# 编译原理实验报告

2016-2017学年第2学期 🞏712班 🞏802班 第\_\_5\_\_组

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 小组成员 共 3 人 | | 学号： 14080214  姓名： 李悦 | 学号： 14080017  姓名： 韩东 | 学号： 14080211  姓名： 盖奕豪 |
| 分工 | | Python代码编写，实验设计 | 实验设计 | 实验设计 |
| 承担工作量 | | \_\_60\_\_% | \_\_20\_\_% | \_\_20\_\_% |
| 🡹 以上内容由学生填写 🡹  🡻 以下内容由教师填写 🡻 | | | | |
| 实验出勤 | | □全勤 □偶尔缺勤  □偶尔出勤 □从未出现 | □全勤 □偶尔缺勤  □偶尔出勤 □从未出现 | □全勤 □偶尔缺勤  □偶尔出勤 □从未出现 |
| 表现 | | □积极提问 □与同学一起商量  □认真设计 □主要靠别人讲解  □完全不参与 | □积极提问 □与同学一起商量  □认真设计 □主要靠别人讲解  □完全不参与 | □积极提问 □与同学一起商量  □认真设计 □主要靠别人讲解  □完全不参与 |
| 程序结构 | | □合理 □基本合理 □较差 | □合理 □基本合理 □较差 | □合理 □基本合理 □较差 |
| 程序风格 | 缩进 | □规范 □较好 □较差 | □规范 □较好 □较差 | □规范 □较好 □较差 |
| 命名 | □规范 □较好 □较差 | □规范 □较好 □较差 | □规范 □较好 □较差 |
| 程序运行 | | □正确 □有问题，能够改正  □有问题，无法改正 | □正确 □有问题，能够改正  □有问题，无法改正 | □正确 □有问题，能够改正  □有问题，无法改正 |
| 演讲 | | PPT设计质量与效果：□优秀 □良好 □一般 □较差  口头表达：□流畅，逻辑清晰 □良好 □一般 □较差 | | |
| 问题回答 | | □正确 □有部分错误  □完全错误 □不能回答 | □正确 □有部分错误  □完全错误 □不能回答 | □正确 □有部分错误  □完全错误 □不能回答 |
| 报告结构 | | □符合要求 □基本符合要求 □有比较多的缺陷 □完全不符合要求 | | |
| 报告内容 | | □充实正确 □基本正确 □有一些问题 □问题很大 | | |
| 成绩 | |  |  |  |
| 评语 | |  |  |  |

**系统功能概述：**

本程序支持在终端启动后不断输入测试代码，不输入任何代码直接输入回车即可退出程序；也支持直接在终端以文件的形式输入测试代码。测试代码可以多行，也可以多条语句。程序以python语言实现，基于第三方库（Python-Lex-Yacc）。

1. 对于测试代码，词法分析程序可以识别出如下内容：  
a) 标识符

b) 16进制实数，8进制实数，10进制实数，16进制整数，8进制整数，10进制整数。

c) 加减乘除运算符

d) 关系运算符<, >, =

e) 左右大括号，左右小括号,{},()

f) 包含（if, then, else, and, or, while, do, for, struct, switch, print）的保留字

g) 以#开头的注释

同时，程序可以跳过空格符和制表符，记录行号。对于未定义的token报错提示非法字符，并且跳过该字符。对于八进制数，如果出现数字8，则可以提示非法八进制字符。

2. 对于测试代码语法分析程序可以识别出以下内容：

a) while-do语句

b) if-then-else if-then 语句（无移进规约冲突）

c) 赋值语句

d) 四则运算语句

f) 关系运算语句

g) 多条语句

同时，语法分析程序可以定位到哪行出错，在转译哪个单词的时候出错，以及是否出错在文件末尾。遇到出错内容时，程序终止。

程序对输入的测试内容输出以缩进形式打印出的抽象语法树，以及正确的三地址代码（该三地址代码以标签的形式打出，无多余标签，无多余行号）。

**词法分析子系统：**

1. 定义非关键字的token列表：

# List of token names.

tokens = [

'IDN',

'COMMENT',

'REAL10',

'REAL8',

'REAL16',

'INT8',

'INT16',

'INT10',

'PLUS',

'MINUS',

'TIMES',

'DIVIDE',

'LPAREN',

'RPAREN',

'RELOP',

'ASSIGNOP',

'SEMI',

'LBRACKET',

'RBRACKET'

]

1. 定义保留字（以键值对的形式保存）：

reserved = {

'if' : 'IF',

'then' : 'THEN',

'else' : 'ELSE',

'while' : 'WHILE',

'for' : 'FOR',

'and' : 'AND',

'or' : 'OR',

'struct' : 'STRUCT',

'switch' : "SWITCH",

'do' : 'DO',

'print' : 'PRINT',

}

tokens = tokens + list(reserved.values())

1. 实现不需要特殊处理的正则表达式匹配的语法规则：

# Regular expression rules for simple tokens

t\_PLUS = r'\+'

t\_MINUS = r'\-'

t\_TIMES = r'\\*'

t\_DIVIDE = r'\/'

t\_LPAREN = r'\('

t\_RPAREN = r'\)'

t\_RELOP = r'[>]|[<]'

t\_ASSIGNOP = r'[=]'

t\_SEMI = r'[;]'

t\_LBRACKET = r'\{'

t\_RBRACKET = r'\}'

1. 实现需要特殊处理的部分，对于十进制、八进制、十六进制的整数将获得到的值转为int；对于十进制的实数则将读入内容转为浮点数。对于八进制、十六进制的实数则将读入内容分成正数和小数部分，整数部分直接转为整数，小数部分则对每一位乘上8或16的对应负指数次幂。对于标识符，则查看其是否为关键字如果是则把之前保存的键值对的值赋值给该token，否则赋值’IDN’。

def t\_REAL8(t):

r'([0][0-7]+)\.[0-7]\*'

item = str(t.value).split('.')

first = int(item[0], 8)

second = 0

for i in range(len(item[1])):

second += (int(item[1][i], 8))\*(8\*\*(-1\*(i+1)))

result = first + second

t.value = result

return t

def t\_REAL16(t):

r'([0][Xx][0-9A-Fa-f]+)\.[0-9A-Fa-f]+'

item = str(t.value).split('.')

first = int(item[0], 16)

second = 0

for i in range(len(item[1])):

second += (int(item[1][i], 16))\*(16\*\*(-1\*(i+1)))

result = first + second

t.value = result

return t

def t\_REAL10(t):

r'([0]|[1-9][0-9]\*)\.[0-9]+'

t.value = float(t.value)

return t

def t\_IDN(t):

r'[a-zA-Z\_][a-zA-Z\_0-9]\*'

# Check for reserved words

t.type = reserved.get(t.value,'IDN')

return t

def t\_INT8(t):

r'([0][0-8][0-8]\*) | ([0][0-7][0-7]\*)'

if '8' in str(t.value):

print "Illegal character 8 is found in an octadecimal number\n"

pass

else:

t.value = int(t.value, 8)

return t

def t\_INT16(t):

r'([0][Xx][0-9\_a-f][0-9\_a-f]\*)'

t.value = int(t.value, 16)

return t

def t\_INT10(t):

#r'(([1-9][0-9]\*|[0])\b)?![.]'

r'0|([1-9]([0-9])\*)'

t.value = int(t.value)

return t

1. 实现需要特殊处理的部分，对于#开头的语句直接忽略，对于输入的空格符和制表符予以忽略，对于每一行记录行号，对于未处理的格式输出错误的字符，同时跳过该字符：

# No return value. Token discarded

def t\_COMMENT(t):

r'\#.\*'

pass

# track line numbers

def t\_newline(t):

r'\n+'

t.lexer.lineno += len(t.value)

# A string containing ignored characters (spaces and tabs)

t\_ignore = ' \t'

# Error handling rule

def t\_error(t):

print "Illegal character '%s'" % t.value[0]

t.lexer.skip(1)

**语法分析子系统：**

1. 全部的语法规则如下（没有dangling else问题，改写语法方法为老师发的虎书的方法）：

Rule 0 S' -> program

Rule 1 program -> statementList

Rule 2 statement -> stmWithoutTrailingSubstm

Rule 3 statement -> whileStatement

Rule 4 statement -> ifThenStatement

Rule 5 statement -> ifThenElseStatement

Rule 6 ifThenStatement -> IF condition THEN statement

Rule 7 ifThenElseStatement -> IF condition THEN statementNoShortIf ELSE statement

Rule 8 statementNoShortIf -> stmWithoutTrailingSubstm

Rule 9 statementNoShortIf -> WHILE condition DO statementNoShortIf

Rule 10 statementNoShortIf -> IF condition THEN statementNoShortIf ELSE statementNoShortIf

Rule 11 whileStatement -> WHILE condition DO statement

Rule 12 condition -> expression RELOP expression

Rule 13 condition -> expression ASSIGNOP expression

Rule 14 stmWithoutTrailingSubstm -> singleStatement

Rule 15 stmWithoutTrailingSubstm -> statementBrackets

Rule 16 singleStatement -> expression SEMI

Rule 17 statementBrackets -> LBRACKET statementList RBRACKET

Rule 18 singleStatement -> IDN ASSIGNOP expression SEMI

Rule 19 statementList -> statement

Rule 20 statementList -> statementList statement

Rule 21 expression -> expression PLUS term

Rule 22 expression -> expression MINUS term

Rule 23 expression -> term

Rule 24 term -> term TIMES factor

Rule 25 term -> term DIVIDE factor

Rule 26 term -> factor

Rule 27 factor -> INT10

Rule 28 factor -> INT8

Rule 29 factor -> INT16

Rule 30 factor -> REAL10

Rule 31 factor -> REAL8

Rule 32 factor -> REAL16

Rule 33 factor -> LPAREN expression RPAREN

Rule 34 factor -> IDN

1. 节点存储形式Node表示所有的节点，所有节点都具有value表示该节点的名称例如’WHILE-DO’，children表示该节点的子节点，默认为空，在语法分析时添加；code表示该节点的三地址代码，初始为空。ConditionNode表示条件节点，除了继承Node节点的属性该节点还有true和false属性表示真假分枝的标签号（python里True和False才是关键字）；StatementNode是while-do节点和if-then等节点，这些节点在生成三地址代码的时候会有单独的标签所以给它们加上开始的标签和吓一条语句的标签。

\_\_repr\_\_是python的用于将对象转为字符串的函数，通过逐层递归，打印抽象语法树。

class Node(object):

def \_\_init\_\_(self, value = None, children = []):

self.value = value

self.children = children

self.code = ''

def \_\_repr\_\_(self, level=0):

result = "\t" \* level + repr(self.value) + "\n"

for child in self.children:

result += child.\_\_repr\_\_(level + 1)

return result

class ConditionNode(Node):

def \_\_init\_\_(self, value = None, children = []):

Node.\_\_init\_\_(self, value, children)

self.true = None

self.false = None

class StatementNode(Node):

def \_\_init\_\_(self, value = None, children = []):

Node.\_\_init\_\_(self, value, children)

self.begin = None

self.next = None

1. 语法规则代码举例：

def p\_statement\_if\_then(p):

'ifThenStatement : IF condition THEN statement'

p[0] = StatementNode('IF-THEN')

p[0].children = [p[2], p[4]]

print 'ifThenStatement -> IF condition THEN statement'

1. 出错处理，输出错误的行号，值和种类：

# Error rule for syntax errors

def p\_error(p):

if p:

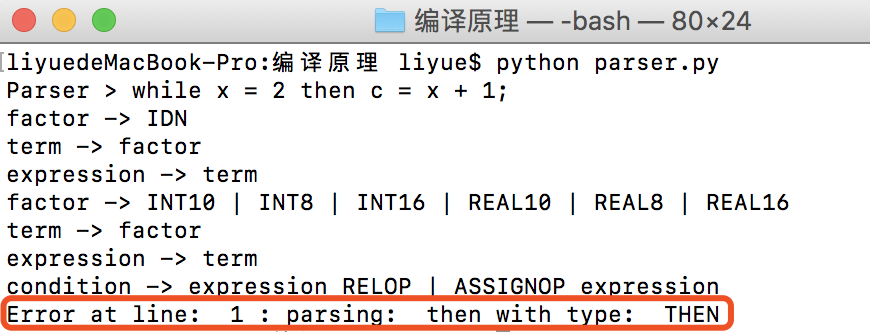
print 'Error at line: ', p.lineno, ': parsing: ', p.value, 'with type: ', p.type

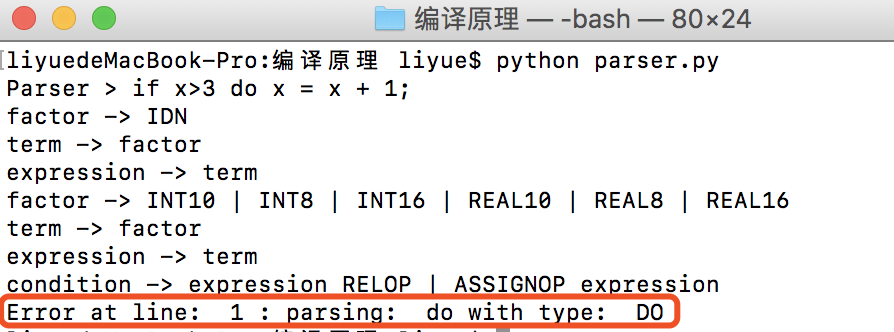
exit(1)

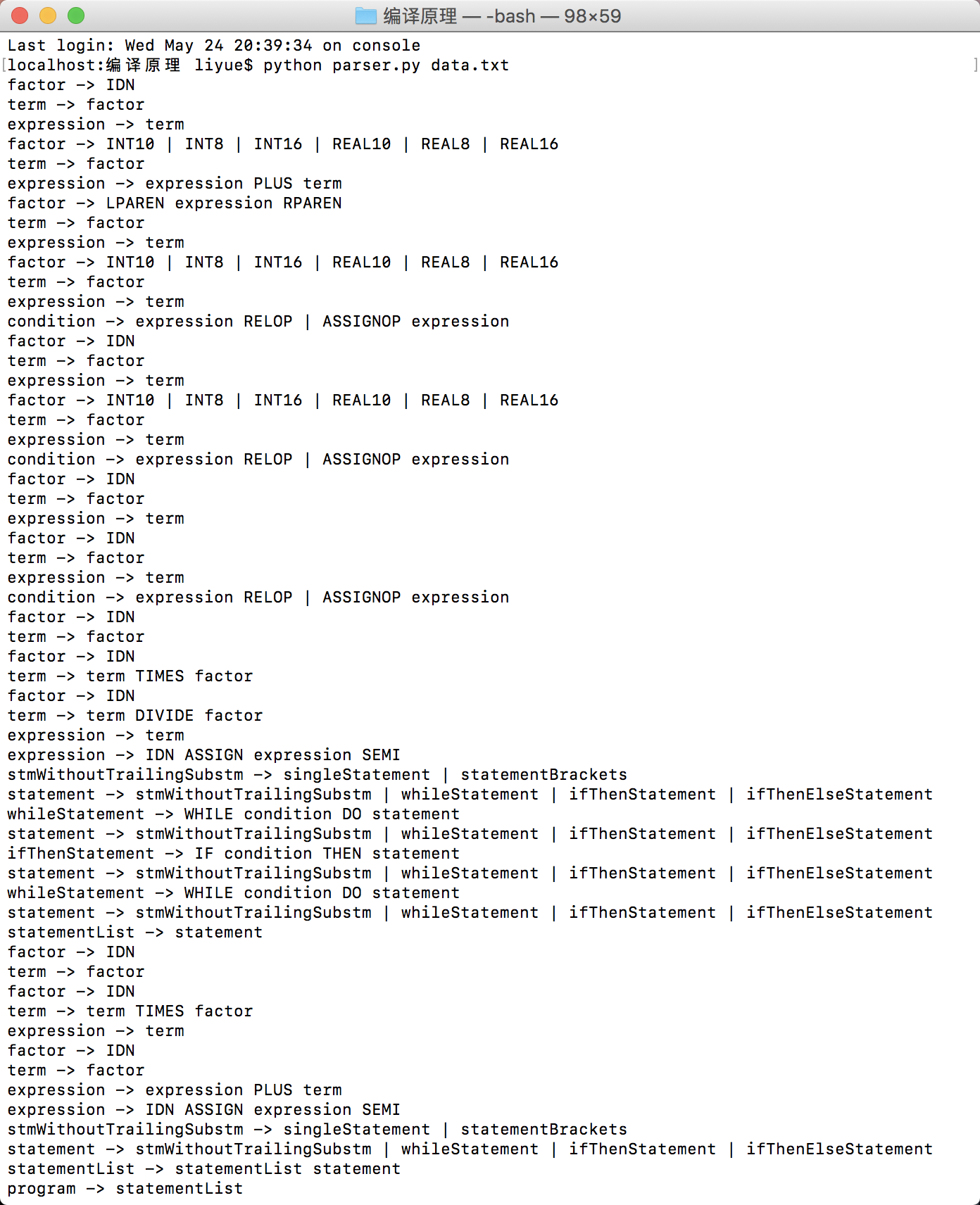
else:

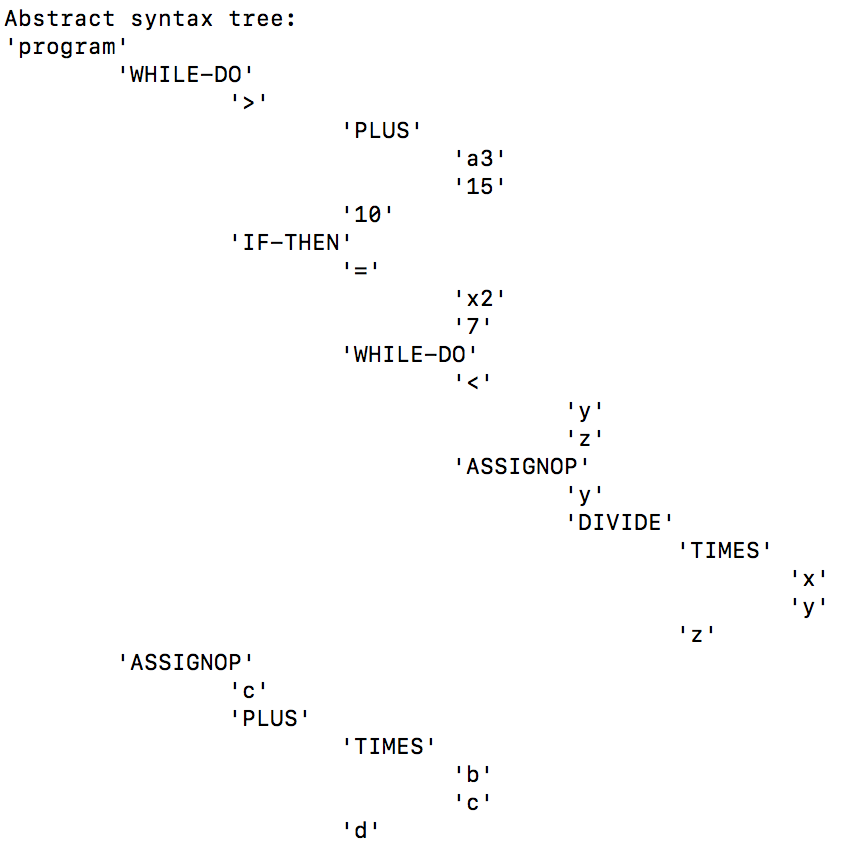
print "Syntax error at EOF"

1. 结果展示









**三地址代码生成器：**

1. 输入：抽象语法树的根节点（其子节点已经在语法分析时添加好了）

输出：三地址代码

1. 实现方法：

两次遍历，第一遍前序遍历，给需要添加新标签的地方添加新标签，添加连接关系。第二遍后序遍历，自底向上合并子节点的三地址代码。最后输出根节点的三地址代码即为结果。

1. 类成员变量及函数定义

tempVarCount = 0 临时变量计数

tacCount = 0 标签计数

def newTemp(self): 临时变量+1

self.tempVarCount += 1

return self.tempVarCount

def newLabel(self): 标签数+1

self.tacCount += 1

return self.tacCount

1. 前序遍历

def PreOrderLabel(self, node):

if node.value not in ['WHILE-DO', 'IF-THEN-ELSE', 'IF-THEN', 'program']: 其他节点不作处理

return

对于节点是program的所有子节点给新的标签，所有节点的’next’赋值为下一个节点的标签

if node.value == 'program':

for i in range(len(node.children)):

node.children[i].begin = self.newLabel()

for i in range(len(node.children) - 1):

node.children[i].next = node.children[i + 1].begin

对于’WHILE-DO’节点，条件分枝的假是该节点的下一条语句，条件分枝的真赋值新的标签

条件分枝为真对应的序号即位该标签；条件分枝为真的下一条语句为循环的开始。

elif node.value == 'WHILE-DO':

node.children[0].false = node.next # C.false := S.next

node.children[0].true = self.newLabel() # C.true := newlabel

node.children[1].begin = node.children[0].true

node.children[1].next = node.begin

对于‘IF-THEN’节点，条件分枝的假是该节点的下一条语句，条件分枝的真赋值新的标签

条件分枝为真对应的序号即位该标签；条件分枝为真的下一条语句为条件分枝为假对应标签

所对应的语句

elif node.value == 'IF-THEN':

node.children[0].true = self.newLabel() # C.true := newlabel

node.children[0].false = node.next # C.false := S.next

node.children[1].next = node.children[0].false # S1.next := C.false

node.children[1].begin = node.children[0].true

对于‘IF-THEN-ELSE’节点，条件分枝的假赋值新的标签，else对应的节点的起点是条件分枝为假

对应的标签，条件分枝的真赋值新的标签；条件分枝为真对应的序号即位该标签；条件分枝为

真和为假的下一条语句为条件分枝为该节点的下一条语句对应的标签。

所对应的语句

elif node.value == 'IF-THEN-ELSE':

node.children[0].true = self.newLabel()

node.children[0].false = self.newLabel()

node.children[2].next = node.next

node.children[1].next = node.next

node.children[1].begin = node.children[0].true

node.children[2].begin = node.children[0].false

后序遍历

for i in range(len(node.children)):

self.PreOrderLabel(node.children[i])

1. 后序遍历

def PostOrderPrint(self, node):

如果没有子节点返回

if node.children == []:

return

for i in range(len(node.children)):

self.PostOrderPrint(node.children[i])

如果是加减乘除节点，需要将该节点的三地址代码复制为两个子节点代码的和，生成新的临时变量

将该节点的value值由加减乘除换为tx

if node.value in ['TIMES', 'DIVIDE', 'PLUS', 'MINUS']:

node.code = node.children[0].code + node.children[1].code + '\tt'

newtemp = str(self.newTemp())

node.code += newtemp

node.code += ' := '

node.code = node.code + node.children[0].value + ' ' + node.value + ' ' + node.children[1].value +'\n'

node.value = 't' + newtemp

如果是’WHILE-DO’节点，需要将条件语句的三地址代码添加进来，之后添加条件真分枝的的标签和其代码，添加该节点的goto语句

elif node.value == 'WHILE-DO':

node.code += node.children[0].code # C.code

node.code = node.code + '\nL' + str(node.children[0].true) + ':' # gen(C.true)

node.code += node.children[1].code # S1.code

node.code = node.code + '\n\tgoto L' + str(node.begin)

如果是‘IF-THEN’节点，需要将条件语句的三地址代码添加进来，之后添加条件真分枝的的标签和其代码

elif node.value == 'IF-THEN':

node.code = node.children[0].code # C.code

node.code = node.code + '\nL' + str(node.children[0].true) + ':' # C.true

node.code += node.children[1].code # S1.code

如果是‘ASSIGNOP’节点，需要添加赋值号右侧节点的三地址代码，同时添加该赋值语句的三地址代码

elif node.value == 'ASSIGNOP':

node.code += node.children[1].code

node.code += ' \t'

node.code += node.children[0].value

node.code += ' := '

node.code += node.children[1].value

如果是关系运算符节点，首先添加赋值号两侧节点对应的三地址代码，之后是该比较语句的三地址代码。

elif node.value in ['<', '>', '=']:

node.code = node.children[0].code + node.children[1].code

node.code = node.code + '\tif ' + node.children[0].value + node.value + node.children[1].value + ' goto L' + str(node.true)

node.code = node.code + '\n' + '\tgoto L' + str(node.false)

如果是‘IF-THEN-ELSE’节点，需要将条件语句的三地址代码添加进来，之后添加条件真分枝的的标签和其代码，之后是goto语句和假分支的标签和三地址代码

elif node.value == 'IF-THEN-ELSE':

node.code = node.children[0].code

node.code = node.code + '\nL' + str(node.children[0].true) + ':' # C.true

node.code += node.children[1].code # S1.code

node.code = node.code + '\n\tgoto L' + str(node.next)

node.code = node.code + '\nL' + str(node.children[0].false) + ':' # C.false

node.code += node.children[2].code

如果是’program’节点，则将所有子节点的三地址代码合并，同时显示每个节点的标签

elif node.value == 'program':

for i in range(len(node.children)):

node.code = node.code + '\nL' + str(node.children[i].begin) + ':'+ node.children[i].code

1. 结果展示：

