编译原理实践第13次课
基于PLY的Python解析-3

张昊　1927405160

概述

使用 Python3 以及 PLY 库实现了简易的 Python 解析器。主要涉及的知识有语法分析，语法制导翻译。
除了赋值语句、完整的四则运算、print语句、选择语句、循环语句、列表、len函数、下标访问，实现了**函数声明与函数调用**的解析。同时也解决了一些上次实验遗留的bug。

编程说明

- **语言**：Python 3
- **文件编码**：UTF-8
- **依赖**：PLY
- **测试环境**：Python 3.8.10

Python程序的解析

设计了如下文法来实现词法分析：

```
运算符定义略
保留字：print len if elif while for break and or def return
ID -> [A-Za-z_][A-Za-z0-9_]*
NUMBER -> \d+
```

【注】这里NUMBER只能识别非负整数，对于负号的实现应该在语法分析中定义产生式来实现。（这里是上一个实验报告遗留的一个bug）但是样例中没有负数出现。因此，这一版本的代码暂不支持纯负数的解析（可以通过0-num来间接实现）。

> 识别 ID，首先检查是否为保留字，若是则申明其类型，否则为 ID

设计了如下语法来实现语法分析

```
program : statements
statements : statements statement | statement
statement : assignment | expr | print | if | while | for | break |
function | return
assignment : variable ASSIGN expr
           | variable MINEQUAL expr
```

```
            | variable PLUSEQUAL expr
            | variable DPLUS
            | variable DMINUS
variable : variable LBRACKET expr RBRACKET | ID
expr : expr PLUS term | expr MINUS term | term | array
term : term TIMES factor | term DIVIDE factor | term EDIVIDE factor | factor
factor : variable | NUMBER | len | call | LPAREN expr RPAREN
exprs : exprs COMMA expr | expr
len : LEN LPAREN variable RPAREN
print : PRINT LPAREN exprs RPAREN | PRINT LPAREN RPAREN
array : LBRACKET exprs RBRACKET | LBRACKET RBRACKET
condition : condition OR join | join
join : join AND equality | equality
equality : equality EQ rel | equality NE rel | rel
rel : expr LT expr | expr LE expr | expr GT expr | expr GE expr | expr
if : IF LPAREN condition RPAREN LBRACE statements RBRACE
   | IF LPAREN condition RPAREN LBRACE statements RBRACE else
else : ELIF LPAREN condition RPAREN LBRACE statements RBRACE
     | ELIF LPAREN condition RPAREN LBRACE statements RBRACE else
     | ELSE LBRACE statements RBRACE
while : WHILE LPAREN condition RPAREN LBRACE statements RBRACE
for : FOR LPAREN assignment SEMICOLON condition SEMICOLON assignment RPAREN LBRACE statements RBRACE
break : BREAK
function : DEF ID LPAREN args RPAREN LBRACE statements RBRACE | DEF ID LPAREN RPAREN LBRACE statements RBRACE
args : args COMMA ID | ID
call : ID LPAREN exprs RPAREN | ID LPAREN RPAREN
return : RETURN | RETURN exprs
```

其中：

- expr、term、factor 定义了四则运算的语法。
- exprs 实现了函数的实参传递。
- args 实现了函数的形参定义。
- variable 定义了ID和数组下标访问语法，使用了C++中左值的概念，为可读可写的引用，对其的读写需要同过符号表。
- len 定义了 Python 函数 len() 的语法，规定传入的只能是左值。
- array 利用 exprs 实现了 Python 的列表定义。
- 将赋值、（右）自增、（右）自减合并为赋值语句，并实现了 `+=`、`-=` 运算符。
  这里考虑到解析的代码中只是对变量进行自增，故未实现左自增（对称实现即可）。
- condition、equality、join、rel 实现了判断中的条件，并在实现了 `and` 和 `or` 表达式以及优

先级。
- if、for、while 实现了分支和循环语句。本次重写了if语句，实现了完整意义上的多路分支语法。
- function 实现了函数的定义，call 实现了函数的调用。

另外定义了一系列节点，与语法分析过程中相对应。

语法制导翻译

如上一小节的代码所示，每个节点都有一个 value 属性（左值Variable的value属性被禁用，而是应该通过符号表来检索值），
用来保存节点的值。
（如没有值则为None，数值类型则会赋给一个自定义的单例NIL，表示未赋值的变量，且与None的区分）
另外设计了一个符号表，用以保存每个变量的值。
具体地，当使用赋值语句为一个变量赋值时，会在符号表中添加名为该变量名的记录；
当访问一个变量的值时，会到表中查找该变量的值，如不存在则报错。
函数同变量存于同一个变量表中。

对于if、for、while，提前翻译条件condition，根据结果来判断分支是否执行或循环是否继续。

对于循环，定义变量loop_flag用来标识循环的层数，大于0为循环层数，等于0为不在循环内，小于0非法。

对于break，定义变量break_flag来指示是否遇到了break，
如果遇到了，则后续节点都不翻译，并跳出循环。

对于函数定义，定义了一个Function类，保存函数的名称、形参和函数体（语法树），调用时传入实参和当前的变量表，遍历语法树（函数体）。

为实现函数的递归调用，将翻译函数封装类，从而可以利用对象来隔离状态，并通过函数调用来实现函数调用的压栈、出栈动作。

以上代码实现详见 translate.py 。

其余部分，采用深度优先的顺序遍历整个语法树，具体实现详见代码。

运行

项目结构为：

```
.
├── README.pdf     # 本文档
├── quick_sort.py # 输入文件
├── main.py        # 主程序
├── node.py        # 节点定义文件
├── parser.out     # PLY生成的文件
├── parsetab.py    # PLY生成的文件
├── py_lex.py      # 词法分析文件
├── py_yacc.py     # 语法分析文件
└── translation.py     # 翻译器
```

主程序接受一个参数，为输入文件的路径。运行方法如下：

```
$ python3 main.py <py-file>
```

输入文件：quick_sort.py

```
def quick_sort(array, left, right){
    if(left >= right){
        return
    }
    low = left
    high = right
    key = array[low]
    while(left < right){
        while(left < right and array[right] > key){
            right -= 1
        }
        array[left] = array[right]
        while(left < right and array[left] <= key){
            left += 1
        }
        array[right] = array[left]
    }
    array[right] = key
    quick_sort(array, low, left - 1)
    quick_sort(array, left + 1, high)
}


a=[1,2,4,3,6,5,7,3]

```

```
quick_sort(a,0,len(a)-1)

print(a)
```

运行结果如下：

```
语法树:  [Program [Statements [Statements [Statements [Statements
[Statement [Function [def] ID('quick_sort') [Args [Args [Args
ID('array')] ID('left')] ID('right')] [Statements [Statements
[Statements [Statements [Statements [Statements [Statements [Statements
[Statement [If [if] [Condition [Join [Equality [Relation [Expr [Term
[Factor [Variable ID('left')]]]] [≥] [Expr [Term [Factor [Variable
ID('right')]]]]]]]] [Statements [Statement [Return [return]]]] ]]]
[Statement [Assignment [Variable ID('low')] [Expr [Term [Factor
[Variable ID('left')]]]]]]] [Statement [Assignment [Variable
ID('high')] [Expr [Term [Factor [Variable ID('right')]]]]]]] [Statement
[Assignment [Variable ID('key')] [Expr [Term [Factor [Variable
[Variable ID('array')] [Expr [Term [Factor [Variable ID('low')]]]]
]]]]]] [Statement [While [Condition [Join [Equality [Relation [Expr
[Term [Factor [Variable ID('left')]]]] [<] [Expr [Term [Factor
[Variable ID('right')]]]]]]] [Statements [Statements [Statements
[Statements [Statement [While [Condition [Join [Join [Equality
[Relation [Expr [Term [Factor [Variable ID('left')]]]] [<] [Expr [Term
[Factor [Variable ID('right')]]]]]]] [and] [Equality [Relation [Expr
[Term [Factor [Variable [Variable ID('array')] [Expr [Term [Factor
[Variable ID('right')]]]] ]]]] [>] [Expr [Term [Factor [Variable
ID('key')]]]]]]] [Statements [Statement [Assignment [Variable
ID('right')] [-=] [Expr [Term [Factor Number(1)]]]]]] ]]] [Statement
[Assignment [Variable [Variable ID('array')] [Expr [Term [Factor
[Variable ID('left')]]]] ] [Expr [Term [Factor [Variable [Variable
ID('array')] [Expr [Term [Factor [Variable ID('right')]]]] ]]]]]]]
[Statement [While [Condition [Join [Join [Equality [Relation [Expr
[Term [Factor [Variable ID('left')]]]] [<] [Expr [Term [Factor
[Variable ID('right')]]]]]]] [and] [Equality [Relation [Expr [Term
[Factor [Variable [Variable ID('array')] [Expr [Term [Factor [Variable
ID('left')]]]] ]]]] [≤] [Expr [Term [Factor [Variable ID('key')]]]]]]]]
[Statements [Statement [Assignment [Variable ID('left')] [+=] [Expr
[Term [Factor Number(1)]]]]]] ]]] [Statement [Assignment [Variable
[Variable ID('array')] [Expr [Term [Factor [Variable ID('right')]]]] ]
[Expr [Term [Factor [Variable [Variable ID('array')] [Expr [Term
[Factor [Variable ID('left')]]]] ]]]]]] ]]] [Statement [Assignment
[Variable [Variable ID('array')] [Expr [Term [Factor [Variable
ID('right')]]]] ] [Expr [Term [Factor [Variable ID('key')]]]]]]]
```

```
[Statement [Expr [Term [Factor [Call ID('quick_sort') [Exprs [Exprs [Exprs [Expr [Term [Factor [Variable ID('array')]]]]] [Expr [Term [Factor [Variable ID('low')]]]]] [Expr [Expr [Term [Factor [Variable ID('left')]]]] [-] [Term [Factor Number(1)]]]] ]]]]] [Statement [Expr [Term [Factor [Call ID('quick_sort') [Exprs [Exprs [Exprs [Expr [Term [Factor [Variable ID('array')]]]]] [Expr [Expr [Term [Factor [Variable ID('left')]]]] [+] [Term [Factor Number(1)]]]] [Expr [Term [Factor [Variable ID('high')]]]]] ]]]]] ]]] [Statement [Assignment [Variable ID('a')] [Expr [Array [Exprs [Exprs [Exprs [Exprs [Exprs [Exprs [Exprs [Exprs [Expr [Term [Factor Number(1)]]]] [Expr [Term [Factor Number(2)]]]] [Expr [Term [Factor Number(4)]]]] [Expr [Term [Factor Number(3)]]]] [Expr [Term [Factor Number(6)]]]] [Expr [Term [Factor Number(5)]]]] [Expr [Term [Factor Number(7)]]]] [Expr [Term [Factor Number(3)]]]] ]]]]] [Statement [Expr [Term [Factor [Call ID('quick_sort') [Exprs [Exprs [Exprs [Expr [Term [Factor [Variable ID('a')]]]]] [Expr [Term [Factor Number(0)]]]] [Expr [Expr [Term [Factor [Len [len] [Variable ID('a')] ]]]] [-] [Term [Factor Number(1)]]]] ]]]]]] [Statement [Print [print] [Exprs [Expr [Term [Factor [Variable ID('a')]]]]] ]]]]
运行结果:
[1, 2, 3, 3, 4, 5, 6, 7]
当前变量表: {'quick_sort': <Function object 'quick_sort'>, 'a': [1, 2, 3, 3, 4, 5, 6, 7]}
```

> 如果图片不清晰，请点击如下链接：http://repo.holgerbest.top/html/quick_sort.png

语法树: