学号 E21514033 专业 软件工程 姓名 何铭春

实验日期12.6 教师签字 成绩

实验报告

【实验名称】 递归下降LL(1)分析器

【实验目的】

输入单词串，以“#”结束，如果是文法正确的句子，

则输出成功信息，打印“success”，否则输出“error”。

输入：begin x:=9; if x>0 then x:=2\*x+1/3 end#

输出：success!

分别验证其他错误（至少2个）

【实验原理】

1.每遇到一个终结符，则判断当前读入的单词符号是否与该终结符相匹配（只要求单词符号与该终结符对应，不必考虑单词自身的值），若匹配，则继续读取下一个单词符号；若不匹配，则进行错误处理。

2.每遇到一个非终结符，则调用相应的分析子程序。

【实验内容】

1.从PPT语法描述中得到的LL(1)文法:

S(程序)->a(begin)A(语句串)b(end)

A->B(语句)A’(提取左公共因子得来)

A’->;A|&

B->C(赋值语句)|D(if语句)

C->c:=E(表达式)

D->e(if)Ef(then)B

E->F(项)E’(提取左公共因子得来)

E’->+E|-E|<E|>E|&

F->G(因子)F’(提取左公共因子得来)

F’->\*F|/F|&

G->c(标识符)|d(常数)|(E)

2.数据结构

以下为词法分析所需数据，即关键字字典，操作符列表，格式字符

global keywords

keywords={"begin":'a',"end":'b',"if":'e',"then":'f'}

global opration

opration=["+","-","\*","/",">","<",";"]

global format

format=[' ','\n','\t']

以下用于词法分析程序的存储，spring存储得到的字符形成列表。spring列表用于进一步的递归子程序法分析。Word用于拼接字符和数字形成字符串，进一步判断是关键字，标识符还是常数。

global spring

spring=[]

global word

word=''

以下为递归子程序法用到的数据，lookahead保存当前的字符，初始状态为spring第一个字符。counter用于保存下一个字符的序号，用于getToken函数，更新lookahead。

global lookahead

lookahead=spring[0]

global counter

counter=1

3.主要函数

getToken函数用于更新lookahead。MatchToken比较字符，匹配则比较下一个，不匹配则报错。

def getToken():

global lookahead

global counter

global spring

if counter==len(spring):

pass

else:

lookahead=spring[counter]

counter=counter+1

def MatchToken(word):

if lookahead!=word:

print("error!")

sys.exit()

else:

getToken()

A7函数，其实指的是A’，但是函数名无法使用单引号，使用7代替。先用if结构通过select集合分不同语法式，然后继续调用子程序。

def ParseA7():

print("A7->",end='')

if lookahead==";":

MatchToken(";")

ParseA()

elif lookahead=="b":

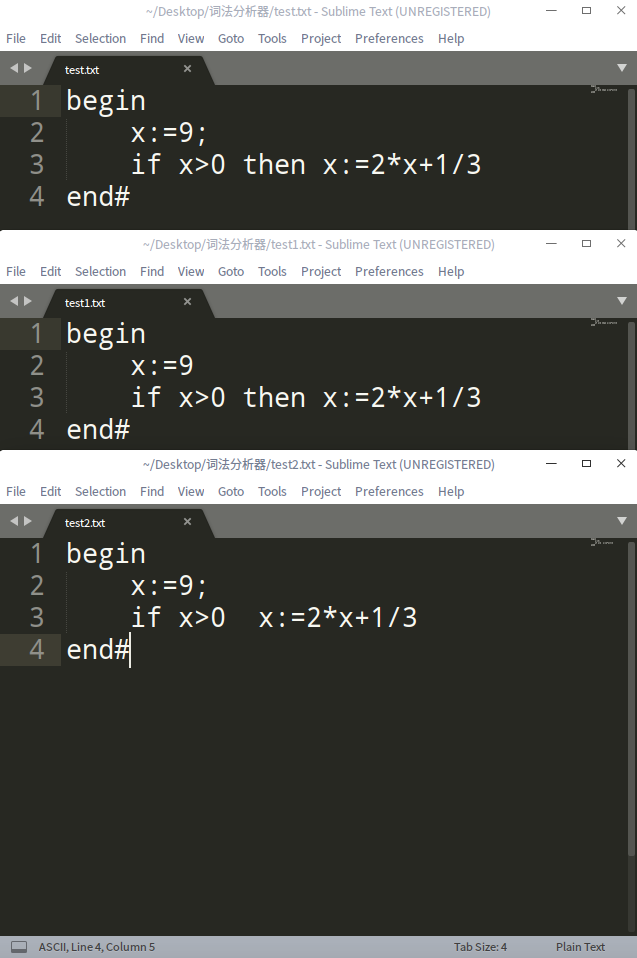
pass

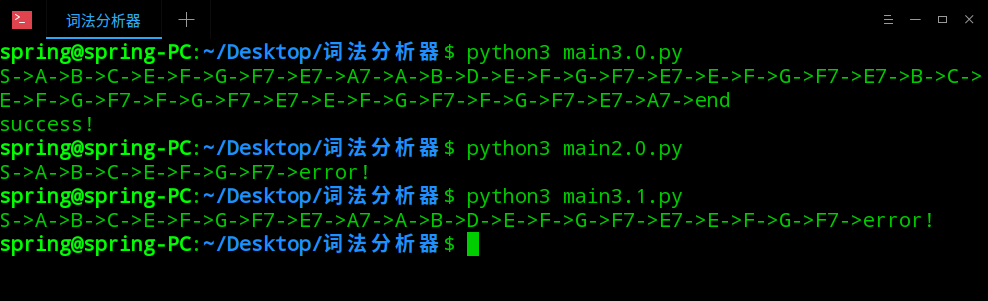
else:

print("error!")

sys.exit()

实验结果：





【小结或讨论】

1.更新使用列表保存由源代码得到的字符数据，使得不需要考虑：=两次访问的情况。

2.由PPT语法写文法时，第一次没有考虑到是否为LL(1)文法，导致出错。后将其修改为LL(1)文法。

3.在获得了正确的文法之后，递归子程序没有其他问题出现。

源代码：

import sys

global keywords

keywords={"begin":'a',"end":'b',"if":'e',"then":'f'}

global opration

opration=["+","-","\*","/",">","<",";"]

global format

format=[' ','\n','\t']

global spring

spring=[]

global word

word=''

def forword():

global spring

global word

global opration

global keywords

global format

#flag用于标记字符串是否为开头数字中间字母的错误

flag=0

#判断word是否为空

if(len(word)!=0):

#判断第一个字符是否字母

if word[0].isalpha():

#判断字符串是否为关键词

if word in keywords:

spring.append(keywords[word])

else:

spring.append("c")

else:

#对于数字开头的字符串，判断其中是否夹杂字母

for xyz in word:

if xyz.isalpha():

flag=1

break

#根据flag判断字符串是否错误

if flag==1:

print('第'+str(num+1)+'行，出现词法错误')

else:

spring.append("d")

word=''

def deal(work):

global word

global spring

global opration

global keywords

global format

for i in range(0,len(work)):

#判断字符是否为字母，如果是拼接word

if work[i].isalpha() or work[i].isdigit():

word=word+work[i]

elif work[i] in opration:

forword()

spring.append(work[i])

elif work[i]==':':

forword()

if work[i+1]=='=':

spring.append(":=")

else:

print('第'+str(num+1)+'行，出现词法错误')

elif work[i]=='=':

if work[i-1]==':':

pass

else:

print('第'+str(num+1)+'行，出现词法错误')

elif work[i] in format:

forword()

elif work[i]=="#":

forword()

break

else:

print('第'+str(num+1)+'行：未规定字符')

def anaysis():

global spring

global word

global opration

global keywords

global format

temp=[]

work=[]

#打开文件

with open('test.txt','r') as f:

temp.extend(f.readlines())

#对于每一行使用deal函数，num用于错误提示表示行数

for num in range(0,len(temp)):

deal(temp[num])

anaysis()

# global spring

# spring=['a', 'c', ':=', 'd', ';', 'e', 'c', '>', 'd', 'f', 'c', ':=', 'd', '\*', 'c', '+', 'd', '/', 'd', 'b']

global lookahead

lookahead=spring[0]

global counter

counter=1

def getToken():

global lookahead

global counter

global spring

if counter==len(spring):

pass

else:

lookahead=spring[counter]

counter=counter+1

def MatchToken(word):

if lookahead!=word:

print("error!")

sys.exit()

else:

getToken()

def ParseS():

print("S->",end='')

MatchToken("a")

ParseA()

MatchToken("b")

print("end")

print("success!")

def ParseA():

print("A->",end='')

ParseB()

ParseA7()

def ParseA7():

print("A7->",end='')

if lookahead==";":

MatchToken(";")

ParseA()

elif lookahead=="b":

pass

else:

print("error!")

sys.exit()

def ParseB():

print("B->",end='')

if lookahead=="c":

ParseC()

elif lookahead=="e":

ParseD()

else:

print("error!")

sys.exit()

def ParseC():

print("C->",end='')

MatchToken("c")

MatchToken(":=")

ParseE()

def ParseD():

print("D->",end='')

MatchToken("e")

ParseE()

MatchToken("f")

ParseB()

def ParseE():

print("E->",end='')

ParseF()

ParseE7()

def ParseE7():

print("E7->",end='')

if lookahead=="+":

MatchToken("+")

ParseE()

elif lookahead=="-":

MatchToken("-")

ParseE()

elif lookahead=="<":

MatchToken("<")

ParseE()

elif lookahead==">":

MatchToken(">")

ParseE()

elif lookahead=="f":

pass

elif lookahead==";":

pass

elif lookahead=="b":

pass

else:

print("error!")

sys.exit()

def ParseF():

print("F->",end='')

ParseG()

ParseF7()

def ParseF7():

print("F7->",end='')

if lookahead=="\*":

MatchToken("\*")

ParseF()

elif lookahead=="/":

MatchToken("/")

ParseF()

elif lookahead in "+-<>f;b":

pass

else:

print("error!")

sys.exit()

def ParseG():

print("G->",end='')

if lookahead=="c":

MatchToken("c")

elif lookahead=="d":

MatchToken("d")

elif lookahead=="(":

MatchToken("(")

ParseE()

MatchToken(")")

else:

print("error!")

sys.exit()

ParseS()