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	2021.11.29

编程语言：***python3.9***

实验内容

本次学习的语法是**选择语句**和**循环语句**，需要注意的是本次使用的语法做了一些改进，**不是纯粹的*python2*语法。**

需要结合上次课四则运算的解析程序

- 利用PLY实现简单的Python程序的解析

  1.示例程序位于example3/

  2.需要进行解析的文件为binary_search.py和select_sort.py，分别对应二分查找和选择排序。

  binary_search.py:

```
a=[1,2,3,4,5,6,7,8,9,10]

key=3

n=len(a)

begin=0
end=n-1

while(begin<=end){
    mid=(begin+end)//2
    if(a[mid]>key){
        end=mid-1
    }
    elif(a[mid]<key){
        begin=mid+1
    }
    else{
        break
    }
}
```

```
print(mid)
```

select_sort.py:

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

n = len(a)

for (i=0;i < n;i++){
    max_v=a[i]
    i_v=i
    for (j=i;j < n;j++){
        if (a[j] > max_v){
            max_v=a[j]
            i_v=j
        }
    }
    t=a[i]
    a[i]=a[i_v]
    a[i_v]=t
}

print(a)
```

3.需要完成以下内容的解析

if

while

for

4.解析结果以语法树的形式呈现

- 编程实现语法制导翻译

  1.语法树上每个节点有一个属性value保存节点的值

  2.设置一个变量表保存每个变量的值

  3.基于深度优先遍历获取整个语法树的分析结果

  在进行翻译条件语句和循环语句时，不能简单的进行深度优先遍历，要对于某些条件节点进行优先翻译

实验步骤

使用lex进行序列标记

在本次实验中要识别的tokens包括以下

```
tokens = ('VARIABLE', 'NUMBER', 'IF', 'WHILE', 'PRINT', 'FOR', 'LEN', 'ELIF',
'ELSE', 'BREAK')

literals = ['=', '+', '-', '*', '/', '(', ')', '{', '}', '<', '>', '[', ']',
',', ';']
```

ply使用"t_"开头的变量来表示规则。如果变量是一个字符串，那么它被解释为一个正则表达式，匹配值是标记的值。 如果变量是函数，则其文档字符串包含模式，并使用匹配的标记调用该函数。该函数可以自由地修改序列或返回一个新的序列来代替它的位置。 如果没有返回任何内容，则忽略匹配。 通常该函数只更改“value”属性，它最初是匹配的文本。

t_FOR()必须写在t_VARIABLE()前面,否则for关键字会被匹配成VARIABLE

t_LEN()与t_FOR同理

VARIABLE的匹配要放在最后

```python
def t_NUMBER(t):
    r'[0-9]+'
    return t
def t_PRINT(t):
    r'print'
    return t
def t_IF(t):
    r'if'
    return t
def t_WHILE(t):
    r'while'
    return t
def t_FOR(t):
    r'for'
    return t
def t_LEN(t):
    r'len'
    return t
def t_ELIF(t):
    r'elif'
    return t
def t_ELSE(t):
    r'else'
    return t
def t_BREAK(t):
    r'break'
    return t
def t_VARIABLE(t):
    r'[a-zA-Z]+_*[a-zA-z]*'
    return t
# Ignored
t_ignore = " \t"
def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)
```

对binary_search.py进行测试，输出每一个识别到的token

```python
from util import clear_text
text1=clear_text(open('binary_search.py','r').readlines())
lexer.input(text1)
for tok in iter(lex.token, None):
    print(repr(tok.type), repr(tok.value))
```

util中的clear_text函数为清除每行的空格

```
def clear_text(textlines):
    lines=[]
    for line in textlines:
        line=line.strip()
        if len(line)>0:
            lines.append(line)
    return ' '.join(lines)
```

使用yacc进行语法分析

PLY 的解析器适用于lex解析出的序列标记。 它使用 BNF 语法来描述这些标记是如何组装的。

对node进行定义

```
class node:
    def __init__(self, data):
        self._data = data
        self._children = []
        self._value=None
    def getdata(self):
        return self._data
    def setvalue(self,value):
        self._value=value
    def getvalue(self):
        return self._value
    def getchild(self,i):
        return self._children[i]
    def getchildren(self):
        return self._children
    def add(self, node):
        self._children.append(node)
    def print_node(self, prefix):
        print ('  '*prefix,'+',self._data)
        for child in self._children:
            child.print_node(prefix+1)
def num_node(data):
    t=node(data)
    t.setvalue(float(data))
    return t
```

定义文法

```
# YACC for parsing Python
def simple_node(t, name):
    t[0] = node(name)
    for i in range(1, len(t)):
        t[0].add(node(t[i]))
    return t[0]
def p_program(t):
    '''program : statements'''
    if len(t) == 2:
        t[0] = node('[PROGRAM]')
        t[0].add(t[1])
def p_statements(t):
    '''statements : statements statement
                  | statement'''
    if len(t) == 3:
        t[0] = node('[STATEMENTS]')
        t[0].add(t[1])
        t[0].add(t[2])
    elif len(t) == 2:
        t[0] = node('[STATEMENTS]')
        t[0].add(t[1])
def p_statement(t):
    ''' statement : assignment
                  | operation
                  | print
                  | if_elif_else
                  | while
                  | for
                  | BREAK'''
    if len(t) == 2:                     # 判断是否为'BREAK'
        if not isinstance(t[1], str):
            t[0] = node('[STATEMENT]')
            t[0].add(t[1])
        else:                           # 为BREAK
            t[0] = node('[STATEMENT]')
            t[0].add(node('[BREAK]'))
def p_assignment(t):
    '''assignment : VARIABLE '=' NUMBER
                  | VARIABLE '[' expr ']' '=' NUMBER
                  | VARIABLE '=' VARIABLE
                  | VARIABLE '=' VARIABLE '[' expr ']'
                  | VARIABLE '=' num_list'''
    if len(t) == 4:
        if isinstance(t[3], str):       # NUMBER or VARIABLE
            if ord('0') <= ord(t[3][0]) <= ord('9'):    # NUMBER
                t[0] = node('[ASSIGNMENT]')
                t[0].add(node(t[1]))
                t[0].add(node(t[2]))
                t[0].add(num_node(t[3]))
            else:                                       # VARIABLE
                t[0] = node('[ASSIGNMENT]')
                t[0].add(node(t[1]))
                t[0].add(node(t[2]))
                t[0].add(node(t[3]))
        else:                           # num_list
            t[0] = node('[ASSIGNMENT]')
            t[0].add(node(t[1]))
            t[0].add(node(t[2]))
            t[0].add(t[3])
    elif len(t) == 7:
        if t[2] == '[':                 # NUMBER
            t[0] = node('[ASSIGNMENT]')
            t[0].add(node(t[1]))
            t[0].add(t[3])
            t[0].add(node(t[5]))
            t[0].add(num_node(t[6]))
        else:                           # VARIABLE '[' expr ']'
            t[0] = node('[ASSIGNMENT]')
            t[0].add(node(t[1]))
            t[0].add(node(t[2]))
            t[0].add(node(t[3]))
            t[0].add(t[5])
```

```
def p_num_list(t):
    '''num_list : '[' numbers ']' '''
    if len(t) == 4:
        t[0] = node('[NUM_LIST]')
        t[0].add(t[2])
def p_numbers(t):
    '''numbers : NUMBER
               | numbers ',' NUMBER'''
    if len(t) == 2:
        t[0] = node('[NUMBERS]')
        t[0].add(num_node(t[1]))
    elif len(t) == 4:
        t[0] = node('[NUMBERS]')
        t[0].add(t[1])
        t[0].add(num_node(t[3]))
def p_operation(t):
    '''operation : VARIABLE '=' expr
                 | VARIABLE '[' expr ']' '=' expr'''
    if len(t) == 4:
        t[0] = node('[OPERATION]')
        t[0].add(node(t[1]))
        t[0].add(node(t[2]))
        t[0].add(t[3])
    elif len(t) == 7:
        t[0] = node('[OPERATION]')
        t[0].add(node(t[1]))
        t[0].add(t[3])
        t[0].add(node(t[5]))
        t[0].add(t[6])
def p_expr(t):
    '''expr : expr '+' term
            | expr '-' term
            | expr DIV expr
            | term
            | LEN '(' factor ')' '''
    if len(t) == 4:
        t[0] = node('[EXPRESSION]')
        t[0].add(t[1])
        t[0].add(node(t[2]))
        t[0].add(t[3])
    elif len(t) == 2:
        t[0] = node('[EXPRESSION]')
        t[0].add(t[1])
    elif len(t) == 5:
        t[0] = node('[EXPRESSION]')
        t[0].add(node('[LEN]'))
        t[0].add(t[3])
def p_term(t):
    '''term : term '*' factor
            | term '/' factor
            | factor'''
    if len(t) == 4:
        t[0] = node('[TERM]')
        t[0].add(t[1])
        t[0].add(node(t[2]))
        t[0].add(t[3])
    elif len(t) == 2:
        t[0] = node('[TERM]')
        t[0].add(t[1])
def p_factor(t):
    '''factor : NUMBER
              | VARIABLE
              | VARIABLE '[' expr ']'
              | '(' expr ')' '''
    if len(t) == 2:
        if ord('0') <= ord(t[1][0]) <= ord('9'):       # NUMBER
            t[0] = node('[FACTOR]')
            t[0].add(num_node(t[1]))
        else:                              # VARIABLE
            t[0] = node('[FACTOR]')
            t[0].add(node(t[1]))
    elif len(t) == 4:
        t[0] = node('[FACTOR]')
        t[0].add(t[2])
    elif len(t) == 5:
        t[0] = node('[FACTOR]')
        t[0].add(node(t[1]))
        t[0].add(t[3])
def p_print(t):
    '''print : PRINT '(' VARIABLE ')' '''
    if len(t) == 5:
        t[0] = node('[PRINT]')
        t[0].add(node(t[3]))
def p_condition(t):
    '''condition : factor '>' factor
                 | factor '<' factor
                 | factor LE factor
                 | factor GE factor'''
    if len(t) == 4:
        t[0] = node('[CONDITION]')
        t[0].add(t[1])
        t[0].add(node(t[2]))
        t[0].add(t[3])
def p_if_elif_else(t):
    '''if_elif_else : if
                    | if elif else'''
    if len(t) == 2:
        t[0] = node('[IF_ELIF_ELSE]')
        t[0].add(t[1])
    elif len(t) == 4:
        t[0] = node('[IF_ELIF_ELSE]')
        t[0].add(t[1])
        t[0].add(t[2])
        t[0].add(t[3])
def p_if(t):
    '''if : IF '(' condition ')' '{' statements '}' '''
    if len(t) == 8:
        t[0] = node('[IF]')
        t[0].add(t[3])
        t[0].add(t[6])
def p_elif(t):
    '''elif : ELIF '(' condition ')' '{' statements '}' '''
    if len(t) == 8:
        t[0] = node('[ELIF]')
        t[0].add(t[3])
        t[0].add(t[6])
```

```
def p_else(t):
    '''else : ELSE '{' statements '}' '''
    if len(t) == 5:
        t[0] = node('[ELSE]')
        t[0].add(t[3])
def p_while(t):
    '''while : WHILE '(' condition ')' '{' statements '}' '''
    if len(t) == 8:
        t[0] = node('[WHILE]')
        t[0].add(t[3])
        t[0].add(t[6])
def p_for(t):
    '''for : FOR '(' conditions ')' '{' statements '}' '''
    if len(t) == 8:
        t[0] = node('[FOR]')
        t[0].add(t[3])
        t[0].add(t[6])
def p_conditions(t):
    '''conditions : assignment ';' condition ';' increment'''
    if len(t) == 6:
        t[0] = node('[CONDITIONS]')
        t[0].add(t[1])
        t[0].add(t[3])
        t[0].add(t[5])
def p_increment(t):
    '''increment : VARIABLE INC'''
    if len(t) == 3:
        t[0] = node('[INCREMENT]')
        t[0].add(node(t[1]))
        t[0].add(node(t[2]))
def p_error(t):
    print("Syntax error at '%s'" % t.value)
yacc.yacc()
```

实行语法制导翻译

定义变量存储字典结构和更新函数

```
v_table = {}  # variable table
def update_v_table(name, value):
    v_table[name] = value
```

翻译函数

在进行翻译条件语句和循环语句时，不能简单的进行深度优先遍历，要对于某些条件节点进行优先翻译

ASSIGNMENT的文法

$$ASSIGENMENT:$$
$$VARIABLE = NUMBER$$
$$|VARIABLE[EXPR] = NUMBER$$
$$|VARIABLE = VARIABLE$$
$$|VARIABLE = VARIABLE[EXPR]|VARIABLE = num\_list$$

```
def trans(node):
    # Translation
    # Assignment
    if node.getdata() == '[ASSIGNMENT]':
        '''assignment : VARIABLE '=' NUMBER
                      | VARIABLE '[' expr ']' '=' NUMBER
                      | VARIABLE '=' VARIABLE
                      | VARIABLE '=' VARIABLE '[' expr ']'
                      | VARIABLE '=' num_list'''
        if len(node.getchildren()) == 3:
            if ord('0') <= ord(node.getchild(2).getdata()[0]) <= ord('9'):     #
NUMBER
                value = node.getchild(2).getvalue()
                # update v_table
                update_v_table(node.getchild(0).getdata(), value)
            elif node.getchild(2).getdata() == '[NUM_LIST]':                   #
num_list
                trans(node.getchild(2))
                value = node.getchild(2).getvalue()
                # update v_table
                update_v_table(node.getchild(0).getdata(), value)
            else:                                                              #
VARIABLE
                value = v_table[node.getchild(2).getdata()]
                # update v_table
                update_v_table(node.getchild(0).getdata(), value)
        elif len(node.getchildren()) == 4:
            if node.getchild(2).getdata() == '=':  # NUMBER
                arg = v_table[node.getchild(0).getdata()]
                trans(node.getchild(1))
                index = int(node.getchild(1).getvalue())
                value = node.getchild(3).getvalue()
                # update VARIABLE
                arg[index] = value
            elif node.getchild(1).getdata() == '=':  # VARIABLE '[' expr ']'
                arg1 = v_table[node.getchild(2).getdata()]
                trans(node.getchild(3))
                index = int(node.getchild(3).getvalue())
                value = arg1[index]
                # update v_table
                update_v_table(node.getchild(0).getdata(), value)
```

NUM_LIST的文法

$$\begin{aligned} &NUM\_LIST: \\ &\quad [numbers] \end{aligned}$$

```
elif node.getdata() == '[NUM_LIST]':
    '''num_list : '[' numbers ']' '''
    if len(node.getchildren()) == 1:
        trans(node.getchild(0))
        value = [float(x) for x in node.getchild(0).getvalue().split()]
        node.setvalue(value)
```

NUMBERS的文法

$$NUMBERS:$$
$$NUMBER$$
$$|numbers, NUMBER$$

```python
elif node.getdata() == '[NUMBERS]':
    '''numbers : NUMBER
               | numbers ',' NUMBER'''
    if len(node.getchildren()) == 1:
        value = str(node.getchild(0).getvalue())
        node.setvalue(value)

    elif len(node.getchildren()) == 2:
        trans(node.getchild(0))

        value0 = node.getchild(0).getvalue()
        value1 = str(node.getchild(1).getvalue())
        value = value0 + ' ' + value1
        node.setvalue(value)
```

OPERATION的文法

$$OPERATION:$$
$$VARIABLE = expr$$
$$|VARIABLE[expr] = expr$$

```python
elif node.getdata() == '[OPERATION]':
    '''operation : VARIABLE '=' expr
                 | VARIABLE '[' expr ']' '=' expr'''
    if len(node.getchildren()) == 3:
        trans(node.getchild(2))
        value = node.getchild(2).getvalue()
        node.getchild(0).setvalue(value)
        # update v_table
        update_v_table(node.getchild(0).getdata(), value)
    elif len(node.getchildren()) == 4:
        arg = v_table[node.getchild(0).getdata()]
        trans(node.getchild(1))
        index = int(node.getchild(1).getvalue())
        trans(node.getchild(3))
        value = node.getchild(3).getvalue()
        # update VARIABLE
        arg[index] = value
```

EXPR的文法

$$EXPR:$$
$$expr + term$$
$$|expr - term$$
$$|exprDIVfactor$$
$$|term$$
$$|LEN(factor)$$

```
elif node.getdata() == '[EXPRESSION]':
    '''expr : expr '+' term
            | expr '-' term
            | expr DIV factor
            | term
            | LEN '(' factor ')' '''
    if len(node.getchildren()) == 3:
        trans(node.getchild(0))
        arg0 = node.getchild(0).getvalue()
        trans(node.getchild(2))
        arg1 = node.getchild(2).getvalue()
        op = node.getchild(1).getdata()
        if op == '+':
            value = arg0 + arg1
        elif op == '-':
            value = arg0 - arg1
        elif op == '//':
            value = arg0 // arg1
        node.setvalue(value)

    elif len(node.getchildren()) == 1:       # term
        trans(node.getchild(0))
        value = node.getchild(0).getvalue()
        node.setvalue(value)

    elif len(node.getchildren()) == 2:
        trans(node.getchild(1))
        value = len(node.getchild(1).getvalue())
        node.setvalue(value)
```

Term的文法

$$
\begin{aligned}
&term : term * factor \\
&\quad |term/factor \\
&\qquad |factor
\end{aligned}
$$

```
elif node.getdata() == '[TERM]':
    '''term : term '*' factor
            | term '/' factor
            | factor'''
    if len(node.getchildren()) == 3:
        trans(node.getchild(0))
        arg0 = node.getchild(0).getvalue()
        trans(node.getchild(2))
        arg1 = node.getchild(2).getvalue()
        op = node.getchild(1).getdata()
        if op == '*':
            value = arg0 + arg1
        elif op == '/':
            value = arg0 - arg1
        node.setvalue(value)
    elif len(node.getchildren()) == 1:
        trans(node.getchild(0))
        value = node.getchild(0).getvalue()
        node.setvalue(value)
```

FACTOR的文法

$$factor:$$
$$NUMBER$$
$$|VARIABLE$$
$$|VARIABLE[expr]$$
$$|(expr)$$

```
elif node.getdata() == '[FACTOR]':
    '''factor : NUMBER
              | VARIABLE
              | VARIABLE '[' expr ']'
              | '(' expr ')' '''
    if len(node.getchildren()) == 1:
        if ord('0') <= ord(node.getchild(0).getdata()[0]) <= ord('9'):  # NUMBER
            value = node.getchild(0).getvalue()
            node.setvalue(value)
        elif node.getchild(0).getdata() == '[EXPRESSION]':           # '('
expr ')'
            trans(node.getchild(0))
            value = node.getchild(0).getvalue()
            node.setvalue(value)
        else:                                                        #
VARIABLE
            value = v_table[node.getchild(0).getdata()]
            node.setvalue(value)
    elif len(node.getchildren()) == 2:
        arg = v_table[node.getchild(0).getdata()]
        trans(node.getchild(1))
        index = int(node.getchild(1).getvalue())
        value = arg[index]
        node.setvalue(value)
```

PRINT的文法

$$print : PRINT(VARIABLE)$$

```
elif node.getdata() == '[PRINT]':
    '''print : PRINT '(' VARIABLE ')' '''
    arg0 = v_table[node.getchild(0).getdata()]
    print (arg0)
```

CONDITION的文法

$$condition:$$
$$factor > factor$$
$$|factor > factor$$
$$|factorLEfactor$$
$$|factorGEfactor$$

```
elif node.getdata() == '[CONDITION]':
    '''condition : factor '>' factor
```

```
                   | factor '<' factor
                   | factor LE factor
                   | factor GE factor'''
    trans(node.getchild(0))
    arg0 = node.getchild(0).getvalue()
    trans(node.getchild(2))
    arg1 = node.getchild(2).getvalue()
    op = node.getchild(1).getdata()
    if op == '>':
        node.setvalue(arg0 > arg1)
    elif op == '<':
        node.setvalue(arg0 < arg1)
    elif op == '<=':
        node.setvalue(arg0 <= arg1)
    elif op == '>=':
        node.setvalue(arg0 >= arg1)
```

if_else_else的文法

$$
\begin{aligned}
&if\_elif\_else : \\
&\qquad if \\
&|if\_elif\_else
\end{aligned}
$$

```
elif node.getdata() == '[IF_ELIF_ELSE]':
    '''if_elif_else : if
                    | if elif else'''
    if len(node.getchildren()) == 1:
        if trans(node.getchild(0)) == '[BREAK]':
            return '[BREAK]'
    elif len(node.getchildren()) == 3:
        trans(node.getchild(0).getchild(0))
        condition = node.getchild(0).getchild(0).getvalue()
        if condition:
            if trans(node.getchild(0)) == '[BREAK]':
                return '[BREAK]'
        else:
            trans(node.getchild(1).getchild(0))
            condition = node.getchild(1).getchild(0).getvalue()
            if condition:
                if trans(node.getchild(1)) == '[BREAK]':
                    return '[BREAK]'
            else:
                if trans(node.getchild(2)) == '[BREAK]':
                    return '[BREAK]'
```

if的文法

$$if : IF(condition)\{statement\}$$

```
elif node.getdata() == '[IF]':
    r'''if : IF '(' condition ')' '{' statements '}' '''
    children = node.getchildren()
    trans(children[0])
    condition = children[0].getvalue()
    if condition:
        for c in children[1:]:
            trans(c)
```

elif的文法

$$elif : ELIF(condition)\{statements\}$$

```
elif node.getdata() == '[ELIF]':
    '''elif : ELIF '(' condition ')' '{' statements '}' '''
    children = node.getchildren()
    trans(children[0])
    condition = children[0].getvalue()
    if condition:
        for c in children[1:]:
            trans(c)
```

else的文法

```
elif node.getdata() == '[ELSE]':
    if trans(node.getchild(0)) == '[BREAK]':
        return '[BREAK]'
```

while的文法

```
elif node.getdata() == '[WHILE]':
    r'''while : WHILE '(' condition ')' '{' statements '}' '''
    children = node.getchildren()
    break_flag = False
    while trans(children[0]):
        for c in children[1:]:
            if trans(c) == '[BREAK]':
                break_flag = True
                break
        if break_flag:
            break
```

for的文法

```
elif node.getdata() == '[FOR]':
    '''for : FOR '(' conditions ')' '{' statements '}' '''
    conditions = node.getchild(0)
    children = node.getchildren()
    break_flag = False
    trans(conditions.getchild(0))           # assignment
    while trans(conditions.getchild(1)):    # condition
        for c in children[1:]:
            if trans(c) == '[BREAK]':
                break_flag = True
                break
        if break_flag:
            break
        trans(conditions.getchild(2))       # increment
```

Increment

```
elif node.getdata() == '[INCREMENT]':
    value = v_table[node.getchild(0).getdata()] + 1
    # update v_table
    update_v_table(node.getchild(0).getdata(), value)
```

Break

```
elif node.getdata() == '[BREAK]':
    return '[BREAK]'
```

递归返回

```
else:
    for c in node.getchildren():
        # 原本仅为trans(c)
        if trans(c) == '[BREAK]':
            return '[BREAK]'

return node.getvalue()
```

将node函数递归输出为字符串语法树形式便于在http://mshang.ca/syntree上检查结果

实验结果

主程序代码

```
from py_yacc import yacc
from util import clear_text
from translation import trans, v_table
def translation(filename):
    text = clear_text(open(filename, 'r').read())
    def put2str(node):
        global res
        if node:
            data = str(node._data)
            data = data.replace("[", "").replace("]", "").replace("/'", "")
            res += data
        if node._children:
            for i in node._children:
                res += "["
                put2str(i)
                res += "]"
    # syntax parse
    root = yacc.parse(text)
    root.print_node(0)
    put2str(root)
    print(res)
    # translation
    trans(root)
    print(v_table)
if __name__ == '__main__':
    res = ""
    translation("select_sort.py")
```

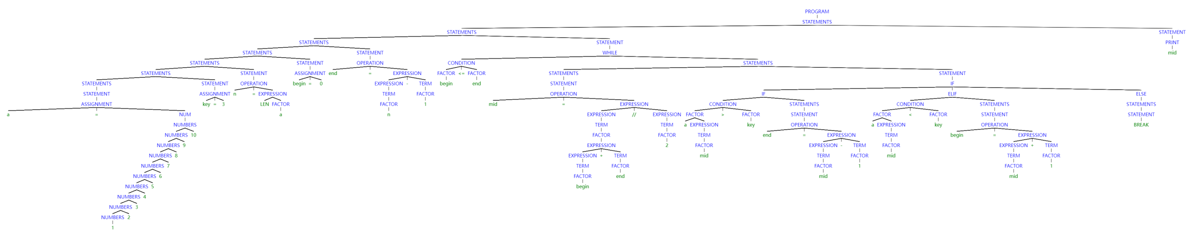
结果

binary_search.py

程序内容

```
a=[1,2,3,4,5,6,7,8,9,10]

key=3

n=len(a)

begin=0
end=n-1

while(begin<=end){
    mid=(begin+end)//2
    if(a[mid]>key){
        end=mid-1
    }
    elif(a[mid]<key){
        begin=mid+1
    }
    else{
        break
    }
}
print(mid)
```

字符串形式的语法树



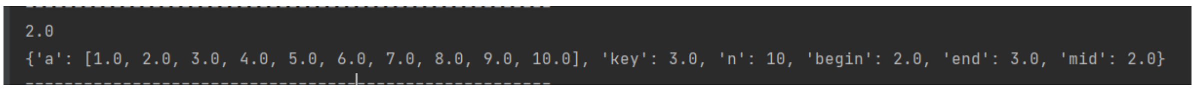
输出

```
+ [PROGRAM]
    + [STATEMENTS]
      + [STATEMENTS]
        + [STATEMENTS]
          + [STATEMENTS]
            + [STATEMENTS]
              + [STATEMENTS]
                + [STATEMENTS]
                  + [STATEMENT]
                    + [ASSIGNMENT]
              + a
              + =
              + [NUM_LIST]
                + [NUMBERS]
                  + [NUMBERS]
                    + [NUMBERS]
                      + [NUMBERS]
                        + [NUMBERS]
                          + [NUMBERS]
                            + [NUMBERS]
                              + [NUMBERS]
                                + [NUMBERS]
                                  + [NUMBERS]
                                    + 1
                                  + 2
                                + 3
                              + 4
                            + 5
                          + 6
                        + 7
                      + 8
                    + 9
                  + 10
        + [STATEMENT]
          + [ASSIGNMENT]
            + key
            + =
            + 3
      + [STATEMENT]
        + [OPERATION]
          + n
          + =
          + [EXPRESSION]
            + [LEN]
            + [FACTOR]
              + a
    + [STATEMENT]
      + [ASSIGNMENT]
        + begin
        + =
        + 0
  + [STATEMENT]
    + [OPERATION]
      + end
      + =
      + [EXPRESSION]
        + [EXPRESSION]
          + [TERM]
            + [FACTOR]
              + n
        + -
        + [TERM]
          + [FACTOR]
            + 1
+ [STATEMENT]
  + [WHILE]
    + [CONDITION]
      + [FACTOR]
```

```
                + begin
              + <=
              + [FACTOR]
                + end
            + [STATEMENTS]
              + [STATEMENTS]
                + [STATEMENT]
                  + [OPERATION]
                    + mid
                    + =
                    + [EXPRESSION]
                      + [EXPRESSION]
                        + [TERM]
                          + [FACTOR]
                            + [EXPRESSION]
                              + [EXPRESSION]
                                + [TERM]
                                  + [FACTOR]
                                    + begin
                              + +
                              + [TERM]
                                + [FACTOR]
                                  + end
                      + //
                      + [EXPRESSION]
                        + [TERM]
                          + [FACTOR]
                            + 2
              + [STATEMENT]
                + [IF_ELIF_ELSE]
                  + [IF]
                    + [CONDITION]
                      + [FACTOR]
                        + a
                        + [EXPRESSION]
                          + [TERM]
                            + [FACTOR]
                              + mid
                      + >
                      + [FACTOR]
                        + key
                    + [STATEMENTS]
                      + [STATEMENT]
                        + [OPERATION]
                          + end
                          + =
                          + [EXPRESSION]
                            + [EXPRESSION]
                              + [TERM]
                                + [FACTOR]
                                  + mid
                            + -
                            + [TERM]
                              + [FACTOR]
                                + 1
                  + [ELIF]
                    + [CONDITION]
                      + [FACTOR]
```

```
                       + a
                       + [EXPRESSION]
                         + [TERM]
                           + [FACTOR]
                             + mid
                     + <
                     + [FACTOR]
                       + key
                   + [STATEMENTS]
                     + [STATEMENT]
                       + [OPERATION]
                         + begin
                         + =
                         + [EXPRESSION]
                           + [EXPRESSION]
                             + [TERM]
                               + [FACTOR]
                                 + mid
                           + +
                           + [TERM]
                             + [FACTOR]
                               + 1
                 + [ELSE]
                   + [STATEMENTS]
                     + [STATEMENT]
                       + [BREAK]
     + [STATEMENT]
       + [PRINT]
         + mid
```

翻译结果



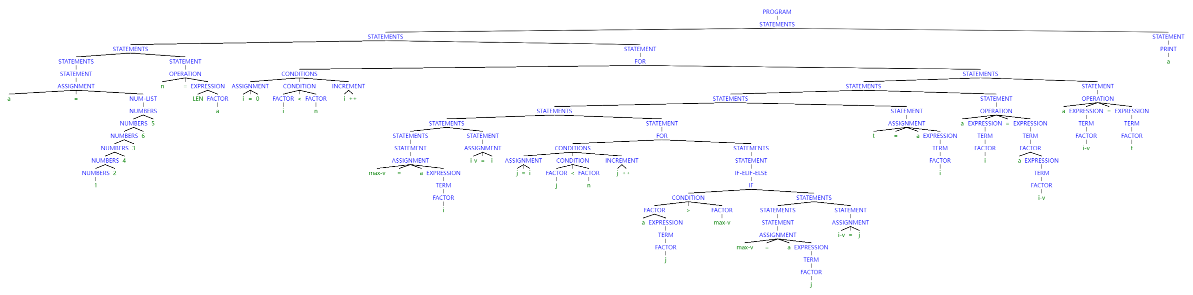
```
2.0
{'a': [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0], 'key': 3.0, 'n': 10, 'begin': 2.0, 'end': 3.0, 'mid': 2.0}
```

select_sort.py

程序内容

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

n=len(a)

for(i=0;i<n;i++){
    max_v=a[i]
    i_v=i

    for(j=i;j<n;j++){
        if(a[j]>max_v){
            max_v=a[j]
            i_v=j
        }
    }

    t=a[i]
    a[i]=a[i_v]
    a[i_v]=t
}

print(a)
```

字符串形式的语法树



```
+ [PROGRAM]
  + [STATEMENTS]
    + [STATEMENTS]
      + [STATEMENTS]
        + [STATEMENTS]
          + [STATEMENT]
            + [ASSIGNMENT]
              + a
              + =
              + [NUM_LIST]
                + [NUMBERS]
                  + [NUMBERS]
                    + [NUMBERS]
                      + [NUMBERS]
                        + [NUMBERS]
                          + [NUMBERS]
                            + 1
                          + 2
                        + 4
                      + 3
                    + 6
                  + 5
        + [STATEMENT]
          + [OPERATION]
            + n
            + =
            + [EXPRESSION]
              + [LEN]
              + [FACTOR]
                + a
      + [STATEMENT]
        + [FOR]
          + [CONDITIONS]
            + [ASSIGNMENT]
              + i
              + =
              + 0
            + [CONDITION]
              + [FACTOR]
                + i
              + <
              + [FACTOR]
```

```
                  + n
              + [INCREMENT]
                + i
                + ++
            + [STATEMENTS]
              + [STATEMENTS]
                + [STATEMENTS]
                  + [STATEMENTS]
                    + [STATEMENTS]
                      + [STATEMENTS]
                        + [STATEMENT]
                          + [ASSIGNMENT]
                            + max_v
                            + =
                            + a
                            + [EXPRESSION]
                              + [TERM]
                                + [FACTOR]
                                  + i
                      + [STATEMENT]
                        + [ASSIGNMENT]
                          + i_v
                          + =
                          + i
                    + [STATEMENT]
                      + [FOR]
                        + [CONDITIONS]
                          + [ASSIGNMENT]
                            + j
                            + =
                            + i
                          + [CONDITION]
                            + [FACTOR]
                              + j
                            + <
                            + [FACTOR]
                              + n
                          + [INCREMENT]
                            + j
                            + ++
                        + [STATEMENTS]
                          + [STATEMENT]
                            + [IF_ELIF_ELSE]
                              + [IF]
                                + [CONDITION]
                                  + [FACTOR]
                                    + a
                                    + [EXPRESSION]
                                      + [TERM]
                                        + [FACTOR]
                                          + j
                                  + >
                                  + [FACTOR]
                                    + max_v
                                + [STATEMENTS]
                                  + [STATEMENTS]
                                    + [STATEMENT]
                                      + [ASSIGNMENT]
```

```
                                        + max_v
                                        + =
                                        + a
                                        + [EXPRESSION]
                                          + [TERM]
                                            + [FACTOR]
                                              + j
                                  + [STATEMENT]
                                    + [ASSIGNMENT]
                                      + i_v
                                      + =
                                      + j
                  + [STATEMENT]
                    + [ASSIGNMENT]
                      + t
                      + =
                      + a
                      + [EXPRESSION]
                        + [TERM]
                          + [FACTOR]
                            + i
                + [STATEMENT]
                  + [OPERATION]
                    + a
                    + [EXPRESSION]
                      + [TERM]
                        + [FACTOR]
                          + i
                    + =
                    + [EXPRESSION]
                      + [TERM]
                        + [FACTOR]
                          + a
                          + [EXPRESSION]
                            + [TERM]
                              + [FACTOR]
                                + i_v
              + [STATEMENT]
                + [OPERATION]
                  + a
                  + [EXPRESSION]
                    + [TERM]
                      + [FACTOR]
                        + i_v
                  + =
                  + [EXPRESSION]
                    + [TERM]
                      + [FACTOR]
                        + t
      + [STATEMENT]
        + [PRINT]
          + a
```

翻译结果

```
[6.0, 5.0, 4.0, 3.0, 2.0, 1.0]
{'a': [6.0, 5.0, 4.0, 3.0, 2.0, 1.0], 'n': 6, 'i': 6.0, 'max_v': 1.0, 'i_v': 5.0, 'j': 6.0, 't': 1.0}
```