# 目录

[1 目录 1](#_Toc513585099)

[2 词法分析单元测试 1](#_Toc513585100)

[2.1 测试环境 1](#_Toc513585101)

[2.1.1 依赖 1](#_Toc513585102)

[2.1.2 调试输出 3](#_Toc513585103)

[2.1.3 获取可执行文件 3](#_Toc513585104)

[2.1.4 输入文件 3](#_Toc513585105)

[2.1.5 输出文件 3](#_Toc513585106)

[2.2 测试计划 4](#_Toc513585107)

[2.3 错误处理单个测试 4](#_Toc513585108)

[2.3.1 行长度超过限制 4](#_Toc513585109)

[2.3.2 标识符长度超过限制 7](#_Toc513585110)

[2.3.3 非法字符 8](#_Toc513585111)

[2.3.4 读取字符常量时遇到文件尾 9](#_Toc513585112)

[2.3.5 读取的字符常量为空 10](#_Toc513585113)

[2.3.6 读取的字符常量不止一个字符 11](#_Toc513585114)

[2.3.7 读取字符常量先遇到换行符而不是单引号 12](#_Toc513585115)

[2.3.8 读取多行注释时遇到文件尾 13](#_Toc513585116)

[2.4 错误处理组合测试 14](#_Toc513585117)

[2.4.1 错误组合测试1 14](#_Toc513585118)

[2.4.2 错误组合测试2 15](#_Toc513585119)

[2.4.3 错误组合测试3 16](#_Toc513585120)

[2.5 综合测试 17](#_Toc513585121)

# 词法分析单元测试

## 测试环境

### 依赖

词法分析单元测试本质上不依赖于其它任何模块，但需要对源程序进行预处理，即在源程序的一开始添加一个换行，并把所有的字母转化为小写，该工作在main.cpp中进行。

词法分析的记号返回值需要用宏进行定义，也可以用YACC编写一个简单的YACC源程序生成yy.tab.cpp和yy.tab.h，里面包含了记号的宏定义。我们采用了后者。所编写的简单YACC程序如下，主要是对记号进行了声明。

1. %{
2. #include "main.h"
3. #include "yacc.tab.h"
5. **extern** "C"
6. {
7. **void** yyerror(**const** **char** \*s);
8. **int** yyparse();
9. **extern** **int** yylex();
10. }
12. %}
14. %token PROGRAM
15. %token CONST
16. %token VAR
17. %token ARRAY
18. %token OF
19. %token PROCEDURE
20. %token FUNCTION
21. %token \_BEGIN
22. %token END
23. %token IF
24. %token THEN
25. %token FOR
26. %token TO
27. %token DO
28. %token ELSE
29. %token REPEAT
30. %token UNTIL
31. %token WHILE
33. %token IDENTIFIER
34. %token UINUM
35. %token UFNUM
36. %token **CHAR**
37. %token TYPE
38. %token ASSIGNOP
39. %token RELOP
40. %token ADDOP
41. %token MULOP
42. %token NOT
43. %token RANGEDOT
45. %start programstruct
47. %locations
49. %%
51. programstruct:
53. %%

56. **void** yyerror(**const** **char** \*s){
57. cout << s << endl;
58. }

### 调试输出

如果定义了宏LEXDEBUG，那么就可以输出词法分析程序的调试输出，即输出按顺序输出每一个记号及其属性。具体做法是，在每一个正则表达式的动作中添加如下代码（以标识符为例）

1. #ifdef LEXDEBUG
2. cout << "identifier: " << yylval->str << endl;
3. #endif

那么只要在一开始定义宏LEXDEBUG，就可以产生调试输出

另外，一旦遇到词法错误，也会输出词法错误的详细信息。

### 获取可执行文件

采用如下命令

1. flex lex.l  //编译lex.l，生成lex.yy.c
2. ren lex.yy.c lex.yy.cpp  //将lex.yy.c更名为lex.yy.cpp
3. bison -vd --debug yacc.y  //编译yacc.y，生成yacc.tab.h和yacc.tab.c
4. ren yacc.tab.c yacc.tab.cpp  //将yacc.tab.c更名为yacc.cpp
5. g++ lex.yy.cpp yacc.tab.cpp main.cpp -o pascal2c.exe
6. //编译生成可执行文件pascal2c.exe

### 输入文件

PascalProgram.pas

### 输出文件

LexOut.txt

## 测试计划

* 有针对性的测试每一种词法错误
* 组合上述词法错误，词法分析程序能否成功处理错误并恢复（有一些词法错误，例如行长度超过限制，会导致词法分析程序直接停止运行）
* 以一个较复杂的，没有错误的程序作为最终测试

## 错误处理单个测试

### 行长度超过限制

* 测试用例

1. program test(input,output);
2. var aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa:integer;
3. begin
4. end.

* 预期结果

报行长度超过10000限制的错误，词法分析程序停止运行

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. [Line length too large, exceed 10000] 2.1-2.10014  //行长度超过限制
10. Lex analyse abort!  //词法分析程序停止运行

* 分析

报错结果和对策符合预期，词法分析程序停止运行。

### 标识符长度超过限制

* 测试用例

1. program test(input,output);
2. var asadhsadsadsadaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa:integer;
3. begin
5. end.

* 预期结果

报标识符长度超过限制100的错误；保留前100个字符作为标识符记号输出

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. keyword: var
10. [Identifier length too large, exceed 100] 2.5-2.133  //标识符长度超过限
11. identifier: asadhsadsadsadaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  //保留前100个字符
12. delimiter: :
13. type: integer
14. delimiter: ;
15. keyword: begin
16. keyword: end
17. delimiter: .

* 分析

报错结果和对策符合预期，保留了前100个字符作为当前识别的标识符，词法分析程序继续运行。

### 非法字符

* 测试用例

1. program test(input,output);
2. var a:integer;
3. begin?
4. $$
5. end.##

* 预期结果

非法字符?出现一次，$出现两次，#出现两次，所以共报五次非法字符的错误，并忽略这些非法字符

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. keyword: var
10. identifier: a
11. delimiter: :
12. type: integer
13. delimiter: ;
14. keyword: begin
15. [Invalid character!] 3.6-3.6  //非法字符
16. begin?
17. ^
18. [Invalid character!] 4.5-4.5  //非法字符
19. $$
20. ^
21. [Invalid character!] 4.6-4.6  //非法字符
22. $$
23. ^
24. keyword: end
25. delimiter: .
26. [Invalid character!] 5.5-5.5  //非法字符
27. end.##
28. ^
29. [Invalid character!] 5.6-5.6  //非法字符
30. end.##
31. ^

* 分析

报错结果和对策符合预期，词法分析程序没有受到非法字符的影响。

### 读取字符常量时遇到文件尾

* 测试用例

1. program test(input,output);
2. **const** a='

* 预期结果

报读取常量时遇到文件尾的错误

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. keyword: **const**
10. identifier: a
11. EQUAL: =
12. [Unexpected end of file when reading a **char** constant] 2.9-2.9  //遇到文件尾
13. **const** a='
14. ^

* 分析

报错结果和对策符合预期。

### 读取的字符常量为空

* 测试用例

1. program test(input,output);
2. **const** a=''
3. begin
5. end.

* 预期结果

报读取的字符常量为空的错误，并提交”\0”，作为当前识别的字符常量

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. keyword: **const**
10. identifier: a
11. EQUAL: =
12. [Char constant missing!] 2.9-2.10  //字符常量为空
13. **const** a=''
14. ^^
15. **char**:   //”\0”作为当前识别的字符常量
16. keyword: begin
17. keyword: end
18. delimiter: .

* 分析

报错结果和对策符合预期，提交了”\0”作为当前识别的字符常量，词法分析程序继续运行。

### 读取的字符常量不止一个字符

* 测试用例

1. program test(input,output);
2. **const** a='abc';
3. begin
4. writeln(a);
5. end.

* 预期结果

报读取的字符常量不止一个字符的错误，并提交第一个字符作为当前识别的字符常量

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. keyword: **const**
10. identifier: a
11. EQUAL: =
12. [Too many characters in a **char** constant!] 2.10-2.12  //识别了多个字符
13. **const** a='abc';
14. ^^^
15. **char**: a  //第一个字符作为当前识别的字符常量
16. delimiter: ;
17. keyword: begin
18. identifier: writeln
19. delimiter: (
20. identifier: a
21. delimiter: )
22. delimiter: ;
23. keyword: end
24. delimiter: .

* 分析

报错结果和对策符合预期，保留了第一个字符作为当前识别的字符常量，词法分析程序继续运行。

### 读取字符常量先遇到换行符而不是单引号

* 测试用例

1. program test(input,output);
2. **const** a='a
3. b='
4. c='cba
5. begin
7. end.

* 预期结果

报三次单引号缺失的错误，第一次提交”a”作为当前识别的字符常量，第二次提交”\0”作为当前识别的字符常量，第三次提交”c”作为当前识别的字符常量

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. keyword: **const**
10. identifier: a
11. EQUAL: =
12. [Right quote missing!] 3.11-3.11  //右单引号缺失
13. **const** a='a
14. ^
15. **char**: a  //提交”a”作为当前识别的字符常量
16. identifier: b
17. EQUAL: =
18. [Right quote missing!] 4.11-4.11  //右单引号缺失
19. b='
20. ^
21. **char**:   //提交”\0”作为当前识别的字符常量
22. identifier: c
23. EQUAL: =
24. [Right quote missing!] 5.11-5.11  //右单引号缺失
25. c='cba
26. ^
27. **char**: c  //提交”c”作为当前识别的字符常量
28. keyword: begin
29. keyword: end
30. delimiter: .

* 分析

报错结果和对策符合预期，三种情况都做出了相应的提交策略，且词法分析程序都能继续运行。

### 读取多行注释时遇到文件尾

* 测试用例

1. program test(input,output);
2. {
3. begin
5. end.

* 预期结果

报读取多行注释时遇到文件尾的错误（右花括号缺失）

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. [Unexpected end of file when reading a multiple line comment, lacking of a right brace] 5.4-5.4  //读取多行注释时遇到文件尾（右花括号缺失）
10. end.
11. ^

* 分析

报错结果和对策符合预期。

## 错误处理组合测试

### 错误组合测试1

* 测试用例

1. program test(input,output);
2. var asadhsadsadsadaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa:integer;
3. #
4. begin
6. end.

* 预期结果

报标识符长度超过限制100的错误，报非法字符的错误

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. keyword: var
10. [Identifier length too large, exceed 100] 2.5-2.133  //标识符长度超过限制
11. identifier: asadhsadsadsadaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
12. delimiter: :
13. type: integer
14. delimiter: ;
15. [Invalid character!] 3.1-3.1  //非法字符
16. #
17. ^
18. keyword: begin
19. keyword: end
20. delimiter: .

* 分析

报错结果和对策符合预期。

### 错误组合测试2

* 测试用例

1. program test(input,output);
2. **const** a=''
3. b='abc

* 预期结果

报读取的字符常量为空的错误，报读取字符常量时遇到文件尾的错误

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. keyword: **const**
10. identifier: a
11. EQUAL: =
12. [Char constant missing!] 2.9-2.10  //读取的字符常量为空
13. **const** a=''
14. ^^
15. **char**:
16. identifier: b
17. EQUAL: =
18. [Unexpected end of file when reading a **char** constant] 3.12-3.12  //遇到文件尾
19. b='abc
20. ^

* 分析

报错结果和对策符合预期。

### 错误组合测试3

* 测试用例

1. program test(input,output);
2. **const** a='abc'
3. b='bac
4. begin
5. {123
6. end.

* 预期结果

报读取的字符常量不止一个字符的错误，报右单引号缺失的错误，报多行注释遇到文件尾的错误

* 测试结果

1. keyword: program
2. identifier: test
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. keyword: **const**
10. identifier: a
11. EQUAL: =
12. [Too many characters in a **char** constant!] 2.10-2.12  //字符常量不止一个字符
13. **const** a='abc'
14. ^^^
15. **char**: a
16. identifier: b
17. EQUAL: =
18. [Right quote missing!] 4.11-4.11  //右单引号缺失
19. b='bac
20. ^
21. **char**: b
22. keyword: begin
23. [Unexpected end of file when reading a multiple line comment, lacking of a right brace] 6.4-6.4  //读取多行注释时遇到文件尾
24. end.
25. ^

* 分析

报错结果和对策符合预期。

## 综合测试

该测试样例不包含错误，实现了快速排序的功能。

* 测试用例

1. program quicksort(input,output);
2. var
3. n,i:integer;
4. a:array[0..100000] of integer;
5. b:**char**;
7. procedure kp(l,r:integer);
8. var
9. i,j,mid:integer;
10. begin
11. b:='a';
12. **if** l>=r then exit;
13. i:=l;j:=r;mid:=a[(l+r) div 2];
14. repeat
15. begin
16. **while** a[i]<mid **do** inc(i);
17. **while** a[j]>mid **do** dec(j);
18. **if** i<=j then
19. begin
20. a[0]:=a[i];a[i]:=a[j];a[j]:=a[0];
21. inc(i);dec(j);
22. end
23. end
24. until i>j;
25. kp l,j);
26. kp(i,r)
27. end;
29. begin
30. readln(n);
31. **for** i:=1 to n **do**
32. read(a[i]);
33. kp(1,n);
34. **for** i:=1 to n **do**
35. write(a[i]);
36. end.

* 预期结果

记号流，不包含任何报错信息

* 测试结果

1. keyword: program
2. identifier: quicksort
3. delimiter: (
4. identifier: input
5. delimiter: ,
6. identifier: output
7. delimiter: )
8. delimiter: ;
9. keyword: var
10. identifier: n
11. delimiter: ,
12. identifier: i
13. delimiter: :
14. type: integer
15. delimiter: ;
16. identifier: a
17. delimiter: :
18. keyword: array
19. delimiter: [
20. integer:0
21. RANGEDOT: ..
22. integer:100000
23. delimiter: ]
24. keyword: of
25. type: integer
26. delimiter: ;
27. identifier: b
28. delimiter: :
29. type: **char**
30. delimiter: ;
31. keyword: procedure
32. identifier: kp
33. delimiter: (
34. identifier: l
35. delimiter: ,
36. identifier: r
37. delimiter: :
38. type: integer
39. delimiter: )
40. delimiter: ;
41. keyword: var
42. identifier: i
43. delimiter: ,
44. identifier: j
45. delimiter: ,
46. identifier: mid
47. delimiter: :
48. type: integer
49. delimiter: ;
50. keyword: begin
51. identifier: b
52. ASSIGNOP: :=
53. **char**: a
54. delimiter: ;
55. keyword: **if**
56. identifier: l
57. RELOP: >=
58. identifier: r
59. keyword: then
60. identifier: exit
61. delimiter: ;
62. identifier: i
63. ASSIGNOP: :=
64. identifier: l
65. delimiter: ;
66. identifier: j
67. ASSIGNOP: :=
68. identifier: r
69. delimiter: ;
70. identifier: mid
71. ASSIGNOP: :=
72. identifier: a
73. delimiter: [
74. delimiter: (
75. identifier: l
76. ADDOP: +
77. identifier: r
78. delimiter: )
79. MULOP: div
80. integer:2
81. delimiter: ]
82. delimiter: ;
83. keyword: repeat
84. keyword: begin
85. keyword: **while**
86. identifier: a
87. delimiter: [
88. identifier: i
89. delimiter: ]
90. RELOP: <
91. identifier: mid
92. keyword: **do**
93. identifier: inc
94. delimiter: (
95. identifier: i
96. delimiter: )
97. delimiter: ;
98. keyword: **while**
99. identifier: a
100. delimiter: [
101. identifier: j
102. delimiter: ]
103. RELOP: >
104. identifier: mid
105. keyword: **do**
106. identifier: dec
107. delimiter: (
108. identifier: j
109. delimiter: )
110. delimiter: ;
111. keyword: **if**
112. identifier: i
113. RELOP: <=
114. identifier: j
115. keyword: then
116. keyword: begin
117. identifier: a
118. delimiter: [
119. integer:0
120. delimiter: ]
121. ASSIGNOP: :=
122. identifier: a
123. delimiter: [
124. identifier: i
125. delimiter: ]
126. delimiter: ;
127. identifier: a
128. delimiter: [
129. identifier: i
130. delimiter: ]
131. ASSIGNOP: :=
132. identifier: a
133. delimiter: [
134. identifier: j
135. delimiter: ]
136. delimiter: ;
137. identifier: a
138. delimiter: [
139. identifier: j
140. delimiter: ]
141. ASSIGNOP: :=
142. identifier: a
143. delimiter: [
144. integer:0
145. delimiter: ]
146. delimiter: ;
147. identifier: inc
148. delimiter: (
149. identifier: i
150. delimiter: )
151. delimiter: ;
152. identifier: dec
153. delimiter: (
154. identifier: j
155. delimiter: )
156. delimiter: ;
157. keyword: end
158. keyword: end
159. keyword: until
160. identifier: i
161. RELOP: >
162. identifier: j
163. delimiter: ;
164. identifier: kp
165. identifier: l
166. delimiter: ,
167. identifier: j
168. delimiter: )
169. delimiter: ;
170. identifier: kp
171. delimiter: (
172. identifier: i
173. delimiter: ,
174. identifier: r
175. delimiter: )
176. keyword: end
177. delimiter: ;
178. keyword: begin
179. identifier: readln
180. delimiter: (
181. identifier: n
182. delimiter: )
183. delimiter: ;
184. keyword: **for**
185. identifier: i
186. ASSIGNOP: :=
187. integer:1
188. keyword: to
189. identifier: n
190. keyword: **do**
191. identifier: read
192. delimiter: (
193. identifier: a
194. delimiter: [
195. identifier: i
196. delimiter: ]
197. delimiter: )
198. delimiter: ;
199. identifier: kp
200. delimiter: (
201. integer:1
202. delimiter: ,
203. identifier: n
204. delimiter: )
205. delimiter: ;
206. keyword: **for**
207. identifier: i
208. ASSIGNOP: :=
209. integer:1
210. keyword: to
211. identifier: n
212. keyword: **do**
213. identifier: write
214. delimiter: (
215. identifier: a
216. delimiter: [
217. identifier: i
218. delimiter: ]
219. delimiter: )
220. delimiter: ;
221. keyword: end
222. delimiter: .

* 分析

记号流输出与源程序相符合，不包含任何错误信息。