院系	年级专业	姓名	学号	实验日期
计算机学院	2019计科	吴家隆	1915404063	2021.9.18

编程语言：**python 3.9**

PLY入门

实验内容

编写程序，识别如下程序中的词法单元

```
int asd = 0;
int bc = 10;
while ( asd < bc)
{
    if(bc - asd < 2)
        cout<<"they are close."<<endl;
    asd = asd + 1;
}
```

代码功能

标记列表

词法分析器必须提供一个标记的列表，这个列表将所有可能的标记告诉分析器，用来执行各种验证。

给定如下标记列表

```
tokens = (
    'IDENTIFIER', 'NUMBER', 'ASSIGN', 'ADDRESS','LSHIFT','RSHIFT','LT', 'GT',
    'SELF_PLUS', 'SELF_MINUS', 'PLUS', 'MINUS', 'MUL', 'DIV', 'GTE',
'LTE','LL_BRACKET', 'RL_BRACKET', 'LB_BRACKET',
    'RB_BRACKET', 'LM_BRACKET', 'RM_BRACKET', 'COMMA',
'DOUBLE_QUOTE','SEMICOLON', 'SHARP','INCLUDE', 'INT', 'FLOAT',
    'CHAR', 'DOUBLE', 'FOR', 'IF', 'ELSE', 'WHILE', 'DO',
'RETURN','STRING_LITERAL'
)
```

标记的规则

每种标记用一个正则表达式规则来表示，每个规则需要以"t_"开头声明，表示该声明是对标记的规则定义。

对于简单的标记

```
t_PLUS = r'\+'
t_MINUS = r'-'
t_MUL = r'\*'
t_DIV = r'/'
t_ASSIGN = r'='
t_ADDRESS = r'&'
```

```
t_LSHIFT = r'<<'
t_RSHIFT = r'>>'
t_LT = r'<'
t_GT = r'>'
t_SELF_PLUS = r'\+\+'
t_SELF_MINUS = r'--'
t_LTE = r'<='
t_GTE = r'>='
t_LL_BRACKET = r'\('
t_RL_BRACKET = r'\)'
t_LB_BRACKET = r'\{'
t_RB_BRACKET = r'}'
t_LM_BRACKET = r'\['
t_RM_BRACKET = r']'
t_COMMA = r','
t_DOUBLE_QUOTE = r'"'
t_SEMICOLON = r';'
t_SHARP = r'\#'
t_STRING_LITERAL = r'"[^"]*"'#处理字符串
```

这里，紧跟在t_后面的单词，必须跟标记列表中的某个标记名称对应。

如果需要执行动作的话，规则可以写成一个方法。

```
def t_IDENTIFIER(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = reserved.get(t.value, 'IDENTIFIER')    # Check for reserved words
    return t
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)
    return t
```

如果使用方法的话，正则表达式写成方法的文档字符串。方法总是需要接受一个 LexToken 实例的参数，该实例有一个`t.type`的属性（字符串表示）来表示标记的类型名称，`t.value`是标记值（匹配的实际的字符串），`t.lineno`表示当前在源输入串中的作业行，`t.lexpos`表示标记相对于输入串起始位置的偏移。默认情况下，`t.type`是以t_开头的变量或方法的后面部分。方法可以在方法体里面修改这些属性。但是，如果这样做，应该返回结果token，否则，标记将被丢弃。

在lex内部，lex.py用`re`模块处理模式匹配，在构造最终的完整的正则式的时候，用户提供的规则按照下面的**顺序**加入：

1. 所有由方法定义的标记规则，按照他们的出现顺序依次加入
2. 由字符串变量定义的标记规则按照其正则式长度倒序后，依次加入（长的先入）
3. 顺序的约定对于精确匹配是必要的。比如，如果你想区分'='和'=='，你需要确保'=='优先检查。如果用字符串来定义这样的表达式的话，通过将较长的正则式先加入，可以帮助解决这个问题。用方法定义标记，可以显示地控制哪个规则优先检查。

为了**处理保留字**，在方法里面作了特殊的查询：

```
reserved = {
    'include': 'INCLUDE',
    'int': 'INT',
    'float': 'FLOAT',
    'char': 'CHAR',
    'double': 'DOUBLE',
    'for': 'FOR',
    'if': 'IF',
    'else': 'ELSE',
    'while': 'WHILE',
    'do': 'DO',
    'return': 'RETURN'
}
```

```
def t_IDENTIFIER(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = reserved.get(t.value, 'IDENTIFIER')    # Check for reserved words
    return t
```

这样做可以大大减少正则式的个数，并稍稍加快处理速度。

行号和位置信息

默认情况下，lex.py对行号一无所知。因为lex.py根本不知道何为"行"的概念（换行符本身也作为文本的一部分）。不过，可以通过写一个特殊的规则来记录行号：

```
def t_newline(t):
    r'\n+'
    t.lexer.lineno += t.value.count("\n")
```

错误处理

在词法分析中遇到非法字符时，`t_error()` 用来处理这类错误。这种情况下，`t.value` 包含了余下还未被处理的输入字串，在之前的例子中，错误处理方法是这样的：

```
def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)
```

构建和使用lex

读取文件夹中的prog.txt，输出每个tok

```
lexer = lex.lex()
with open("prog.txt","r",encoding= "utf-8") as x:
    str = x.read()
lexer.input(str)
while True:
    tok = lexer.token()
    if not tok: break      # No more input
    print(tok)
```

实验结果

```
LexToken(INT,'int',1,0)
LexToken(IDENTIFIER,'asd',1,4)
LexToken(ASSIGN,'=',1,8)
LexToken(NUMBER,0,1,10)
LexToken(SEMICOLON,';',1,11)
LexToken(INT,'int',2,13)
LexToken(IDENTIFIER,'bc',2,17)
LexToken(ASSIGN,'=',2,20)
LexToken(NUMBER,10,2,22)
LexToken(SEMICOLON,';',2,24)
LexToken(WHILE,'while',3,26)
LexToken(LL_BRACKET,'(',3,32)
LexToken(IDENTIFIER,'asd',3,34)
LexToken(LT,'<',3,38)
LexToken(IDENTIFIER,'bc',3,40)
LexToken(RL_BRACKET,')',3,42)
LexToken(LB_BRACKET,'{',4,44)
LexToken(IF,'if',5,47)
LexToken(LL_BRACKET,'(',5,49)
LexToken(IDENTIFIER,'bc',5,50)
LexToken(MINUS,'-',5,53)
LexToken(IDENTIFIER,'asd',5,55)
LexToken(LT,'<',5,59)
LexToken(NUMBER,2,5,61)
LexToken(RL_BRACKET,')',5,62)
LexToken(IDENTIFIER,'cout',6,66)
```

```
LexToken(LSHIFT,'<<',6,70)
LexToken(STRING_LITERAL,'"they are close."',6,72)
LexToken(LSHIFT,'<<',6,89)
LexToken(IDENTIFIER,'endl',6,91)
LexToken(SEMICOLON,';',6,95)
LexToken(IDENTIFIER,'asd',7,98)
LexToken(ASSIGN,'=',7,102)
LexToken(IDENTIFIER,'asd',7,104)
LexToken(PLUS,'+',7,108)
LexToken(NUMBER,1,7,110)
LexToken(SEMICOLON,';',7,111)
LexToken(RB_BRACKET,'}',8,113)
```

运行结果正确

运行说明

解压ply，运行clex.py，可以修改prog.txt来修改要识别语法单元的程序。

运行后将按顺序打印出结果。

该程序只适用于部分C语言风格的代码