# 编译原理实验三-递归下降子程序

## 1.实验目的

掌握最基本的自顶向下分析方法，即递归下降子程序方法，理解其特点和适用范围（回溯，左递归等现象），锻炼递归调用程序的构造方法。

## 2.实验内容

对于文法:

E→E+T | T

T→T\*F |F

F→( E ) | i

在这个实验中我们进行手工的进行左递归的消去

我们首先发现他是一个左递归的文法，

进行直接左递归的消除，

E→TE1

E1→+TE1 |ε

T→FT1

T1→\*FT1 |ε

F→(E) | i

经观察，这个文法没有间接左递归

然后事情就比较简单了，我们依照以上五条文法，依次实现读掉相应的终结符，当所有读进去的终结符都能够被正确的解析的时候，这个输入的串是合法的；反之，只要有一个字符不能被文法解析，就停止递归下降程序，此程序不合法。

## 3.具体实现

**class** ArithParser:  
 **def** \_\_init\_\_(self, symbol: str):  
 self.symbol = symbol  
 self.i = 0  
 self.is\_valid = **True** self.token = self.get\_token()

成员函数:

判断输入的串是否合法，只需要从开始符号E开始执行，然后查看is\_valid的值

**def** is\_legal(self) -> bool:  
 self.E()  
 **return** self.is\_valid

**错误处理**

**def** error(self):  
 self.is\_valid = **False  
 return**

获取下一个token以供分析

**def** get\_token(self):  
 **if** self.i < len(self.symbol):  
 idx = self.i  
 self.i = self.i + 1  
 **return** self.symbol[idx]  
 **return None**

判断文法给出的终结符与实际获得的token是否一致，一致的话继续获得下一个token,不一致执行出错处理

**def** match(self, t):  
 **if** self.token == t:  
 self.token = self.get\_token()  
 **else**:  
 self.error()

**规则**E→TE1

**def** E(self):  
 self.T()  
 self.E1()

规则E1→+TE1 |ε

**def** E1(self):  
 **if** self.token == **'+'**:  
 self.match(**'+'**)  
 self.T()  
 self.E1()  
 **else**:  
 **pass**

T→FT1

**def** T(self):  
 self.F()  
 self.T1()

T1→\*FT1 |ε

**def** T1(self):  
 **if** self.token == **'\*'**:  
 self.match(**'\*'**)  
 self.F()  
 self.T1()  
 **else**:  
 **pass**

F→(E) | i

F是比较特殊的，因为其两个产生式的第一个字符都是终结符，如果说token不在(或i中，需要错误处理

**def** F(self):  
 *# F->(E)|i* **if** (self.token == **'i'**):  
 self.match(**'i'**)  
 **elif** self.token == **'('**:  
 self.match(**'('**)  
 self.E()  
 self.match(**')'**)  
 **else**:  
 self.error()

### 测试数据如下:

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 arith = ArithParser(**'i+(i\*i)'**)  
 print(arith.is\_legal())  
 arith = ArithParser(**'(i+(i+(i\*i)+i))'**)  
 print(arith.is\_legal())  
  
 arith = ArithParser(**'i+(i\*i)+(i+i)\*i'**)  
 print(arith.is\_legal())  
 *# 不合法输入* arith = ArithParser(**'+i+(i\*i)'**)  
 print(arith.is\_legal())  
 arith = ArithParser(**'(i+(i+(i\*i)+i)'**)  
 print(arith.is\_legal())

结果如下:

True

True

True

False

False

### 总结:

这个实验总体来说还是比较简单的一个，因为我手工构造了这个文法的左递归消去，其他的工作就基本上是按照文法去将相应的函数填进函数体就好。

当然，在单步跟踪这个递归下降程序的时候，能够很清晰的感受到这个程序在回溯的情况还是比较多的，因此就像课本上以及理论上所展示的那样，直接的递归下降子程序效率很糟糕，解析复杂的串需要非常长的时间，这也是为什么会有后面的LL(1)等的原因。