编译原理词法分析器实验报告

班级:2017211301 学号:2017211137 姓名: 王亚宇

2019 年 10 月 24 日

1 实验内容

1.1 题目

词法分析程序的设计与实现

1.2 实验目的

设计一个并实现 C 语言的词法分析程序, 要求实现如下功能.

(1) 可以识别出用 C 语言编写的源程序中的每次单词符号, 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, 并报告错误所在位置.

(5) 队源程序中出现的错误进行适当的恢复, 使此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此法错误.

2 实验环境

2.1 操作系统

Ubuntu 18.04 LTS

2.2 编译器

flex 2.6.4

g++ (Ubuntu 7.4.0-1ubuntu1 18.04.1) 7.4.0

3 实验原理

3.1 LEX

Lex 是 LEXical compiler 的缩写, 是 Unix 环境下非常著名的工具, 主要功能是生成一个词法分析器 (scanner) 的 C 源码, 描述规则采用正则表达式 (regular expression). 描述词法分析器的文件 *.l, 经过 lex 编译后, 生成一个 lex.yy.c 的文件, 然后由 C 编译器编译生成一个词法分析器.

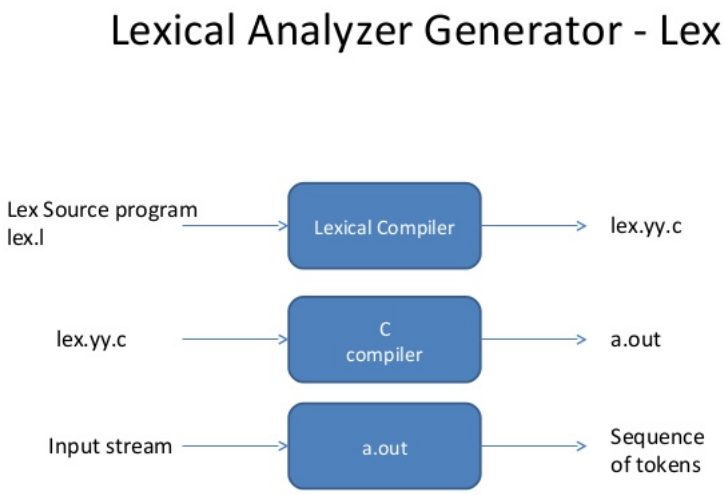

图 1: 使用 LEX 生成词法分析程序的流程

一种匹配的常规表达式可能会包含相关的动作. 这一动作可能还包括返回一个标记. 当 Lex 接收到文件或文本形式的输入时, 它试图将文本与常规表达式进行匹配. 它一次读入一个输入字符, 直到找到一个匹配的模式. 如果能够找到一个匹配的模式,Lex 就执行相关的动作（可能包括返回一个标记）. 另一方面, 如果没有可以匹配的常规表达式, 将会停止进一步的处理,Lex 将显示一个错误消息.

Lex 文件结构分为三个部分:

```
declarations
%%
```

```
3 translation rules
4 %%
5 auxiliary procedures
```

分别是声明, 翻译规则和辅助过程.

声明部分包括变量的声明, 符号常量的声明和正则表达式定义. 正规定义中定义的名字可以出现在翻译规则的正规表达式中.

翻译规则部分是由正规表达式和相应的动作组成的具有如下形式的语句序列:

$$\begin{array}{ll} p_1 & action_1 \\ p_2 & action_2 \\ \vdots & \\ p_n & action_n \end{array}$$

其中 p_i 是正规表达式, 描述一种记号的模式; $action_i$ 是用 C 语言描述的程序段.

由 LEX 生成的词法分析程序在识别单词符号时, 遵循以下的匹配规则:

(1) 最长匹配原则: 当有几条规则都适用时, 实施匹配最长输入串的那个规则.

(2) 有限匹配规则: 当有几条规则都适用, 并且匹配长度相同时, 则实施排在最前面的那条规则.

辅助规程是对翻译规则的补充. 翻译规则部分中某些动作需要调用的过程或函数, 如果不是 C 语言的库函数, 要在此给出具体的定义.

Variable	Function
yytext	匹配模式的文本存储在这一变量中（char*）
yyleng	匹配模式串的长度

表 1: Lex 变量

3.2 设计思路

通过规翻译规则部分使用正则表达式对 C 语言保留字、表示符、注释等进行匹配, 并使用 C 语言对匹配内容进行分析以统计行数、各类单词的个数、以及字符总数.

4 代码实现

4.1 声明

```
%{
#include <bits/stdc++.h>
#include "identifier.h"
int wordCount = 0;
int lineCount = 1;
int charCount = 0;
int column = 0;
bool comment_multi();
bool comment_line();
void handle(std::string s="");
void lexicalError(std::string);
%}

digit               [0-9]
letter              [a-zA-Z_]
hexdigit            [a-fA-F0-9]
exponent            ([Ee][+-]?{digit}+)
power               ([Pp][+-]?{digit}+)
delimiter           [ \t\r\v\f]
newline             \n
whitespace          {delimiter}+
intsymbol           ((u|U)|(u|U)?(l|L|ll|LL)|(l|L|ll|LL)(u|U))
floatsymbol         (f|F|l|L)
```

- 声明处理函数
- 初始化行数、词数、字符数统计值, 设置当前列为 0
- 定义类型

ID	Describe
digit	数字
letter	字符
hexdigit	十六进制字符
exponent	十指数幂后缀
power	二指数幂后缀
delimiter	除\n 外分界符
newline	\n
whitespace	连续空白字符
intsymbol	整数后缀
floatsymbol	浮点数后缀

表 2: 声明类型

4.2 翻译规则

```
<<EOF>>             { return 0; }
```

- 设置 EOF 范围值为 0

```
{newline}           { ++lineCount; ++charCount; column = 0 ; }
```

- 对于单独出现的换行符\n, 行数加一, 字符数加一, 当前列重置 0

```
"auto"              { handle("Reserved word"); return(AUTO); }
"break"             { handle("Reserved word"); return(BREAK); }
"case"              { handle("Reserved word"); return(CASE); }
"char"              { handle("Reserved word"); return(CHAR); }
"const"             { handle("Reserved word"); return(CONST); }
"continue"          { handle("Reserved word"); return(CONTINUE); }
"default"           { handle("Reserved word"); return(DEFAULT); }
"do"                { handle("Reserved word"); return(DO); }
"double"            { handle("Reserved word"); return(DOUBLE); }
"else"              { handle("Reserved word"); return(ELSE); }
"enum"              { handle("Reserved word"); return(ENUM); }
"extern"            { handle("Reserved word"); return(EXTERN); }
```

```
"float"             { handle("Reserved word"); return(FLOAT); }
"for"               { handle("Reserved word"); return(FOR); }
"goto"              { handle("Reserved word"); return(GOTO); }
"if"                { handle("Reserved word"); return(IF); }
"inline"            { handle("Reserved word"); return(INLINE); }
"int"               { handle("Reserved word"); return(INT); }
"long"              { handle("Reserved word"); return(LONG); }
"register"          { handle("Reserved word"); return(REGISTER); }
"restrict"          { handle("Reserved word"); return(RESTRICT); }
"return"            { handle("Reserved word"); return(RETURN); }
"short"             { handle("Reserved word"); return(SHORT); }
"signed"            { handle("Reserved word"); return(SIGNED); }
"sizeof"            { handle("Reserved word"); return(SIZEOF); }
"static"            { handle("Reserved word"); return(STATIC); }
"struct"            { handle("Reserved word"); return(STRUCT); }
"switch"            { handle("Reserved word"); return(SWITCH); }
"typedef"           { handle("Reserved word"); return(TYPEDEF); }
"union"             { handle("Reserved word"); return(UNION); }
"unsigned"          { handle("Reserved word"); return(UNSIGNED); }
"void"              { handle("Reserved word"); return(VOID); }
"volatile"          { handle("Reserved word"); return(VOLATILE); }
"while"             { handle("Reserved word"); return(WHILE); }
```

- 处理 C 保留字, 返回各自类型

```
"/*"                { if( comment_multi() ) return ERROR; return
↪  (COMMENT); }
"//"(\\\n|[^\n])*  { comment_line(); return (COMMENT); }
```

- 分别处理行注释与多行注释
- 行注释仅允许在\后使用\n 换行

```
[1-9]{digit}*{intsymbol}?     { handle("Decimal integer");
↪  return(CONSTANT); }
0[bB][01]+{intsymbol}         { handle("Binary integer");
↪  return(CONSTANT); }
```

```
0[xX]{hexdigit}+{intsymbol}?  { handle("Hexadecimal integer");
↪  return(CONSTANT); }
0[0-7]*{intsymbol}?           { handle("Octonary integer");
↪  return(CONSTANT); }
```

- 分别识别十进制整数、二进制整数、十六进制整数和八进制整数

```
{digit}+{exponent}{floatsymbol}?                    { handle("Decimal
↪  float"); return(CONSTANT); }
{digit}*"."{digit}+{exponent}?{floatsymbol}?        { handle("Decimal
↪  float"); return(CONSTANT); }
{digit}+"."{digit}*{exponent}?{floatsymbol}?        { handle("Decimal
↪  float"); return(CONSTANT); }
0[xX]{hexdigit}+{power}{floatsymbol}?               {
↪  handle("Hexadecimal float"); return(CONSTANT); }
0[xX]{hexdigit}*"."{hexdigit}+{power}{floatsymbol}?  {
↪  handle("Hexadecimal float"); return(CONSTANT); }
0[xX]{hexdigit}+"."{hexdigit}*{power}{floatsymbol}?  {
↪  handle("Hexadecimal float"); return(CONSTANT); }
```

- 识别十进制、十六进制浮点数

```
L?\'(\\n|[^\'\n])+\'  { handle("Character"); return(CONSTANT); }
L?\'(\\n|[^\'\n])+    { charCount += yyleng;
↪  lexicalError("Unterminated character"); return (PASS); }

L?\"(\\n|[^\"\n])*\"  { handle("String"); return(STRING_LITERAL); }
L?\"(\\n|[^\"\n])*    { charCount += yyleng; lexicalError("Unterminated
↪  string"); return (PASS); }
```

- 处理字符、字符串
- 对于无终结符的字符、字符串, 以\n 视为结尾, 并跳过该错误继续向后执行

```
{letter}({letter}|{digit})*  { handle("Identifier"); return(VARIANT); }
{digit}({letter}|{digit})+   { charCount += yyleng; lexicalError("Bad
↪  identifier format"); return (PASS); }
```

- 识别表示符
- 将数字开头且不被整数、浮点数格式匹配的标识符认为是错误格式并跳过

```
"..."          { handle("Operator"); return(ELLIPSIS); }
">>="          { handle("Operator"); return(RIGHT_ASSIGN); }
"<<="          { handle("Operator"); return(LEFT_ASSIGN); }
"+="           { handle("Operator"); return(ADD_ASSIGN); }
"-="           { handle("Operator"); return(SUB_ASSIGN); }
"*="           { handle("Operator"); return(MUL_ASSIGN); }
"/="           { handle("Operator"); return(DIV_ASSIGN); }
"%="           { handle("Operator"); return(MOD_ASSIGN); }
"&="           { handle("Operator"); return(AND_ASSIGN); }
"^="           { handle("Operator"); return(XOR_ASSIGN); }
"|="           { handle("Operator"); return(OR_ASSIGN); }
">>"           { handle("Operator"); return(RIGHT_OP); }
"<<"           { handle("Operator"); return(LEFT_OP); }
"++"           { handle("Operator"); return(INC_OP); }
"--"           { handle("Operator"); return(DEC_OP); }
"->"           { handle("Operator"); return(PTR_OP); }
"&&"           { handle("Operator"); return(AND_OP); }
"||"           { handle("Operator"); return(OR_OP); }
"<="           { handle("Operator"); return(LE_OP); }
">="           { handle("Operator"); return(GE_OP); }
"=="           { handle("Operator"); return(EQ_OP); }
"!="           { handle("Operator"); return(NE_OP); }
";"            { handle("Operator"); return(';'); }
("{"|"<%")     { handle("Operator"); return('{'); }
("}"|"%>")     { handle("Operator"); return('}'); }
","            { handle("Operator"); return(','); }
":"            { handle("Operator"); return(':'); }
"="            { handle("Operator"); return('='); }
"("            { handle("Operator"); return('('); }
```

```
")"             { handle("Operator"); return(')'); }
("["|"<:")      { handle("Operator"); return('['); }
("]"|":>")      { handle("Operator"); return(']'); }
"."             { handle("Operator"); return('.'); }
"&"             { handle("Operator"); return('&'); }
"!"             { handle("Operator"); return('!'); }
"~"             { handle("Operator"); return('~'); }
"-"             { handle("Operator"); return('-'); }
"+"             { handle("Operator"); return('+'); }
"*"             { handle("Operator"); return('*'); }
"/"             { handle("Operator"); return('/'); }
"%"             { handle("Operator"); return('%'); }
"<"             { handle("Operator"); return('<'); }
">"             { handle("Operator"); return('>'); }
"^"             { handle("Operator"); return('^'); }
"|"             { handle("Operator"); return('|'); }
"?"             { handle("Operator"); return('?'); }
```

- 处理所有运算符

```
{whitespace}    { charCount += yyleng; column += yyleng; }
```

- 处理空白字符

```
. { lexicalError("Unknown symbol"); return (PASS); }
```

- 将未能被以上格式匹配的字符认为是不可识别字符并跳过该错误

4.3 辅助过程

```
int yywrap(){
    return 1;
}
```

- 自定义 yywrap

```
void lexicalError(std::string errorif){
    fprintf(stderr, "Error: %s, in line %d, column %d:\n%s\n\n",
    ↪  errorif.c_str(), lineCount, column, yytext);
}
```

- 从 stderr 输出词法错误信息

```
bool comment_multi(){
  int st_lino = lineCount, st_cono = column;
  fprintf(stdout, "Multi-line comment at line: %d, column: %d\n",
  ↪  lineCount, column);
  char c, pre = 0;
  fprintf(stdout, "/*");
  charCount += 2;
  while ( (c = yyinput()) != 0 ){
    fprintf(stdout, "%c", c);
    ++charCount;
    if (c == '/' && pre == '*'){
      fprintf(stdout, "\n\n");
      return 0;
    }
    else if( c=='\n' ){
        ++lineCount;
    }
    pre = c;
  }
  fprintf(stderr, "\nUnterminated comment starting at line: %d, column:
  ↪  %d\n\n", st_lino, st_cono);
  return ERROR;
}
```

- 在多行注释中处理行数、字符数信息
- 处理没有终结符的多行注释错误

```
bool comment_line(){
     fprintf(stdout, "Line comment at line %d:\n %s\n\n", lineCount,
     ↪  yytext);
     lineCount += std::count(yytext, yytext + yyleng, '\n');
     charCount += yyleng;
     return 0;
}
```

- 处理行注释

```
void handle(std::string s){
    fprintf(stdout, "%s at line %d, column %d:\n%s\n\n", s.c_str(),
    ↪  lineCount, column, yytext);
    if( s == "Identifier" || s == "Reserved word" || s == "Character" ||
    ↪  s == "String" ){
        wordCount += 1;
    }
    lineCount += std::count(yytext, yytext + yyleng, '\n');
    charCount += yyleng;
    for (int i = 0; yytext[i] != '\0'; i++){
        if (yytext[i] == '\t'){
            column += 4 - (column % 4);
        }
        else{
            ++column;
        }
    }
}
```

- 处理一般保留字、标识符、字符串等

```
int main(){
    int res, cnt=0;
    while( (res=yylex())>0 ) ;
    fprintf(stdout, "\n# The number of lines = %d, of words = %d, of
    ↪  chars = %d ", lineCount-1, wordCount, charCount);
```

```
    return 0;
}
```

- 一直从输入流读取字符, 直到文件结尾或出现不可修复错误
- 最后输出统计信息

5 样例测试

5.1 测试用例

test.c

```c
// #include<stdio.h>

int main()
{
  "test0
  char s1[] = "hello\
world";// test1\
       test2
  char c = '\0337';// test3
  int 03f;
  // printf("%s\n", s1);
  printf("%c\n", c);
  int a = 0b111111u;
  double b = 12.88e15l;
  printf("%d", a);
  /* test */
  return 0;
}

/*
int main()<%
  int a[5] = {1,2,3};
int b = a<:2:>;
// cout << a<:2:>;
```

```
//_Imaginary a;
%>
```

5.2 运行分析程序

在 **terminal** 中执行

```
flex lexical.l && g++ lex.yy.c -g -o lyser -lfl
g++ lex.yy.c -o lyser
./lyser < test.c 1> out.txt 2> err.txt
```

5.3 分析结果

分别得到标准输出和错误信息输出

5.3.1 标准输出

```
Line comment at line 1:
 // #include<stdio.h>

Reserved word at line 3, column 0:
int

Identifier at line 3, column 4:
main

Operator at line 3, column 8:
(

Operator at line 3, column 9:
)

Operator at line 4, column 0:
{

Reserved word at line 6, column 2:
```

```
char

Identifier at line 6, column 7:
s1

Operator at line 6, column 9:
[

Operator at line 6, column 10:
]

Operator at line 6, column 12:
=

String at line 6, column 14:
"hello\
world"

Operator at line 7, column 28:
;

Line comment at line 7:
 // test1\
       test2

Reserved word at line 9, column 2:
char

Identifier at line 9, column 7:
c

Operator at line 9, column 9:
=

Character at line 9, column 11:
'\0337'
```

```
56
57 Operator at line 9, column 18:
58 ;
59
60 Line comment at line 9:
61  // test3
62
63 Reserved word at line 10, column 2:
64 int
65
66 Operator at line 10, column 6:
67 ;
68
69 Line comment at line 11:
70  // printf("%s\n", s1);
71
72 Identifier at line 12, column 2:
73 printf
74
75 Operator at line 12, column 8:
76 (
77
78 String at line 12, column 9:
79 "%c\n"
80
81 Operator at line 12, column 15:
82 ,
83
84 Identifier at line 12, column 17:
85 c
86
87 Operator at line 12, column 18:
88 )
89
90 Operator at line 12, column 19:
91 ;
```

```
92
93  Reserved word at line 13, column 2:
94  int
95
96  Identifier at line 13, column 6:
97  a
98
99  Operator at line 13, column 8:
100 =
101
102 Binary integer at line 13, column 10:
103 0b111111u
104
105 Operator at line 13, column 19:
106 ;
107
108 Reserved word at line 14, column 2:
109 double
110
111 Identifier at line 14, column 9:
112 b
113
114 Operator at line 14, column 11:
115 =
116
117 Decimal float at line 14, column 13:
118 12.88e15l
119
120 Operator at line 14, column 22:
121 ;
122
123 Identifier at line 15, column 2:
124 printf
125
126 Operator at line 15, column 8:
127 (
```

```
128
129 String at line 15, column 9:
130 "%d"
131
132 Operator at line 15, column 13:
133 ,
134
135 Identifier at line 15, column 15:
136 a
137
138 Operator at line 15, column 16:
139 )
140
141 Operator at line 15, column 17:
142 ;
143
144 Multi-line comment at line: 16, column: 2
145 /* test */
146
147 Reserved word at line 17, column 2:
148 return
149
150 Octonary integer at line 17, column 9:
151 0
152
153 Operator at line 17, column 10:
154 ;
155
156 Operator at line 18, column 0:
157 }
158
159 Multi-line comment at line: 20, column: 0
160 /*
161 int main()<%
162   int a[5] = {1,2,3};
163 int b = a<:2:>;
```

```
// cout << a<:2:>;
//_Imaginary a;
%>

# The number of lines = 26, of words = 20, of chars = 365
```

识别结果良好.

5.3.2 错误输出

```
Error: Unterminated string, in line 5, column 2:
"test0

Error: Bad identifier format, in line 10, column 6:
03f


Unterminated comment starting at line: 20, column: 0
```

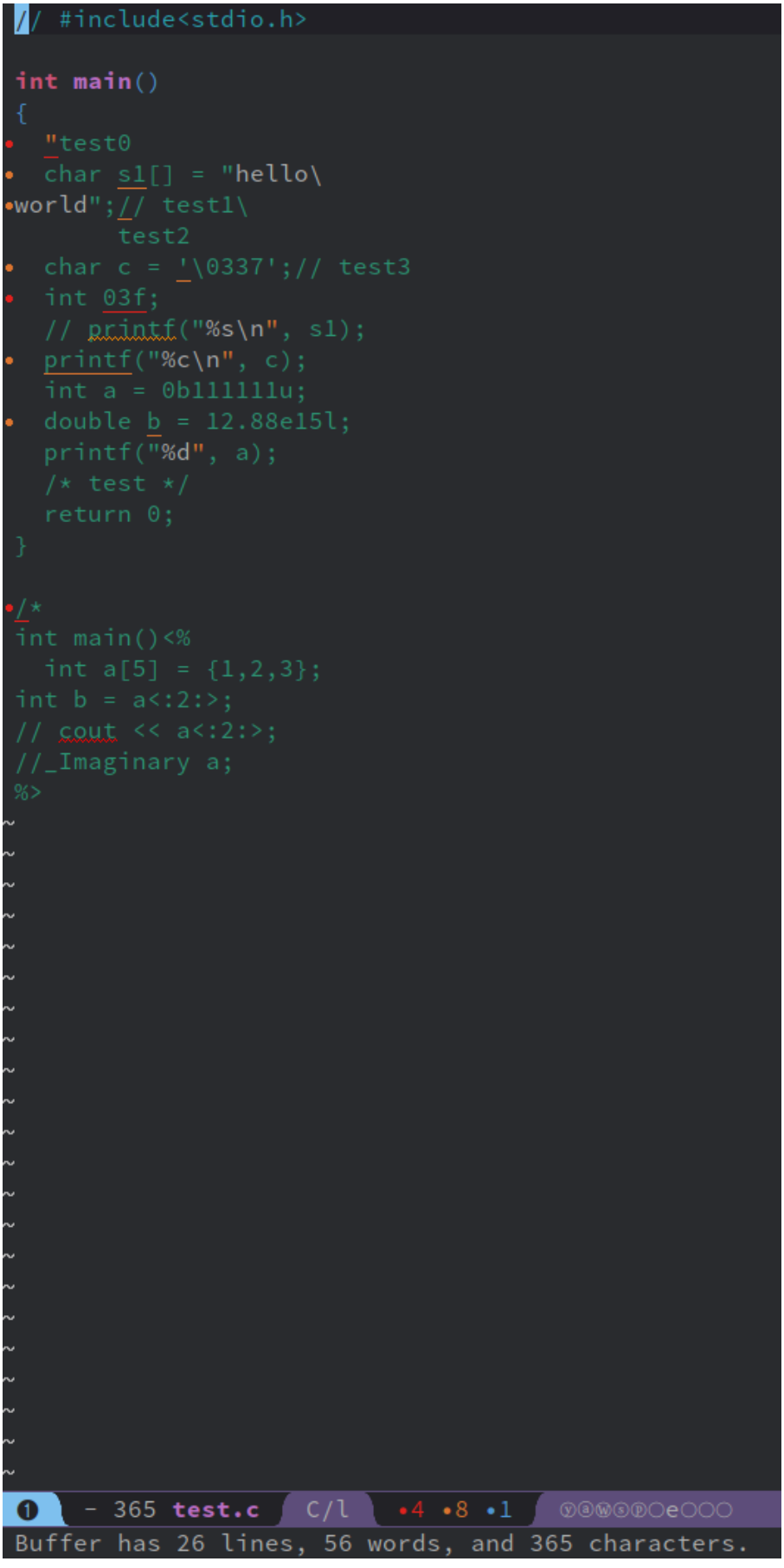
// #include<stdio.h>
int main()
{
"test0
char s1[] = "hello\
world";// test1\
test2
char c = '\0337';// test3
int 03f;
// printf("%s\n", s1);
printf("%c\n", c);
int a = 0b111111u;
double b = 12.88e15l;
printf("%d", a);
/* test */
return 0;
}
/*
int main()<%
int a[5] = {1,2,3};
int b = a<:2:>;
// cout << a<:2:>;
//_Imaginary a;
%>
1 - 365 test.c C/l •4 •8 •1
Buffer has 26 lines, 56 words, and 365 characters.

图 2: 统计结果

除去单词数为统计注释中内容外, 行数和字符数与编辑器统计结果吻合极好.

6 实验总结

学会了 LEX 词法分析器的使用方法, 熟练掌握了正则表达式的使用技巧, 对 C 语言词法、词法有了更深入理解.

7 Appendix

7.1 lexical.l

```
%{
#include <bits/stdc++.h>
#include "identifier.h"
int wordCount = 0;
int lineCount = 1;
int charCount = 0;
int column = 0;
bool comment_multi();
bool comment_line();
void handle(std::string s="");
void lexicalError(std::string);
%}

digit               [0-9]
letter              [a-zA-Z_]
hexdigit            [a-fA-F0-9]
exponent            ([Ee][+-]?{digit}+)
power               ([Pp][+-]?{digit}+)
delimiter           [ \t\r\v\f]
newline             \n
whitespace          {delimiter}+
intsymbol           ((u|U)|(u|U)?(l|L|ll|LL)|(l|L|ll|LL)(u|U))
floatsymbol         (f|F|l|L)

%%
<<EOF>>             { return 0; }

{newline}           { ++lineCount; ++charCount ;column = 0 ; }
"/*"                { if( comment_multi() ) return ERROR; return
 ↪  (COMMENT); }
"//"(\\\n|[^\n])*   { comment_line(); return (COMMENT); }
"auto"              { handle("Reserved word"); return(AUTO); }
```

```
32 "break"            { handle("Reserved word"); return(BREAK); }
33 "case"             { handle("Reserved word"); return(CASE); }
34 "char"             { handle("Reserved word"); return(CHAR); }
35 "const"            { handle("Reserved word"); return(CONST); }
36 "continue"         { handle("Reserved word"); return(CONTINUE); }
37 "default"          { handle("Reserved word"); return(DEFAULT); }
38 "do"               { handle("Reserved word"); return(DO); }
39 "double"           { handle("Reserved word"); return(DOUBLE); }
40 "else"             { handle("Reserved word"); return(ELSE); }
41 "enum"             { handle("Reserved word"); return(ENUM); }
42 "extern"           { handle("Reserved word"); return(EXTERN); }
43 "float"            { handle("Reserved word"); return(FLOAT); }
44 "for"              { handle("Reserved word"); return(FOR); }
45 "goto"             { handle("Reserved word"); return(GOTO); }
46 "if"               { handle("Reserved word"); return(IF); }
47 "inline"           { handle("Reserved word"); return(INLINE); }
48 "int"              { handle("Reserved word"); return(INT); }
49 "long"             { handle("Reserved word"); return(LONG); }
50 "register"         { handle("Reserved word"); return(REGISTER); }
51 "restrict"         { handle("Reserved word"); return(RESTRICT); }
52 "return"           { handle("Reserved word"); return(RETURN); }
53 "short"            { handle("Reserved word"); return(SHORT); }
54 "signed"           { handle("Reserved word"); return(SIGNED); }
55 "sizeof"           { handle("Reserved word"); return(SIZEOF); }
56 "static"           { handle("Reserved word"); return(STATIC); }
57 "struct"           { handle("Reserved word"); return(STRUCT); }
58 "switch"           { handle("Reserved word"); return(SWITCH); }
59 "typedef"          { handle("Reserved word"); return(TYPEDEF); }
60 "union"            { handle("Reserved word"); return(UNION); }
61 "unsigned"         { handle("Reserved word"); return(UNSIGNED); }
62 "void"             { handle("Reserved word"); return(VOID); }
63 "volatile"         { handle("Reserved word"); return(VOLATILE); }
64 "while"            { handle("Reserved word"); return(WHILE); }
65
66 [1-9]{digit}*{intsymbol}?     { handle("Decimal integer");
   ↪  return(CONSTANT); }
```

```
0[bB][01]+{intsymbol}          { handle("Binary integer");
↪  return(CONSTANT); }
0[xX]{hexdigit}+{intsymbol}?   { handle("Hexadecimal integer");
↪  return(CONSTANT); }
0[0-7]*{intsymbol}?            { handle("Octonary integer");
↪  return(CONSTANT); }


{digit}+{exponent}{floatsymbol}?                    { handle("Decimal
↪  float"); return(CONSTANT); }
{digit}*"."{digit}+{exponent}?{floatsymbol}?        { handle("Decimal
↪  float"); return(CONSTANT); }
{digit}+"."{digit}*{exponent}?{floatsymbol}?        { handle("Decimal
↪  float"); return(CONSTANT); }
0[xX]{hexdigit}+{power}{floatsymbol}?               {
↪  handle("Hexadecimal float"); return(CONSTANT); }
0[xX]{hexdigit}*"."{hexdigit}+{power}{floatsymbol}?  {
↪  handle("Hexadecimal float"); return(CONSTANT); }
0[xX]{hexdigit}+"."{hexdigit}*{power}{floatsymbol}?  {
↪  handle("Hexadecimal float"); return(CONSTANT); }

L?\'(\\n|[^\'\n])+\'  { handle("Character"); return(CONSTANT); }
L?\'(\\n|[^\'\n])+    { charCount += yyleng;
↪  lexicalError("Unterminated character"); return (PASS); }

L?\"(\\n|[^\"\n])*\"  { handle("String"); return(STRING_LITERAL); }
L?\"(\\n|[^\"\n])*    { charCount += yyleng; lexicalError("Unterminated
↪  string"); return (PASS); }

{letter}({letter}|{digit})*  { handle("Identifier"); return(VARIANT); }
{digit}({letter}|{digit})+   { charCount += yyleng; lexicalError("Bad
↪  identifier format"); return (PASS); }

"..."          { handle("Operator"); return(ELLIPSIS); }
">>="          { handle("Operator"); return(RIGHT_ASSIGN); }
"<<="          { handle("Operator"); return(LEFT_ASSIGN); }
```

```
91  "+="            { handle("Operator"); return(ADD_ASSIGN); }
92  "-="            { handle("Operator"); return(SUB_ASSIGN); }
93  "*="            { handle("Operator"); return(MUL_ASSIGN); }
94  "/="            { handle("Operator"); return(DIV_ASSIGN); }
95  "%="            { handle("Operator"); return(MOD_ASSIGN); }
96  "&="            { handle("Operator"); return(AND_ASSIGN); }
97  "^="            { handle("Operator"); return(XOR_ASSIGN); }
98  "|="            { handle("Operator"); return(OR_ASSIGN); }
99  ">>"            { handle("Operator"); return(RIGHT_OP); }
100 "<<"            { handle("Operator"); return(LEFT_OP); }
101 "++"            { handle("Operator"); return(INC_OP); }
102 "--"            { handle("Operator"); return(DEC_OP); }
103 "->"            { handle("Operator"); return(PTR_OP); }
104 "&&"            { handle("Operator"); return(AND_OP); }
105 "||"            { handle("Operator"); return(OR_OP); }
106 "<="            { handle("Operator"); return(LE_OP); }
107 ">="            { handle("Operator"); return(GE_OP); }
108 "=="            { handle("Operator"); return(EQ_OP); }
109 "!="            { handle("Operator"); return(NE_OP); }
110 ";"             { handle("Operator"); return(';'); }
111 ("{"|"<%")      { handle("Operator"); return('{'); }
112 ("}"|"%>")      { handle("Operator"); return('}'); }
113 ","             { handle("Operator"); return(','); }
114 ":"             { handle("Operator"); return(':'); }
115 "="             { handle("Operator"); return('='); }
116 "("             { handle("Operator"); return('('); }
117 ")"             { handle("Operator"); return(')'); }
118 ("["|"<: ")     { handle("Operator"); return('['); }
119 ("]"|":  >")    { handle("Operator"); return(']'); }
120 "."             { handle("Operator"); return('.'); }
121 "&"             { handle("Operator"); return('&'); }
122 "!"             { handle("Operator"); return('!'); }
123 "~"             { handle("Operator"); return('~'); }
124 "-"             { handle("Operator"); return('-'); }
125 "+"             { handle("Operator"); return('+'); }
126 "*"             { handle("Operator"); return('*'); }
```

```
"/"             { handle("Operator"); return('/'); }
"%"             { handle("Operator"); return('%'); }
"<"             { handle("Operator"); return('<'); }
">"             { handle("Operator"); return('>'); }
"^"             { handle("Operator"); return('^'); }
"|"             { handle("Operator"); return('|'); }
"?"             { handle("Operator"); return('?'); }

{whitespace}    { charCount += yyleng; column += yyleng; }

. { lexicalError("Unknown symbol"); return (PASS); }


%%

int yywrap(){
    return 1;
}

void lexicalError(std::string errorif){
    fprintf(stderr, "Error: %s, in line %d, column %d:\n%s\n\n",
    ↪  errorif.c_str(), lineCount, column, yytext);
}

bool comment_multi(){
  int st_lino = lineCount, st_cono = column;
  fprintf(stdout, "Multi-line comment at line: %d, column: %d\n",
  ↪  lineCount, column);
  char c, pre = 0;
  fprintf(stdout, "/*");
  charCount += 2;
  while ( (c = yyinput()) != 0 ){
    fprintf(stdout, "%c", c);
    ++charCount;
    if (c == '/' && pre == '*'){
      fprintf(stdout, "\n\n");
```

```
161       return 0;
162     }
163     else if( c=='\n' ){
164         ++lineCount;
165     }
166     pre = c;
167   }
168   fprintf(stderr, "\nUnterminated comment starting at line: %d, column:
      %d\n\n", st_lino, st_cono);
169   return ERROR;
170 }
171
172 bool comment_line(){
173      fprintf(stdout, "Line comment at line %d:\n %s\n\n", lineCount,
       yytext);
174      lineCount += std::count(yytext, yytext + yyleng, '\n');
175      charCount += yyleng;
176      return 0;
177 }
178
179 void handle(std::string s){
180     fprintf(stdout, "%s at line %d, column %d:\n%s\n\n", s.c_str(),
      lineCount, column, yytext);
181     if( s == "Identifier" || s == "Reserved word" || s == "Character" ||
      s == "String" ){
182         wordCount += 1;
183     }
184     lineCount += std::count(yytext, yytext + yyleng, '\n');
185     charCount += yyleng;
186     for (int i = 0; yytext[i] != '\0'; i++){
187         if (yytext[i] == '\t'){
188             column += 4 - (column % 4);
189         }
190         else{
191             ++column;
192         }
```

```
193     }
194 }
195
196 int main(){
197     int res, cnt=0;
198     while( (res=yylex())>0 ) ;
199     fprintf(stdout, "\n# The number of lines = %d, of words = %d, of
        ↪  chars = %d ", lineCount-1, wordCount, charCount);
200     return 0;
201 }
```

7.2 identifier.h

```
1  #define ERROR      -1
2  #define END        0
3  #define AUTO       1
4  #define BREAK      2
5  #define CASE       3
6  #define CHAR       4
7  #define CONST      5
8  #define CONTINUE   6
9  #define DEFAULT    7
10 #define DO         8
11 #define DOUBLE     9
12 #define ELSE       10
13 #define ENUM       11
14 #define EXTERN     12
15 #define FLOAT      13
16 #define FOR        14
17 #define GOTO       15
18 #define IF         16
19 #define INLINE     17
20 #define INT        18
21 #define LONG       19
22 #define REGISTER   20
```

```
23 #define RESTRICT   21
24 #define RETURN     22
25 #define SHORT      23
26 #define SIGNED     24
27 #define SIZEOF     25
28 #define STATIC     26
29 #define STRUCT     27
30 #define SWITCH     28
31 #define TYPEDEF    29
32 #define UNION      30
33 #define UNSIGNED   31
34 #define VOID       32
35 #define VOLATILE   33
36 #define WHILE      34
37 #define CONSTANT   35
38 #define STRING_LITERAL 36
39 #define ELLIPSIS   37
40 #define RIGHT_ASSIGN 38
41 #define LEFT_ASSIGN 39
42 #define ADD_ASSIGN 40
43 #define SUB_ASSIGN 41
44 #define MUL_ASSIGN 42
45 #define DIV_ASSIGN 43
46 #define MOD_ASSIGN 44
47 #define AND_ASSIGN 45
48 #define XOR_ASSIGN 46
49 #define OR_ASSIGN  47
50 #define RIGHT_OP   48
51 #define LEFT_OP    49
52 #define INC_OP     50
53 #define DEC_OP     51
54 #define PTR_OP     52
55 #define AND_OP     53
56 #define OR_OP      54
57 #define LE_OP      55
58 #define GE_OP      56
```

```
59 #define EQ_OP       57
60 #define NE_OP       58
61 #define VARIANT     59
62 #define COMMENT     60
63 #define PASS        61
```