

电子科技大学

计算机专业类课程

实验报告

课程名称：编译原理

学 院：计算机科学与工程学院

专 业：计算机科学与技术

学生姓名：徐贤达

学 号：2016060601018

指导教师：陈昆

评 分：

日 期：2019 年 5 月 9 日

电子科技大学

实验报告

实验一：词法分析器的设计与实现

一. 实验目的

了解和掌握词法分析的方法。编程实现给定源语言程序的词法分析器。并利用该分析器扫描源语言程序的字符串，按照给定的词法规则，识别出单词符号作为输出，发现其中的词法错误。

二. 实验内容

1. 源语言：求 $n!$ 的极小语言

文法如下：

<程序>→<分程序>

<分程序>→begin <说明语句表>; <执行语句表> end

<说明语句表>→<说明语句> | <说明语句表> ; <说明语句>

<说明语句>→<变量说明> | <函数说明>

<变量说明>→integer <变量>

<变量>→<标识符>

<标识符>→<字母> | <标识符><字母> | <标识符><数字>

<字母>→a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s
| t | u | v | w | x | y | z

<数字>→0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<函数说明>→integer function <标识符> (<参数>); <函数体>

<参数>→<变量>

<函数体>→begin <说明语句表>; <执行语句表> end

<执行语句表>→<执行语句> | <执行语句表>; <执行语句>

<执行语句>→<读语句> | <写语句> | <赋值语句> | <条件语句>

<读语句>→read(<变量>)

<写语句>→write(<变量>)

<赋值语句>→<变量>:=<算术表达式>

<算术表达式>→<算术表达式>*<项> | <项>

<项>→<项>*<因子> | <因子>

<因子>→<变量> | <常数> | <函数调用>

<常数>→<无符号整数>

<无符号整数>→<数字> | <无符号整数><数字>

<条件语句>→if<条件表达式>then<执行语句>else <执行语句>

<条件表达式>→<算术表达式><关系运算符><算术表达式>

<关系运算符> →< | <= | > | >= | = | <>

测试程序：

begin

integer k;

integer function F(n);

begin

integer n;

if n<=0 then F:=1

else F:=n*F(n-1)

end;

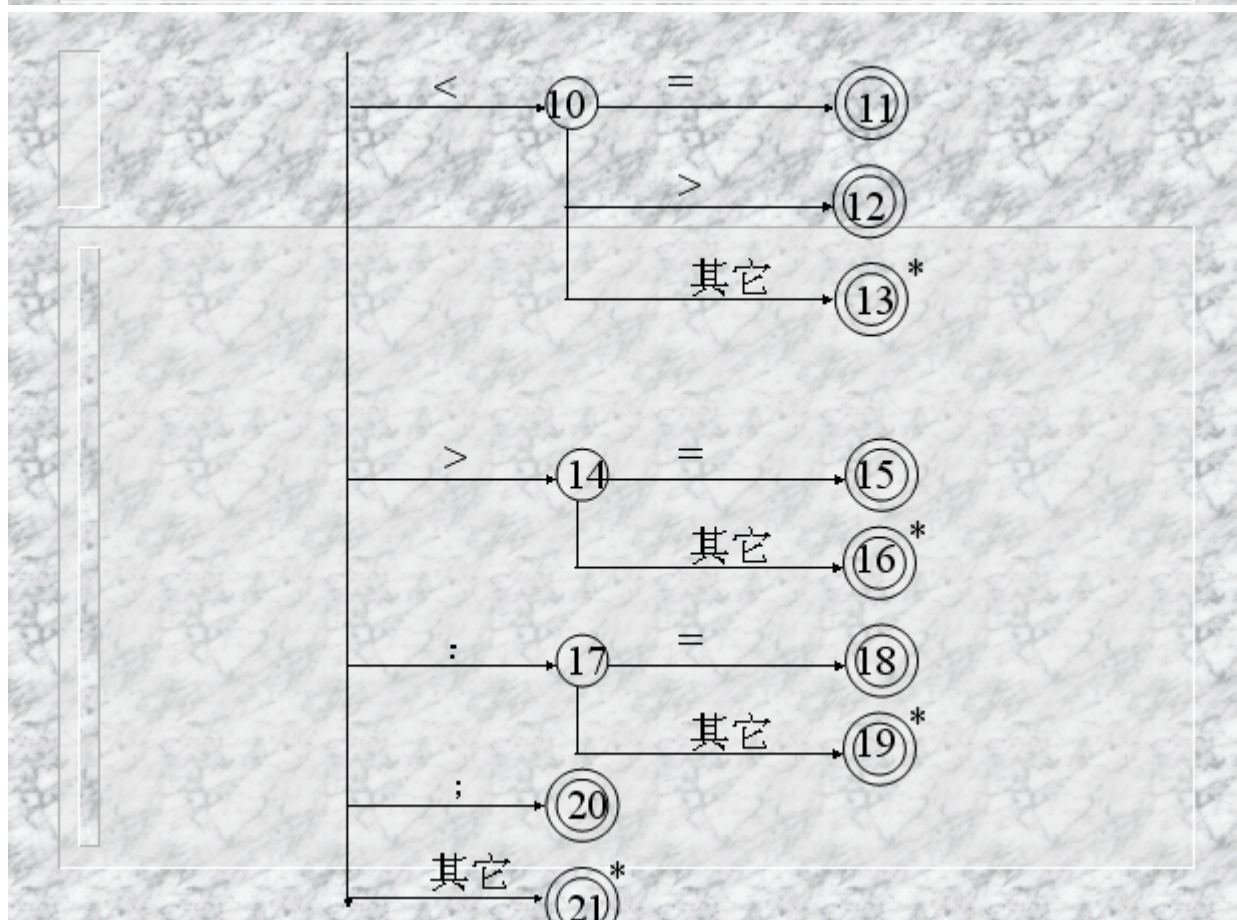
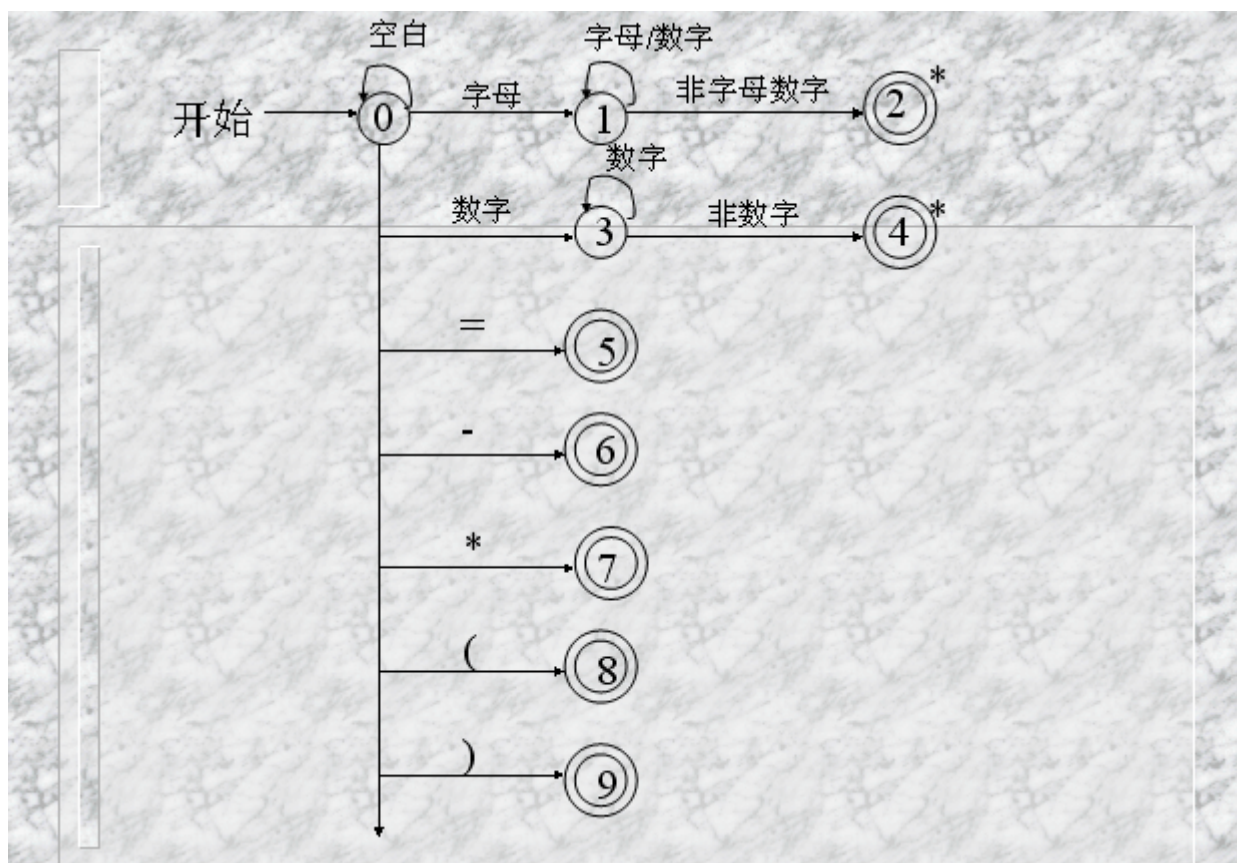
read(m);

k:=F(m);

write(k)

end

2.状态转换图

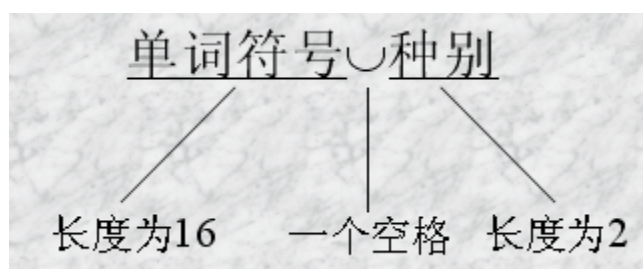


3.单词编码表

单词符号与种别对照表					
单词符号	种别	单词符号	种别	单词符号	种别
begin	1	end	2	integer	3
if	4	then	5	else	6
function	7	read	8	write	9
标识符	10	常数	11	=	12
<>	13	<=	14	<	15
>=	16	>	17	-	18
*	19	:=	20	(21
)	22	;	23		

4.构造词法分析器要按照给定的单词编码表组织成一个指定的二元式序列文件*.dyd, 词法分析中要求产生一个词法错误提示文件*.err, 产生的二元式序列文件*.dyd 将作为语法分析器的输入文件。产生的二元式文件有如下要求:

1) 二元式形式



2) 每行后加一

“ ∪ ∪ ∪ ... ∪ EOLN ∪ 24 ”

3) 文件结尾加

“...EOF25”

2.) 错误信息文件 *.err

(1) 错误信息格式

***LINE:行号 错误性质

(2) 注意: 进入每一阶段, 首先打开 *.err, 如果无错误, 则 *.err 为空。

5.实现提示

- 1) 多数文件均和源文件同名, 仅扩展名不同。
- 2) 词法分析时, 注意行尾和文件尾。
- 3) 词法分析的实现方法——利用状态转换图

三 . 实验要求

实验前要做好充分准备, 包括程序清单、调试步骤、调试方法, 以及对程序结果的分析等。

四 . 实验报告

1. 设计思想

词法分析器是编译程序中重要的一个环节, 它将源程序字符串转换成二元式序列交给下面的语法分析器进行进一步的处理。在将源程序字符串转换成二元式过程中, 可能会遇到诸如非法字符、冒号不匹配等多种错误, 这就同时要求了错误的处理。

本实验词法分析器的设计思想如下图所示:

首先, 对测试程序进行预处理, 将其转化为字符串。其中, 对空格进行保留, 对换行用标识符“EOLN”表示, 对文件结束用标识符“EOF”表示。

然后, 将字符串输入主词法分析算法进行处理, 将得到的二元式序列写入文件 result.dyd 文件中, 将包括非法字符和冒号不匹配错误的错误信息写入文件 error.err 文件中。

对于出错处理, 存在 3 种错误: 非法字符、冒号不匹配以及非法标识符。其中, 非法标识符的错误是本人在实验中另外添加的 1 种错误。它指的是在命名标

识符的时候,不能以数字开头,如 ‘integer 2k;’ 否则将进行非法标识符错误处理。

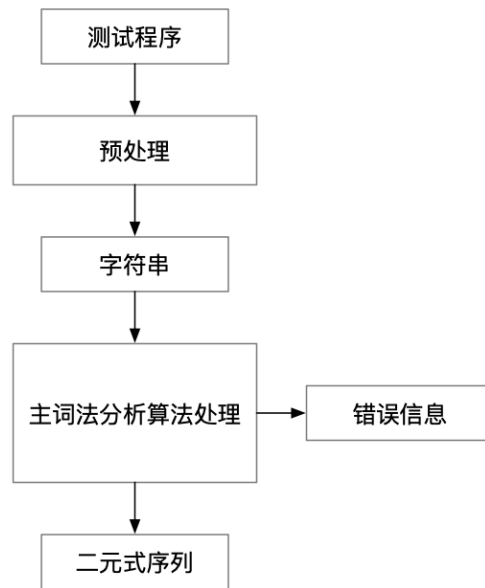


图 1. 词法分析器设计思想

2. 总控算法

本人设计的词法分析器中主词法分析算法如下:

```
start: chars[]          // 存放字符数组
      token = ""        // 存放单词符号
      flag = false      // 判断当前字符和下一字符能否构成 2 位符号
begin
  if flag = true        // 如果上一字符和当前字符构成 2 位符号,则当前跳过
    continue
  getchar               // 获取下一个符号
  case character of
    'a'...'z' or '0'...'9' begin
      concatenate       // 将 ch 中的字符连接到 token 后
      continue
    end
    '=' begin           // =
      return('=',12)
    end
    '-' begin           // -
      return('-',18)
    end
  end
```

```

'*' begin          // *
    return('*',19)
end
'(' begin          // (
    return('(',21)
end
')' begin          // )
    return(')',22)
end
';' begin          // ;
    return('; ',23)
end
'<' begin
    if ch == '>'
        return('<>',13)    // <>
    else if ch == '='
        return('<=',14)    // <=
    else
        return('<',15)      // <
    end
end
'>' begin
    if ch == '='
        return('>=',16)    // >=
    else
        return('>',17)      // >
    end
end
':' begin
    if ch == '='
        return(':',20)      // :=
    else
        error              // error:冒号不匹配
    end
end
'' begin
    continue

```



```

end
default begin
    if token isNot empty begin
        c := reserve    // 用 token 中的字符串查保留字表,
                        // 若查到则返回该保留字的种别编码

        if c=0 begin
            if character of token is 'a'...'z'
                return(token,10)
            if character of token is '0'...'9'
                return(token,11)
        end
    end
    else begin
        if character of token[0] is '0' ... '9' and exists token[i] is 'a'...'z'
            error      // error:非法标识符
        else
            return(token,c)
        token = ""      // 将 token 置空
    end
    else begin
        error          // error:非法字符
    end
end of case
end

```

3. 主要服务子程序算法

本人设计的主要服务子程序算法，即预处理算法如下所示：

```

start: file          // 存放测试程序的文件
    in              // 存放测试程序的字符串
    str            // 存放测试程序的一行的字符串
begin
    while str = file.readLine isNot null do
        if in isNot null
            in += "EOLN" // 在每行后面加上"EOLN"代表换行
            in += str    // 将 str 接到 in 后面
        end
    end

```

```
        in += "EOF"           // 在文件后面加上"EOF"代表结束
end
```

4. 程序说明

本人设计的词法分析器的代码及注释如下：（Java 语言）

```
public static void main(String[] args) throws Exception {
    // 从文件中读取测试程序
    File file = new File("/Users/XuXianda/Downloads/test.txt");
    BufferedReader bufferedReader = new BufferedReader(new FileReader(file));
    File result = new File("/Users/XuXianda/Downloads/result.dyd");
    BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(result));
    File error = new File("/Users/XuXianda/Downloads/error.err");
    BufferedWriter bufferedWriter1 = new BufferedWriter(new FileWriter(error));

    String in = "";
    String str;
    // 字符串 in 用来存储测试程序的内容
    while ((str = bufferedReader.readLine()) != null) {
        // 在每行的末尾添加 EOLN 代表换行
        if (!in.equals(""))
            in += " EOLN ";
        in += str;
    }
    // 在程序的末尾添加 EOF 代表测试程序结束
    in += " EOF ";
    // 字符串 token 用来存放单词符号
    String token = "";
    // 布尔值 flag 用来判断当前字符和下一字符能否构成 2 位运算符或关系符
    Boolean flag = false;
    // int 值 line 用来记录当前行号
    int line = 1;
    // 字符数组 chars 用来存放字符串 in 的每个字符值
    char[] chars = in.toCharArray();
    // 依次读取 chars 中的每个字符值
    for (int i = 0; i < chars.length; i++) {
        // 如果之前已经将 2 位运算符或关系符返回,则跳过这一步
```

```

if (flag) {
    flag = false;
    continue;
}
// 字符值 ch 记录当前一步的字符值
char ch = chars[i];
// 如果当前字符值 ch 是字母或者数字,将它连接到 token
if (Character.isLetter(ch) || Character.isDigit(ch)) {
    token = token + ch;
}
else if (!token.equals("")) {
    // 关键字 begin
    if (token.equals("begin")) {
        System.out.println("        begin 01");
        bufferedWriter.write("        begin 01\n");
    }
    // 关键字 end
    else if (token.equals("end")) {
        System.out.println("        end 02");
        bufferedWriter.write("        end 02\n");
    }
    // 关键字 integer
    else if (token.equals("integer")) {
        System.out.println("        integer 03");
        bufferedWriter.write("        integer 03\n");
    }
    // 关键字 if
    else if (token.equals("if")) {
        System.out.println("        if 04");
        bufferedWriter.write("        if 04\n");
    }
    // 关键字 then
    else if (token.equals("then")) {
        System.out.println("        then 05");
    }
}

```

```

        bufferedWriter.write("        then 05\n");
    }
    // 关键字 else
    else if (token.equals("else")) {
        System.out.println("        else 06");
        bufferedWriter.write("        else 06\n");
    }
    // 关键字 function
    else if (token.equals("function")) {
        System.out.println("        function 07");
        bufferedWriter.write("        function 07\n");
    }
    // 关键字 read
    else if (token.equals("read")) {
        System.out.println("        read 08");
        bufferedWriter.write("        read 08\n");
    }
    // 关键字 write
    else if (token.equals("write")) {
        System.out.println("        write 09");
        bufferedWriter.write("        write 09\n");
    }
    // 关键字 EOLN
    else if (token.equals("EOLN")) {
        line++;
        System.out.println("        EOLN 24");
        bufferedWriter.write("        EOLN 24\n");
    }
    // 关键字 EOF
    else if (token.equals("EOF")) {
        System.out.println("        EOF 25");
        bufferedWriter.write("        EOF 25");
    }
    else {

```

```

        int spaceNumber = 16 - token.length();
        // 标识符
        if (Character.isLetter(token.charAt(0))) {
            for (int j = 0; j < spaceNumber; j++) {
                System.out.print(" ");
                bufferedWriter.write(" ");
            }
            System.out.println(token + " 10");
            bufferedWriter.write(token + " 10\n");
        }
        // 常量
        if (Character.isDigit(token.charAt(0))) {
            // 出错处理:非法标识符
            boolean checkError = false;
            for (int m = 0; m < token.length(); m++)
                if (!Character.isDigit(token.charAt(m)))
                    checkError = true;
            if (checkError) {
                bufferedWriter1.write("ERROR! LINE:" + line + " 非法标识符\n");
            } else {
                for (int j = 0; j < spaceNumber; j++) {
                    System.out.print(" ");
                    bufferedWriter.write(" ");
                }
                System.out.println(token + " 11");
                bufferedWriter.write(token + " 11\n");
            }
        }
    }
    // token 清空
    token = "";
    // retract 回退
    i = i - 1;
}

```

```

// 运算符 =
else if (ch == '=') {
    System.out.println("                = 12");
    bufferedWriter.write("                = 12\n");
}
// 运算符 -
else if (ch == '-') {
    System.out.println("                - 18");
    bufferedWriter.write("                - 18\n");
}
// 运算符 *
else if (ch == '*') {
    System.out.println("                * 19");
    bufferedWriter.write("                * 19\n");
}
// 界符 (
else if (ch == '(') {
    System.out.println("                ( 21");
    bufferedWriter.write("                ( 21\n");
}
// 界符 )
else if (ch == ')') {
    System.out.println("                ) 22");
    bufferedWriter.write("                ) 22\n");
}
// 界符 ;
else if (ch == ';') {
    System.out.println("                ; 23");
    bufferedWriter.write("                ; 23\n");
}
else if (ch == '<') {
    // ch2 记录下一位字符
    char ch2 = chars[i+1];
    // 2 位界符 <>

```



```

        if (ch2 == '>') {
            System.out.println("                ◇ 13");
            bufferedWriter.write("                ◇ 13\n");
            flag = true;
        }
        // 2 位运算符 <=
        else if (ch2 == '=') {
            System.out.println("                <= 14");
            bufferedWriter.write("                <= 14\n");
            flag = true;
        }
        // 1 位运算符 <
        else {
            System.out.println("                < 15");
            bufferedWriter.write("                < 15\n");
        }
    }
    else if (ch == '>') {
        // ch2 记录下一位字符
        char ch2 = chars[i+1];
        // 2 位运算符 >=
        if (ch2 == '=') {
            System.out.println("                >= 16");
            bufferedWriter.write("                >= 16\n");
            flag = true;
        }
        // 1 位运算符 >
        else {
            System.out.println("                > 17");
            bufferedWriter.write("                > 17\n");
        }
    }
    else if (ch == ':') {
        // ch2 记录下一位字符

```

```

        char ch2 = chars[i+1];
        // 2 位运算符 :=
        if (ch2 == '=') {
            System.out.println("                := 20");
            bufferedWriter.write("                := 20\n");
            flag = true;
        }
        // 出错处理:冒号不匹配
        else {
            bufferedWriter1.write("ERROR! LINE:" + line + "    冒号不匹配\n");
        }
    }
    // 空格
    else if (ch == ' ') {
        continue;
    }
    // 出错处理:非法字符
    else {
        bufferedWriter1.write("ERROR! LINE:" + line + "    非法字符\n");
    }
}
bufferedReader.close();
bufferedWriter.close();
bufferedWriter1.close();
}

```

5. 测试程序及运行结果

首先测试的是没有错误的标准测试程序：

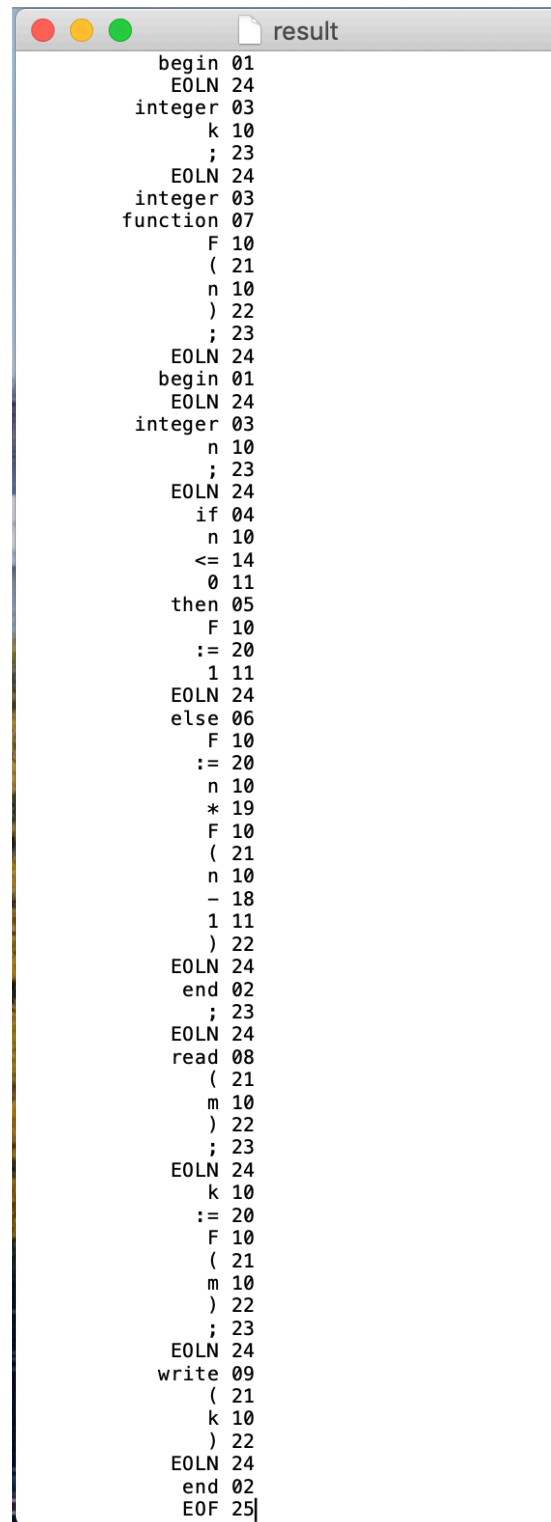
```

begin
    integer k;
    integer function F(n);
    begin
        integer n;
        if n<=0 then F:=1
        else F:=n*F(n-1)
    end
end

```

```
end;  
read(m);  
k:=F(m);  
write(k)  
end
```

本测试程序的输出结果是：



```
begin 01  
EOLN 24  
integer 03  
k 10  
; 23  
EOLN 24  
integer 03  
function 07  
F 10  
( 21  
n 10  
) 22  
; 23  
EOLN 24  
begin 01  
EOLN 24  
integer 03  
n 10  
; 23  
EOLN 24  
if 04  
n 10  
<= 14  
0 11  
then 05  
F 10  
:= 20  
1 11  
EOLN 24  
else 06  
F 10  
:= 20  
n 10  
* 19  
F 10  
( 21  
n 10  
- 18  
1 11  
) 22  
EOLN 24  
end 02  
; 23  
EOLN 24  
read 08  
( 21  
m 10  
) 22  
; 23  
EOLN 24  
k 10  
:= 20  
F 10  
( 21  
m 10  
) 22  
; 23  
EOLN 24  
write 09  
( 21  
k 10  
) 22  
EOLN 24  
end 02  
EOF 25
```

图 2 没有错误的测试程序输出结果

其次测试的是有非法字符错误的测试程序：

```
begin
  integer k;
  integer function F(n);
    begin
      integer &n;      // 非法字符 &
      if n<=0 then F:=1
      else F:=n*F(n-1)
    end;
  read(?m);           // 非法字符 ?
  k:=F(m);
  write(k)
end
```

本测试程序的输出结果是：



图 3 带有非法字符错误的测试程序的输出结果

其次测试的是有冒号不匹配错误的测试程序

```
begin
  integer k;
  integer function F(n);
    begin
      integer n;
      if n<=0 then F:=1
      else F:n*F(n-1)  // 冒号不匹配
    end;
  read(m);
  k:=F(m);
  write(k)
end
```

本测试程序的输出结果是：

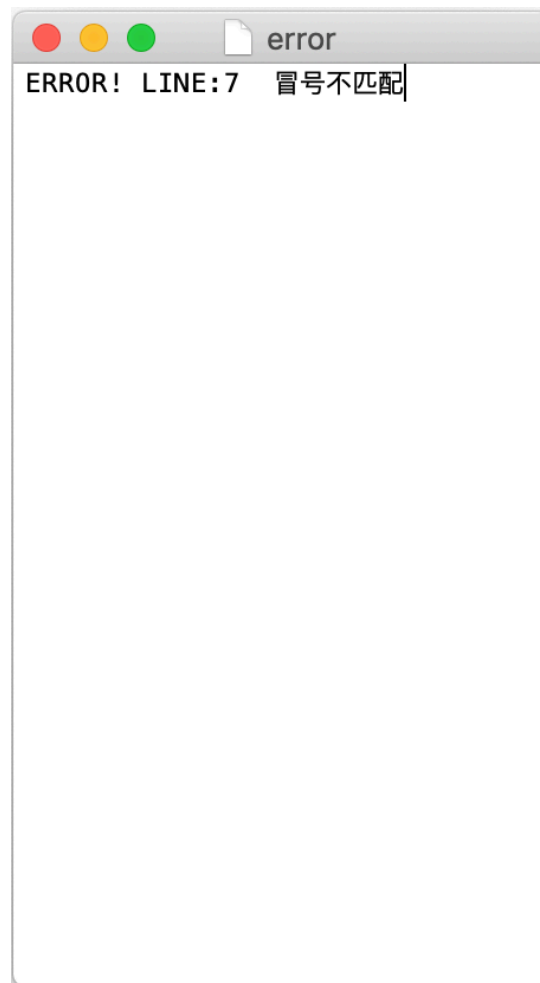


图 4 带有冒号不匹配错误的测试程序的输出结果
最后测试的是有非法标识符错误的测试程序

```
begin
  integer 2k;           // 非法标识符
  integer function F(n);
  begin
    integer n;
    if n<=0 then F:=1
    else F:n*F(n-1)
  end;
  read(m);
  k:=F(m);
  write(k)
end
```

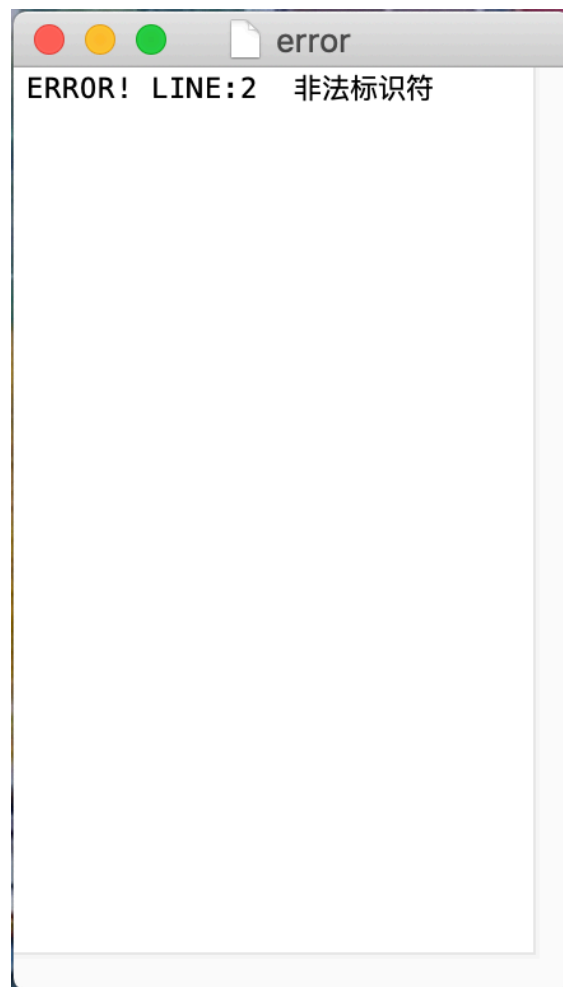


图 5 带有非法标识符错误的测试程序的输出结果

6. 调试过程

本实验一开始本人想在预处理中把所有的空格给去掉，但是后来发现不能去掉。如果仅仅依靠运算符或者界符来识别关键字或标识符，会出现识别错误的情况，比方说 `integer k` 会识别成标识符 `integerk`，导致错误的发生。后来，经过多次调试，本人设计出了符合要求的算法完成了词法分析的工作。

五 . 总结心得与建议

本实验进行了对于编译过程中词法分析器的设计。词法分析器的工作就是将源程序字符串转换成二元式序列交给下面的语法分析器进行进一步的处理。首先，我们得将源测试程序进行预处理，转换成符合要求的字符串，输入词法分析器。然后，我们的词法分析器对于关键字、标识符、常量、运算符和界符进行识别，转换成二元式写入文件。同时，我们的词法分析器需要对两种错误，即非法字符和冒号不匹配进行识别，写入错误信息文件。

本人通过此次实验更好地了解了词法分析器的设计原理以及对于错误的处理方式，感谢老师的讲授。