编译原理 Lab3: 词法分析实验

郑子帆 1120200822

北京理工大学 计算机学院 07112002 班

日期：2023 年 3 月 25 日

摘　　要

本文为北京理工大学《编译原理与设计 2023》课程的 Lab3 实验报告。在本文中，我们学习了 flex 和 Bison 的基本写法和用法，BIT-MinCC 框架的使用方法。结合课上所学内容和实验文档 [2] 中所给出的 C11 语言中的主要单词类的规范定义，利用 Python 编写词法分析器 *MyScanner.py*；将 *MyScanner.py* 嵌入到 BIT-MinCC 框架中；设计简单的 C 语言程序用于测试所写的词法分析器。

1　实验简介

1.1　实验目的

1. 熟悉 C 语言的词法规则，了解编译器词法分析器的主要功能和实现技术，掌握典型词法分析器构造方法，设计并实现 C 语言词法分析器;
2. 了解 Flex 工作原理和基本思想，学习使用工具自动生成词法分析器;
3. 掌握编译器从前端到后端各个模块的工作原理，词法分析模块与其他模块之间的交互过程。

1.2　实验内容

根据 C 语言的词法规则，设计识别 C 语言所有单词类的词法分析器的确定有限状态自动机，并使用 **Python 语言**，采用**程序中心法**实现词法分析器。词法分析器的输入为 C 语言源程序，输出为属性字流。

下载并学习 BIT-MinCC 框架，将自己编写的词法分析器程序 *MyScanner.py* 替换框架中的扫描器。设计测试程序，测试并对比自己编写的词法分析器和默认的分析器的结果。

2 实验准备

2.1 Windows 系统上的 Java 环境配置

首先，在 Java SE 官网上下载对应版本的 jdk，在这里我下载的版本为 19.0.2。

在安装完成之后，进行环境变量的配置。具体地，在系统变量中新建变量，名为 JAVA_HOME，变量值为刚才所安装的 jdk 的具体地址。再新建变量 CLASSPATH。在 Path 中加入%JAVA_HOME%\bin 和%JAVA_HOME%\jre\bin。

在 cmd 中检查 JAVA 是否安装成功，如下图。

```
C:\Users\steve>java --version
java 19.0.2 2023-01-17
Java(TM) SE Runtime Environment (build 19.0.2+7-44)
Java HotSpot(TM) 64-Bit Server VM (build 19.0.2+7-44, mixed mode, sharing)
```

图 1: JAVA 版本

2.2 测试程序的准备

在 BIT-MinCC 框架中自带了一些测试程序，在本实验中，我们选用了 *1_Fibonacci.c* 文件中的程序作为测试程序，其代码详见附录A。

3 实验过程

3.1 总体设计思路

根据实验文档，我们将标识符和关键字、运算符和界限符、字符、字符串、以及整型和浮点型，根据这五类分别进行 DFA 的设计，并采用**超前搜索法**进行扫描。具体如下图：

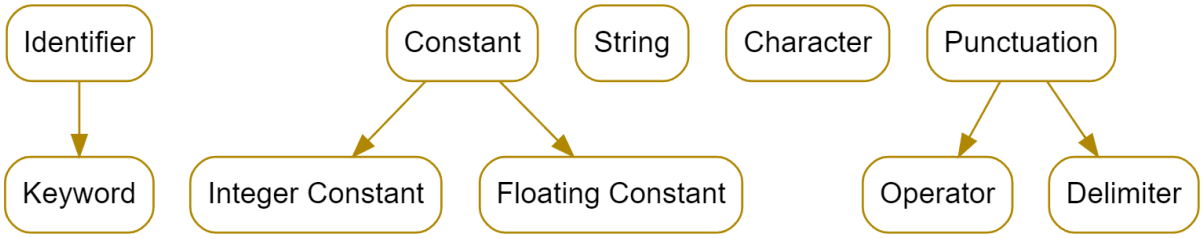

图 2: 五类划分

3.2 具体 DFA 设计

3.2.1 标识符号、关键字 DFA 设计

在程序中，我们先判断它是否是一个标识符 or 关键字，再进一步判断它是否关键字。DFA 如下图。

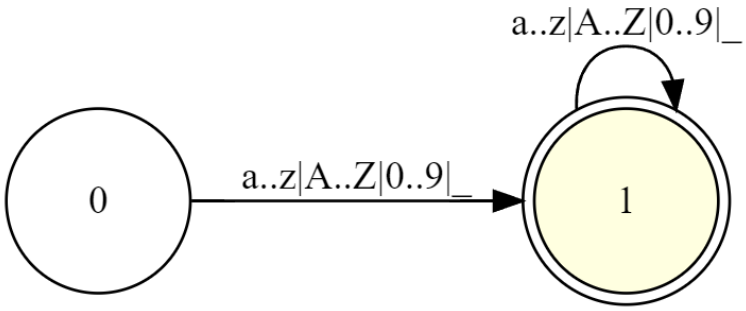

图 3: 标识符号、关键字 DFA 设计

3.2.2 常量 DFA 设计

我们需要将整数和浮点数分开讨论，并要考虑到整数中加 ul，浮点数中 1.0e-4 等等这些情况，具体的 DFA 如下图。

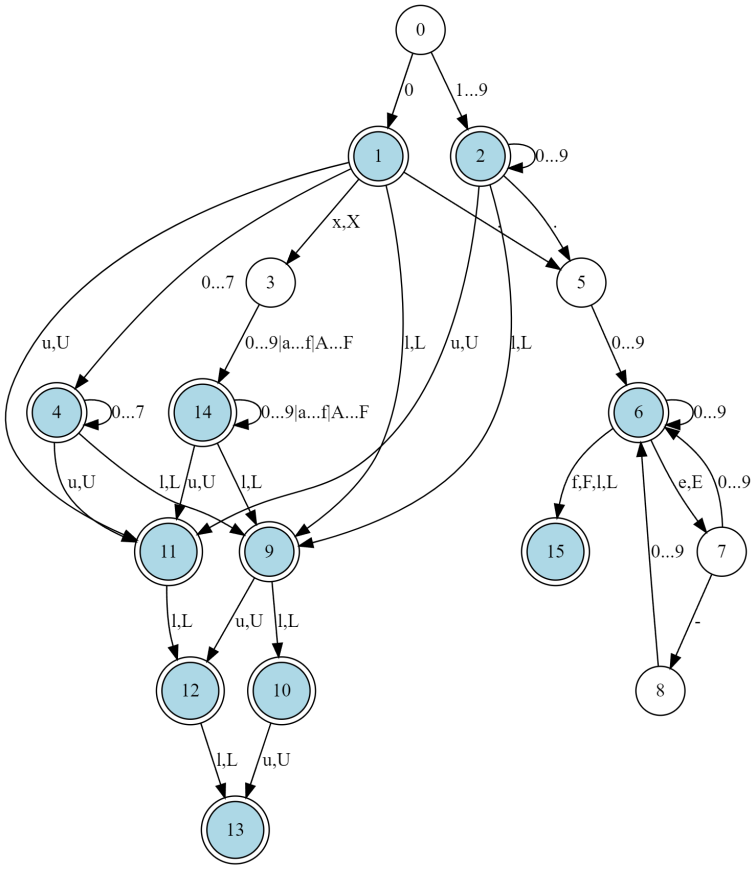

图 4: 常量 DFA 设计

3.2.3 运算符、界限符 DFA 设计

在程序实现中，我们将这些字符分成了三类，分别为：只能表示一字符的算符的字符数组、能表示两个及以上字符的算符的字符数组和界限符数组。

对于界限符数组直接判断，剩下分别进行判断即可。

3.2.4 字符 DFA 设计

DFA 如下图。

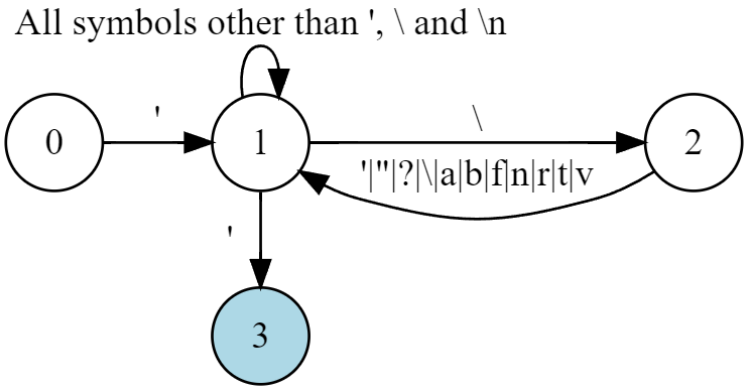

图 5: 字符 DFA 设计

3.2.5 字符串 DFA 设计

和上面类似，DFA 如下图。

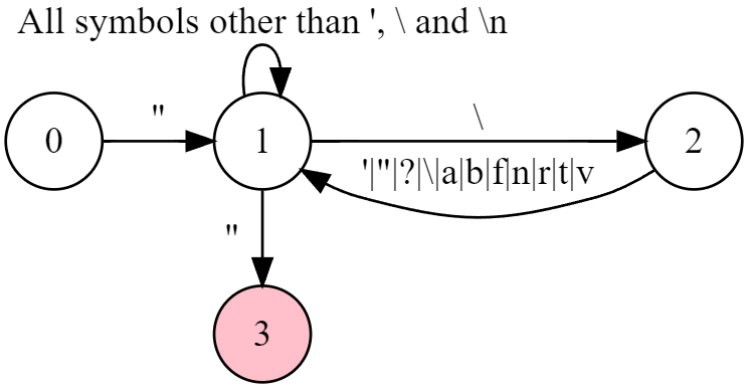

图 6: 字符串 DFA 设计

3.3 程序实现

我采用了程序中心控制法编写了程序，并按照上面所设计的 DFA 进行状态转换。具体代码详见附录B。

3.4 BIT-MiniCC 的使用

3.4.1 框架代码修改

在这次实验中，编写完 python 代码后，在将其嵌入到 BIT-MiniCC 框架后发现程序运行有诸多问题，但是在 python 环境中单独运行 *MyScanner.py* 文件并没有报错。经过调试和试验，发现存在以下几个问题和报错：

1. Python 默认为 ASCII 编码，但是我使用了一些非英文字符，导致报错: Python: SyntaxError: Non-ASCII character...。
2. Jython.jar 包版本过低，不兼容 Python2.7 及更新版本，导致报错: 'with' will become a reserved keyword in Python 2.6。
3. Jython.jar 包包含内容不全，导致会有如下类似报错"No module name sys found"。
4. pyi.exec("sys.argv()") 在加入文件路径到参数后，因为"\" 的存在，会导致传到 python 程序中的文件路径名部分字符被转义，从而找不到文件。

对于上述问题，我对 BIT-MinCC 进行了一些修改。首先，在本地电脑上安装了版本为 2.7.3 的 Jython。由于像 os、string 等包在安装目录下的 Lib 和 Lib/site-packages 中，所以需要在 python 程序中加入这两个路径。

另外，根据编译过程中出现的一些报错，我们需要把实例化的 PythonInterpreter 对象中的属性值重新设置，比如 $python.import.site = false$ 和 $python.console.encoding = UTF-8$。

具体地，我更改的代码均为 *MiniCCompiler.java* 文件中的 MiniCCompiler 类中的 runPy() 方法，更改后的代码如下。

```
private void runPy(String iFile, String oFile, String path)
throws IOException{
    Properties props = new Properties();
    props.setProperty("python.import.site", "false");
    props.setProperty("python.console.encoding", "UTF-8");
    PythonInterpreter.initialize(System.getProperties(), props,
    new String[] {});

    PythonInterpreter pyi = new PythonInterpreter();
    // DIRTY HACK! Apparently the retard who wrote this before don't know

    pyi.exec("import sys");
    System.out.println(MiniCCUtil.escape(iFile));
    pyi.exec("sys.argv = ['<string>', r\'" + MiniCCUtil.escape(iFile)
    + "\', r\'" + MiniCCUtil.escape(oFile) + "\']");
```

```
    pyi.exec("sys.path.append('D:/Program Files (x86)/Jython2.7.3/Lib')")
    pyi.exec("sys.path.append('D:/Program Files (x86)/Jython2.7.3/Lib/sit
    pyi.setOut(System.out);
    pyi.execfile(path);
    pyi.cleanup();
    return;
}
```

3.4.2 配置文件修改

根据 BIT-MinCC 简介文档 [1]，我可以将自己编写的 python 版本的扫描器替换 BIT-MinCC 自带的扫描器并进行测试。

具体地，我们需要把除 Scan 外的其余步骤中的 skip 属性设为 true，即我们只执行扫描部分。另外，还需要将 Scan 中的 type 属性设为 python，path 设为 *MyScanner.py* 的路径。具体的配置后的代码如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<config name="config.xml">
  <phases>
    <phase>
      <phase skip="true" type="java" path="" name="preprocess" />
      <phase skip="false" type="python"
      path="C:\Users\steve\Desktop\Compiler\BIT-MiniCC-master\src\bit\min
      <phase skip="true" type="java" path="" name="parse" />
      <phase skip="true" type="java" path="" name="semantic" />
      ...
    </phase>
  </phases>
</config>
```

4 实验结果测试

使用 *1_Fibonacci.c* 程序，分别用我自己编写的词法扫描器和 BIT-MinCC 中自带的词法扫描器进行扫描，得到的结果如下两张图。

```
[@0,0:2='int',<'int'>,1:0]
[@1,4:12='fibonacci',<Identifier>,1:4]
[@2,13:13='(',<'('>,1:13]
[@3,14:16='int',<'int'>,1:14]
[@4,18:20='num',<Identifier>,1:18]
[@5,21:21=')',<')'>,1:21]
[@6,22:22='{',<'{'>,1:22]
[@7,25:27='int',<'int'>,2:1]
[@8,29:31='res',<Identifier>,2:5]
```

图 7: BIT-MinCC 自带扫描器结果

```
[@0,0:2='int',<'int'>,1:0]
[@1,4:12='fibonacci',<Identifier>,1:4]
[@2,13:13='(',<'('>,1:13]
[@3,14:16='int',<'int'>,1:14]
[@4,18:20='num',<Identifier>,1:18]
[@5,21:21=')',<')'>,1:21]
[@6,22:22='{',<'{'>,1:22]
[@7,25:27='int',<'int'>,2:1]
[@8,29:31='res',<Identifier>,2:5]
```

图 8: MyScanner.py 结果

可以看到结果一样。

5 实验心得

本次实验可以说是整个编译原理实验中的第一步，后续的实验都要基于语法分析器开展，所以在设计语法分析器时，要将所有的单词类型都考虑到，否则语法分析器的输出结果错误的话，后续的实验也很难继续开展。总的来说，语法分析器的实现需要的是耐心，做到不重不漏。在 bit-mini-cc 框架和 c 语言 +bison 之间，我选择了 bit-mini-cc，但由于对 java 不够熟悉，所以没有选择 jflex 自动生成语法分析器，转而自己设计状态，用 python 语言编写语法分析器程序，再修改 bit-mini-cc 中的配置文件，放入该框架运行。过程中遇到了一些难题，也通过不同方法较好地解决了。做完本次实验后，我对后面的实验也产生了更大的兴趣，产生了自己一步步做出 c 语言编译器的实感。

参考文献

[1] *BIT-MinCC 简介.pdf*. zh. 2023.

[2] *Lab3-词法分析实验.pdf*. zh. 2023.

A 测试程序 *1_Fibonacci.c*

```
int fibonacci(int num){
    int res;
    if(num < 1){
        res = 0;
    }else if(num <= 2){
        res = 1;
    }else{
        res = fibonacci(num-1)+fibonacci(num-2);
    }
    return res;
}
int main(){
    Mars_PrintStr("Please input a number:\n");
    int n = Mars_GetInt();
    int res = fibonacci(n);
    Mars_PrintStr("This number's fibonacci value is :\n");
    Mars_PrintInt(res);
    return 0;
}
```

B MyScanner 程序

```
# -*- coding: utf-8 -*-
#! /usr/local/bin/python3

import os
import sys
import string
import xml.etree.cElementTree as ElementTree
from xml.dom import minidom

# 关键词
cKeywords = ['auto', 'break', 'case', 'char', 'const',
             'continue', 'default', 'do', 'double', 'else',
```

```
                'enum', 'extern', 'float', 'for', 'goto',
                'if', 'inline', 'int', 'long', 'register',
                'restrict', 'return', 'short', 'signed', 'sizeof',
                'static', 'struct', 'switch', 'typedef', 'union',
                'unsigned', 'void', 'volatile', 'while']
# 转义字符
cEscSequence = ['\'', '"', '?', '\\', 'a', 'b', 'f', 'n', 'r', 't', 'v']
# 运算符
cOperator = ['+', '-', '&', '*', '~', '!', '/',
                '^', '%', '=', '.', ':', '?', '#', '<', '>', '|', '`']
# 可作为二元运算符首字符的算符
cBinaryOp = ['+', '-', '>', '<', '=', '!',
                '&', '|', '*', '/', '%', '^', '#', ':', '.']
# 界限符
cDelimiter = ['[', ']', '(', ')', '{', '}', '\'', '"', ',', ';', '\\']

# 指针查找位置
index = 0

# Token 属性
codeNum = 1
codeType = ''
codeLine = 1
codeValue = ''
codeValid = 0

# 自动机状态
charState = 0
stringState = 0
constantState = 0
operatorState = 0


def preProcess(content):
    code = ''
    for line in content:
```

```
        if line != '\n':
            code = code + line
        else:
            code = code + line
    code = code + '@'    # 加一个 @ 表示 EOF
    return code


def scanner(code):
    # 从左到右分别为: 当前扫描代码位置、当前识别符数、当前代码行、识别到词语的类别、识
    global index, codeNum, codeLine, codeType, codeValue
    codeType = ''
    codeValue = ''

    # 当前识别字符
    character = code[index]
    index = index + 1

    # 空格直接跳过
    while character == ' ':
        character = code[index]
        index = index + 1

    # 标识符 or 关键字
    if character.isalpha() or character == '_':
        while character.isalpha() or character.isdigit() or character ==
            codeValue = codeValue + character
            character = code[index]
            index = index + 1
        codeType = 'identifier'
        index = index - 1
        # 关键字
        for keyword in cKeywords:
            if codeValue == keyword:
                codeType = 'keyword'
                break
```

```
# 字符串
elif character == '"':
    global stringState
    while index < len(code):
        codeValue = codeValue + character
        if stringState == 0:
            if character == '"':
                stringState = 1
        elif stringState == 1:
            if character == '\\':
                stringState = 3
            elif character == '"':
                stringState = 2
                break
        elif stringState == 2:
            break
        elif stringState == 3:
            if character in cEscSequence:
                stringState = 1
        character = code[index]
        index = index + 1

    if stringState == 2:
        codeType = 'string'
        stringState = 0
    else:
        print('Illegal string.')
        stringState = 0

# 字符
elif character == '\'':
    global charState
    while index < len(code):
        codeValue = codeValue + character
        if charState == 0:
```

```
                if character == '\'':
                    charState = 1
            elif charState == 1:
                if character == '\'':
                    charState = 2
                    break
                elif character == '\\':
                    charState = 3
            elif charState == 2:
                break
            elif charState == 3:
                if character in cEscSequence:
                    charState = 1
            character = code[index]
            index = index + 1
        if charState == 2:
            codeType = 'character'
            charState = 0
        else:
            codeType = 'illegal char'
            charState = 0

    # 整型、浮点型变量
    elif character.isdigit():
        global constantState
        while character.isdigit() or character in '-.xXeEaAbBcCdDfFuUlL':
            codeValue = codeValue + character
            if constantState == 0:
                if character == '0':
                    constantState = 1
                elif character in '123456789':
                    constantState = 2
            elif constantState == 1:
                if character in 'xX':
                    constantState = 3
                elif character in '01234567':
```

```
            constantState = 4
        elif character == '.':
            constantState = 5
        elif character in 'lL':
            constantState = 9
        elif character in 'uU':
            constantState = 11
        else:
            constantState = -1
    elif constantState == 2:
        if character.isdigit():
            constantState = 2
        elif character == '.':
            constantState = 5
        elif character in 'lL':
            constantState = 9
        elif character in 'uU':
            constantState = 11
    elif constantState == 3:
        if character in 'aAbBcCdDeEfF' or character.isdigit():
            constantState = 14
    elif constantState == 4:
        if character in '01234567':
            constantState = 4
        elif character in 'lL':
            constantState = 9
        elif character in 'uU':
            constantState = 11
        else:
            constantState = -1
    elif constantState == 5:
        if character.isdigit():
            constantState = 6
    elif constantState == 6:
        if character.isdigit():
            constantState = 6
```

```
        elif character in 'eE':
            constantState = 7
        elif character in 'fFlL':
            constantState = 15
        else:
            constantState = -1
    elif constantState == 7:
        if character.isdigit():
            constantState = 6
        elif character == '-':
            constantState = 8
    elif constantState == 8:
        if character.isdigit():
            constantState = 6
    elif constantState == 9:
        if character in 'lL':
            constantState = 10
        elif character in 'uU':
            constantState = 12
        else:
            constantState = -1
    elif constantState == 10:
        if character in 'uU':
            constantState = 13
        else:
            constantState = -1
    elif constantState == 11:
        if character in 'lL':
            constantState = 12
        else:
            constantState = -1
    elif constantState == 12:
        if character in 'lL':
            constantState = 13
        else:
            constantState = -1
```

```
            elif constantState == 14:
                if character.isdigit() or character in 'aAbBcCdDeEfF':
                    constantState = 14
                elif character in 'lL':
                    constantState = 9
                elif character in 'uU':
                    constantState = 11
                else:
                    constantState = -1
            elif constantState == 13 or constantState == 15:
                if character:
                    constantState = -1

            character = code[index]
            index = index + 1
        index = index - 1
        if constantState in (1, 2, 4, 9, 10, 11, 12, 13, 14):
            codeType = 'integer constant'
            constantState = 0
        elif constantState == 6 or constantState == 15:
            codeType = 'floating constant'
            constantState = 0
        else:
            codeType = 'illegal constant'
            constantState = 0

    # 界限符
    elif character in cDelimiter:
        codeValue = codeValue + character
        codeType = 'delimiter'

    # 运算符
    elif character in cOperator:
        global operatorState
        while character in cOperator:
            codeValue = codeValue + character
```

```
if operatorState == 0:
    if not character in cBinaryOp:
        operatorState = 20
        break
    else:
        if character == '+':
            operatorState = 2
        elif character == '-':
            operatorState = 3
        elif character == '<':
            operatorState = 4
        elif character == '>':
            operatorState = 5
        elif character == '=':
            operatorState = 6
        elif character == '!':
            operatorState = 7
        elif character == '&':
            operatorState = 8
        elif character == '|':
            operatorState = 9
        elif character == '*':
            operatorState = 10
        elif character == '/':
            operatorState = 11
        elif character == '%':
            operatorState = 12
        elif character == '^':
            operatorState = 13
        elif character == '#':
            operatorState = 14
        elif character == ':':
            operatorState = 15
        elif character == '.':
            operatorState = 18
```

```
        elif operatorState == 1:
            break
        elif operatorState == 2:
            if character in '+=':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 3:
            if character in '-=':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 4:
            if character in '=:%':
                operatorState = 1
                break
            elif character == '<':
                operatorState = 16
            else:
                operatorState = -1
        elif operatorState == 5:
            if character in '=':
                operatorState = 1
                break
            elif character == '>':
                operatorState = 17
            else:
                operatorState = -1
        elif operatorState == 6:
            if character == '=':
                operatorState = 1
                break
            else:
                operatorState = -1
```

```
        elif operatorState == 7:
            if character == '=':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 8:
            if character in '&=':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 9:
            if character in '|=':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 10:
            if character == '=':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 11:
            if character == '=':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 12:
            if character in '=>:':
                operatorState = 1
                break
            else:
                operatorState = -1
```

```
        elif operatorState == 13:
            if character == '=':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 14:
            if character == '#':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 15:
            if character == '>':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 16:
            if character == '=':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 17:
            if character == '=':
                operatorState = 1
                break
            else:
                operatorState = -1
        elif operatorState == 18:
            if character == '.':
                operatorState = 19
            else:
                operatorState = -1
        elif operatorState == 19:
```

```
                if character == '.':
                    operatorState = 1
                    break
                else:
                    operatorState = -1

            character = code[index]
            index = index + 1

        if operatorState >= 2 and operatorState <= 18:
            index = index - 1
            codeType = 'Unary operator'
            operatorState = 0
        elif operatorState == 20:
            codeType = 'Unary operator'
            operatorState = 0
        elif operatorState == 1:
            codeType = 'Multicast operator'
            operatorState = 0
        else:
            index = index - 1
            codeType = 'Illegal operator'
            operatorState = 0
    # 换行
    elif character == '\n':
        codeLine = codeLine + 1
    # EOF
    elif character == '@':
        codeValue = codeValue + character
        codeType = 'END OF FILE'


def main():
    # 读文件
    filePath = sys.argv[1]
    f = open(filePath, 'r')
```

```
content = f.readlines()

# 预处理文件
code = preProcess(content)
print(code)

fileName = os.path.basename(filePath)
# XML output file name
xmlFileName = os.path.splitext(fileName)[0] + '.token.xml'

# Create XML tree
xmlTree = ElementTree.Element('project', {
    'name': fileName
})
xmlTokens = ElementTree.SubElement(xmlTree, 'tokens')

# 扫描
global codeNum
while index <= len(code) - 1:
    scanner(code)
    # Print identified word type and word itself
    if codeType != '':
        xmlToken = ElementTree.SubElement(xmlTokens, 'token')

        xmlNumber = ElementTree.SubElement(xmlToken, 'numbers')
        xmlValue = ElementTree.SubElement(xmlToken, 'value')
        xmlType = ElementTree.SubElement(xmlToken, 'keyword')
        xmlLine = ElementTree.SubElement(xmlToken, 'line')
        xmlValid = ElementTree.SubElement(xmlToken, 'true')

        xmlNumber.text = str(codeNum)
        xmlValue.text = str(codeValue)
        xmlType.text = codeType.lower()
        xmlLine.text = str(codeLine)
        if not 'illegal' in codeType.lower():
            xmlValid.text = 'true'
```

```
            else:
                xmlValid.text = 'false'

            # Test output, debugging purposes
            print('Num', '{:>2}'.format(codeNum), 'Line', '{:>2}'.format(
                  '{:>18}'.format(codeType.upper()) + ': ' + '{:<5}'.form

            codeNum = codeNum + 1

    # Turn XML tree to string
    xmlString = ElementTree.tostring(xmlTree)
    # Set XML indent and encoding (This returns a byte for the file to re
    xml = minidom.parseString(xmlString).toprettyxml(
        indent='  ', encoding='utf-8')
    # Write XML to file
    f = open(xmlFileName, 'wb')
    f.write(xml)
    print('Written XML token processing results to file:', xmlFileName)


if __name__ == "__main__":
    main()
```