

# 编译原理实验报告

## 词法分析程序的设计与实现

班级：-----

学号：-----

姓名：-----

# 目录

一、 实验内容和要求 .....	3
二、 实验分析 .....	3
三、 Lex 源代码.....	4
四、 实验测试 .....	6
五、 实验总结 .....	11

## 一、实验内容和要求

**题目** 词法分析程序的设计与实现。

**实验内容** 设计并实现 C 语言的词法分析程序，要求如下：

- 1) 可以识别出用 C 语言编写的源程序中的每个单词符号，并以记号的形式输出每个单词符号。
- 2) 可以识别并读取源程序中的注释。
- 3) 可以统计源程序汇总的语句行数、单词个数和字符个数，其中标点和空格不计算为单词，并输出统计结果
- 4) 检查源程序中存在的错误，并可以报告错误所在的行列位置。
- 5) 发现源程序中存在的错误后，进行适当的恢复，使词法分析可以继续进行，通过一次词法分析处理，可以检查并报告源程序中存在的所有错误。

**实验要求** 方法 1：采用 C/C++ 作为实现语言，手工编写词法分析程序。

方法 2：通过编写 LEX 源程序，利用 LEX 软件工具自动生成词法分析程序。

## 二、实验分析

词法分析的任务在于将字符序列识别为词法所描述的各种记号。通过将标识符以正则定义式的形式进行描述，可以用词法分析器去识别字符序列中的标识符。

语言所定义的记号共有以下几种：

- 1) 标识符：以字母开头的，后跟字母和数字组成的符号串
- 2) 关键字：标识符的子集，语言的保留关键字
- 3) 无符号数：数字和小数点组成的符号串
- 4) 运算符：+、\*、\、()、{}、[]、= 以及 == 等等符号
- 5) 注释标记：分为单行注释和多行注释

**输出形式** 由于本次实验没有语法分析部分，因此将词法分析的结果直接输出至屏幕上，

**全局变量的设计** 通过全局变量来计算字符数、单词数和行数，同时鉴于标识符将会被加入符号表，因此通过对标识符计数来模拟标识符被加入符号表。

同时 lex 中可以设计全局函数，通过 main 函数，将字符数、单词数和行数的统计结果输出。

Lex 在处理二义性问题上遵循两个原则：最长匹配原则和优先匹配原则。通过优先匹配原则，可以在匹配标识符的基础上匹配关键字。通过最长匹配原则，可以识别字符串（String 类型）而不会将其中的内容识别为标识符等等。

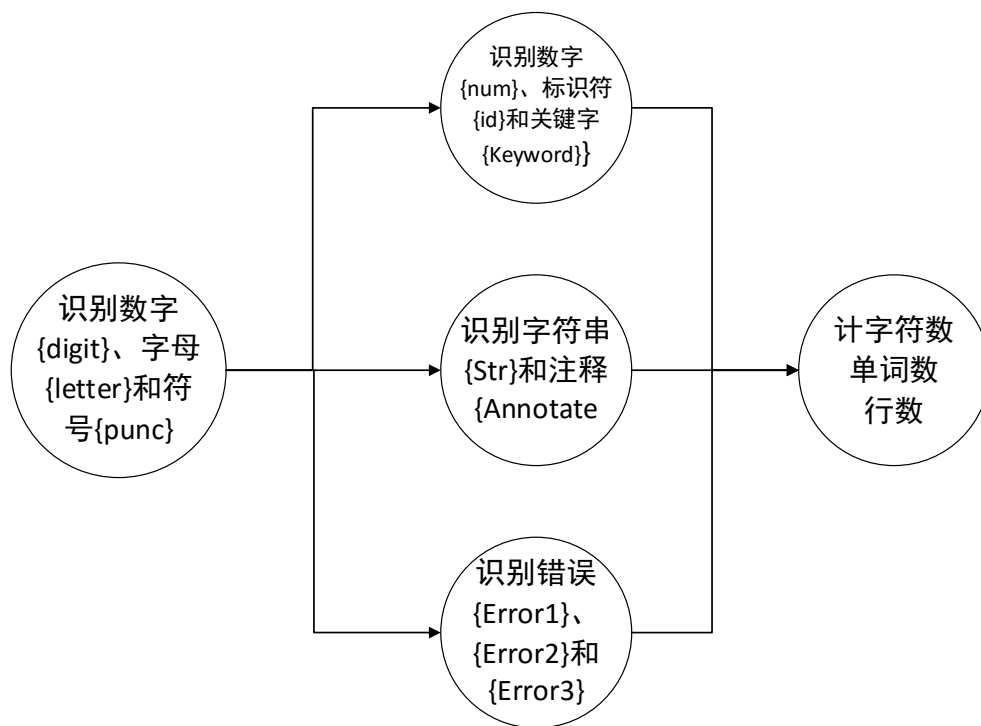


图 1

### 三、Lex 源代码

```
%{
#include <stdio.h>

int wordCount=0;           //单词数
int charCount=0;           //字符数
int columnCount=0;         //当前行的字符数
int lineCount=1;           //行数
int IDCCount=0;            //标识符数
int stringCount=0;         //字符串数

%}

delim      [' '\t]+
wrap       [\n]
notwrap    [^\n]
ws         {delim}+
letter     [A-Za-z]
digit      [0-9]
notpunc    [^"]
keyword    char|short|int|unsigned|long|float|double|struct
           |union|void|enum|signed|const|volatile|typedef|auto|register
           |static|extern|break|case|continue|default|do|
```

```

else|for|goto|if|return|switch|while|sizeof
punc    \+|\-|\*|\\|\&|\(|\)|\{|\}|\[|\]||=|==|\||\^|\?|\.|\\,
id       {letter}({letter}|{digit})*
num      {digit}+(\.{digit}+)?
str      \"{notwrap}*\"
linecomment  \/\{notwrap}*
error1   {digit}+{letter}+({letter}|{digit})*
error2   \"{notpunc}*
error3   @|~|`
%x       IN_COMMENT

```

/\*wrap 指换行, notwrap 指除了换行以外的其他字符, ws 是指大段空格, punc 指标点符号, linecomment 指行注释。error1 指数字开头的标识符, error2 指只有左引号, 没有右引号, error3 指非法字符例如@, ~。%x 定义了 IN\_COMMENT 函数。\*/

```

%%
{delim}      {columnCount+=yyleng;charCount+=yyleng;}
{keyword}    {printf("KEYWORD: %s\n",yytext);
              columnCount+=yyleng;charCount+=yyleng;wordCount++;}
{id}         {IDCount++;printf("ID%d: %s\n",IDCount,yytext);
              columnCount+=yyleng;charCount+=yyleng;wordCount++;}
{num}        {printf("NUM: %s\n",yytext);columnCount+=yyleng;
              charCount+=yyleng;}
{linecomment} {printf("LINENOTE: %s\n",yytext);}
{str}        {stringCount++;printf("STRING%d: %s\n",
              stringCount,yytext);columnCount+=yyleng;charCount+=
              =yyleng;}
{punc}       {printf("PUNCTUATION: %s\n",yytext);
              columnCount++;charCount++;}
{error1}     {printf("ERROR: Line %d, Column %d:
              Identity cannot start with number.\"%s\"\\n",
              lineCount,columnCount+1,yytext);columnCount+=yylen
              g;charCount+=yyleng;}
{error3}     {printf("ERROR: Line %d, Column %d: Invalid
              Symbol.\"%s\"\\n",lineCount,columnCount+1,yytext);c
              olumnCount+=yyleng;charCount+=yyleng;}

<INITIAL>{
"/*"        BEGIN(IN_COMMENT);yymore();
}
<IN_COMMENT>{
"/*"        printf("ANNODATE: %s\n",yytext);BEGIN(INITIAL);
[^*\n]+     yyomore();
"*"         yyomore();
}

```

```

\n                lineCount++;yymore();
}

{error2}          {printf("ERROR:Line %d, Column %d: Cannot Match the
                    Left Puncuation.\n",lineCount,columnCount+1);
                    columnCount+=yyleng;charCount+=yyleng;}
{wrap}            {lineCount++;columnCount=0;}
.                 {printf("%s",yytext);charCount++;columnCount++;}

%%

int main(){
    yylex();
    printf("\nFinished.\nResult:\n");
    printf("Chars: %d \n",charCount);
    printf("Lines: %d \n",lineCount);
    printf("Words: %d \n",wordCount);
    return 0;
}

int yywrap(){
    return 1;
}

```

## 四、实验测试

### 测试 1

```

int main(void)
{
    int a=0;
    if(a<0)
    {
        printf("this"); //print String
    }
    else{
        continue;
    }
    float b=5.6;
    b=b*a;
    /*example */
    return 0;
}

```

## 测试结果

```
KEYWORD: int
ID1: main
PUNCTUATION: (
KEYWORD: void
PUNCTUATION: )
PUNCTUATION: {
KEYWORD: int
ID2: a
PUNCTUATION: =
NUM: 0
PUNCTUATION: ;
KEYWORD: if
PUNCTUATION: (
ID3: a
PUNCTUATION: <
NUM: 0
PUNCTUATION: )
PUNCTUATION: {
ID4: printf
PUNCTUATION: (
STRING1: "this"
PUNCTUATION: )
PUNCTUATION: ;
LINENOTE: //print String
PUNCTUATION: }
KEYWORD: else
PUNCTUATION: {
KEYWORD: continue
PUNCTUATION: ;
PUNCTUATION: }
KEYWORD: float
ID5: b
PUNCTUATION: =
NUM: 5.6
PUNCTUATION: ;
ID6: b
PUNCTUATION: =
ID7: b
PUNCTUATION: *
ID8: a
PUNCTUATION: ;
ANNODATE: /*example */
```

KEYWORD: return  
NUM: 0  
PUNCTUATION: ;  
PUNCTUATION: }

Finished.

Result:

Chars: 120

Lines: 17

Words: 16

Ca. C:\WINDOWS\system32\cmd.exe

```
D:\>lex.yy < abc.txt
KEYWORD: int
ID1: main
PUNCTUATION: <
KEYWORD: void
PUNCTUATION: >
PUNCTUATION: <
KEYWORD: int
ID2: a
PUNCTUATION: =
NUM: 0
PUNCTUATION: ;
KEYWORD: if
PUNCTUATION: <
ID3: a
PUNCTUATION: <
NUM: 0
PUNCTUATION: >
PUNCTUATION: <
ID4: printf
PUNCTUATION: <
STRING1: "this"
PUNCTUATION: >
PUNCTUATION: ;
LINENOTE: //print String
PUNCTUATION: >
KEYWORD: else
PUNCTUATION: <
KEYWORD: continue
PUNCTUATION: ;
PUNCTUATION: >
KEYWORD: float
ID5: b
PUNCTUATION: =
NUM: 5.6
PUNCTUATION: ;
ID6: b
PUNCTUATION: =
ID7: b
PUNCTUATION: *
ID8: a
PUNCTUATION: ;
ANNODATE: /*example */
KEYWORD: return
NUM: 0
```



```
PUNCTUATION: ;
PUNCTUATION: >
STRING2: "this"
ERROR:Line 17, Column 1: Cannot Match the Left Punctuation.

Finished.
Result:
Chars: 120
Lines: 17
Words: 16
```

图 2 测试 1 的输出结果

测试 1 旨在测试标识符的识别和字符统计，测试 2 将测试错误识别。设计的 Lex 程序共可以识别三种错误，错误 1 是以数字为开头的标识符，错误 2 是无法配对的左引号，错误 3 是检查非法符号。

## 测试 2

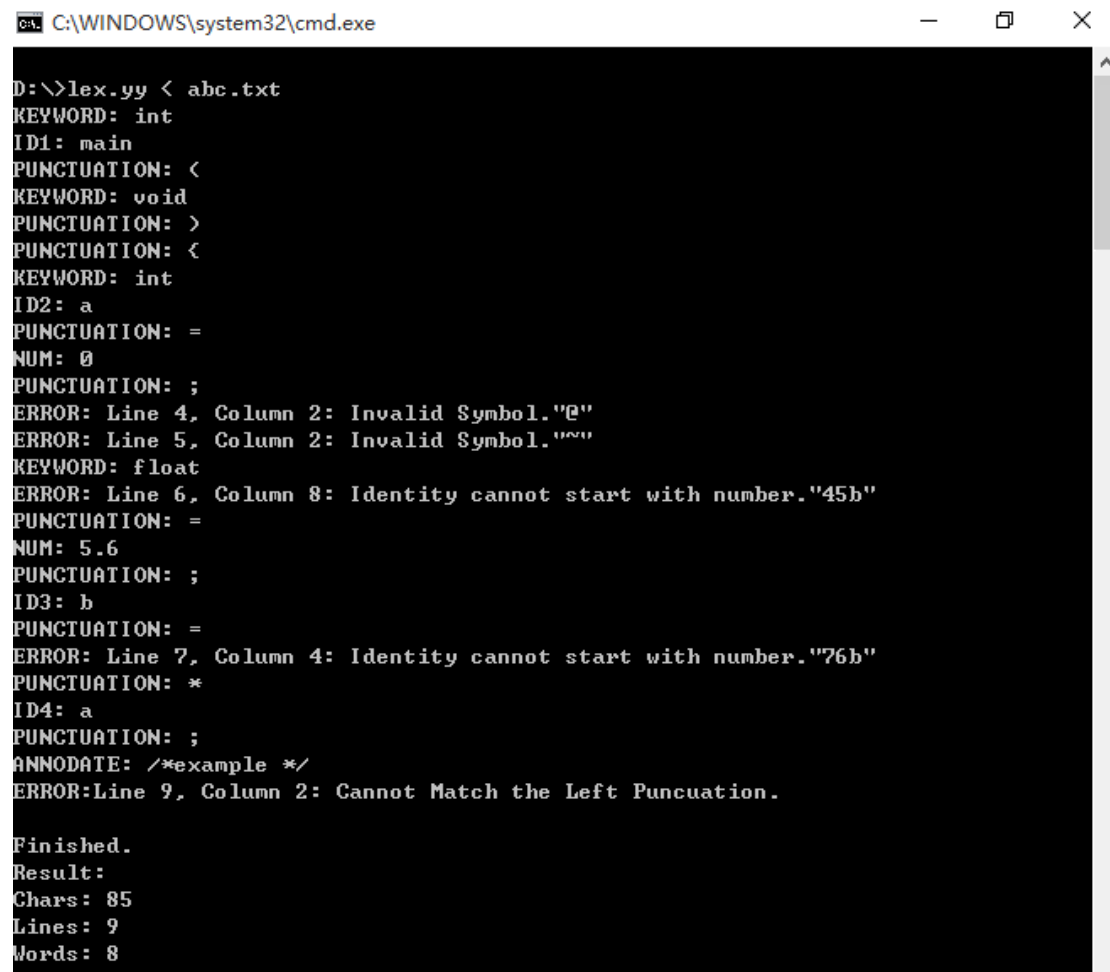
```
int main(void)
{
    int a=0;
    @
    ~
    float 45b=5.6;
    b=76b*a;
    /*example */
    "Only Left Punc
    return 0;
}
```

## 测试结果

```
KEYWORD: int
ID1: main
PUNCTUATION: (
KEYWORD: void
PUNCTUATION: )
PUNCTUATION: {
KEYWORD: int
ID2: a
PUNCTUATION: =
NUM: 0
PUNCTUATION: ;
ERROR: Line 4, Column 2: Invalid Symbol."@"
ERROR: Line 5, Column 2: Invalid Symbol."~"
KEYWORD: float
ERROR: Line 6, Column 8: Identity cannot start with number."45b"
```

```
PUNCTUATION: =
NUM: 5.6
PUNCTUATION: ;
ID3: b
PUNCTUATION: =
ERROR: Line 7, Column 4: Identity cannot start with number."76b"
PUNCTUATION: *
ID4: a
PUNCTUATION: ;
ANNODATE: /*example */
ERROR:Line 9, Column 2: Cannot Match the Left Punctuation.

Finished.
Result:
Chars: 85
Lines: 9
Words: 8
```



```
C:\WINDOWS\system32\cmd.exe

D:\>lex.yy < abc.txt
KEYWORD: int
ID1: main
PUNCTUATION: (
KEYWORD: void
PUNCTUATION: )
PUNCTUATION: {
KEYWORD: int
ID2: a
PUNCTUATION: =
NUM: 0
PUNCTUATION: ;
ERROR: Line 4, Column 2: Invalid Symbol."@"
ERROR: Line 5, Column 2: Invalid Symbol."^"
KEYWORD: float
ERROR: Line 6, Column 8: Identity cannot start with number."45b"
PUNCTUATION: =
NUM: 5.6
PUNCTUATION: ;
ID3: b
PUNCTUATION: =
ERROR: Line 7, Column 4: Identity cannot start with number."76b"
PUNCTUATION: *
ID4: a
PUNCTUATION: ;
ANNODATE: /*example */
ERROR:Line 9, Column 2: Cannot Match the Left Punctuation.

Finished.
Result:
Chars: 85
Lines: 9
Words: 8
```

图3 测试2的输出结果

可以看出,设计的 Lex 程序可以有效地识别标识符、关键字、标点符号、数字、字符串、行注释和块注释等等,能够较为准确地统计出字符序列的字符数、单词数和行数,同时能够识别词法分析阶段常见的三种错误,并指出错误的位置。由于识别注释和字符串时只能将其看作整体,无法分割成单词元计算单词个数,因此单词数是不包含字符串和注释里的单词的。

## 五、实验总结

本次实验主要是对课上学习的词法分析的基本原理进行应用。无论载体是 C 语言还是 Lex,其实质都在于对字符序列进行分析并识别出标识符,进而根据不同的标识符进行相应处理。Lex 相比于 C 语言来说好处在于让我能够投入更多精力去研究如何设计正则表达式以正确处理所有情况,不用考虑在 C 语言中如何实现。通过这次实验设计,我对课上所讲的词法分析方法有了更为深刻的了解。通过查阅帮助和参考资料,我学习到了 Lex 的一些比较实用的用法(例如如何识别 C 语言的注释,以及通过 `yymore()` 函数来输出注释等等),虽然对于 Lex 的了解还只是皮毛,但是对之后的学习会有很大的帮助。