责对源代码进行词法分析，包括预处理（如去除注释和多余空格）和词法单元识别（自动机状态转换实现）。

**用例模型：**

用例1：文件读取。通过命令行参数输入文件名，将文件内容读取至字符串，以便后续分析。

用例2：词法分析。对源代码字符串逐字符扫描，识别出关键字、标识符、数字和符号，并生成Token序列。

用例3：输出Token序列。根据生成的Token序列，逐一输出每个词法单元的种别码和内容，按关键字、符号、标识符和数字分类。

词法分析器的状态转换图如图1所示：

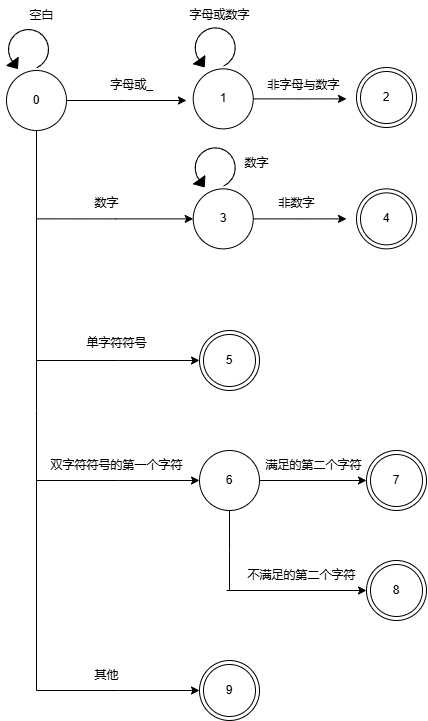


图1 状态转换图

## 设计模型

**模块结构：**

该词法分析器主要分为Lexer模块、Token模块、Table模块，模块结构图如图2所示。

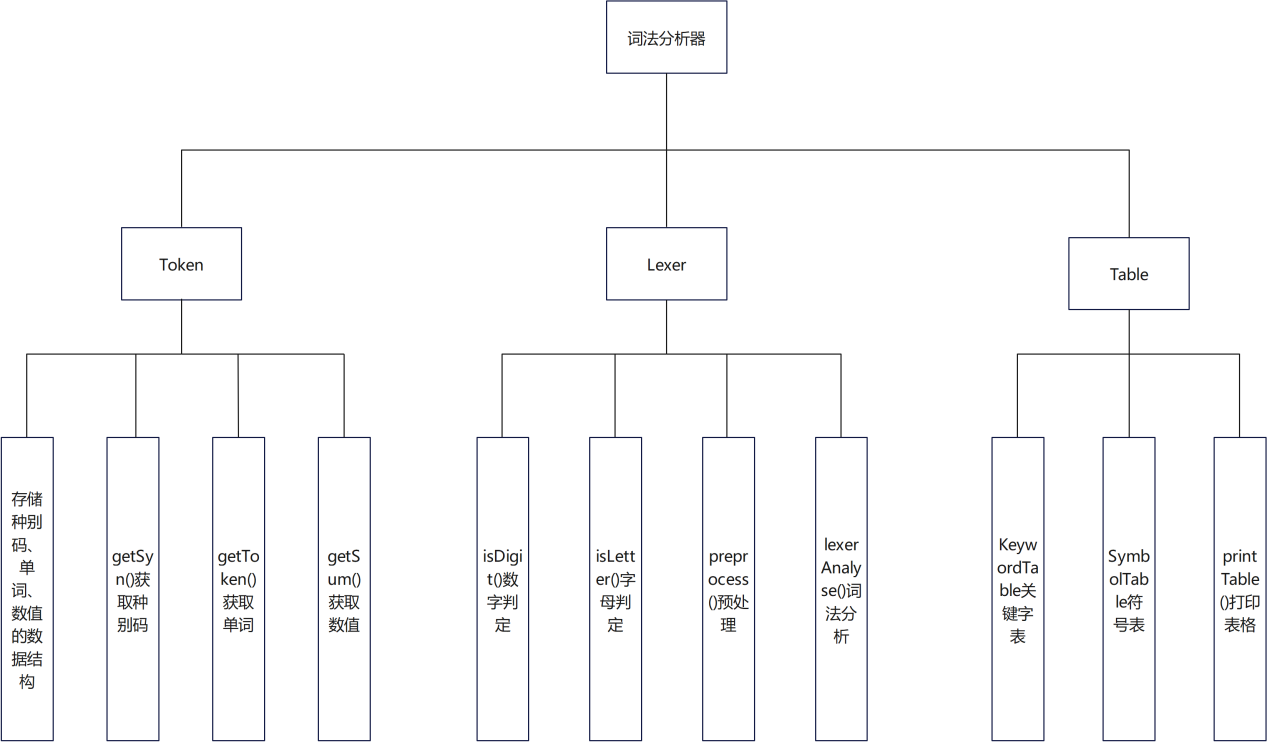
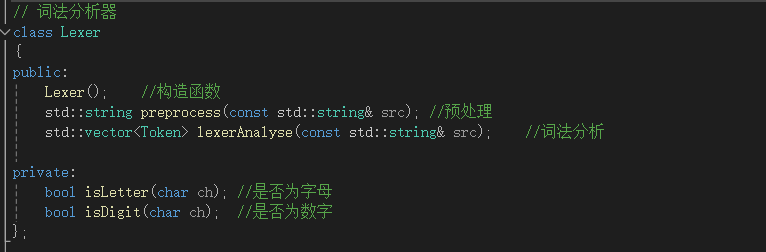


图2 词法分析器模块结构图

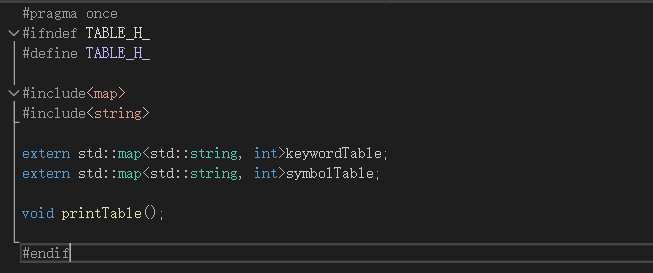
Lexer模块：包括preprocess()和lexerAnalyse()方法，分别实现代码预处理和词法分析的功能。

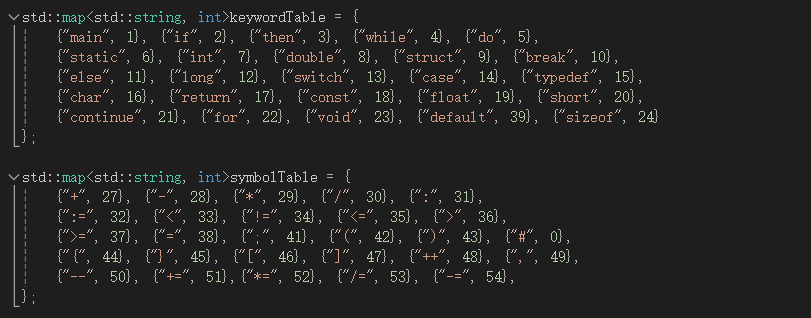


Token模块：包含Token的数据结构，提供获取种别码和内容的方法。



Table模块：保存关键字表和符号表，辅助词法分析过程中的查找和验证。





主要数据结构：

使用std::map<std::string, int>来存储关键字和符号的映射表，实现快速查找。

使用std::vector<Token>存储分析生成的Token序列，以便逐一输出。

## 主要算法描述

词法分析的实现分为三个步骤：预处理、自动机状态转换和Token生成。

1. **预处理：清理源代码，去除注释和多余空白字符，便于后续分析。**
2. 注释清理：多行注释使用/\* \*/标识，单行注释使用//标识，预处理函数preprocess()跳过这些内容。
3. 空白字符规范化：将多余的空格、制表符和换行符简化为单个空格，以分隔Token。
4. **自动机状态转换：逐字符读取并识别词法单元。**
5. 识别关键字和标识符：若以字母或下划线开头，进入识别标识符的状态，检查是否在关键字表中，若在则标识为关键字，否则标识为标识符。
6. 识别数字：若以数字开头，进入识别数字的状态，支持整数和小数。
7. 识别运算符和分隔符：针对单字符和双字符运算符（如>=, <=），直接通过符号表查找种别码。
8. 未定义符号：若字符未被识别，则输出为“未定义符号”。
9. **Token生成：将识别出的词法单元保存为Token对象，包括种别码、字符串内容和数值形式，最终将结果存入Token序列中。**

**主要算法流程图如下：**

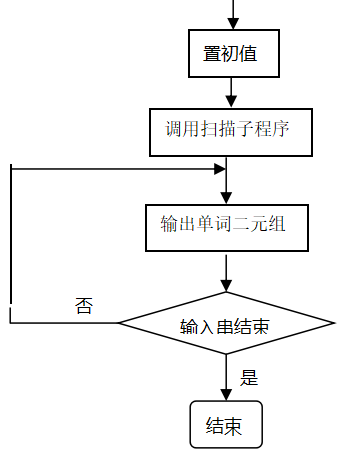


图3 主程序流程图

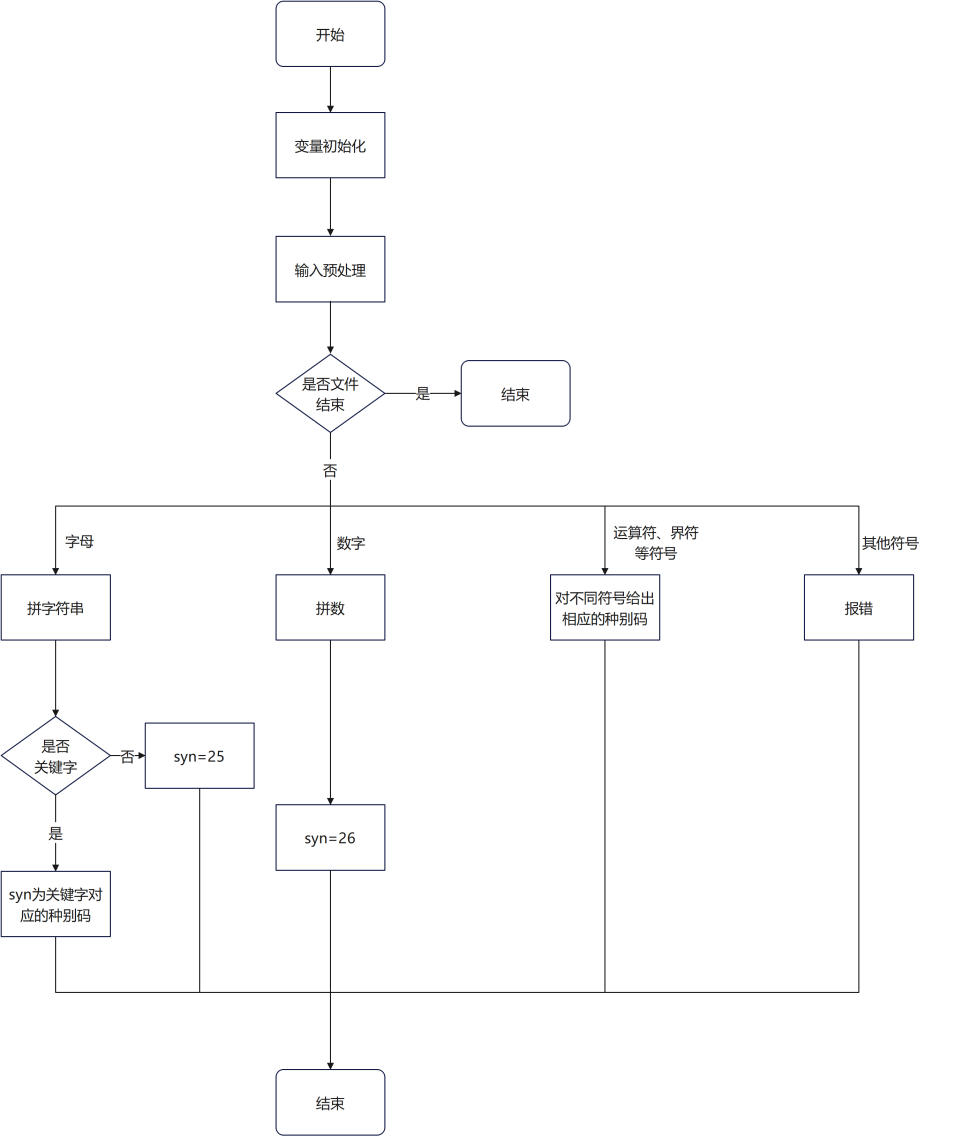


图4 词法分析程序流程图

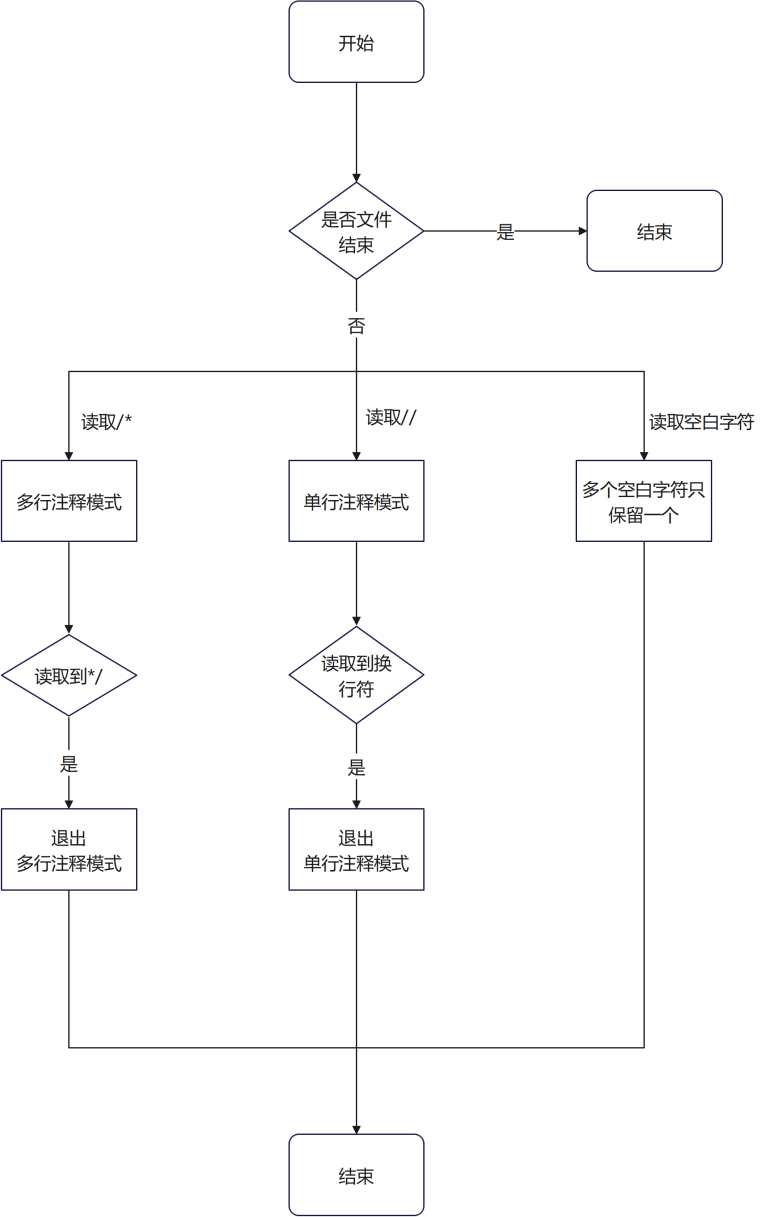


图5 预处理程序流程图

## 测试数据与测试效果

**测试样例1：**

**输入文件内容：**

1. */\*这是一个测试文件\*/*
2. void main()
3. {
4. */\**
5. 在本文件中定义基本类型、各种程序结构以及自定义类型
6. \*/
7. double \_a = -2.5;     *// 整型*
8. float b\_123 = 3 - 2;
9. double c = 2;
10. const char \_1 = 'd';
11. int arr[5] = {0, 1, 2, 3, 4};
12. int a = -1;
13. float b = 1;
14. *// 单行注释测试*
15. *// 单行注释测试*
16. typedef struct
17. {
18. long e;
19. short f;
20. static int g;
21. }
22. if(a == -1)
23. {
24. b = 1;
25. }
26. else
27. {
28. b = 2;
29. }
30. for(int i=0;i<5;i++)
31. {
32. if(i==0)
33. continue;
34. a++;
35. }
36. while(1)    a--;
37. switch(a)
38. {
39. case 0:
40. return sizeof(int);
41. case 1:
42. break;
43. default:
44. return 0;
45. }
46. return 0;
47. }

**输出：**

1. Keyword/Symbol      Type      syn
2. --------------------------------------------------
3. break               Keyword   10
4. case                Keyword   14
5. char                Keyword   16
6. const               Keyword   18
7. continue            Keyword   21
8. default             Keyword   39
9. do                  Keyword   5
10. double              Keyword   8
11. else                Keyword   11
12. float               Keyword   19
13. for                 Keyword   22
14. if                  Keyword   2
15. int                 Keyword   7
16. long                Keyword   12
17. main                Keyword   1
18. return              Keyword   17
19. short               Keyword   20
20. sizeof              Keyword   24
21. static              Keyword   6
22. struct              Keyword   9
23. switch              Keyword   13
24. then                Keyword   3
25. typedef             Keyword   15
26. void                Keyword   23
27. while               Keyword   4
28. !=                  Symbol    34
29. *#                   Symbol    0*
30. (                   Symbol    42
31. )                   Symbol    43
32. \*                   Symbol    29
33. \*=                  Symbol    52
34. +                   Symbol    27
35. ++                  Symbol    48
36. +=                  Symbol    51
37. ,                   Symbol    49
38. -                   Symbol    28
39. --                  Symbol    50
40. -=                  Symbol    54
41. /                   Symbol    30
42. /=                  Symbol    53
43. :                   Symbol    31
44. :=                  Symbol    32
45. ;                   Symbol    41
46. <                   Symbol    33
47. <=                  Symbol    35
48. =                   Symbol    38
49. >                   Symbol    36
50. >=                  Symbol    37
51. [                   Symbol    46
52. ]                   Symbol    47
53. {                   Symbol    44
54. }                   Symbol    45
55. syn: 23 Keyword: void
56. syn: 1 Keyword: main
57. syn: 42 Symbol: (
58. syn: 43 Symbol: )
59. syn: 44 Symbol: {
60. syn: 8 Keyword: double
61. syn: 25 Token: \_a
62. syn: 38 Symbol: =
63. syn: 26 Digit: -2.5
64. syn: 41 Symbol: ;
65. syn: 19 Keyword: float
66. syn: 25 Token: b\_123
67. syn: 38 Symbol: =
68. syn: 26 Digit: 3
69. syn: 28 Symbol: -
70. syn: 26 Digit: 2
71. syn: 41 Symbol: ;
72. syn: 8 Keyword: double
73. syn: 25 Token: c
74. syn: 38 Symbol: =
75. syn: 26 Digit: 2
76. syn: 41 Symbol: ;
77. syn: 18 Keyword: const
78. syn: 16 Keyword: char
79. syn: 25 Token: \_1
80. syn: 38 Symbol: =
81. syn: -1 ERROR: Undefined Symbol : [']
82. syn: 25 Token: d
83. syn: -1 ERROR: Undefined Symbol : [']
84. syn: 41 Symbol: ;
85. syn: 7 Keyword: int
86. syn: 25 Token: arr
87. syn: 46 Symbol: [
88. syn: 26 Digit: 5
89. syn: 47 Symbol: ]
90. syn: 38 Symbol: =
91. syn: 44 Symbol: {
92. syn: 26 Digit: 0
93. syn: 49 Symbol: ,
94. syn: 26 Digit: 1
95. syn: 49 Symbol: ,
96. syn: 26 Digit: 2
97. syn: 49 Symbol: ,
98. syn: 26 Digit: 3
99. syn: 49 Symbol: ,
100. syn: 26 Digit: 4
101. syn: 45 Symbol: }
102. syn: 41 Symbol: ;
103. syn: 7 Keyword: int
104. syn: 25 Token: a
105. syn: 38 Symbol: =
106. syn: 26 Digit: -1
107. syn: 41 Symbol: ;
108. syn: 19 Keyword: float
109. syn: 25 Token: b
110. syn: 38 Symbol: =
111. syn: 26 Digit: 1
112. syn: 41 Symbol: ;
113. syn: 15 Keyword: typedef
114. syn: 9 Keyword: struct
115. syn: 44 Symbol: {
116. syn: 12 Keyword: long
117. syn: 25 Token: e
118. syn: 41 Symbol: ;
119. syn: 20 Keyword: short
120. syn: 25 Token: f
121. syn: 41 Symbol: ;
122. syn: 6 Keyword: static
123. syn: 7 Keyword: int
124. syn: 25 Token: g
125. syn: 41 Symbol: ;
126. syn: 45 Symbol: }
127. syn: 2 Keyword: if
128. syn: 42 Symbol: (
129. syn: 25 Token: a
130. syn: 38 Symbol: =
131. syn: 38 Symbol: =
132. syn: 26 Digit: -1
133. syn: 43 Symbol: )
134. syn: 44 Symbol: {
135. syn: 25 Token: b
136. syn: 38 Symbol: =
137. syn: 26 Digit: 1
138. syn: 41 Symbol: ;
139. syn: 45 Symbol: }
140. syn: 11 Keyword: else
141. syn: 44 Symbol: {
142. syn: 25 Token: b
143. syn: 38 Symbol: =
144. syn: 26 Digit: 2
145. syn: 41 Symbol: ;
146. syn: 45 Symbol: }
147. syn: 22 Keyword: for
148. syn: 42 Symbol: (
149. syn: 7 Keyword: int
150. syn: 25 Token: i
151. syn: 38 Symbol: =
152. syn: 26 Digit: 0
153. syn: 41 Symbol: ;
154. syn: 25 Token: i
155. syn: 33 Symbol: <
156. syn: 26 Digit: 5
157. syn: 41 Symbol: ;
158. syn: 25 Token: i
159. syn: 48 Symbol: ++
160. syn: 43 Symbol: )
161. syn: 44 Symbol: {
162. syn: 2 Keyword: if
163. syn: 42 Symbol: (
164. syn: 25 Token: i
165. syn: 38 Symbol: =
166. syn: 38 Symbol: =
167. syn: 26 Digit: 0
168. syn: 43 Symbol: )
169. syn: 21 Keyword: continue
170. syn: 41 Symbol: ;
171. syn: 25 Token: a
172. syn: 48 Symbol: ++
173. syn: 41 Symbol: ;
174. syn: 45 Symbol: }
175. syn: 4 Keyword: while
176. syn: 42 Symbol: (
177. syn: 26 Digit: 1
178. syn: 43 Symbol: )
179. syn: 25 Token: a
180. syn: 50 Symbol: --
181. syn: 41 Symbol: ;
182. syn: 13 Keyword: switch
183. syn: 42 Symbol: (
184. syn: 25 Token: a
185. syn: 43 Symbol: )
186. syn: 44 Symbol: {
187. syn: 14 Keyword: case
188. syn: 26 Digit: 0
189. syn: 31 Symbol: :
190. syn: 17 Keyword: return
191. syn: 24 Keyword: sizeof
192. syn: 42 Symbol: (
193. syn: 7 Keyword: int
194. syn: 43 Symbol: )
195. syn: 41 Symbol: ;
196. syn: 14 Keyword: case
197. syn: 26 Digit: 1
198. syn: 31 Symbol: :
199. syn: 10 Keyword: break
200. syn: 41 Symbol: ;
201. syn: 39 Keyword: default
202. syn: 31 Symbol: :
203. syn: 17 Keyword: return
204. syn: 26 Digit: 0
205. syn: 41 Symbol: ;
206. syn: 45 Symbol: }
207. syn: 17 Keyword: return
208. syn: 26 Digit: 0
209. syn: 41 Symbol: ;
210. syn: 45 Symbol: }

**测试样例2：**

**输入文件：**

1. /\*这是一个测试文件
2. void main()
3. {
4. int a = -1;     // 整型
5. float b = 3 - 2;
6. double c = 2;
7. const char d = 'd';
8. int arr[5] = {0, 1, 2, 3, 4};
9. // 单行注释测试
10. // 单行注释测试
11. typedef struct
12. {
13. long e;
14. short f;
15. static int g;
16. }
17. if(a == -1)
18. {
19. b = 1;
20. }
21. else
22. {
23. b = 2;
24. }
25. for(int i=0;i<5;i++)
26. {
27. if(i==0)
28. continue;
29. a++;
30. }
31. while(1)    a--;
32. switch(a)
33. {
34. case 0:
35. return sizeof(int);
36. case 1:
37. break;
38. default:
39. return 0;
40. }
41. return 0;
42. }

**输出：**

1. Keyword/Symbol      Type      syn
2. --------------------------------------------------
3. break               Keyword   10
4. case                Keyword   14
5. char                Keyword   16
6. const               Keyword   18
7. continue            Keyword   21
8. default             Keyword   39
9. do                  Keyword   5
10. double              Keyword   8
11. else                Keyword   11
12. float               Keyword   19
13. for                 Keyword   22
14. if                  Keyword   2
15. int                 Keyword   7
16. long                Keyword   12
17. main                Keyword   1
18. return              Keyword   17
19. short               Keyword   20
20. sizeof              Keyword   24
21. static              Keyword   6
22. struct              Keyword   9
23. switch              Keyword   13
24. then                Keyword   3
25. typedef             Keyword   15
26. void                Keyword   23
27. while               Keyword   4
28. !=                  Symbol    34
29. #                   Symbol    0
30. (                   Symbol    42
31. )                   Symbol    43
32. \*                   Symbol    29
33. \*=                  Symbol    52
34. +                   Symbol    27
35. ++                  Symbol    48
36. +=                  Symbol    51
37. ,                   Symbol    49
38. -                   Symbol    28
39. --                  Symbol    50
40. -=                  Symbol    54
41. /                   Symbol    30
42. /=                  Symbol    53
43. :                   Symbol    31
44. :=                  Symbol    32
45. ;                   Symbol    41
46. <                   Symbol    33
47. <=                  Symbol    35
48. =                   Symbol    38
49. >                   Symbol    36
50. >=                  Symbol    37
51. [                   Symbol    46
52. ]                   Symbol    47
53. {                   Symbol    44
54. }                   Symbol    45

## 实验总结

在本实验的实现过程中，遇到了以下问题及解决方法：

1. 区分负号与减号：在处理负数时，需要判断是否为减号。例如在x = -10;中，-10应视为负数，而在x - 1;中，-应视为减号。通过检查符号前后字符类型，成功区分了两种情况。
2. 注释处理的边界条件：处理多行和单行注释时，需要确保正确识别注释的开始和结束符号。同时注意注释可能嵌套或在代码行内，已通过设置标志位控制进入和退出注释模式。
3. 空白字符的规范化：处理多余空白字符时，确保空白符仅在需要的情况下保留一个空格，避免影响Token分隔。通过检查结果序列末尾字符是否为空格，实现了多余空白的去除。

本实验帮助我加深了对词法分析器工作流程的理解，特别是自动机的状态转换在词法分析中的应用。通过实现代码，提升了C++编程能力，熟悉了类和模块的使用，以及如何通过分离功能模块提升代码的可读性和可维护性。