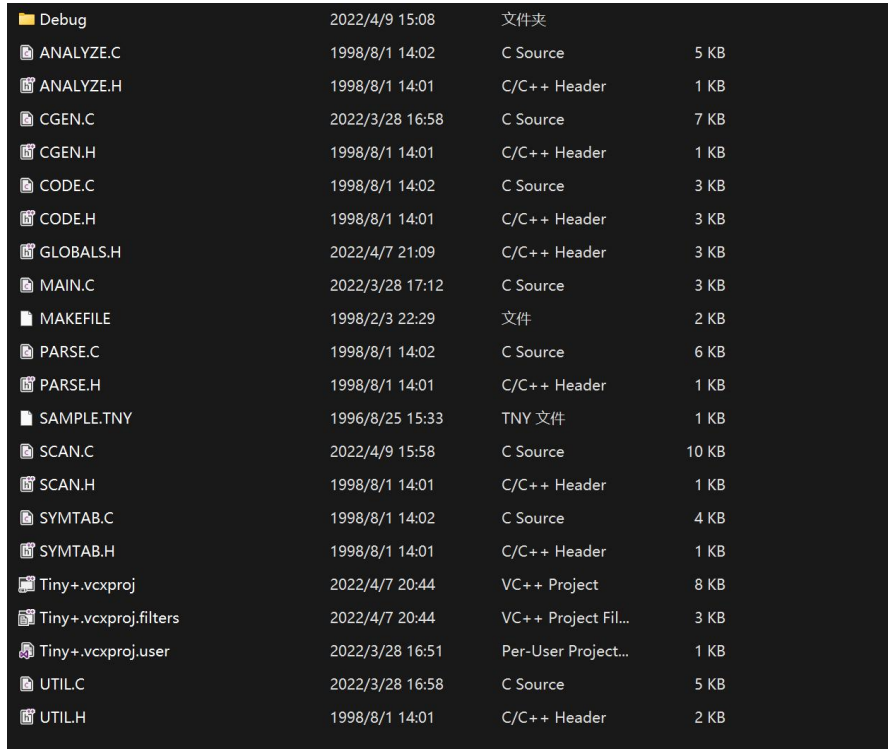
编译原理词法分析报告

陆逸凡

2019141090163

1.编程语言及开发环境

- 编程语言：C语言
- 开发环境：
  - 硬件：处理器AMD Ryzen 7 5800H with Radeon Graphics　　3.20 GHz
  - 软件：Windows11 家庭版
  - IDE: Microsoft Visual Studio 2019



Debug	2022/4/9 15:08	文件夹	
ANALYZE.C	1998/8/1 14:02	C Source	5 KB
ANALYZE.H	1998/8/1 14:01	C/C++ Header	1 KB
CGEN.C	2022/3/28 16:58	C Source	7 KB
CGEN.H	1998/8/1 14:01	C/C++ Header	1 KB
CODE.C	1998/8/1 14:02	C Source	3 KB
CODE.H	1998/8/1 14:01	C/C++ Header	3 KB
GLOBALS.H	2022/4/7 21:09	C/C++ Header	3 KB
MAIN.C	2022/3/28 17:12	C Source	3 KB
MAKEFILE	1998/2/3 22:29	文件	2 KB
PARSE.C	1998/8/1 14:02	C Source	6 KB
PARSE.H	1998/8/1 14:01	C/C++ Header	1 KB
SAMPLE.TNY	1996/8/25 15:33	TNY 文件	1 KB
SCAN.C	2022/4/9 15:58	C Source	10 KB
SCAN.H	1998/8/1 14:01	C/C++ Header	1 KB
SYMTAB.C	1998/8/1 14:02	C Source	4 KB
SYMTAB.H	1998/8/1 14:01	C/C++ Header	1 KB
Tiny+.vcxproj	2022/4/7 20:44	VC++ Project	8 KB
Tiny+.vcxproj.filters	2022/4/7 20:44	VC++ Project Fil...	3 KB
Tiny+.vcxproj.user	2022/3/28 16:51	Per-User Project...	1 KB
UTIL.C	2022/3/28 16:58	C Source	5 KB
UTIL.H	1998/8/1 14:01	C/C++ Header	2 KB

2. 工作内容

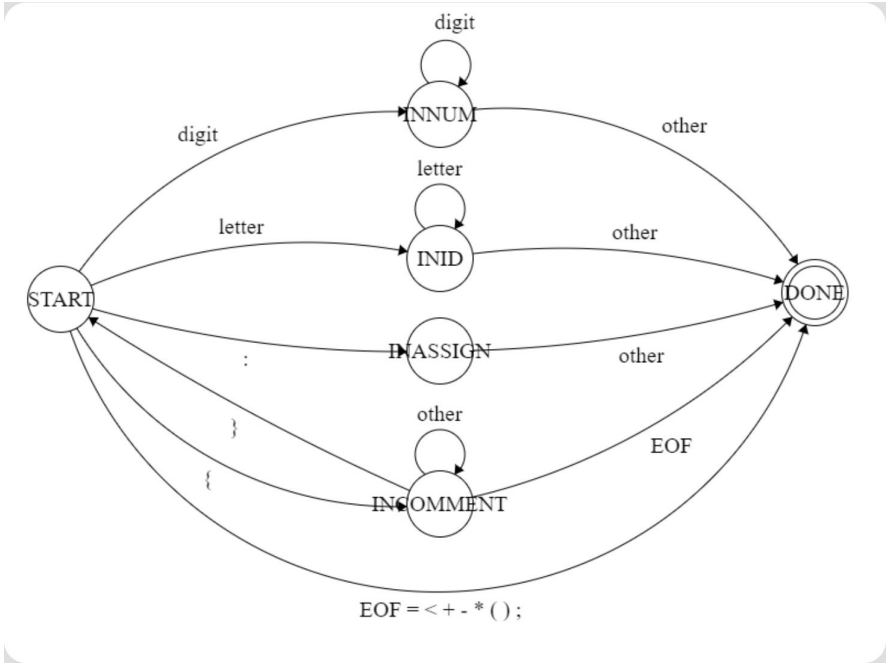
2.1 Tiny词法分析器的原本功能

分析：源程序中，使用SCAN.C中的getToken()函数实现词法分析功能。具体利用双层嵌套的case语句实现。

画出修改前的DFA如右图：

可以看到原词法分析器只对整数进行了简单的判定。

其注释是采用“{”与“}”来进行标识。



DFA在线制作网址：Finite State Machine Designer - by Evan Wallace (madebyevan.com)

2. 工作内容

2.1 Tiny词法分析器的原本功能

原词法分析器一共有6种状态：

START,

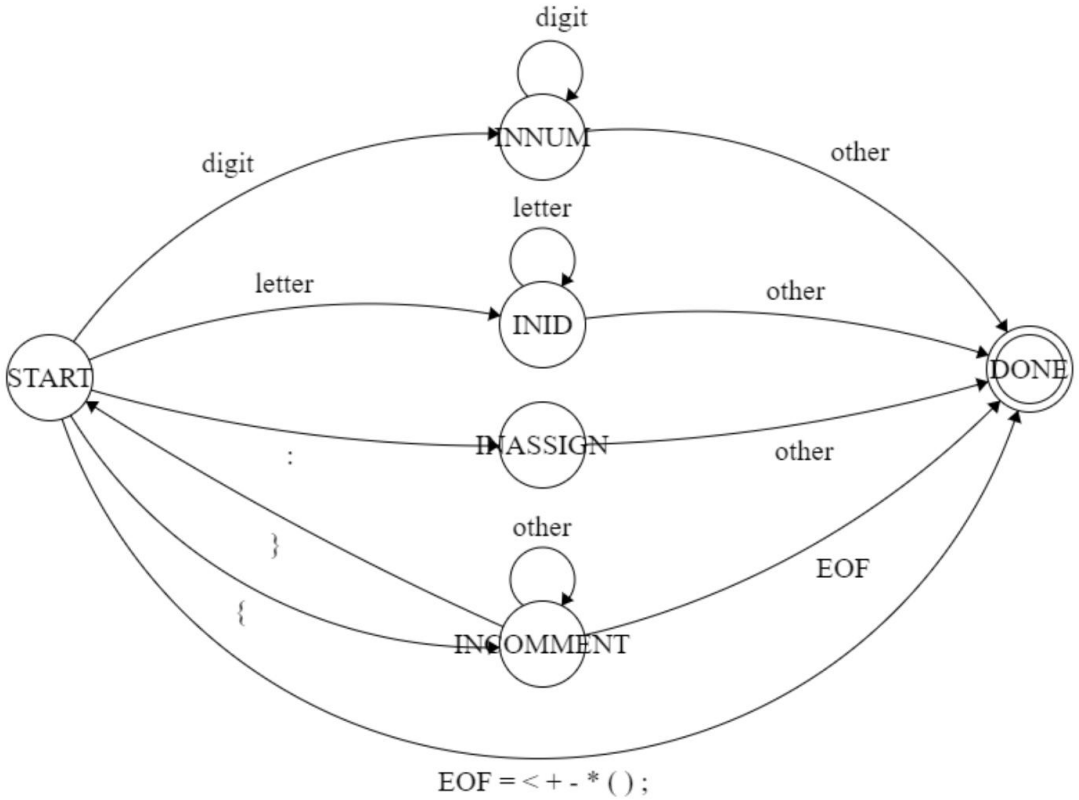
INASSIGN,

INCOMMENT,

INNUM,

INID,

DONE



DFA在线制作网址：Finite State Machine Designer - by Evan Wallace (madebyevan.com)

2. 工作内容

2.2 Tiny词法分析器的新增功能

（一）浮点数&科学计数法的识别

分析：浮点数科学计数法的结构形式的例子为：3.5e+2，5E-1

需要在原基于整数的分析器上增加关于小数点“.”、字符“e”“E”以及加减号“+”“-”的识别。

需要注意的点是：e/E之前的数可以为小数，而之后的数含义为10的次方，因此为整数。

2. 工作内容

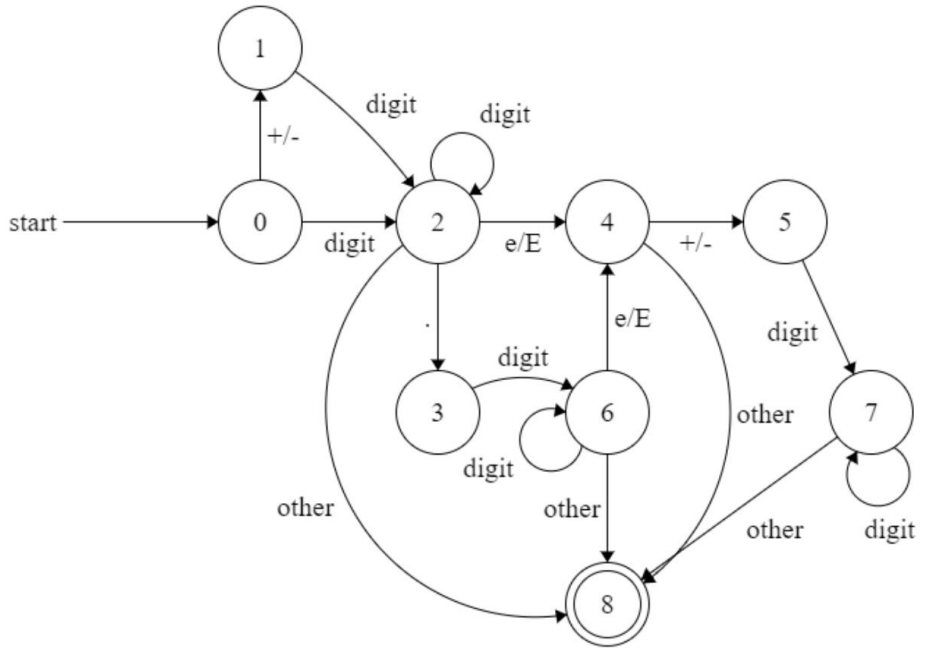
2.2 Tiny词法分析器的新增功能

（一）浮点数&科学计数法的识别

画出DFA图：

增加了8个状态。

最后一个状态为原本的DONE状态



DFA在线制作网址：Finite State Machine Designer - by Evan Wallace (madebyevan.com)

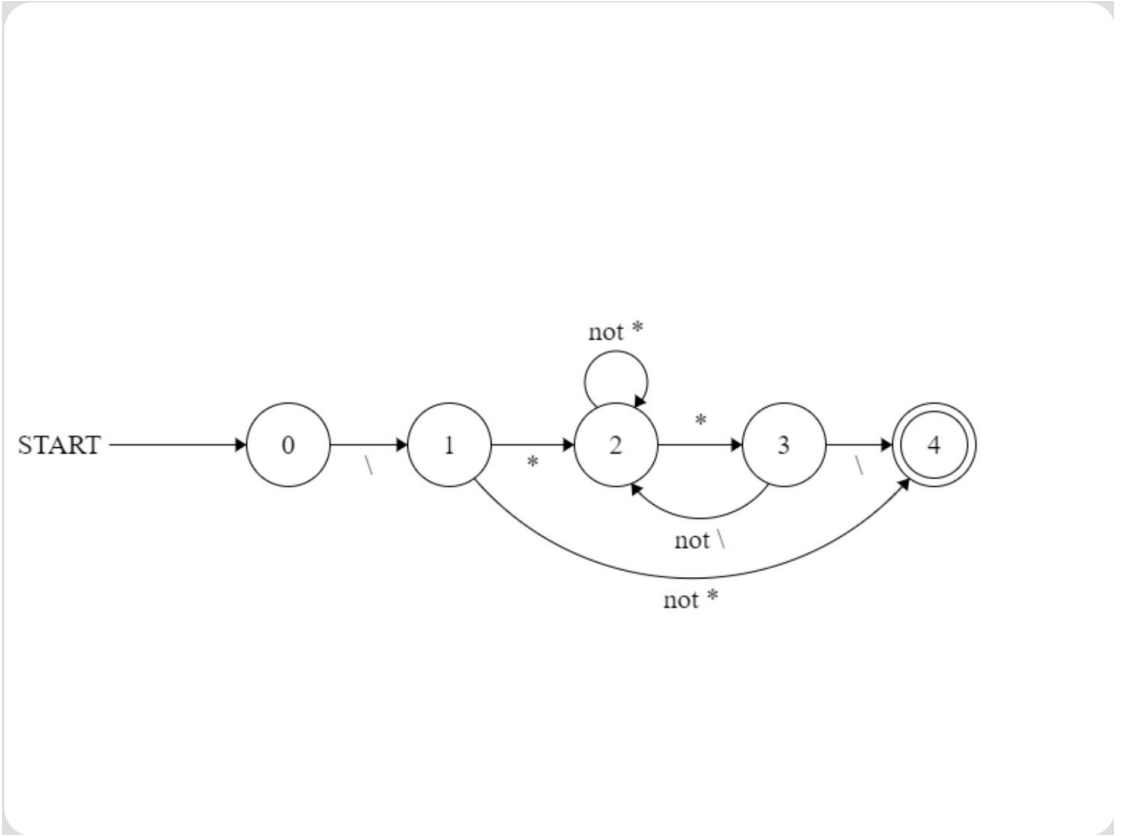
2. 工作内容

2.2 Tiny词法分析器的新增功能

（二）多行注释/* */的识别

画出DFA图：

增加了4个状态。



与此同时，将原本“{}”风格注释保留。

2. 工作内容

2.2 Tiny词法分析器的新增功能

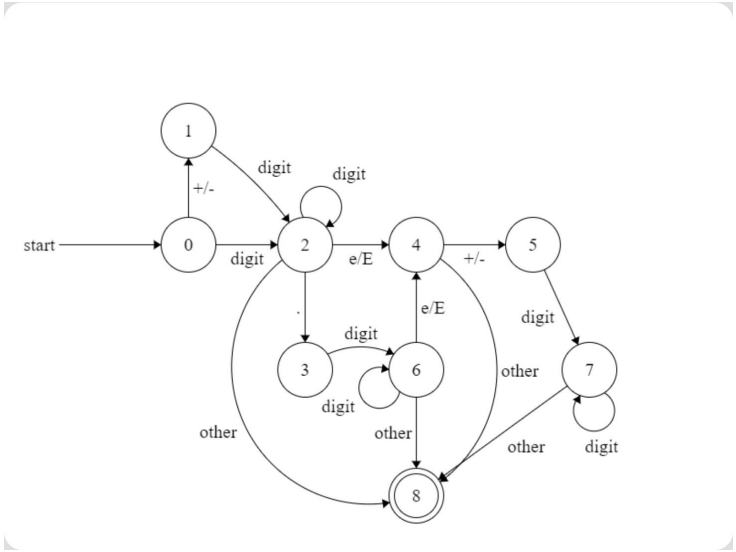
（三）出错处理

针对科学计数法中e/E后不能有小数点进行处理。检测到小数点则进行报错。

针对变量名的出错，增加了以数字开头的变量名的识别报错。

3.实现方式

- 主要采用的实现方式：双层嵌套的case语句
- （一）科学计数法的实现

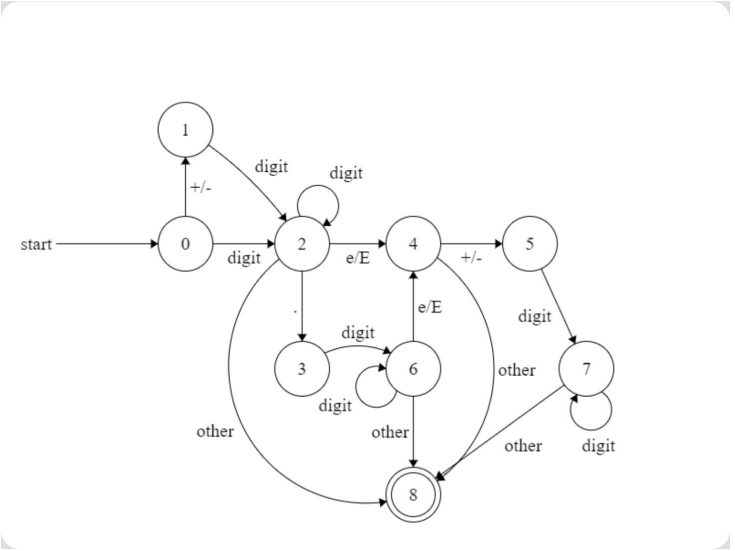



```
  /*科学计数法的修改*/
case float_1:
    if (isdigit(c))
    {
        state = float_2;
    }
    else if (c == '+') {
        ungetNextChar();
        state = DONE;
        currentToken = PLUS;//加法的判断
    }
    else if (c == '-') {
        ungetNextChar();
        state = DONE;
        currentToken = MINUS;//加法的判断
    }
    else
    {
        ungetNextChar();
        state = START;
    }
    break;
```

```
case float_2:
    if (isdigit(c)) state = float_2;
    else if (c == '.') state = float_3;
    else if (c == 'e' || c == 'E') state = float_4;
    else
    {
        ungetNextChar();
        save = FALSE;
        state = DONE;
        currentToken = NUM;
    }
    break;
```

3.实现方式

- 主要采用的实现方式：双层嵌套的case语句
- （一）科学计数法的实现




```
case float_3:
    if (isdigit(c)) state = float_6;
    else
    {
        state = DONE;
        currentToken = ERROR;
    }
    break;
```

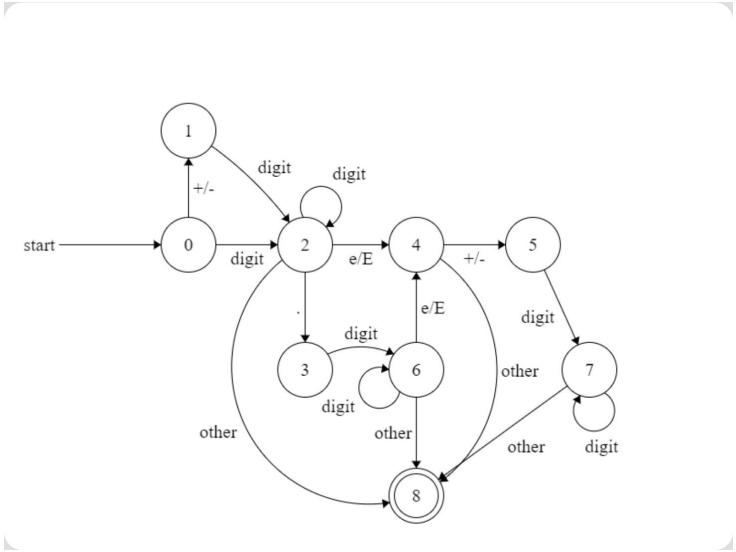
```
case float_4:
    if (c == '+' || c == '-') state = float_5;
    else if (isdigit(c))
    {
        state = DONE;
        currentToken = ERROR;
        while (isdigit(c)||c=='.') getNextChar();
    }
    else
    {

        ungetNextChar();
        save = FALSE;
        state = DONE;
        currentToken = NUM;
    }
    break;
```

```
case float_5:
    if (isdigit(c)) state = float_7;
    else {
        state = DONE;
        currentToken = ERROR;
     }
    break;
```

3.实现方式

- 主要采用的实现方式：双层嵌套的case语句
- （一）科学计数法的实现



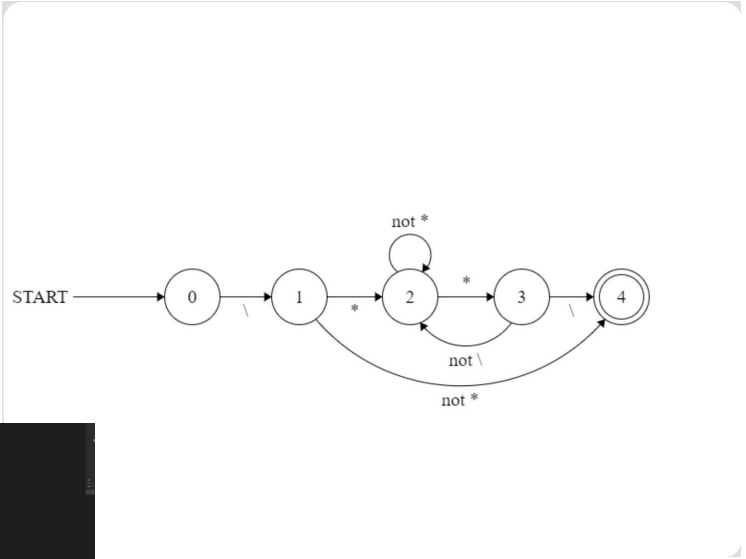

```
case float_6:
    if (isdigit(c)) state = float_6;
    else if (c == 'e' || c == 'E') state = float_4;
    else
    {
        ungetNextChar();
        save = FALSE;
        state = DONE;
        currentToken = NUM;
    }
    break;
```

```
case float_7:
    if (isdigit(c)) state = float_7;
    else if (c == '.')
    {
        state = DONE;
        currentToken = ERROR;
    }
    else
    {
        ungetNextChar();
        save = FALSE;
        state = DONE;
        currentToken = NUM;

    }
    break;
    /*科学计数法修改结束*/
```

3.实现方式

- 主要采用的实现方式：双层嵌套的case语句
- （二）多行注释识别的实现

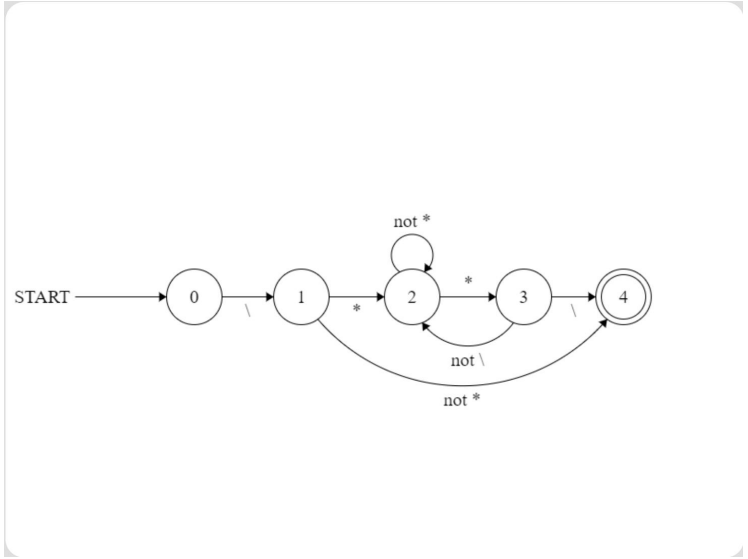



```
        currentToken = ENDFILE;
        break;
      case '=':
        currentToken = EQ;
        break;
      case '<':
        currentToken = LT;
        break;
      case '+':
        currentToken = PLUS;
        break;
      case '-':
        currentToken = MINUS;
        break;
      case '*':
        currentToken = TIMES;
        break;
      case '/':
        currentToken = OVER;
        break;
      case '(':
        currentToken = LPAREN;
        break;
      case ')':
        currentToken = RPAREN;
        break;
      case ';':
```

注释"\"与除号相同，需要将原始的除号状态删去

3.实现方式

- 主要采用的实现方式：双层嵌套的case语句
- （二）多行注释识别的实现




```
case INCOMMENT_O:
  save = FALSE;
  if (c == EOF)
  {
      state = DONE;
      currentToken = ENDFILE;
  }
  else if (c == '}') state = START;
  break;

  /*注释的修改*/
case INCOMMENT:
  save = FALSE;

  if (c == EOF)
  {
      state = DONE;
      currentToken = ENDFILE;
  }
  else if (c == '*' && getNextChar() == '/') state = START;//注释结束，不需要回滚了
  else state = INCOMMENT;
  break;
```

```
case COMMENTSTART:
    save = FALSE;
    if (c == '/')
    {
        state = INCOMMENT;
    }
    else
    {
        state = START;
        currentToken = OVER;//除法的判断
    }
    //注释修改结束//
```

3.实现方式

- 主要采用的实现方式：双层嵌套的case语句
- （三）出错处理的实现

```
case float_4:
    if (c == '+' || c == '-') state = float_5;
    else if (isdigit(c))
    {
        state = DONE;
        currentToken = ERROR;
        while (isdigit(c)||c=='.') getNextChar();
    }
    else
    {
        ungetNextChar();
        save = FALSE;
        state = DONE;
        currentToken = NUM;
    }
    break;
```

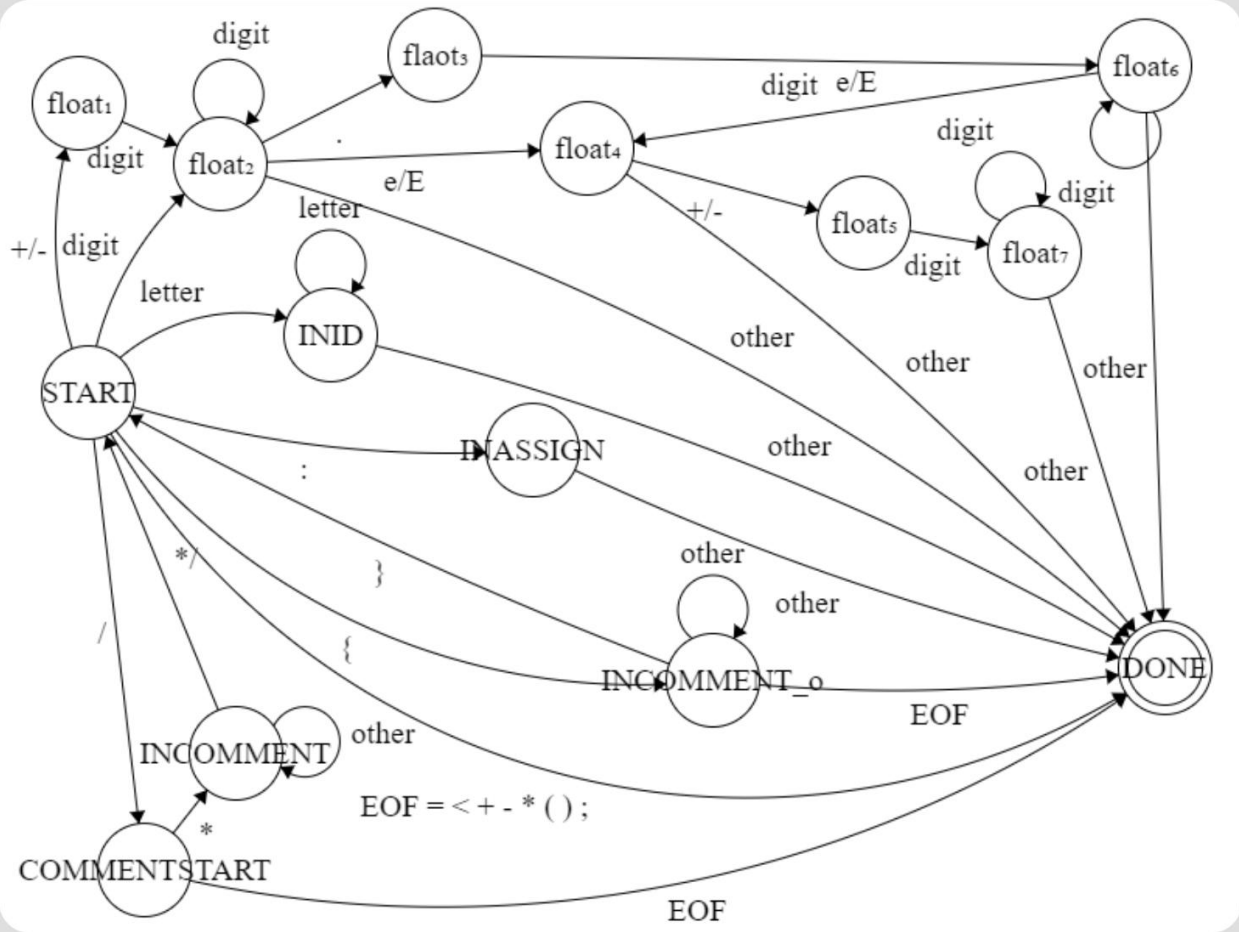
在e/E后出现数字或小数点的报错

```
case float_2:
    if (isdigit(c)) state = float_2;
    else if (c == '.') state = float_3;
    else if (c == 'e' || c == 'E') state = float_4;
    else if (isalpha(c))
    {
        state = DONE;
        currentToken = ERROR;
    }
    else
    {
        ungetNextChar();
        save = FALSE;
        state = DONE;
        currentToken = NUM;
    }
    break;
```

出现以数字开头的变量名的报错

4. 核心函数的复杂度分析

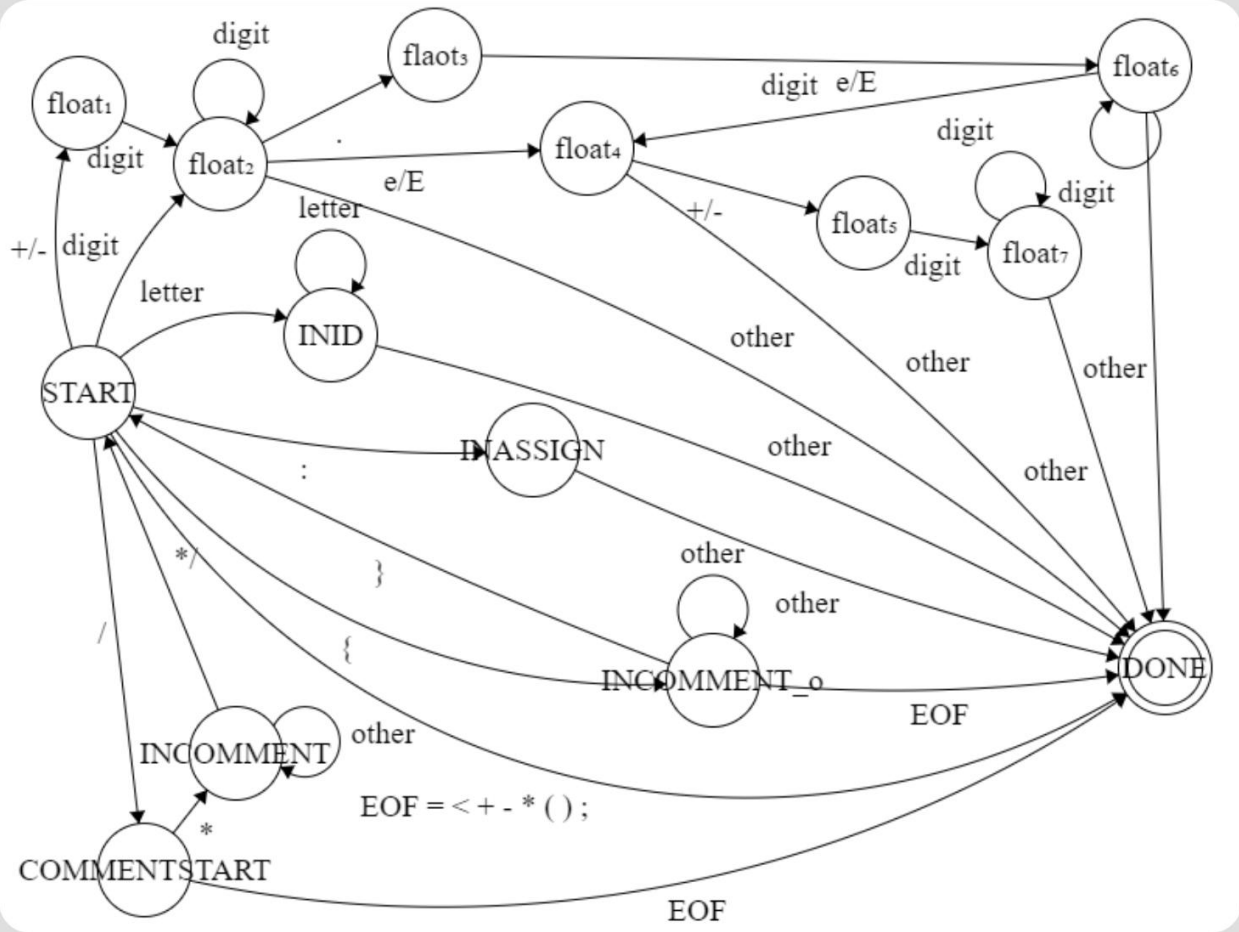
- 首先画出核心函数的DFA图如下：



根据公式环复杂度=E-N+2
分析计算可得环复杂度为
31-14+2=19

4. 核心函数的复杂度分析

- 采用基本路径测试法，实现全覆盖，应该设计19个测试用例

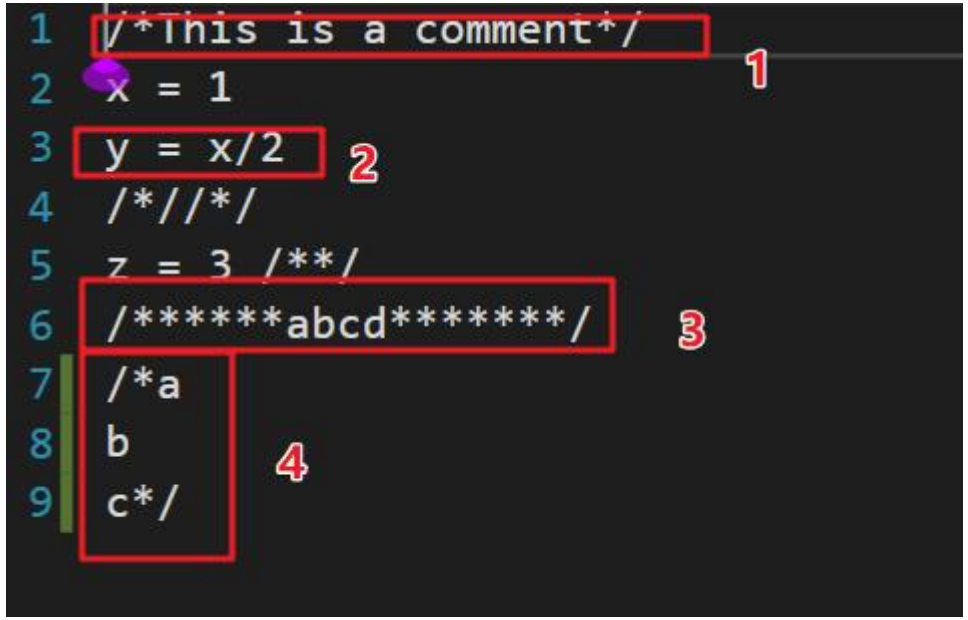


5. 实际测试

- 针对2种实现的功能，分别编写测试文件进行测试。

5. 实际测试

- 在对注释的测试文件commentTest.TNY中，根据各种情况设计了4种测试用例，包含了多行注释DFA的所有路径。




```
D:\2022春\编译原理\课设\Tiny+\Debug>Tiny+.exe commentTest.TNY

TINY COMPILATION: commentTest.TNY
   1: /*This is a comment*/
   2: x = 1
        2: ID, name= x
        2: =
        2: NUM, val= 1
   3: y = x/2
        3: ID, name= y
        3: =
        3: ID, name= x
        3: /
        3: NUM, val= 2
   4: /*//*/
   5: z = 3 /**/
        5: ID, name= z
        5: =
        5: NUM, val= 3
   6: /******abcd*******/
   7: /*a
   8: b
   9: c*/       10: EOF
```

5. 实际测试

- 在对注释的测试文件float_test.TNY中，根据各种情况设计了10种测试用例以及2种错误用例，包含了科学计数法识别的DFA的所有路径。

```
1  1.0
2  3.333333
3  2.
4  -1
5  -5.4
6  +1E+1
7  10e+5
8  2.5E-3
9  -3E
10 5e
11 4E+  10
12 4e-5.33333
13 1.e10.0
```

```
D:\2022春\编译原理\课设\Tiny+\Debug>Tiny+.exe float_test.TNY

TINY COMPILATION: float_test.TNY
   1: 1.0
        1: NUM, val= 1.0
   2: 3.333333
        2: NUM, val= 3.333333
   3: 2.
        3: ERROR: 2.

   4: -1
        4: NUM, val= -1
   5: -5.4
        5: NUM, val= -5.4
   6: +1E+1
        6: NUM, val= +1E+1
   7: 10e+5
        7: NUM, val= 10e+5
   8: 2.5E-3
        8: NUM, val= 2.5E-3
   9: -3E
        9: NUM, val= -3E
  10: 5e
        10: NUM, val= 5e
  11: 4E+  10
        11: ERROR: 4E+
        11: NUM, val= 10
  12: 4e-5.33333
        12: ERROR: 4e-5.
        12: NUM, val= 33333
  13: 1.e10.0   13: ERROR: 1.e
        14: NUM, val= 10.0
        15: EOF
```

5. 实际测试

- 针对格式错误的报错，能够做到识别并进行错误提示。

```
12  4e-5.33333
13  1.e10.0
```

```
12: 4e-5.33333
      12: ERROR: 4e-5.
      12: NUM, val= 33333
13: 1.e10.0    13: ERROR: 1.e
      14: NUM, val= 10.0
      15: EOF
```

5. 实际测试

- 测试总结：

编写三个测试程序，覆盖了所有19种路径。测试结果表明，本次实验中新增部分的预期功能均已达到。

谢谢！

陆逸凡

20191410901163