# 编译器专题实验报告

#### 目录

##### 实验一:词法部分

###### 实验内容（必做）：

【问题描述】：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. 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)

2. 单词的类别码请统一按如下形式定义：

词法分析类别码定义如下：

| **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** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Ident** | IDENFR | ! | NOT | \* | MULT | = | ASSIGN |
| **IntConst** | INTCON | && | AND | / | DIV | ; | SEMICN |
| **FormatString** | STRCON | || | OR | % | MOD | , | COMMA |
| main | MAINTK | while | WHILETK | < | LSS | ( | LPARENT |
| const | CONSTTK | getint | GETINTTK | <= | LEQ | ) | RPARENT |
| int | INTTK | printf | PRINTFTK | > | GRE | [ | LBRACK |
| break | BREAKTK | return | RETURNTK | >= | GEQ | ] | RBRACK |
| continue | CONTINUETK | + | PLUS | == | EQL | { | LBRACE |
| if | IFTK | - | MINU | != | NEQ | } | RBRACE |
| else | ELSETK | void | VOIDTK |  |  |  |  |

【输入形式】testfile.txt中的符合文法要求的测试程序。

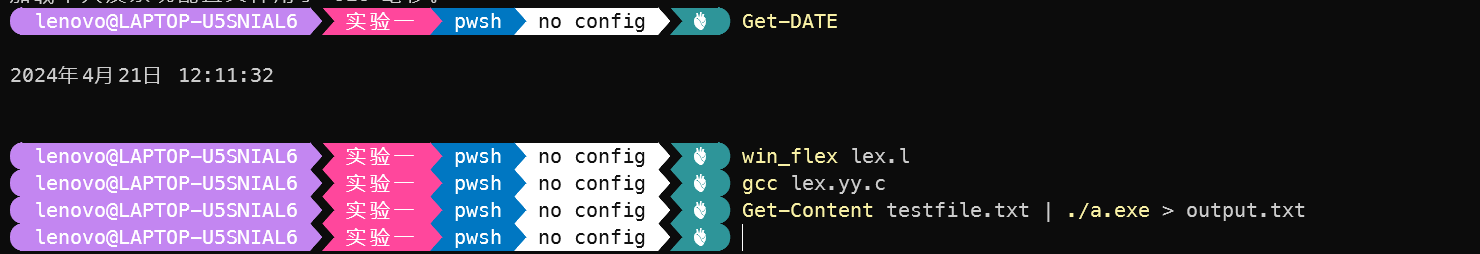
【输出形式】要求将词法分析结果输出至output.txt中。

【特别提醒】

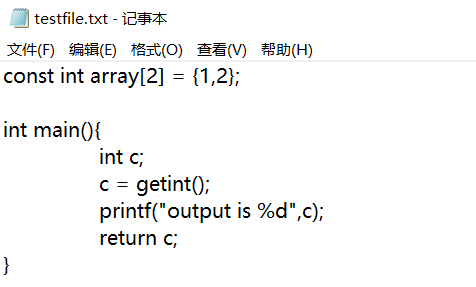
1. 读取的字符串要原样保留着便于输出，特别是数字，这里输出的并不是真正的单词值，其实是读入的字符串，单词值需另行记录。（存储时直接复制字符串来保存）
2. 本次作业只考核对正确程序的处理，但需要为今后可能出现的错误情况预留接口。（错误情况目前考虑了错误的标识符和不可识别的符号）
3. 在今后的错误处理作业中，需要输出错误的行号，在词法分析的时候注意记录该信息。（行号用line来记录，设置了一个变量决定输出的时候是否输出行号）
4. 单词的类别和单词值以及其他关注的信息，在词法分析阶段获取后，后续的分析阶段会使用，请注意记录；当前要求的输出只是为了便于评测，完成编译器中无需出现这些信息，请设计为方便打开/关闭这些输出的方案。（用一个链表来存储所有词法单元，在测试时选择用输出函数来显示，可以决定是否显示）

###### 实验内容（选做）：

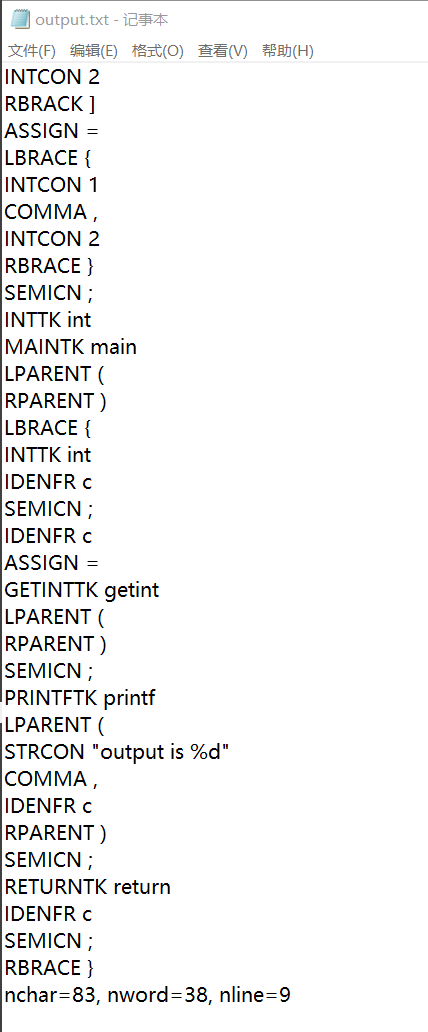
实验结果： （截图。实验结束截图提供实验完成时间，这一点也比较重要，截图中体现提示符，提示符揭示了自己是否独立完成，是合作模式还是独立模式等等。）：



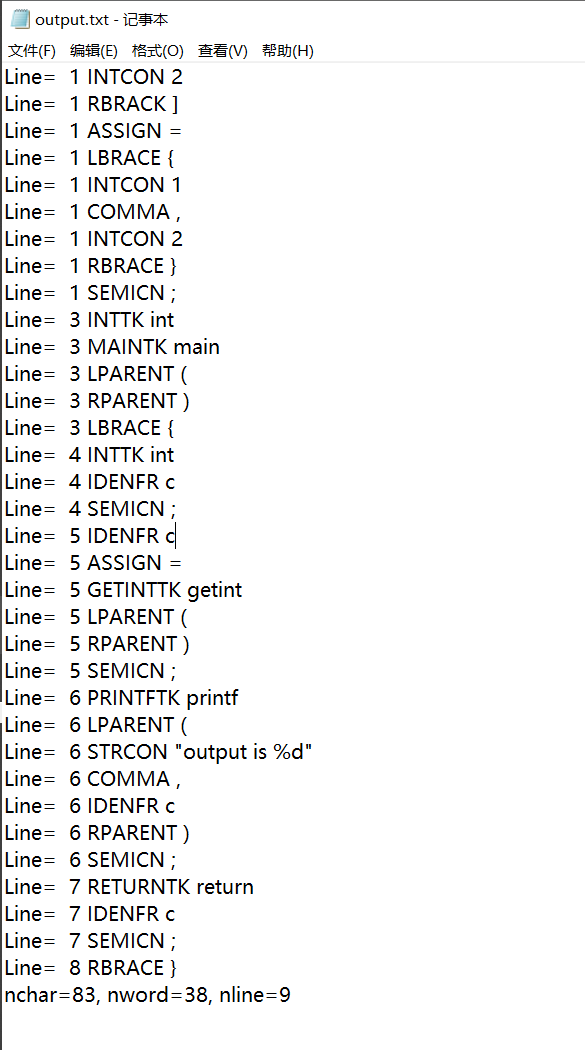
以上是命令行输入，显示了完成实验的时间



以上是testfile.txt文件的内容，为实验的样例输入。



以上是实验的output.txt的内容，与实验样例输出完全一致。



以上是输出行数的结果。

另外遇到的问题和解决思路（可选）：

解决思路：

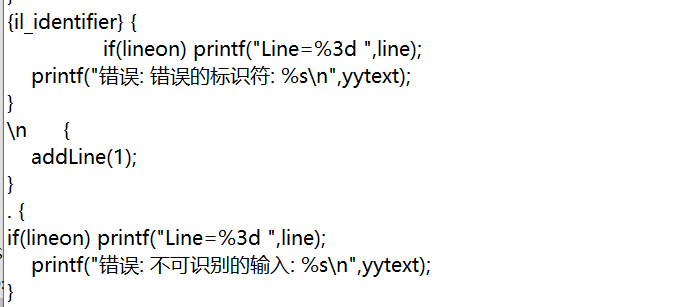
1. 用链表来存储每一个词法单元，保存下来了具体的字符串（输入）的值。
2. 首先用正则式去定义每一个词法单元，之后再在模式匹配规则区域，去写匹配后的操作，存入链表（结构体）+调用输出函数+对nchar、nword，nline进行处理。

考虑的错误情况：

错误的标识符



错误处理情况



遇到的问题：

1. 概念问题：在定义全局变量时只能定义静态变量。

代码很原创（√）：

代码：

1. %{
2. #include <stdio.h>
3. #include <string.h>
4. **int** lineon = 1;
5. **int** line=1;
6. **int** nchar,nword;
7. **char**  token\_name[50][20];
8. **typedef** **enum** {
9. head,
10. NCHAR,
11. letter,
12. digit,
13. IDENFR,
14. INTCON,
15. STRCON,
16. MAINTK,
17. CONSTTK,
18. INTTK,
19. BREAKTK,
20. CONTINUETK,
21. IFTK,
22. ELSETK,
23. WHILETK,
24. GETINTTK,
25. PRINTFTK,
26. RETURNTK,
27. PLUS,
28. MINU,
29. VOIDTK,
30. NOT,
31. AND,
32. OR,
33. MULT,
34. DIV,
35. MOD,
36. LSS,
37. LEQ,
38. GRE,
39. GEQ,
40. EQL,
41. NEQ,
42. ASSIGN,
43. SEMICN,
44. COMMA,
45. LPARENT,
46. RPARENT,
47. LBRACK,
48. RBRACK,
49. LBRACE,
50. RBRACE,
51. **FLOAT**,
52. il\_identifier
53. } TokenType;
55. **typedef** **struct** Token{
56. TokenType type;
57. **char** \* value;
58. **int** line;
59. **struct** Token \*next;
60. } Token;
61. Token \* head\_node;
62. Token \* last\_node;
64. Token\* token\_new(TokenType type, **char**\* value, **int** line,Token\* node);
65. **void** token\_free(Token\* token);
66. **void** token\_print(Token\* token);
67. **void** addLine(**int**);
68. %}
70. line\_comment    (\/\/.\*\n)
72. **char**    '[^']'
73. letter  [A-Za-z\_]
74. digit   [0-9]
75. IDENFR  {letter}({letter}|{digit})\*
76. INTCON  {digit}+
77. STRCON   \"[^\"]\*\"
78. MAINTK  "main"
79. CONSTTK "const"
80. INTTK   "int"
81. BREAKTK "break"
82. CONTINUETK "continue"
83. IFTK    "if"
84. ELSETK  "else"
85. WHILETK "while"
86. GETINTTK    "getint"
87. PRINTFTK    "printf"
88. RETURNTK "return"
89. PLUS    "+"
90. MINU    "-"
91. VOIDTK  "void"
92. NOT "!"
93. AND "&&"
94. OR  "||"
95. MULT    "\*"
96. DIV "/"
97. MOD "%"
98. LSS "<"
99. LEQ "<="
100. GRE ">"
101. GEQ ">="
102. EQL "=="
103. NEQ "!="
104. ASSIGN  "="
105. SEMICN  ";"
106. COMMA   ","
107. LPARENT "("
108. RPARENT ")"
109. LBRACK  "["
110. RBRACK  "]"
111. LBRACE  "{"
112. RBRACE  "}"
113. **FLOAT**                 {digit}\*(\.{digit}+)?(e|E[+\-]?{digit}+)?
114. il\_identifier          ({digit}|{digit}\*(\.{digit}+)?(e|E[+\-]?{digit}+)?)({letter}|{digit})\*{letter}({letter}|{digit})\*

117. %%
119. {line\_comment}  {
120. **if**(lineon) printf("Line=%3d ",line);
121. printf("line\_comment %s",yytext);
122. addLine(1);
123. }
124. {**char**}  {
125. Token\* Tnode =token\_new(NCHAR,yytext,line,NULL);
126. last\_node->next = Tnode;
127. last\_node = Tnode;
128. token\_print(Tnode);
129. nchar+=yyleng;
130. nword++;
131. }
132. {STRCON}  {
133. Token\* Tnode =token\_new(STRCON,yytext,line,NULL);
134. token\_print(Tnode);
135. last\_node->next = Tnode;
136. last\_node = Tnode;
137. nchar+=yyleng;
138. nword++;
139. }
140. {MAINTK}   {
141. Token\* Tnode =token\_new(MAINTK,yytext,line,NULL);
142. token\_print(Tnode);
143. last\_node->next = Tnode;
144. last\_node = Tnode;
145. nchar+=yyleng;
146. nword++;
147. }
148. {CONSTTK}   {
149. Token\* Tnode =token\_new(CONSTTK,yytext,line,NULL);
150. token\_print(Tnode);
151. last\_node->next = Tnode;
152. last\_node = Tnode;
153. nchar+=yyleng;
154. nword++;
155. }
156. {INTTK}   {
157. Token\* Tnode =token\_new(INTTK,yytext,line,NULL);
158. token\_print(Tnode);
159. last\_node->next = Tnode;
160. last\_node = Tnode;
161. nchar+=yyleng;
162. nword++;
163. }
164. {BREAKTK}   {
165. Token\* Tnode =token\_new(BREAKTK,yytext,line,NULL);
166. token\_print(Tnode);
167. last\_node->next = Tnode;
168. last\_node = Tnode;
169. nchar+=yyleng;
170. nword++;
171. }
172. {CONTINUETK}   {
173. Token\* Tnode =token\_new(CONTINUETK,yytext,line,NULL);
174. token\_print(Tnode);
175. last\_node->next = Tnode;
176. last\_node = Tnode;
177. nchar+=yyleng;
178. nword++;
179. }
180. {IFTK}   {
181. Token\* Tnode =token\_new(IFTK,yytext,line,NULL);
182. token\_print(Tnode);
183. last\_node->next = Tnode;
184. last\_node = Tnode;
185. nchar+=yyleng;
186. nword++;
187. }
188. {ELSETK}   {
189. Token\* Tnode =token\_new(ELSETK,yytext,line,NULL);
190. token\_print(Tnode);
191. last\_node->next = Tnode;
192. last\_node = Tnode;
193. nchar+=yyleng;
194. nword++;
195. }
196. {WHILETK}   {
197. Token\* Tnode =token\_new(WHILETK,yytext,line,NULL);
198. token\_print(Tnode);
199. last\_node->next = Tnode;
200. last\_node = Tnode;
201. nchar+=yyleng;
202. nword++;
203. }
204. {GETINTTK}   {
205. Token\* Tnode =token\_new(GETINTTK,yytext,line,NULL);
206. token\_print(Tnode);
207. last\_node->next = Tnode;
208. last\_node = Tnode;
209. nchar+=yyleng;
210. nword++;
211. }
212. {PRINTFTK}   {
213. Token\* Tnode =token\_new(PRINTFTK,yytext,line,NULL);
214. token\_print(Tnode);
215. last\_node->next = Tnode;
216. last\_node = Tnode;
217. nchar+=yyleng;
218. nword++;
219. }
220. {RETURNTK}   {
221. Token\* Tnode =token\_new(RETURNTK,yytext,line,NULL);
222. token\_print(Tnode);
223. last\_node->next = Tnode;
224. last\_node = Tnode;
225. nchar+=yyleng;
226. nword++;
227. }
228. {VOIDTK}   {
229. Token\* Tnode =token\_new(VOIDTK,yytext,line,NULL);
230. token\_print(Tnode);
231. last\_node->next = Tnode;
232. last\_node = Tnode;
233. nchar+=yyleng;
234. nword++;
235. }
236. {IDENFR}    {
237. Token\* Tnode =token\_new(IDENFR,yytext,line,NULL);
238. token\_print(Tnode);
239. last\_node->next = Tnode;
240. last\_node = Tnode;
241. nchar+=yyleng;
242. nword++;
243. }
245. {INTCON}   {
246. Token\* Tnode =token\_new(INTCON,yytext,line,NULL);
247. token\_print(Tnode);
248. last\_node->next = Tnode;
249. last\_node = Tnode;
250. nchar+=yyleng;
251. nword++;
252. }
253. {**FLOAT**}   {
254. Token\* Tnode =token\_new(**FLOAT**,yytext,line,NULL);
255. token\_print(Tnode);
256. last\_node->next = Tnode;
257. last\_node = Tnode;
258. nchar+=yyleng;
259. nword++;
260. }
261. {NOT}   {
262. Token\* Tnode =token\_new(NOT,yytext,line,NULL);
263. token\_print(Tnode);
264. last\_node->next = Tnode;
265. last\_node = Tnode;
266. nchar+=yyleng;
267. nword++;
268. }
269. {AND}   {
270. Token\* Tnode =token\_new(AND,yytext,line,NULL);
271. token\_print(Tnode);
272. last\_node->next = Tnode;
273. last\_node = Tnode;
274. nchar+=yyleng;
275. nword++;
276. }
277. {OR}   {
278. Token\* Tnode =token\_new(OR,yytext,line,NULL);
279. token\_print(Tnode);
280. last\_node->next = Tnode;
281. last\_node = Tnode;
282. nchar+=yyleng;
283. nword++;
284. }
285. {PLUS}   {
286. Token\* Tnode =token\_new(PLUS,yytext,line,NULL);
287. token\_print(Tnode);
288. last\_node->next = Tnode;
289. last\_node = Tnode;
290. nchar+=yyleng;
291. nword++;
292. }
293. {MINU}   {
294. Token\* Tnode =token\_new(MINU,yytext,line,NULL);
295. token\_print(Tnode);
296. last\_node->next = Tnode;
297. last\_node = Tnode;
298. nchar+=yyleng;
299. nword++;
300. }
301. {MULT}   {
302. Token\* Tnode =token\_new(MULT,yytext,line,NULL);
303. token\_print(Tnode);
304. last\_node->next = Tnode;
305. last\_node = Tnode;
306. nchar+=yyleng;
307. nword++;
308. }
309. {DIV}   {
310. Token\* Tnode =token\_new(DIV,yytext,line,NULL);
311. token\_print(Tnode);
312. last\_node->next = Tnode;
313. last\_node = Tnode;
314. nchar+=yyleng;
315. nword++;
316. }
317. {MOD}   {
318. Token\* Tnode =token\_new(MOD,yytext,line,NULL);
319. token\_print(Tnode);
320. last\_node->next = Tnode;
321. last\_node = Tnode;
322. nchar+=yyleng;
323. nword++;
324. }
325. {LSS}   {
326. Token\* Tnode =token\_new(LSS,yytext,line,NULL);
327. token\_print(Tnode);
328. last\_node->next = Tnode;
329. last\_node = Tnode;
330. nchar+=yyleng;
331. nword++;
332. }
333. {LEQ}   {
334. Token\* Tnode =token\_new(LEQ,yytext,line,NULL);
335. token\_print(Tnode);
336. last\_node->next = Tnode;
337. last\_node = Tnode;
338. nchar+=yyleng;
339. nword++;
340. }
341. {GRE}   {
342. Token\* Tnode =token\_new(GRE,yytext,line,NULL);
343. token\_print(Tnode);
344. last\_node->next = Tnode;
345. last\_node = Tnode;
346. nchar+=yyleng;
347. nword++;
348. }
349. {GEQ}   {
350. Token\* Tnode =token\_new(GEQ,yytext,line,NULL);
351. token\_print(Tnode);
352. last\_node->next = Tnode;
353. last\_node = Tnode;
354. nchar+=yyleng;
355. nword++;
356. }
357. {EQL}   {
358. Token\* Tnode =token\_new(EQL,yytext,line,NULL);
359. token\_print(Tnode);
360. last\_node->next = Tnode;
361. last\_node = Tnode;
362. nchar+=yyleng;
363. nword++;
364. }
365. {NEQ}   {
366. Token\* Tnode =token\_new(NEQ,yytext,line,NULL);
367. token\_print(Tnode);
368. last\_node->next = Tnode;
369. last\_node = Tnode;
370. nchar+=yyleng;
371. nword++;
372. }
373. {ASSIGN}   {
374. Token\* Tnode =token\_new(ASSIGN,yytext,line,NULL);
375. token\_print(Tnode);
376. last\_node->next = Tnode;
377. last\_node = Tnode;
378. nchar+=yyleng;
379. nword++;
380. }
381. {SEMICN}   {
382. Token\* Tnode =token\_new(SEMICN,yytext,line,NULL);
383. token\_print(Tnode);
384. last\_node->next = Tnode;
385. last\_node = Tnode;
386. nchar+=yyleng;
387. nword++;
388. }
389. {COMMA}   {
390. Token\* Tnode =token\_new(COMMA,yytext,line,NULL);
391. token\_print(Tnode);
392. last\_node->next = Tnode;
393. last\_node = Tnode;
394. nchar+=yyleng;
395. nword++;
396. }
397. {LPARENT}   {
398. Token\* Tnode =token\_new(LPARENT,yytext,line,NULL);
399. token\_print(Tnode);
400. last\_node->next = Tnode;
401. last\_node = Tnode;
402. nchar+=yyleng;
403. nword++;
404. }
405. {RPARENT}   {
406. Token\* Tnode =token\_new(RPARENT,yytext,line,NULL);
407. token\_print(Tnode);
408. last\_node->next = Tnode;
409. last\_node = Tnode;
410. nchar+=yyleng;
411. nword++;
412. }
413. {LBRACK}   {
414. Token\* Tnode =token\_new(LBRACK,yytext,line,NULL);
415. token\_print(Tnode);
416. last\_node->next = Tnode;
417. last\_node = Tnode;
418. nchar+=yyleng;
419. nword++;
420. }
421. {RBRACK}   {
422. Token\* Tnode =token\_new(RBRACK,yytext,line,NULL);
423. token\_print(Tnode);
424. last\_node->next = Tnode;
425. last\_node = Tnode;
426. nchar+=yyleng;
427. nword++;
428. }
429. {LBRACE}   {
430. Token\* Tnode =token\_new(LBRACE,yytext,line,NULL);
431. token\_print(Tnode);
432. last\_node->next = Tnode;
433. last\_node = Tnode;
434. nchar+=yyleng;
435. nword++;
436. }
437. {RBRACE}   {
438. Token\* Tnode =token\_new(RBRACE,yytext,line,NULL);
439. token\_print(Tnode);
440. last\_node->next = Tnode;
441. last\_node = Tnode;
442. nchar+=yyleng;
443. nword++;
444. }
445. {il\_identifier} {
446. **if**(lineon) printf("Line=%3d ",line);
447. printf("错误: 错误的标识符: %s\n",yytext);
448. }
449. \n      {
450. addLine(1);
451. }
452. . {
453. **if**(lineon) printf("Line=%3d ",line);
454. printf("错误: 不可识别的输入: %s\n",yytext);
455. }

458. %%
459. **int** main(**void**)
460. {
461. head\_node = token\_new(head,"begin",0,NULL);
462. last\_node = head\_node;
464. strcpy(token\_name[0], "head");
465. strcpy(token\_name[1], "NCHAR");
466. strcpy(token\_name[2], "letter");
467. strcpy(token\_name[3], "digit");
468. strcpy(token\_name[4], "IDENFR");
469. strcpy(token\_name[5], "INTCON");
470. strcpy(token\_name[6], "STRCON");
471. strcpy(token\_name[7], "MAINTK");
472. strcpy(token\_name[8], "CONSTTK");
473. strcpy(token\_name[9], "INTTK");
474. strcpy(token\_name[10], "BREAKTK");
475. strcpy(token\_name[11], "CONTINUETK");
476. strcpy(token\_name[12], "IFTK");
477. strcpy(token\_name[13], "ELSETK");
478. strcpy(token\_name[14], "WHILETK");
479. strcpy(token\_name[15], "GETINTTK");
480. strcpy(token\_name[16], "PRINTFTK");
481. strcpy(token\_name[17], "RETURNTK");
482. strcpy(token\_name[18], "PLUS");
483. strcpy(token\_name[19], "MINU");
484. strcpy(token\_name[20], "VOIDTK");
485. strcpy(token\_name[21], "NOT");
486. strcpy(token\_name[22], "AND");
487. strcpy(token\_name[23], "OR");
488. strcpy(token\_name[24], "MULT");
489. strcpy(token\_name[25], "DIV");
490. strcpy(token\_name[26], "MOD");
491. strcpy(token\_name[27], "LSS");
492. strcpy(token\_name[28], "LEQ");
493. strcpy(token\_name[29], "GRE");
494. strcpy(token\_name[30], "GEQ");
495. strcpy(token\_name[31], "EQL");
496. strcpy(token\_name[32], "NEQ");
497. strcpy(token\_name[33], "ASSIGN");
498. strcpy(token\_name[34], "SEMICN");
499. strcpy(token\_name[35], "COMMA");
500. strcpy(token\_name[36], "LPARENT");
501. strcpy(token\_name[37], "RPARENT");
502. strcpy(token\_name[38], "LBRACK");
503. strcpy(token\_name[39], "RBRACK");
504. strcpy(token\_name[40], "LBRACE");
505. strcpy(token\_name[41], "RBRACE");
506. strcpy(token\_name[42], "FLOAT");
507. strcpy(token\_name[43], "il\_identifier");
509. yylex();
510. printf("nchar=%d, nword=%d, nline=%d\n",nchar, nword, line);
511. **return** 0;
512. }
513. **int** yywrap()    {
514. **return** 1;
515. }
517. **void** addLine(**int** cnt)   {
518. line += cnt;
519. }
521. Token\* token\_new(TokenType type, **char**\* value, **int** line,Token\* node) {
522. Token\* token = (Token\*)malloc(**sizeof**(Token));
523. token->type = type;
524. token->value = strdup(value); // 使用strdup分配内存并复制字符串
525. token->line = line;
526. token->next = node;
527. **return** token;
528. }
530. **void** token\_free(Token\* token) {
531. **if** (token != NULL) {
532. free(token->value);
533. free(token);
534. }
535. }
537. **void** token\_print(Token\* token) {
538. **if** (token != NULL) {
539. **if**(lineon) printf("Line=%3d ",token->line);
540. printf("%s %s\n",
541. token\_name[token->type], token->value);
542. }
543. }