编译原理实践第3次课
PLY入门

张昊　1927405160

概述

使用 Python3 实现了类 C++ 语法的词法分析。

项目只有一个源文件：`cpp_lex.py`

编程环境说明

- **语言**：Python3
- **依赖**：ply 库（使用附件中的安装包）
- **文件编码**：UTF-8
- **开发&测试环境**：Ubuntu 20.04, Python 3.8.10

源代码功能

代码中封装了一个词法分析器类 `CPlusPlusLexer`，将 lexer 的操作定义在类中：

- 构造函数初始化成员变量 `lexer` 为 None，使用 `build` 方法显示构造 Lexer，此时 `lexer` 初始化。
- 类中以类变量的形式定义了保留字与 token 列表等预定义常量。
- 提供识别新行的方法 `t_newline`（正则表达式：`r'\n+'`），该方法将行号 + 1。
- 定义了忽略的字符为：`' \t'`（空格和制表符）
- 提供错误处理的方法 `t_error`，该方法将不能识别的 token 识别出来并忽略掉。
- 定义了一个迭代器 `lex_tokens` 该迭代器接受一个保存有类 C++ 源代码的字符串，通过遍历 `self.lexer.token()`，从 Lexer 中获取词法单元。
- token 名与正则表达式定义见 附录1 。

定义为函数的 token 解释说明如下：

1. **字符串** STRING

   单纯使用正则表达式 `r'"(.*)"'` 无法解决 `"string" <token> "string"` 的输入，会将中间的 token 包含在字符串中，从而产生错误。

   原因是这一文法不属于正则文法，无法使用正则表达式解决。

解决办法是在匹配到整个字符串的基础上做一次遍历，找到第一个结束引号的位置：除第一个引号外，首次出现的不在 `\` 字符后面的引号；并修改 `lexer.lexpos`（分析点）和 value 即可。

2. **标识符** ID

   由于关建字（保留字）会影响标识符的判断，需要提前检查匹配到的是否为保留字。

   若是则申明其类型为相应保留字，否则为 ID。

3. **数字** NUMBER

   数字的词法单元的值应为整型数字，而不是字符串，故需要在函数中将其转换为字符串再存入词法单元的值。

运行示例

以在 Linux 系统下运行为例，输入文件为 `prog.txt`：

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

运行代码，接受一个命令行参数作为输入文件路径，如不提供，则从标准输入读取。

```
$ python3 ./cpp_lex.py prog.txt
```

或

```
$ python3 ./cpp_lex.py < prog.txt
```

输出如下：

```
<INT, 'int'> <ID, 'asd'> <ASSIGN, '='> <NUMBER, 0> <SEMICOLON, ';'>
<INT, 'int'> <ID, 'bc'> <ASSIGN, '='> <NUMBER, 10> <SEMICOLON, ';'>
<WHILE, 'while'> <LPAREN, '('> <ID, 'asd'> <LT, '<'> <ID, 'bc'> <RPAREN, ')'>
<LBRACE, '{'>
<IF, 'if'> <LPAREN, '('> <ID, 'bc'> <MINUS, '-'> <ID, 'asd'> <LT, '<'> <NUMBER,
2> <RPAREN, ')'>
<COUT, 'cout'> <INSERT, '<<'> <STRING, '"they are close."'> <INSERT, '<<'> <ENDL,
'endl'> <SEMICOLON, ';'>
<ID, 'asd'> <ASSIGN, '='> <ID, 'asd'> <PLUS, '+'> <NUMBER, 1> <SEMICOLON, ';'>
<RBRACE, '}'>
```

输出结果为输入程序的词法单元，同一行的词法单元输出在一行中。

附录1 正则表达式定义对照表

token	正则表达式	说明
PLUS	`r'\+'`	加号
MINUS	`r'-'`	减号
TIMES	`r'\*'`	乘号
DIVIDE	`r'/'`	除号
LPAREN	`r'\('`	左小括号
RPAREN	`r'\)'`	右小括号
LBRACE	`r'\{'`	左大括号
RBRACE	`r'\}'`	右大括号
ASSIGN	`r'='`	赋值
INSERT	`r'<<'`	流插入运算符
LT	`r'<'`	小于
GT	`r'>'`	大于
SEMICOLON	`r';'`	分号
STRING	`r'"(.*)"'`	字符串（定义为函数）
ID	`r'[a-zA-Z]+'`	标识符（定义为函数）
NUMBER	`r'\d+'`	数字（定义为函数）
IF	`'if'`	if 关键字
WHILE	`'while'`	while 关键字
INT	`'int'`	int 关键字
COUT	`'cout'`	cout （按关键字处理）
ENDL	`'endl'`	endl （按关键字处理）