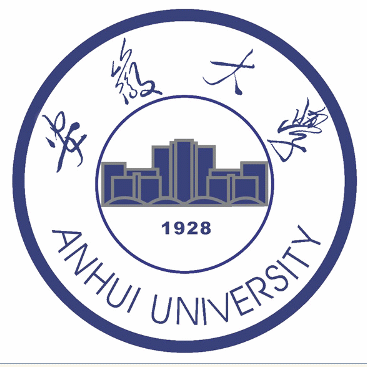
**《编译原理实验报告》**

****

**论文名称：**　　 递归下降翻译程序

**年级专业：** 15级软件工程

**姓名学号：**何铭春 E21514033

## 一.实验内容

递归子程序法又称递归下降分析法

基本原理：

1.原本递归下降分析法

对每个非终结符按其产生式结构构造相应语法分析子程序，其中终结符产生匹配命令，而非终结符则产生过程调用命令。

特点：子程序结构与产生式几乎一致

要求：文法是LL(1)

对每个非终结符 A，构造一个函数，以 A 的每个继承属性为形参，以 A 的综合属性为返回值。

2.增加语义翻译

对终结符 X，保存其综合属性x的值至专为 X.x 而声明的变量；然后调用匹配终结符（match\_token） 和取下一输入符号（next\_token）的函数；

对非终结符 B，利用相应于 B 的函数 ParseB 产生赋值语句 c:=B(b1, b2, …, bk)，其中变量 b1, b2, …, bk 对应 B的各继承属性，变量c对应B的综合属性

对语义规则集，直接copy其中每一语义规则来产生代码，只是将对属性的访问替换为对相应变量的访问。

## 二.基本思想

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。

3.对语义规则集，直接copy其中每一语义规则来产生代码，只是将对属性的访问替换为对相应变量的访问。

## 三.核心算法

1.从PPT描述中得到的LL(1)文法:

N -> . { S.f : =1} S { print(S.v) }

S -> { B.f : =S.f } B { S1.f := S.f +1 } S1 {S.v := S1.v+B.v }

S -> & { S.v := 0 }

B -> 0 { B.v := 0 }

B -> 1 { B.v := 2-B.f }

2.数据结构

spring字符串保存要转化的二进制小数,并程序自动添加#作为结尾。

global spring

spring=spring=input("请输入要转化的小数：")

spring=spring+"#"

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()

ParseN函数，在语句中增加了语义计算。

def ParseN():

print("N->",end="")

MatchToken(".")

sf=1

sv=ParseS(sf)

print("end")

print(sv)

ParseS函数，用于调用

def ParseS(sf):

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

if lookahead=="0" or lookahead=="1":

bf=sf

bv=ParseB(bf)

s1f=sf+1

s1v=ParseS(s1f)

sv=s1v+bv

elif lookahead=="#":

sv=0

else:

print("error")

sys.exit()

return sv

ParseB函数，同上

def ParseB(bf):

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

if lookahead=="0":

MatchToken("0")

bv=0

elif lookahead=="1":

MatchToken("1")

bv=2\*\*(-bf)

else:

print("error")

sys.exit()

return bv

## 四.实验结果



## 五.实验心得

1.类似上一次实验，只需要改变Parse函数的参数和返回值。增加了语义计算的语句即可。

2.例子中原本没有结束标志，导致刚写完的程序出现不断循环现象。添加#作为字符串结束标志，解决了这个问题。

3.Python中幂计算采用\*\*运算符，按习惯使用^作为运算符导致结果错误，

## 六.源代码

import sys

global spring

spring=input("请输入要转化的小数：")

spring=spring+"#"

global lookahead

lookahead=spring[0]

global counter

counter=1

def getToken():

global lookahead

global counter

global spring

lookahead=spring[counter]

counter=counter+1

def MatchToken(word):

if lookahead!=word:

print("error!")

sys.exit()

else:

getToken()

def ParseN():

print("N->",end="")

MatchToken(".")

sf=1

sv=ParseS(sf)

print("end")

print(sv)

def ParseS(sf):

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

if lookahead=="0" or lookahead=="1":

bf=sf

bv=ParseB(bf)

s1f=sf+1

s1v=ParseS(s1f)

sv=s1v+bv

elif lookahead=="#":

sv=0

else:

print("error")

sys.exit()

return sv

def ParseB(bf):

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

if lookahead=="0":

MatchToken("0")

bv=0

elif lookahead=="1":

MatchToken("1")

bv=2\*\*(-bf)

else:

print("error")

sys.exit()

return bv

ParseN()