

北 京 邮 电 大 学

# 实 验 报 告

课程名称：编译原理

实验名称：词法分析

学院：计算机学院

班级：2018211304

学号：2018211208

姓名：谢睿

教师：李文生

2020 年 10 月 17 日

## 目录

一. 实验题目及要求.....	3
1. 实验题目：C 语言词法分析程序的设计与实现.....	3
2. 实验要求：.....	3
3. 实现方法：.....	3
二. 程序设计说明.....	3
1. 需求分析：.....	3
2. 算法流程：.....	3
3. 程序中使用的主要变量以及结构体说明：.....	5
4. 程序中功能模块设计：.....	6
5. 记号流输出含义：.....	7
6. 可识别的错误类型：.....	8
三. 程序测试：.....	8
四. C++实现-代码.....	10
五. 利用 LEX 实现.....	28
1.lex 环境安装与配置：.....	28
2.编写 lex.l 源程序.....	29
3.测试 lex 程序.....	30
4.lex.l 源代码：.....	31
六. 实验心得与总结.....	38
1.实验心得：.....	38
2.实验总结与改进方向：.....	38

# 一. 实验题目及要求

## 1. 实验题目：C 语言词法分析程序的设计与实现

## 2. 实验要求：

- 1) 可以识别出用 C 语言编写的源程序中的每个单词符号,并以记号的形式输出每个单词符号。
- 2) 可以识别并跳过源程序中的注释。
- 3) 可以统计源程序中的语句行数、各类单词的个数、以及字符总数,并输出统计结果。
- 4) 检查源程序中存在的词法错误,并报告错误所在的位置。
- 5) 对源程序中出现的错误进行适当的恢复,使词法分析可以继续进行,对源程序进行一次扫描,即可检查并报告源程序中存在的所有词法错误。

## 3. 实现方法：

- 1) 采用从 C/C++ 作为实现语言,手工编写词法分析程序。
- 2) 编写 LEX 源程序,利用 LEX 编译程序自动生成词法分析程序。

# 二. 程序设计说明

## 1. 需求分析：

词法分析主要完成识别代码中的单词并输出记号流。

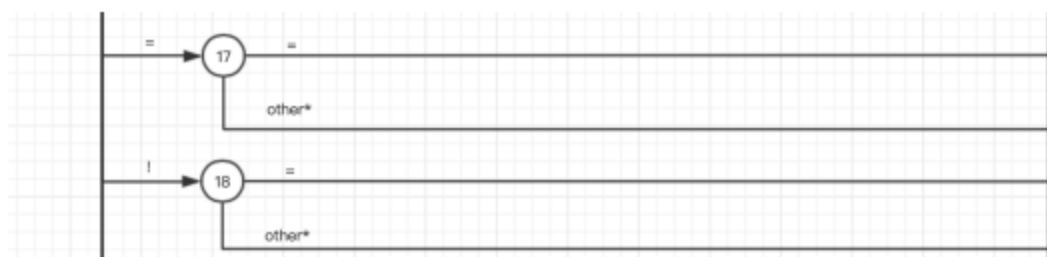
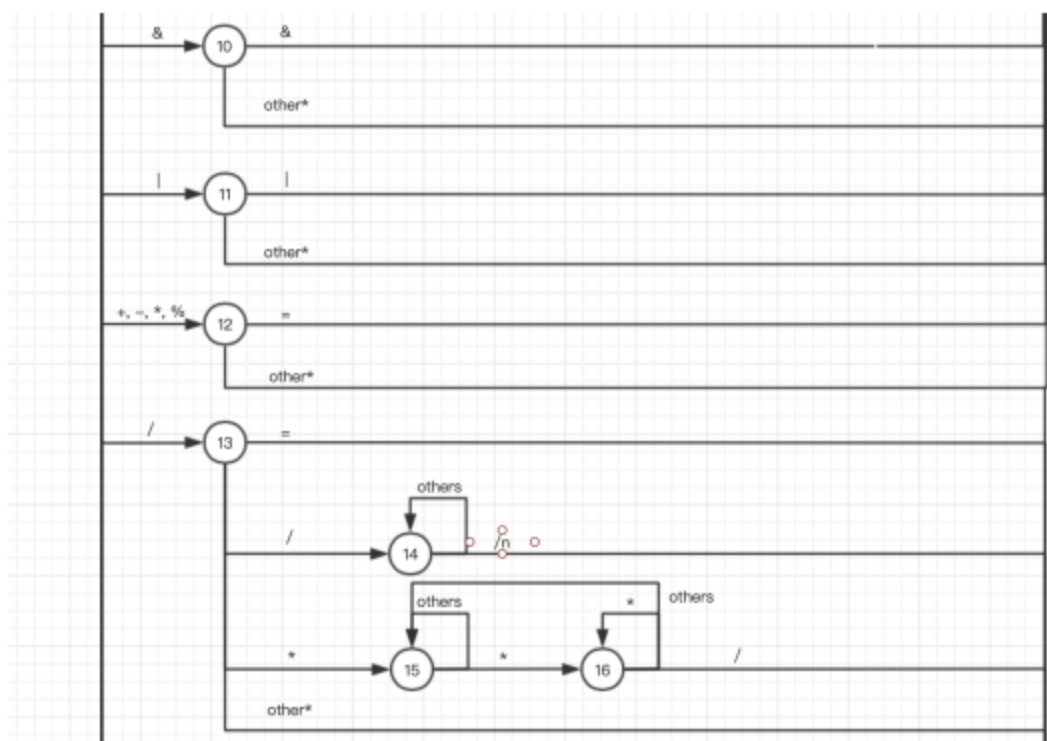
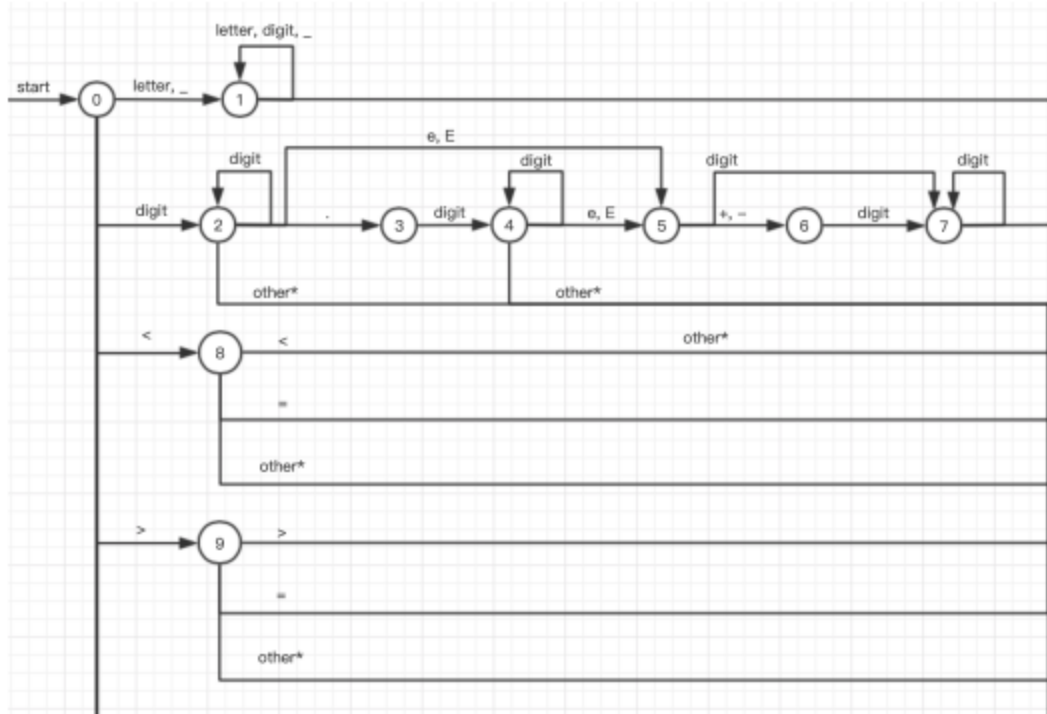
对于 C 语言,需要识别的单词类型主要有标识符、关键字(保留字)、宏定义以及预处理(以 # 开头的语句)、数字、字符串、字符、运算符、标点符号、赋值号(= 号以及 += 等包含运算的赋值)、比较运算符、注释。

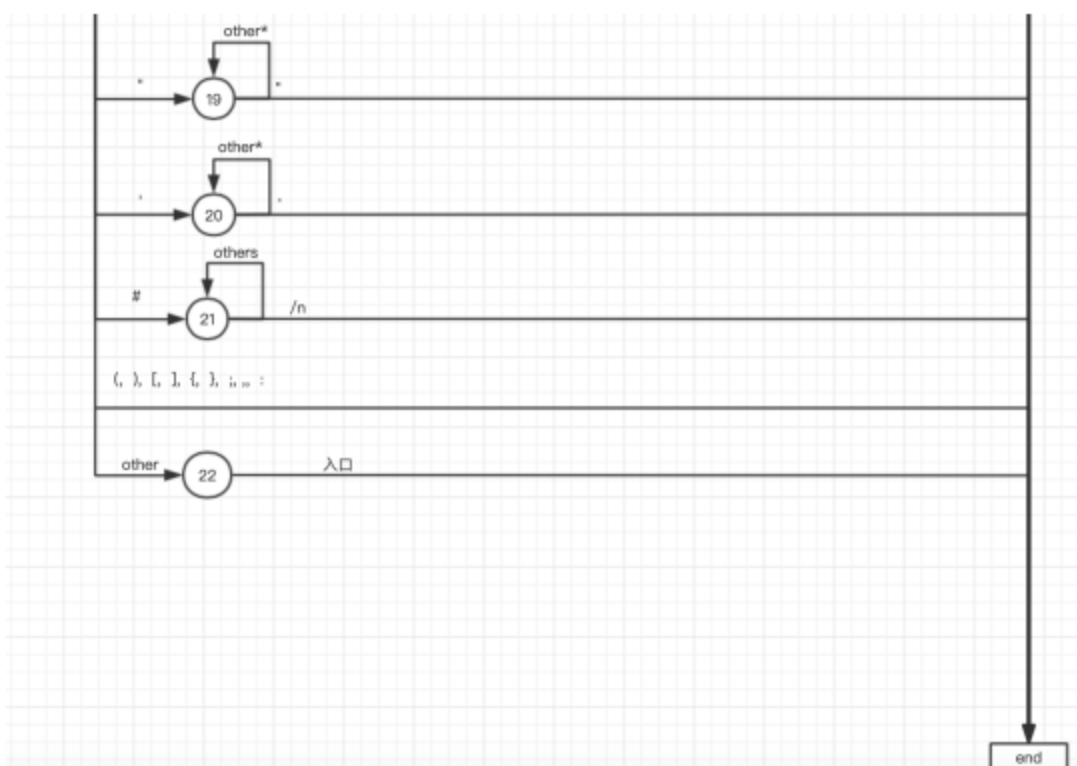
我们要根据每种单词的构词规则识别并分辨单词。对于出现的不符合构词规则的单词应该舍弃或作一定恢复,并记录为一个错误,然后继续读取并分析。所以程序采用有限状态自动机转移的设计方法,可以有效识别各种单词。

最后将识别出的记号流以(记号,属性)的二元组形式记录并输出。

## 2. 算法流程：

使用状态转换图描述算法流程





其中需要注意的是，为了分辨出以数字开头的标识符，上图状态 2, 3, 4, 5, 6, 7 在接收单词或下划线后，会转移到状态 1，为简洁起见图中没有全部表示。

### 3. 程序中使用的主要变量以及结构体说明：

#### 3.1 结构体定义：

结构“all\_words”，用于以二元组的形式储存记号流中的每个单词。两个成员分别代表单词的记号与属性。

```

1. // 储存记号流中每个词以及相关信息
2. typedef struct all_words
3. {
4.     string sign; // 记号
5.     string attribute; // 属性
6. }ALL_WORDS;
  
```

结构“all\_errors”，用于以三元组的形式记录程序中的错误，三个成员分别表示错误的信息，错误的单词（token），错误的行数。

```

1. // 记录程序中的错误
2. typedef struct all_errors
3. {
4.     string errorInf; // 记录错误信息
5.     string errorToken; // 记录错误 token
6.     int lines; // 记录错误行数
  
```

```
7. }ALL_ERRORS;
```

### 3.2 宏定义:

```
1. #define BUF_SIZE 2048 // 输入缓存区大小
2. #define KEY_NUM 28 // 关键字数里
```

### 3.3 变量:

关键字(保留字)字符串数组: 用于记录所有关键字, 便于查询识别出的标识符是否为用户自定义。

```
1. // 关键字
2. string key[28] =
3. { "int", "long", "short", "float", "double", "char", "unsigned", "signed", "
  const", "void", "struct", "union",
4. "if", "else", "goto", "switch", "case", "do", "while", "for", "continue", "b
  reak", "return", "default", "typedef",
5. "extern", "static", "main" };
```

### 其他全局变量:

```
1. vector<string> IDs; // 识别出的符号表
2. vector<all_words> wordFlow; // 识别出的记号流
3. vector<ALL_ERRORS> errorFlow; // 识别出的错误
4. string FILE_NAME = "test.txt"; // 默认测试文件
5. FILE *filePtr; // 打开的文件指针
6. ofstream fout; // 输出文件流
7. char buf[BUF_SIZE]; // 输入缓存(分为左右两半)
8. int forwardPtr; // 缓存区前向指针
9. int validChNum = 0; // 有效字符数(不含空格, 换行等)
10. int allChNum = 0; // 所有字符数(含空格, 换行等)
11. int allRows = 0; // 总行数
12. int nowRows = 0; // 当前行数
```

### 局部变量:

```
1. int state = 0; // 状态
2. char C = buf[0]; // 当前处理的一个字符
3. string token = ""; // 当前处理的字符串
4. ALL_WORDS tmpWord = { "", "" }; // 待加入记号流的单词
5. ALL_ERRORS tmpError = { "", "", 0 }; // 待记录的错误
```

## 4. 程序中功能模块设计:

名称	功能
----	----

<code>int get_buf(int part)</code>	从文件获取输入并存入 buf，分左右两半区。
<code>void get_char(char&amp; C)</code>	根据 forwardPtr 的指示获取一个字符 c 并移动 forwardPtr
<code>int iskey(const string token)</code>	判断 token 是否为关键字
<code>void get_nbc(char&amp; C)</code>	跳过空白
<code>void cat(string&amp; token, const char C)</code>	把 C 拼接在 token 后
<code>bool isLetter(const char C)</code>	判断是否为字母
<code>bool isDigit(const char C)</code>	判断是否为数字
<code>void retract()</code>	forwardPtr 后退一个字符
<code>int inIDs(const string token)</code>	判断并查询标识符是否存在
<code>int table_insert(const string token)</code>	在符号表中插入单词 token
<code>void words_insert(const ALL_WORDS newWord)</code>	在记号流中插入一个新记号
<code>void error(ALL_ERRORS newError)</code>	记录一个新错误
<code>int lexical_analysis()</code>	词法分析主模块
<code>int lexInit()</code>	初始化模块
<code>void print_result()</code>	输出模块（到命令行和文件

## 5. 记号流输出含义：

类型	记号	属性
标识符	Id	标识符内容
关键字	关键字本身	
数字	num	数字内容
移位运算	shift	<<或>>
比较运算符	relop	<, >, <= 等
逻辑运算符	logic	&&或
取地址运算符	addre	&
赋值运算符	assin	+=, -= 等
运算符	opera	+, -, *, / 等
注释	annota	//或/**/及内容
字符串	str	字符串内容
字符	char	字符内容
预处理，宏定义	pretrt	#以及之后的内容
标点，分界符	delimi	(, ), {, }, 等

## 6. 可识别的错误类型：

错误	输出错误信息
标识符以数字开头	Identifier starts with a number
小数点后缺少内容	Missing content after the decimal point
指数 E 后缺少内容	Missing sign after exponent
含有+号的指数不合法（缺少内容）	The number after the index is illegal
逻辑运算符  不完整	"  " is incomplete
比较运算符!=不完整	"!=" is incomplete
不合法的字符	Unrecognized symbol

## 三. 程序测试：

将如下内容保存至 test.txt 作为输入：其中包含两处错误，以及各类单词。

```
1. #include <stdio.h>
2. int main()
3. {
4.     int 2a = 1;
5.     float i =1.2;
6.     _b = c + i;
7.     printf("Hello world");
8.     // this ia a line of annotation
9.     /* this is sevel lines of
10.    annotationsssss*/
11.     return 0;
12.     ^
13. }
```

运行程序，输入要分析的文件名：

```
Please enter the file name to be analyzed: test.txt
```

输出记号流：



```

Token stream:32
sign      attribute
pretrt    #include <stdio.h>
           int
           main
delimi    (
delimi    )
delimi    {
           int
assin     =
num       1
delimi    ;
           float
id        i
assin     =
num       1.2
delimi    ;
id        _b
assin     =
id        c
opera     +
id        i
delimi    ;
id        printf
delimi    (
str       "Hello world"
delimi    )
delimi    ;
annota    // this ia a line of annotation
annota    /* this is sevel lines of
           annotationsssss*/
           return
num       0
delimi    ;
delimi    }

```

输出标识符表：

```

Symbol table:4
i
_b
c
printf

```

输出错误：

```

Errors:2
line      token      Inf
4         2a         Identifier starts with a number
12        ^         Unrecognized symbol

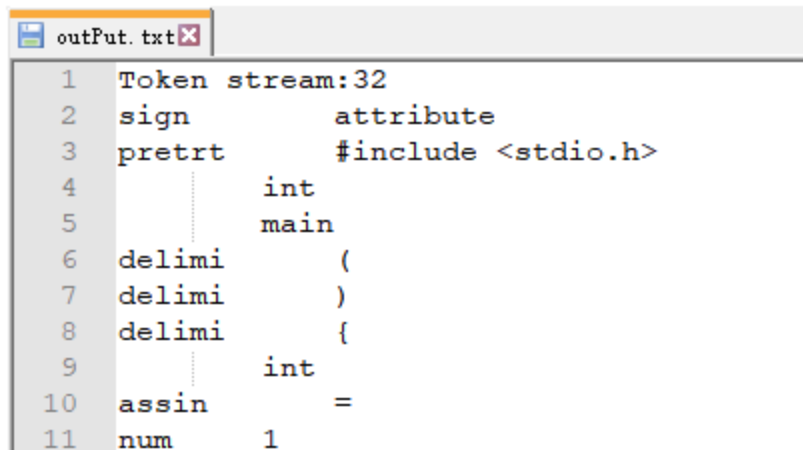
```

输出统计信息：

```
Total number of rows: 13
Total characters: 192
Total valid characters: 146
Total words: 4
```

可以看到，程序正常运行并输出了正确的结果。

在向命令行输出的同时，还将结果存于文件“outPut.txt”中。



```
1 Token stream:32
2 sign      attribute
3 prettrt   #include <stdio.h>
4           int
5           main
6 delimi     (
7 delimi     )
8 delimi     {
9           int
10 assin      =
11 num       1
```

## 四. C++实现-代码

```
1. #include <iostream>
2. #include <fstream>
3. #include <stdio.h>
4. #include <string.h>
5. #include <stdlib.h>
6. #include <vector>
7. #include <map>
8. #define BUF_SIZE 2048 // 输入缓存区大小
9. #define KEY_NUM 28 // 关键字数里
10. using namespace std;
11.
12. // 储存记号流中每个词以及相关信息
13. typedef struct all_words
14. {
15.     string sign; // 记号
16.     string attribute; // 属性
17. }ALL_WORDS;
18.
19. // 记录程序中的错误
20. typedef struct all_errors
```

```

21. {
22.     string errorInf; // 记录错误信息
23.     string errorToken; // 记录错误 token
24.     int lines; // 记录错误行数
25. }ALL_ERRORS;
26.
27. // 关键字
28. string key[28] =
29. { "int", "long", "short", "float", "double", "char", "unsigned", "signed", "
    const", "void", "struct", "union",
30. "if", "else", "goto", "switch", "case", "do", "while", "for", "continue", "b
    reak", "return", "default", "typedef",
31. "extern", "static", "main" };
32.
33. vector<string> IDs; // 识别出的符号表
34. vector<all_words> wordFlow; // 识别出的记号流
35. vector<ALL_ERRORS> errorFlow; // 识别出的错误
36. string FILE_NAME = "test.txt"; // 默认测试文件
37. FILE *filePtr; // 打开的文件指针
38. ofstream fout; // 输出文件流
39. char buf[BUF_SIZE]; // 输入缓存(分为左右两半)
40. int forwardPtr; // 缓存区前向指针
41. int validChNum = 0; // 有效字符数(不含空格, 换行等)
42. int allChNum = 0; // 所有字符数(含空格, 换行等)
43. int allRows = 0; // 总行数
44. int nowRows = 0; // 当前行数
45.
46.
47.
48. int get_buf(int part) // 返回读取的字符数
49. {
50.     char tmp = '\0';
51.     int i = 0;
52.     int offset = 0; // buf 偏移量
53.
54.     if (part == 0) // 获取右半区
55.         offset = BUF_SIZE / 2;
56.     else // 获取左半区
57.         offset = 0;
58.
59.     for (i = 0; i < BUF_SIZE / 2; i++)
60.     {
61.         tmp = fgetc(filePtr);
62.         if (tmp == EOF) // 判断文件是否结束

```

```

63.     {
64.         break;
65.     }
66.     else
67.     {
68.         buf[offset + i] = tmp;
69.         //cout << buf[offset + i] << " " << endl; // test code
70.         if (tmp == '\n') // 如果是换行，则行数计数加一
71.             allRows++;
72.         if (tmp != ' ' && tmp != '\t' && tmp != '\n') // 如果是有效字符，
            则有效字符计数加一
73.         {
74.             validChNum++;
75.         }
76.     }
77. }
78.
79. allChNum += i;
80. return i;
81. }
82.
83. void get_char(char& C) // 根据 forwardPtr 的指示获取一个字符 C 并移动 forwardPtr
84. {
85.     C = buf[forwardPtr];
86.     if (C == '\n')
87.         nowRows++;
88.
89.     if (forwardPtr == BUF_SIZE / 2)
90.         get_buf(1); // 获取右半区
91.     else if (forwardPtr == BUF_SIZE - 1)
92.         get_buf(0); // 获取左半区
93.
94.     forwardPtr = (forwardPtr + 1) % BUF_SIZE;
95. }
96.
97. int iskey(const string token)
98. {
99.     int i;
100.    for (i = 0; i < KEY_NUM; i++)
101.    {
102.        if (token == key[i])
103.            return i; // 返回 key 的序号
104.    }
105.    return -1; // 不是 key

```

```

106. }
107.
108. void get_nbc(char& C) // 跳过空白
109. {
110.     while (C == ' ' || C == '\t' || C == '\n')
111.         get_char(C);
112. }
113.
114. void cat(string& token, const char C) // 把 C 拼接在 token 后
115. {
116.     token = token + C;
117. }
118.
119. bool isLetter(const char C) // 判断是否为字母
120. {
121.     if ((C >= 'a' && C <= 'z') || (C >= 'A' && C <= 'Z'))
122.         return true;
123.     return false;
124. }
125.
126. bool isDigit(const char C) // 判断是否为数字
127. {
128.     if (C >= '0' && C <= '9')
129.         return true;
130.     return false;
131. }
132.
133. void retract() // forwardPtr 后退一个字符
134. {
135.     forwardPtr = (forwardPtr - 1 + BUF_SIZE) % BUF_SIZE;
136. }
137.
138. int inIDs(const string token) // 查询单词在符号表中的位置(行数)
139. {
140.     int i;
141.     for (i = 0; i < IDs.size(); i++)
142.     {
143.         if (token == IDs[i])
144.             return i + 1; // 返回行号(从 1 开始数)
145.     }
146.     return -1; // 不在符号表中(即这是一个新符号)
147. }
148.
149. int table_insert(const string token) // 在符号表中插入单词 token

```

```

150. {
151.     if (inIDs(token) == -1) // 是新单词
152.     {
153.         IDs.push_back(token);
154.         return IDs.size();
155.     }
156.     else
157.     {
158.         return inIDs(token);
159.     }
160. }
161.
162. void words_insert(const ALL_WORDS newWord) // 插入记号流
163. {
164.     wordFlow.push_back(newWord);
165. }
166.
167. void error(ALL_ERRORS newError)
168. {
169.     errorFlow.push_back(newError);
170. }
171.
172. int lexical_analysis() // 词法分析主模块
173. {
174.     int state = 0; // 状态
175.     char C = buf[0]; // 当前处理的一个字符
176.     string token = ""; // 当前处理的字符串
177.     ALL_WORDS tmpWord = { "", "" }; // 待加入记号流的单词
178.     ALL_ERRORS tmpError = { "", "", 0 }; // 待记录的错误
179.     while (C != '\0')
180.     {
181.         switch (state)
182.         {
183.             case 0: // 初始状态
184.                 token = "";
185.                 get_char(C);
186.                 get_nbc(C);
187.                 if (isLetter(C) || C == '_')
188.                     state = 1;
189.                 else if (isDigit(C))
190.                     state = 2;
191.                 else
192.                 {
193.                     switch (C)

```

```

194.         {
195.             case '<':state = 8; break;
196.             case '>':state = 9; break;
197.             case '&':state = 10; break;
198.             case '|':state = 11; break;
199.             case '+':
200.             case '-':
201.             case '*':
202.             case '%':state = 12; break;
203.             case '/':state = 13; break;
204.             case '=':state = 17; break;
205.             case '!':state = 18; break;
206.             case '"':state = 19; break;
207.             case '\\':state = 20; break;
208.             case '#':state = 21; break;
209.             case '(':
210.             case ')':
211.             case '[':
212.             case ']':
213.             case '{':
214.             case '}':
215.             case ';':
216.             case ':':
217.             case ',':
218.                 state = 0;
219.                 tmpWord.attribute = C;
220.                 tmpWord.sign = "delimi";
221.                 words_insert(tmpWord);
222.                 break;
223.             case EOF:return 0;
224.             default:
225.                 state = 22;
226.                 break;
227.         }
228.     }
229.     break;
230. case 1: // 标识符状态
231.     cat(token, C);
232.     get_char(C);
233.     if (isLetter(C) || isDigit(C) || C == '_')
234.     {
235.         state = 1;
236.     }
237.     else

```

```

238.         {
239.             retract();
240.             state = 0;
241.             if (iskey(token) != -1)
242.             {
243.                 tmpWord.attribute = key[iskey(token)];
244.                 tmpWord.sign = "";
245.                 words_insert(tmpWord);
246.             }
247.             else
248.             {
249.                 if (isDigit(token[0]))
250.                 {
251.                     tmpError.errorInf = "Identifier starts with a numbe
r";
252.                     tmpError.errorToken = token;
253.                     tmpError.lines = nowRows;
254.                     error(tmpError);
255.                 }
256.                 else
257.                 {
258.                     tmpWord.attribute = token;
259.                     tmpWord.sign = "id";
260.                     words_insert(tmpWord);
261.                     table_insert(token);
262.                 }
263.             }
264.         }
265.         break;
266.     case 2: // 常数状态
267.         cat(token, C);
268.         get_char(C);
269.         if (isDigit(C))
270.             state = 2;
271.         else if ((isLetter(C) && C != 'e' && C != 'E') || C == '_')
272.             state = 1;
273.         else
274.         {
275.             switch (C)
276.             {
277.                 case '.': state = 3; break;
278.                 case 'e':
279.                 case 'E': state = 5; break;
280.                 default:

```



```
281.             retract();
282.             state = 0;
283.             tmpWord.attribute = token;
284.             tmpWord.sign = "num";
285.             words_insert(tmpWord);
286.             break;
287.         }
288.     }
289.     break;
290.     case 3: // 小数点状态
291.         cat(token, C);
292.         get_char(C);
293.         if (isDigit(C))
294.             state = 4;
295.         else if (isLetter(C) || C == '_')
296.             state = 1;
297.         else
298.         {
299.             state = 0;
300.             tmpError.errorInf = "Missing content after the decimal poin
t";
301.             tmpError.errorToken = token;
302.             tmpError.lines = nowRows;
303.             error(tmpError);
304.         }
305.         break;
306.     case 4: // 小数状态
307.         cat(token, C);
308.         get_char(C);
309.         if (isDigit(C))
310.             state = 4;
311.         else if ((isLetter(C) && C != 'e' && C != 'E') || C == '_')
312.             state = 1;
313.         else
314.         {
315.             switch (C)
316.             {
317.             case 'e':
318.             case 'E': state = 5; break;
319.             default:
320.                 retract();
321.                 state = 0;
322.                 tmpWord.attribute = token;
323.                 tmpWord.sign = "num";
```

```
324.         words_insert(tmpWord);
325.         break;
326.     }
327. }
328. break;
329. case 5: // 指数状态
330.     cat(token, C);
331.     get_char(C);
332.     if (isDigit(C))
333.         state = 7;
334.     else if (isLetter(C) || C == '_')
335.         state = 1;
336.     else
337.     {
338.         switch (C)
339.         {
340.             case '+':
341.             case '-': state = 6; break;
342.             default:
343.                 retract();
344.                 state = 0;
345.                 tmpError.errorInf = "Missing sign after exponent";
346.                 tmpError.errorToken = token;
347.                 tmpError.lines = nowRows;
348.                 error(tmpError);
349.                 break;
350.         }
351.     }
352.     break;
353. case 6:
354.     cat(token, C);
355.     get_char(C);
356.     if (isDigit(C))
357.         state = 7;
358.     else if (isLetter(C) || C == '_')
359.         state = 1;
360.     else
361.     {
362.         retract();
363.         state = 0;
364.         tmpError.errorInf = "The number after the index is illegal";
365.         tmpError.errorToken = token;
366.         tmpError.lines = nowRows;
```

```

367.             error(tmpError);
368.         }
369.         break;
370.     case 7:
371.         cat(token, C);
372.         get_char(C);
373.         if (isDigit(C))
374.             state = 7;
375.         else if (isLetter(C) || C == '_')
376.             state = 1;
377.         else
378.         {
379.             retract();
380.             state = 0;
381.             tmpWord.attribute = token;
382.             tmpWord.sign = "num";
383.             words_insert(tmpWord);
384.         }
385.         break;
386.     case 8: // '<'状态
387.         cat(token, C);
388.         get_char(C);
389.         switch (C)
390.         {
391.             case '<':
392.                 cat(token, C);
393.                 state = 0;
394.                 tmpWord.attribute = token;
395.                 tmpWord.sign = "shift";
396.                 words_insert(tmpWord);
397.                 break;
398.             case '=':
399.                 cat(token, C);
400.                 state = 0;
401.                 tmpWord.attribute = token;
402.                 tmpWord.sign = "relop";
403.                 words_insert(tmpWord);
404.                 break;
405.             default:
406.                 retract();
407.                 state = 0;
408.                 tmpWord.attribute = token;
409.                 tmpWord.sign = "relop";
410.                 words_insert(tmpWord);

```

```
411.         break;
412.     }
413.     break;
414.     case 9: // '>'状态
415.         cat(token, C);
416.         get_char(C);
417.         switch (C)
418.         {
419.             case '>':
420.                 cat(token, C);
421.                 state = 0;
422.                 tmpWord.attribute = token;
423.                 tmpWord.sign = "shift";
424.                 words_insert(tmpWord);
425.                 break;
426.             case '=':
427.                 cat(token, C);
428.                 state = 0;
429.                 tmpWord.attribute = token;
430.                 tmpWord.sign = "relop";
431.                 words_insert(tmpWord);
432.                 break;
433.             default:
434.                 retract();
435.                 state = 0;
436.                 tmpWord.attribute = token;
437.                 tmpWord.sign = "relop";
438.                 words_insert(tmpWord);
439.                 break;
440.         }
441.     break;
442.     case 10: // '&'状态
443.         cat(token, C);
444.         get_char(C);
445.         if (C == '&')
446.         {
447.             cat(token, C);
448.             state = 0;
449.             tmpWord.attribute = token;
450.             tmpWord.sign = "logic";
451.             words_insert(tmpWord);
452.         }
453.     else
454.     {
```

```
455.         retract();
456.         state = 0;
457.         tmpWord.attribute = "addre";
458.         tmpWord.sign = token;
459.         words_insert(tmpWord);
460.     }
461.     break;
462. case 11: // '|'状态
463.     cat(token, C);
464.     get_char(C);
465.     if (C == '|')
466.     {
467.         cat(token, C);
468.         state = 0;
469.         tmpWord.attribute = token;
470.         tmpWord.sign = "logic";
471.         words_insert(tmpWord);
472.     }
473.     else
474.     {
475.         retract();
476.         state = 0;
477.         tmpError.errorInf = "\\|\" is incomplete";
478.         tmpError.errorToken = token;
479.         tmpError.lines = nowRows;
480.         error(tmpError);
481.     }
482.     break;
483. case 12: // '+' '-' '*' '%'状态
484.     cat(token, C);
485.     get_char(C);
486.     if (C == '=')
487.     {
488.         cat(token, C);
489.         state = 0;
490.         tmpWord.attribute = token;
491.         tmpWord.sign = "assin";
492.         words_insert(tmpWord);
493.     }
494.     else
495.     {
496.         retract();
497.         state = 0;
498.         tmpWord.attribute = token;
```

```

499.         tmpWord.sign = "opera";
500.         words_insert(tmpWord);
501.     }
502.     break;
503. case 13: // '/'状态
504.     cat(token, C);
505.     get_char(C);
506.     switch (C)
507.     {
508.     case '=':
509.         cat(token, C);
510.         state = 0;
511.         tmpWord.attribute = token;
512.         tmpWord.sign = "assin";
513.         words_insert(tmpWord);
514.         break;
515.     case '/':state = 14; break;
516.     case '*':state = 15; break;
517.     default:
518.         retract();
519.         state = 0;
520.         tmpWord.attribute = token;
521.         tmpWord.sign = "opera";
522.         words_insert(tmpWord);
523.         break;
524.     }
525.     break;
526. case 14: // "///"状态(单行注释状态)
527.     cat(token, C);
528.     get_char(C);
529.     if (C == '\n')
530.     {
531.         state = 0;
532.         tmpWord.attribute = token;
533.         tmpWord.sign = "annota";
534.         words_insert(tmpWord);
535.     }
536.     else
537.     {
538.         state = 14;
539.     }
540.     break;
541. case 15: // "/*"状态(多行注释状态)
542.     cat(token, C);

```

```
543.         get_char(C);
544.         if (C == '*')
545.         {
546.             state = 16;
547.         }
548.         else
549.         {
550.             state = 15;
551.         }
552.         break;
553.     case 16: // "/*...*/"状态(等待完整的"*/")
554.         cat(token, C);
555.         get_char(C);
556.         if (C == '*')
557.         {
558.             state = 16;
559.         }
560.         else if (C == '/')
561.         {
562.             cat(token, C);
563.             state = 0;
564.             tmpWord.attribute = token;
565.             tmpWord.sign = "annota";
566.             words_insert(tmpWord);
567.         }
568.         else
569.         {
570.             state = 15;
571.         }
572.         break;
573.     case 17: // '='状态
574.         cat(token, C);
575.         get_char(C);
576.         if (C == '=')
577.         {
578.             cat(token, C);
579.             state = 0;
580.             tmpWord.attribute = token;
581.             tmpWord.sign = "relop";
582.             words_insert(tmpWord);
583.         }
584.         else
585.         {
586.             retract();
```

```
587.         state = 0;
588.         tmpWord.attribute = token;
589.         tmpWord.sign = "assin";
590.         words_insert(tmpWord);
591.     }
592.     break;
593.     case 18: // '!'状态
594.         cat(token, C);
595.         get_char(C);
596.         if (C == '=')
597.         {
598.             cat(token, C);
599.             state = 0;
600.             tmpWord.attribute = token;
601.             tmpWord.sign = "relop";
602.             words_insert(tmpWord);
603.         }
604.         else
605.         {
606.             retract();
607.             state = 0;
608.             tmpError.errorInf = "\"!=\" is incomplete";
609.             tmpError.errorToken = token;
610.             tmpError.lines = nowRows;
611.             error(tmpError);
612.         }
613.         break;
614.     case 19: // '''状态
615.         cat(token, C);
616.         get_char(C);
617.         if (C == '')
618.         {
619.             cat(token, C);
620.             state = 0;
621.             tmpWord.attribute = token;
622.             tmpWord.sign = "str";
623.             words_insert(tmpWord);
624.         }
625.         else
626.         {
627.             state = 19;
628.         }
629.         break;
630.     case 20: // '''状态
```



```

631.         cat(token, C);
632.         get_char(C);
633.         if (C == '\\')
634.         {
635.             cat(token, C);
636.             state = 0;
637.             tmpWord.attribute = token;
638.             tmpWord.sign = "char";
639.             words_insert(tmpWord);
640.         }
641.         else
642.         {
643.             state = 20;
644.         }
645.         break;
646.     case 21: // '#'状态
647.         cat(token, C);
648.         get_char(C);
649.         if (C == '\\n')
650.         {
651.             state = 0;
652.             tmpWord.attribute = token;
653.             tmpWord.sign = "pretrt";
654.             words_insert(tmpWord);
655.         }
656.         else
657.         {
658.             state = 21;
659.         }
660.         break;
661.     case 22: // 错误状态
662.         state = 0;
663.         cat(token, C);
664.         tmpError.errorInf = "Unrecognized symbol";
665.         tmpError.errorToken = token;
666.         tmpError.lines = nowRows;
667.         error(tmpError);
668.         break;
669.     default:
670.         cout << "STATE ERROR!" << endl;
671.         exit(-1);
672.     }
673. }
674. }

```

```

675.
676. int lexInit() // 初始化模块
677. {
678.     fout.open("output.txt");
679.     filePtr = fopen(FILE_NAME.c_str(), "r");
680.     if (!filePtr)
681.     {
682.         printf("Error in opening file");
683.         return -1; // 初始化失败
684.     }
685.
686.     int i;
687.     for (i = 0; i < BUF_SIZE; i++)
688.         buf[i] = '\0';
689.
690.
691.     forwardPtr = 0;
692.     validChNum = 0;
693.     allChNum = 0;
694.     allRows = 0;
695.     nowRows = 1;
696.     get_buf(1);
697.     return 1; // 初始化成功
698. }
699.
700. void print_result() // 输出模块
701. {
702.     int i = 0;
703.
704.     cout << "Token stream:" << wordFlow.size() << endl;
705.     cout << "sign\t\tattribute" << endl;
706.     fout << "Token stream:" << wordFlow.size() << endl;
707.     fout << "sign\t\tattribute" << endl;
708.     for (i = 0; i < wordFlow.size(); i++)
709.     {
710.         cout << wordFlow[i].sign << "\t\t" << wordFlow[i].attribute << endl;
711.
712.         fout << wordFlow[i].sign << "\t\t" << wordFlow[i].attribute << endl;
713.
714.     }
715.     cout << endl;
716.     fout << endl;
717.
718.     cout << "Symbol table:" << IDs.size() << endl;

```

```

717.     fout << "Symbol table:" << IDs.size() << endl;
718.     for (i = 0; i < IDs.size(); i++)
719.     {
720.         cout << IDs[i] << endl;
721.         fout << IDs[i] << endl;
722.     }
723.     cout << endl;
724.     fout << endl;
725.
726.     cout << "Errors:" << errorFlow.size() << endl;
727.     cout << "line\t\ttoken\t\tInf" << endl;
728.     fout << "Errors:" << errorFlow.size() << endl;
729.     fout << "line\t\ttoken\t\tInf" << endl;
730.     for (i = 0; i < errorFlow.size(); i++)
731.     {
732.         cout << errorFlow[i].lines << "\t\t" << errorFlow[i].errorToken <<
            "\t\t" << errorFlow[i].errorInf << endl;
733.         fout << errorFlow[i].lines << "\t\t" << errorFlow[i].errorToken <<
            "\t\t" << errorFlow[i].errorInf << endl;
734.     }
735.     cout << endl;
736.     fout << endl;
737.
738.     cout << "Total number of rows: " << allRows << endl;
739.     cout << "Total characters: " << allChNum << endl;
740.     cout << "Total valid characters: " << validChNum << endl;
741.     cout << "Total words: " << IDs.size() << endl;
742.     fout << "Total number of rows: " << allRows << endl;
743.     fout << "Total characters: " << allChNum << endl;
744.     fout << "Total valid characters: " << validChNum << endl;
745.     fout << "Total words: " << IDs.size() << endl;
746. }
747.
748. int main() // 主函数
749. {
750.     // 获取文件名
751.     cout << "Please enter the file name to be analyzed: ";
752.     cin >> FILE_NAME;
753.
754.     if (lexInit())
755.     {
756.         lexical_analysis(); // 开始词法分析
757.         print_result(); // 输出分析结果
758.         fclose(filePtr); // 关闭文件

```

```

759.         fout.close();
760.     }
761.     else // 初始化失败
762.         cout << "Initialization failed!" << endl;
763.
764.     return 0;
765. }

```

## 五. 利用 LEX 实现

### 1.lex 环境安装与配置:

采用在 Windows 下安装 cygwin 的方式。

首先安装 cygwin，接着选择安装 gcc，lex，bison，make 环境包。

安装完成后测试安装结果：

```

B7290@NANA-Y7000P ~
$ gcc -v
使用内建 specs。
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-pc-cygwin/10/lto-wrapper.exe
目标: x86_64-pc-cygwin
配置为: /mnt/share/cygpkgs/gcc/gcc.x86_64/src/gcc-10.2.0/configure --srcdir=/mnt/share/cygpkgs/gcc/gcc
x86_64/src/gcc-10.2.0 --prefix=/usr --exec-prefix=/usr --localstatedir=/var --sysconfdir=/etc --docdi
r=/usr/share/doc/gcc --htmldir=/usr/share/doc/gcc/html -C --build=x86_64-pc-cygwin --host=x86_64-pc-cy
gwin --target=x86_64-pc-cygwin --without-libiconv-prefix --without-libintl-prefix --libexecdir=/usr/li
b --with-gcc-major-version-only --enable-shared --enable-shared-libgcc --enable-static --enable-versio
n-specific-runtime-libs --enable-bootstrap --enable-_cxa_atexit --with-dwarf2 --with-tune=generic --e
nable-languages=c,c++,fortran,lto,objc,obj-c++ --enable-graphite --enable-threads=posix --enable-libat
omic --enable-libgomp --enable-libquadmath --enable-libquadmath-support --disable-libssp --enable-liba
da --disable-symvers --with-gnu-ld --with-gnu-as --with-cloog-include=/usr/include/cloog-isl --without
-libiconv-prefix --without-libintl-prefix --with-system-zlib --enable-linker-build-id --with-default-l
ibstdcxx-abi=gcc4-compatible --enable-libstdcxx-filesystem-ts
线程模型: posix
Supported LTO compression algorithms: zlib zstd
gcc 版本 10.2.0 (GCC)

```

```

B7290@NANA-Y7000P ~
$ flex -help
用法: flex [选项] [文件]...
Generates programs that perform pattern-matching on text.

Table Compression:
  -Ca, --align      trade off larger tables for better memory alignment
  -Ce, --ecs        construct equivalence classes
  -Cf               do not compress tables; use -f representation
  -CF               do not compress tables; use -F representation
  -Cm, --meta-ecs   construct meta-equivalence classes
  -Cr, --read       use read() instead of stdio for scanner input
  -f, --full        generate fast, large scanner. Same as -Cfr
  -F, --fast        use alternate table representation. Same as -Cfr
  -Cem             default compression (same as --ecs --meta-ecs)

Debugging:
  -d, --debug       enable debug mode in scanner

```

输命令“gcc-v”与“flex -help”，出现上图说明成功配置好了 lex 的环境。

## 2.编写 lex.l 源程序

最重要的部分是正规定义式的书写。

与上述 C++的代码相同，设置以下几种单词识别规则：

```
1. delim      [ \t \n]
2. ws         {delim}+
3. letter     [A-Za-z_]
4. digit      [0-9]
5. id         {letter}({letter}|{digit})*
6. number     {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
7. annota     "/*"([^\*]|(\*[^\/])*(\/)*)*"*/"
8. str        \"(\\.|[^\"])*\"
9. achar      '.'
```

相应的词语含义也与 C++程序相同。

此外，还需定义一些辅助函数：

int installID ()	把 id 插入符号表
int installNum ()	把 num 插入符号表
int installAnnota ()	把 annota 插入符号表
int installStr ()	把 str 插入符号表
int installAChar()	把一个 char 插入符号表
int installPretprt()	把 pretprt 插入符号表
void writeout(int c)	输出单词 c 相关信息

最后是主函数：用于接收参数与打开文件

```
1. int main (int argc, char ** argv){
2.     int c,j=0;
3.     if (argc>=2){
4.         if ((yyin = fopen(argv[1], "r")) == NULL){
5.             printf("Can't open file %s\n", argv[1]);
6.             return 1;
7.         }
8.         if (argc>=3){
9.             yyout=fopen(argv[2], "w");
10.        }
11.    }
12.    while (c = yylex()){
```

```

13.     writeout(c);
14.     j++;
15.     if (j%5 == 0) writeout(NEWLINE);
16. }
17. if(argc>=2){
18.     fclose(yyin);
19.     if (argc>=3) fclose(yyout);
20. }
21. return 0;
22. }

```

### 3.测试 lex 程序

测试分 3 步。

第一步，要根据 lex.l 生成 lex.yy.c:

```

87290@NANA-Y7000P ~
$ flex lex.l

```

没有报错说明生成成功。

第二步，使用 gcc 编译 lex.yy.c 成为可执行文件 a.exe:

```

87290@NANA-Y7000P ~
$ gcc lex.yy.c -lfl
lex2.l: 在函数 'yylex' 中:
lex2.l:128:11: 警告: 隐式声明函数 'installID' [-Wimplicit-function-declaration]
128 | [id] {yy|val = installID (); return (ID);}
lex2.l:129:11: 警告: 隐式声明函数 'installNum' [-Wimplicit-function-declaration]
129 | [number] {yy|val = installNum (); return (NUMBER);}
lex2.l:130:11: 警告: 隐式声明函数 'installAnnota' [-Wimplicit-function-declaration]
130 | [annota] {yy|val = installAnnota (); return (ANNOTA);}
lex2.l:131:11: 警告: 隐式声明函数 'installStr' [-Wimplicit-function-declaration]
131 | [str] {yy|val = installStr (); return (STR);}
lex2.l:132:11: 警告: 隐式声明函数 'installAChar' [-Wimplicit-function-declaration]
132 | [achar] {yy|val = installAChar (); return (ACHAR);}
lex2.l:133:11: 警告: 隐式声明函数 'installPretrt' [-Wimplicit-function-declaration]
133 | [pretrt] {yy|val = installPretrt (); return (PRETRT);}

```

编译产生部分警告信息，但对程序功能没有影响，故忽略警告。

第三步，分析测试文件：

采用以下输入文件：

```

1. #include <stdio.h>
2. int main()
3. {
4.     float i =1.2;

```

```

5.     _b = c + i;
6.     printf("Hello world");
7.     /*this ia a line of annotation*/
8.     return 0;
9. }

```

分析结果如下：

```

87290@NANA-Y7000P ~
$ ./a.exe test.txt
") ETRT, "#include <stdio.h>"
(INT, "int")
(MAIN, "main")
(DELIMIT, "(")
(DELIMIT, ")")

(DELIMIT, "{")
(FLOAT, "float")
(ID, "i")

(ASSIN, "=")
(NUM, "1.2")
(DELIMIT, ";")
(ID, "_b")

(ASSIN, "=")
(ID, "c")
(OPERA, "+")
(ID, "i")
(DELIMIT, ";")

(ID, "printf")
(DELIMIT, "(")
(STR, "Hello world")
(DELIMIT, ")")

(DELIMIT, ";")
(ANNOA, "/*this ia a line of annotation*/")
(RETURN, "return")

(NUM, "0")
(DELIMIT, ";")
(DELIMIT, "}")

```

可以看到，lex生成的分析程序正确的完成了分析任务，将记号流输出到命令行中。

## 4.lex.l 源代码：

```

1. %{
2. #include <stdio.h>
3. #define LT 1
4. #define LE 2

```

5. #define GT	3
6. #define GE	4
7. #define EQ	5
8. #define NE	6
9.	
10. #define INT	7
11. #define LONG	8
12. #define SHORT	9
13. #define FLOAT	10
14. #define DOUBLE	11
15. #define CHAR	12
16. #define UNSIGNED	13
17. #define SIGNED	14
18. #define CONST	15
19. #define VOID	16
20. #define STRUCT	17
21. #define UNION	18
22. #define IF	19
23. #define ELSE	20
24. #define GOTO	21
25. #define SWITCH	22
26. #define CASE	23
27. #define DO	24
28. #define WHILE	25
29. #define FOR	26
30. #define CONTINUE	27
31. #define BREAK	28
32. #define RETURN	29
33. #define DEFAULT	30
34. #define TYPRDEF	31
35. #define EXTERN	32
36. #define STATIC	33
37. #define MAIN	34
38.	
39. #define ID	35
40. #define NUMBER	36
41. #define SHIFT	37
42. #define RELOP	38
43. #define LOGIC	39
44. #define ADDRE	40
45. #define ASSIN	41
46. #define OPERA	42
47. #define ANNOTA	43
48. #define STR	44



```

49. #define ACHAR          45
50. #define PRETRT         46
51. #define DELIMI         47
52.
53. #define NEWLINE         48
54. #define ERRORCHAR      49
55.
56. #define LK1             50
57. #define RK1             51
58. #define LK2             52
59. #define RK2             53
60. #define LK3             54
61. #define RK3             55
62. #define COM             56 /*逗号      */
63. #define SEM             57 /*分号      */
64. #define COL             58 /*冒号      */
65. #define DQM             59 /*双引号    */
66. #define SQM             60 /*单引号    */
67. #define LM              61 /*左移位    */
68. #define RM              62 /*右移位    */
69. #define AND             63 /*与        */
70. #define OR              64 /*或        */
71. #define ADD             65 /*+         */
72. #define SUB             66 /*-         */
73. #define MUL             67 /***        */
74. #define DIV             68 /*/         */
75. #define REM             69 /*%         */
76. #define EQU            70 /*=         */
77. #define EADD            71 /*+=        */
78. #define ESUB            72 /*-=        */
79. #define EMUL            73 /**=        */
80. #define EDIV            74 /*/=        */
81. #define EREM            75 /*%=        */
82.
83.
84. int yylval;
85. %}
86. delim      [ \t \n]
87. ws         {delim}+
88. letter     [A-Za-z_]
89. digit      [0-9]
90. id         {letter}({letter}|{digit})*
91. number     {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
92. annota     "/*"([^\*]|(\*[^\/])*(\*))*"/"

```

```

93. str      \"(\\.|[^\"])*\"
94. achar    '.'
95. pretrt   #^[^\\n]*
96.
97. %%
98. {ws}     {}
99. int      {return (INT);}
100. long    {return (LONG);}
101. short    {return (SHORT);}
102. float    {return (FLOAT);}
103. double   {return (DOUBLE);}
104. char     {return (CHAR);}
105. unsigned {return (UNSIGNED);}
106. signed   {return (SIGNED);}
107. const    {return (CONST);}
108. void     {return (VOID);}
109. struct   {return (STRUCT);}
110. union    {return (UNION);}
111. if       {return (IF);}
112. else     {return (ELSE);}
113. goto     {return (GOTO);}
114. switch   {return (SWITCH);}
115. case     {return (CASE);}
116. do       {return (DO);}
117. while    {return (WHILE);}
118. for      {return (FOR);}
119. continue {return (CONTINUE);}
120. break    {return (BREAK);}
121. return   {return (RETURN);}
122. default  {return (DEFAULT);}
123. typedef  {return (TYPRDEF);}
124. extern   {return (EXTERN);}
125. static   {return (STATIC);}
126. main     {return (MAIN);}
127.
128. {id}      {yyval = installID (); return (ID);}
129. {number}   {yyval = installNum (); return (NUMBER);}
130. {annota}   {yyval = installAnnota (); return (ANNOTA);}
131. {str}      {yyval = installStr (); return (STR);}
132. {achar}    {yyval = installAChar (); return (ACHAR);}
133. {pretrt}   {yyval = installPretrt (); return (PRETRT);}
134. "<"        {yyval = LT; return (RELOP);}
135. "<="      {yyval = LE; return (RELOP);}
136. ">"        {yyval = GT; return (RELOP);}

```

```

137. ">="      {yyval = GE; return (RELOP);}
138. "=="      {yyval = EQ; return (RELOP);}
139. "!="      {yyval = NE; return (RELOP);}
140.
141. "("        {yyval = LK1; return (DELIMI);}
142. ")"        {yyval = RK1; return (DELIMI);}
143. "["        {yyval = LK2; return (DELIMI);}
144. "]"        {yyval = RK2; return (DELIMI);}
145. "{"        {yyval = LK3; return (DELIMI);}
146. "}"        {yyval = RK3; return (DELIMI);}
147. ","        {yyval = COM; return (DELIMI);}
148. ";"        {yyval = SEM; return (DELIMI);}
149. ":"        {yyval = COL; return (DELIMI);}
150. "\"        {yyval = DQM; return (DELIMI);}
151. '\''       {yyval = SQM; return (DELIMI);}
152. "<<"      {yyval = LT; return (SHIFT);}
153. ">>"      {yyval = RM; return (SHIFT);}
154. "&&"      {yyval = AND; return (LOGIC);}
155. "||"       {yyval = OR; return (LOGIC);}
156. "="        {yyval = EQU; return (ASSIN);}
157. "+="       {yyval = EADD; return (ASSIN);}
158. "-="       {yyval = ESUB; return (ASSIN);}
159. "*="       {yyval = EMUL; return (ASSIN);}
160. "/="       {yyval = EDIV; return (ASSIN);}
161. "%="       {yyval = EREM; return (ASSIN);}
162. "+"        {yyval = ADD; return (OPERA);}
163. "-"        {yyval = SUB; return (OPERA);}
164. "*"        {yyval = MUL; return (OPERA);}
165. "/"        {yyval = DIV; return (OPERA);}
166. "%"        {yyval = REM; return (OPERA);}
167.
168. .          {yyval = ERRORCHAR; return ERRORCHAR;}
169.
170. %%
171. int installID () {
172.     return ID;
173. }
174.
175.
176. int installNum () {
177.     return NUMBER;
178. }
179.
180.

```

```

181. int installAnnota (){
182.     return ANNOTA;
183. }
184.
185. int installStr (){
186.     return STR;
187. }
188.
189. int installAChar (){
190.     return CHAR;
191. }
192.
193. int installPretrt (){
194.     return PRETRT;
195. }
196.
197.
198. int yywrap (){
199.     return 1;
200. }
201.
202. void writeout(int c){
203.     switch(c){
204.         case ERRORCHAR: break;
205.
206.         case INT:      fprintf(yyout, "(INT, \"%s\") \n", yytext);break;
207.         case LONG:     fprintf(yyout, "(LONG, \"%s\") \n", yytext);break;
208.         case SHORT:    fprintf(yyout, "(SHORT, \"%s\") \n", yytext);break;
209.         case FLOAT:    fprintf(yyout, "(FLOAT, \"%s\") \n", yytext);break;
210.         case DOUBLE:   fprintf(yyout, "(DOUBLE, \"%s\") \n", yytext);break;
211.         case CHAR:     fprintf(yyout, "(CHAR, \"%s\") \n", yytext);break;
212.         case UNSIGNED: fprintf(yyout, "(UNSIGNED, \"%s\") \n", yytext);break;
213.         case SIGNED:   fprintf(yyout, "(SIGNED, \"%s\") \n", yytext);break;
214.         case CONST:    fprintf(yyout, "(CONST, \"%s\") \n", yytext);break;
215.         case VOID:     fprintf(yyout, "(VOID, \"%s\") \n", yytext);break;
216.         case STRUCT:   fprintf(yyout, "(STRUCT, \"%s\") \n", yytext);break;
217.         case UNION:    fprintf(yyout, "(UNION, \"%s\") \n", yytext);break;
218.         case IF:       fprintf(yyout, "(IF, \"%s\") \n", yytext);break;
219.         case ELSE:     fprintf(yyout, "(ELSE, \"%s\") \n", yytext);break;
220.         case GOTO:     fprintf(yyout, "(GOTO, \"%s\") \n", yytext);break;
221.         case SWITCH:   fprintf(yyout, "(SWITCH, \"%s\") \n", yytext);break;
222.         case CASE:     fprintf(yyout, "(CASE, \"%s\") \n", yytext);break;
223.         case DO:       fprintf(yyout, "(DO, \"%s\") \n", yytext);break;
224.         case WHILE:    fprintf(yyout, "(WHILE, \"%s\") \n", yytext);break;

```

```

225.     case FOR:      fprintf(yyout, "(FOR, \"%s\") \n", yytext);break;
226.     case CONTINUE: fprintf(yyout, "(CONTINUE, \"%s\") \n", yytext);break;
227.     case BREAK:     fprintf(yyout, "(BREAK, \"%s\") \n", yytext);break;
228.     case RETURN:    fprintf(yyout, "(RETURN, \"%s\") \n", yytext);break;
229.     case DEFAULT:   fprintf(yyout, "(DEFAULT, \"%s\") \n", yytext);break;
230.     case TYPRDEF:   fprintf(yyout, "(TYPRDEF, \"%s\") \n", yytext);break;
231.     case EXTERN:    fprintf(yyout, "(EXTERN, \"%s\") \n", yytext);break;
232.     case STATIC:    fprintf(yyout, "(STATIC, \"%s\") \n", yytext);break;
233.     case MAIN:      fprintf(yyout, "(MAIN, \"%s\") \n", yytext);break;
234.
235.     case ID:        fprintf(yyout, "(ID, \"%s\") \n", yytext);break;
236.     case NUMBER:    fprintf(yyout, "(NUM, \"%s\") \n", yytext);break;
237.     case SHIFT:     fprintf(yyout, "(SHIFT, \"%s\") \n", yytext);break;
238.     case RELOP:     fprintf(yyout, "(RELOP, \"%s\") \n", yytext);break;
239.
240.     case LOGIC:     fprintf(yyout, "(LOGIC, \"%s\") \n", yytext);break;
241.     case ADDRE:     fprintf(yyout, "(ADDRE, \"%s\") \n", yytext);break;
242.     case ASSIN:     fprintf(yyout, "(ASSIN, \"%s\") \n", yytext);break;
243.     case OPERA:     fprintf(yyout, "(OPERA, \"%s\") \n", yytext);break;
244.     case ANNOTA:    fprintf(yyout, "(ANNOTA, \"%s\") \n", yytext);break;
245.     case STR:       fprintf(yyout, "(STR, \"%s\") \n", yytext);break;
246.     case ACHAR:     fprintf(yyout, "(ACHAR, \"%s\") \n", yytext);break;
247.     case PRETRT:    fprintf(yyout, "(PRETRT, \"%s\") \n", yytext);break;
248.     case DELIMI:    fprintf(yyout, "(DELIMI, \"%s\") \n", yytext);break;
249.
250.     case NEWLINE:   fprintf(yyout, "\n");break;
251.     default:        break;
252. }
253. return;
254. }
255.
256. int main (int argc, char ** argv){
257.     int c,j=0;
258.     if (argc>=2){
259.         if ((yyin = fopen(argv[1], "r")) == NULL){
260.             printf("Can't open file %s\n", argv[1]);
261.             return 1;
262.         }
263.         if (argc>=3){
264.             yyout=fopen(argv[2], "w");
265.         }
266.     }
267.     while (c = yylex()){

```

```

268.     writeout(c);
269.     j++;
270.     if (j%5 == 0) writeout(NEWLINE);
271. }
272. if(argc>=2){
273.     fclose(yyin);
274.     if (argc>=3) fclose(yyout);
275. }
276. return 0;
277. }

```

## 六. 实验心得与总结

### 1.实验心得：

通过这次实际编写词法分析源程序，深入具体地学习了编译的第一个步骤，对词法分析的原理和方法有了深刻的认识。

同时，在搭建学习使用 lex 的过程中，也锻炼了我解决问题，获取知识的能力。

对于类似 lex 这种基于 unix 的工具，环境配置往往是一个繁杂的过程，本次实验中选择合适的方式以及实际的环境配置都耗费了大量时间。

对 lex 语法的不熟悉也导致编写 lex 源程序时产生诸多 bug。

```

87290@NANA-Y7000P ~
$ flex lex2.l
lex2.l:130: 不能识别的规则

```

```

lex2.l:151:23: 错误: 'SHIFT' undeclared (first use in this function); did you mean 'SHITF' ?
151  "\"" {yy1val = SQM; return (DELIMI);}

```

在修复 bug 的同时也加深了对 lex 的理解与运用

### 2.实验总结与改进方向：

采用两种方法实现词法分析，其中使用 C++达到了较好的效果。

由于 lex 使用不熟练，最终没有实现检错与计数功能，这是一个值得继续研究的方面。