

词法分析器 Lexical Analyzer

学号：19335182

姓名：唐晨轩

1.实验目的

设计，编写并调试一个词法分析器，能够读取.c文件并进行正确地词法分析。加深对词法分析原理地理解。

2.实验要求

手动设计实现，或者使用Lex实现词法分析器。

这里我选择使用 **手动设计实现** 来实现词法分析器。

词法分析器需要将输入进来的源语言，转换成词法单元的Token序列。

3.实验过程

3.1 分析词法分析器应具有的功能

输入：所给文法的源程序字符串

输出：词法单元的token序列

3.2 确认待分析的简单词法

- 首先要确认待分析的简单语言词法：

(1) 关键字：int,float,double,long,if,while,else,end,for等等

(2) 运算符和定界符：{,},(,),+,-,/,++,--,>,<,>=,<=,|,&,||,&&等等

(3) 其他单词是字符串(*I*)和数字(*D*)，通过以下正规式定义：

$I = letter(letter | digit)^*$ //代表字符串

$N = digit(digit)^*$ //代表数字

(4) 空格由空白、制表符和换行符组成。空格一般用来分隔标识符、数字、运算符、界符和关键字。

(5) 注释：包括//和/**/

3.3 单词符号对应的种别码表

- 以下为各种单词符号对应的种别码：

(由于markdown建表很麻烦所以我在word下建表后截图至此)

(单词符号的搜集来自网络)

单词符号	种别码	单词符号	种别码	单词符号	种别码
digit digit*	0	--	23	struct	45
letter(letter digit)*	1	&&	24	union	46
#	2		25	if	47
<	3	>=	26	else	48
>	4	<=	27	goto	49
(5	==	28	switch	50
)	6	!=	29	case	51
[7	>>	30	do	52
]	8	<<	31	while	53
{	9	:=	32	for	54
}	10	int	33	continue	55
,	11	long	34	break	56
;	12	short	35	return	57
+	13	float	36	default	58
-	14	double	37	typedef	59
*	15	char	38	auto	60
/	16	unsigned	39	register	61
%	17	signed	40	extern	62
!	18	const	41	static	63
\'	19	void	42	sizeof	64
"	20	volatile	43	include	65
=	21	enum	44	letter letter*.h	66
\++	22				

3.4 实验流程思想

本次实验基本任务是：

把源程序识别出具有独立意义的单词符号，输出对应的二元组，我在设计单词符号和种别码的时候，直接采用 c 语言保留字和运算符，.h 文件采用字符串闭包表示，经过测试，对任意 c 语言文件都可以进行词法分析。

本实验程序功能强大，可以对所有 c 语言基础语法文件进行词法分析，可扩展性强，可以轻松扩展后识别 c++ 等高级语言(改变一些单词符号和对应的种别码即可)，鲁棒性强等特点。

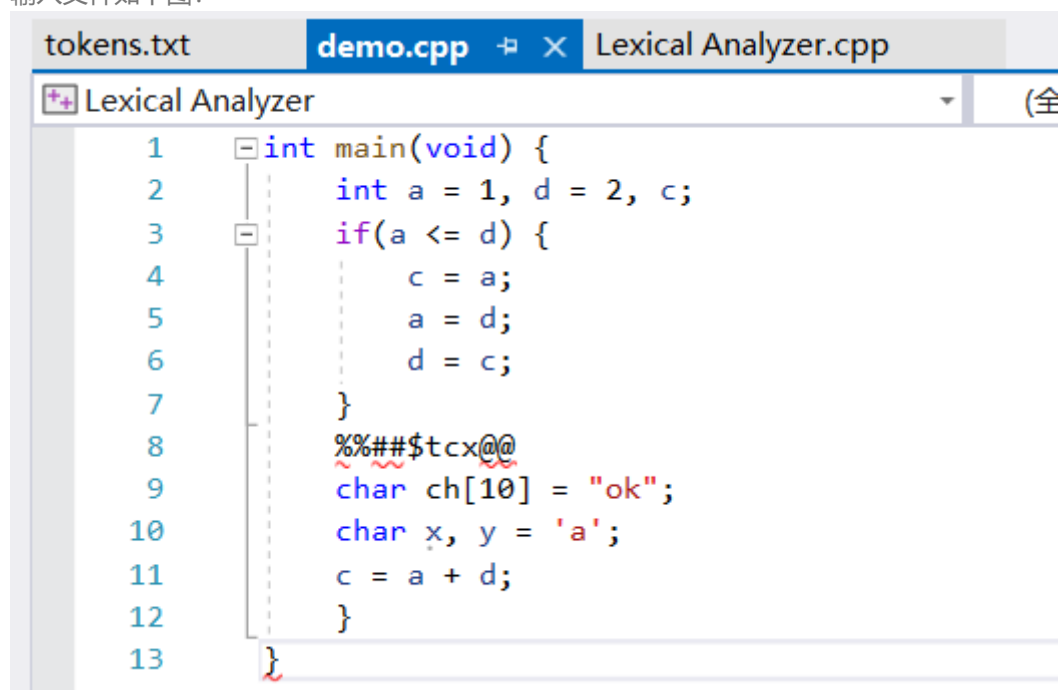
程序设计思路为：

首先按行读入字符串后，先对字符串中多余空格进行处理，主要目的是为了减少后续不必要的判断，加快运行速度。

处理完空格后对代码中的单行注释进行处理，由于是按行读入，所以多行注释暂时不处理，经过初始化处理掉冗余元素后进行字符串分割，在此处也增加了对多行注释的特判，对于分割后的字符串对我设计的种别码进行 hash 映射输出二元组即可。

3.5 实验结果展示

输入文件如下图：



```
tokens.txt demo.cpp Lexical Analyzer.cpp
Lexical Analyzer (全)
1  int main(void) {
2      int a = 1, d = 2, c;
3      if(a <= d) {
4          c = a;
5          a = d;
6          d = c;
7      }
8      %$tcx@@
9      char ch[10] = "ok";
10     char x, y = 'a';
11     c = a + d;
12 }
13 }
```


输出文件如下图:

tokens.txt	demo.cpp	Lexical Analyzer.cpp
1	第1行读进来的是: int main(void) {	
2	<33,int>	
3	<1,main>	
4	<5,(>	
5	<42,void>	
6	<6,)>	
7	<9,{>	
8	第2行读进来的是: int a = 1, d = 2, c;	
9	<33,int>	
10	<1,a>	
11	<21,=>	
12	<0,binary:1>	
13	<11,,>	
14	<1,d>	
15	<21,=>	
16	<0,binary:10>	
17	<11,,>	
18	<1,c>	
19	<12,;>	
20	第3行读进来的是: if(a <= d) {	
21	<47,if>	
22	<5,(>	
23	<1,a>	
24	<27,<=>	
25	<1,d>	
26	<6,)>	
27	<9,{>	
28	第4行读进来的是: c = a;	
29	<1,c>	
30	<21,=>	
31	<1,a>	
32	<12,;>	
33	第5行读进来的是: a = d;	
34	<1,a>	
35	<21,=>	
36	<1,d>	
37	<12,;>	
38	第6行读进来的是: d = c;	
39	<1,d>	
40	<21,=>	
41	<1,c>	
42	<12,;>	
43	第7行读进来的是: }	

100 % 未找到相关问题

tokens.txt	demo.cpp	Lexical Analyzer.cpp
28	第4行读进来的是: c = a;	
29	<1,c>	
30	<21,=>	
31	<1,a>	
32	<12,;>	
33	第5行读进来的是: a = d;	
34	<1,a>	
35	<21,=>	
36	<1,d>	
37	<12,;>	
38	第6行读进来的是: d = c;	
39	<1,d>	
40	<21,=>	
41	<1,c>	
42	<12,;>	
43	第7行读进来的是: }	

```

44 <10,>
45 第8行读进来的是: %$tcx@@
46 <17,>
47 <17,>
48 <2,>
49 <2,>
50 第8行出现错误: $
51 <1,tcx>
52 第8行出现错误: @
53 第8行出现错误: @
54 第9行读进来的是: char ch[10] = "ok";
55 <38,char>
56 <1,ch>

```

100 %

未找到相关问题

tokens.txt

demo.cpp

Lexical Analyzer.cpp

```

52 第8行出现错误: @
53 第8行出现错误: @
54 第9行读进来的是: char ch[10] = "ok";
55 <38,char>
56 <1,ch>
57 <7,>
58 <0,binary:1010>
59 <8,>
60 <21,>
61 <20,>
62 <1,ok>
63 <20,>
64 <12,>
65 第10行读进来的是: char x, y = 'a';
66 <38,char>
67 <1,x>
68 <11,>
69 <1,y>
70 <21,>
71 <19,'>
72 <1,a>
73 <19,'>
74 <12,>
75 第11行读进来的是: c = a + d;
76 <1,c>
77 <21,>
78 <1,a>
79 <13,+>
80 <1,d>

```

100 %

未找到相关问题

tokens.txt	demo.cpp	Lexical Analyzer.cpp
79	<13,+>	
80	<1,d>	
81	<12,;>	
82	第12行读进来的是: }	
83	<10,}>	
84	第13行读进来的是: }	
85	<10,}>	
86		

4.实验总结

本次实验的输出与预期一致；

可以正确快速的完成词法分析任务，而且当遇到词法错误的时候也会提出报错，且不会对后续程序的执行造成影响。

总的来说本次实验圆满完成了。后续会将代码和实验报告以及输入输出样例都上传到我的Github仓库中。

我的Github编译原理仓库地址为：<https://github.com/Yuhiman7Xc/Compilation-Principle/tree/main>