

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	PRINTF TK	>	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

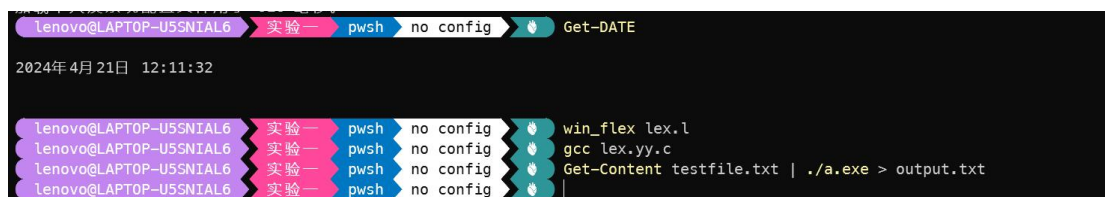
是否输出行号)

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

 testfile.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
const int array[2] = {1,2};
```

```
int main(){
    int c;
    c = getint();
    printf("output is %d",c);
    return c;
}
```

以上是 testfile.txt 文件的内容，为实验的样例输入。

output.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
INTCON 2
RBRACK ]
ASSIGN =
LBRACE {
INTCON 1
COMMA ,
INTCON 2
RBRACE }
SEMICN ;
INTTK int
MAINTK main
LPARENT (
RPARENT )
LBRACE {
INTTK int
IDENFR c
SEMICN ;
IDENFR c
ASSIGN =
GETINTTK getint
LPARENT (
RPARENT )
SEMICN ;
PRINTFTK printf
LPARENT (
STRCON "output is %d"
COMMA ,
IDENFR c
RPARENT )
SEMICN ;
RETURNTK return
IDENFR c
SEMICN ;
RBRACE }
nchar=83, nword=38, nline=9
```

以上是实验的 `output.txt` 的内容，与实验样例输出完全一致。

output.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Line= 1 INTCON 2
Line= 1 RBRACK ]
Line= 1 ASSIGN =
Line= 1 LBRACE {
Line= 1 INTCON 1
Line= 1 COMMA ,
Line= 1 INTCON 2
Line= 1 RBRACE }
Line= 1 SEMICN ;
Line= 3 INTTK int
Line= 3 MAINTK main
Line= 3 LPARENT (
Line= 3 RPARENT )
Line= 3 LBRACE {
Line= 4 INTTK int
Line= 4 IDENFR c
Line= 4 SEMICN ;
Line= 5 IDENFR d
Line= 5 ASSIGN =
Line= 5 GETINTTK getint
Line= 5 LPARENT (
Line= 5 RPARENT )
Line= 5 SEMICN ;
Line= 6 PRINTFTK printf
Line= 6 LPARENT (
Line= 6 STRCON "output is %d"
Line= 6 COMMA ,
Line= 6 IDENFR c
Line= 6 RPARENT )
Line= 6 SEMICN ;
Line= 7 RETURNTK return
Line= 7 IDENFR c
Line= 7 SEMICN ;
Line= 8 RBRACE }
nchar=83, nword=38, nline=9
```

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

```
il_identifier      ((digit){digit}*\\(digit)+)?(e|E[+-]?{digit}+)?((letter){digit})*{letter}((letter){digit})*
```

错误处理情况

```
{il_identifier} {  
    if(lineon) printf("Line=%3d ",line);  
    printf("错误: 错误的标识符: %s\n",yytext);  
}  
\n {  
    addLine(1);  
}  
. {  
    if(lineon) printf("Line=%3d ",line);  
    printf("错误: 不可识别的输入: %s\n",yytext);  
}
```

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;
54.
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;
63.
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. %}
69.
70. line_comment  (\\/\\..*\\n)
71.
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. id_identifier ({digit}|{digit}*(\.{digit}+)?(e|E[+-]?{digit}+)?){letter}|{digit})*{letter}|{digit})*
115.
116.
117. %%
118.
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+=yytext;
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+=yytext;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yylen;
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+=yylen;
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+=yylen;
210.     nword++;
211. }
212. {PRINTF TK} {
213.     Token* Tnode =token_new(PRINTFTK,yytext,line,NULL);
214.     token_print(Tnode);
215.     last_node->next = Tnode;
216.     last_node = Tnode;
217.     nchar+=yylen;
218.     nword++;
219. }
220. {RETURN TK} {
221.     Token* Tnode =token_new(RETURN TK,yytext,line,NULL);
222.     token_print(Tnode);
223.     last_node->next = Tnode;
224.     last_node = Tnode;
225.     nchar+=yylen;
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+=yy leng;
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+=yy leng;
242.     nword++;
243. }
244.
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yy leng;
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+=yylen;
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+=yylen;
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+=yylen;
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+=yylen;
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+=yylen;
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+=yytext;
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+=yytext;
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+=yytext;
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+=yytext;
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+=yytext;
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. }
456.
457.
458. %%
459. int main(void)
460. {
461.     head_node = token_new(head,"begin",0,NULL);
462.     last_node = head_node;
463.
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], "PRINTF TK");
481.     strcpy(token_name[17], "RETURN TK");
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");
508.
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. }
516.
517. void addLine(int cnt) {
518.     line += cnt;
519. }
520.
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. }
529.
530. void token_free(Token* token) {
531.     if (token != NULL) {
532.         free(token->value);
533.         free(token);
534.     }
535. }
536.
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. }
```