输入输出说明：

本程序的运行需要在 **example.py 文件**中输入需要解析的表达式，（example.py 文件需放在和所有程序放在一个目录）执行本程序则只需要运行 **main.py 文件即可**。

解析样例如下：

```
a=1
b=2
c=a+b
d=c-1+a
print(c)
print(a,b,c)
```

程序输出结果如下：

```
C:\Users\pc\AppData\Local\Programs\Python\Python37\python.exe D:/编译原理实践/四则运算/main.py
 + [PROGRAM]
   + [STATEMENTS]
     + [STATEMENTS]
       + [STATEMENTS]
         + [STATEMENTS]
           + [STATEMENTS]
             + [STATEMENTS]
               + [STATEMENTS]
                 + ['STATEMENT']
                   + [ASSIGNMENT]
                     + a
                     + =
                     + 1
               + ['STATEMENT']
                 + [ASSIGNMENT]
                   + b
                   + =
                   + 2
           + ['STATEMENT']
             + [OPERATION]
               + c
               + =
               + [OPERATION_EXPR]
                 + [OPERATION_EXPR]
                   + [OPERATION_TERM]
                     + [FACTOR]
                       + [VARIABLE]
                       + a
                 + +
                 + [OPERATION_TERM]
                   + [FACTOR]
```

```
                + [OPERATION_EXPR]
                  + [OPERATION_TERM]
                    + [FACTOR]
                      + [VARIABLE]
                      + c
                + -
                + [OPERATION_TERM]
                  + [FACTOR]
                    + [NUMBER]
                    + 1
              + +
              + [OPERATION_TERM]
                + [FACTOR]
                  + [VARIABLE]
                  + a
      + ['STATEMENT']
        + [PRINT]
          + [PRINT_ELEMENTS]
            + c
    + ['STATEMENT']
      + [PRINT]
        + [PRINT_ELEMENTS]
          + a
          + [PRINT_ELEMENTS]
            + b
            + [PRINT_ELEMENTS]
              + c
c 3.0
a 1.0
b 2.0
c 3.0
{'a': 1.0, 'b': 2.0, 'c': 3.0, 'd': 3.0}
```

设计思路：

本次实验的难点主要在于遍历生成语法树，所以着重分析 translation.py 中的方法：
getExpr() getTerm()分别负责加减部分和乘除部分，使用递归
首先是 getExpr()部分

```
def getExpr(node):
    '''expr  : expr '+' term
             | expr '-' term
             | term'''
    leftNode = node.getchild(0)
    if(leftNode.getdata()=="[OPERATION_TERM]"):
        return getTerm(leftNode)

    op = node.getchild(1).getdata()
    rightNode=node.getchild(2)

    if(leftNode.getdata()=="[OPERATION_EXPR]" and rightNode.getdata()=="[OPERATION_TERM]"):
        if(op=='+'):
            return getExpr(leftNode) + getTerm(rightNode)
        else:
            return getExpr(leftNode) - getTerm(rightNode)
```

首先判断是否为文法的第三种情况 是的话直接返回值
若不是则获取二三节点的值
根据运算符不同 进行递归调用

getTerm 部分也同理 根据乘除号进行递归调用 或者直接返回值

```
def getTerm(node):
    '''term  : term '*' factor
             | term '/' factor
             | factor'''
    leftNode=node.getchild(0)
    if(leftNode.getdata()=="[FACTOR]"):return getFactor(leftNode)

    op = node.getchild(1).getdata()
    rightNode=node.getchild(2)

    if(leftNode.getdata()=="[OPERATION_TERM]" and rightNode.getdata()=="[FACTOR]"):
        if(op=='*'):
            return getTerm(leftNode) * getFactor(rightNode)
        else:
            return getTerm(leftNode) / getFactor(rightNode)
```

getFactor 部分

```
def getFactor(node):
    '''factor  : VARIABLE
               | NUMBER'''
    nodeId = node.getchild(0)
    ValueId = node.getchild(1)
    if (nodeId.getdata() == "[NUMBER]"):
        return eval(ValueId.getdata())
    else:
        return v_table[ValueId.getdata()]
```

根据标签判断是常数还是变量 是变量则去变量表中获取
多变量 PRINT 函数部分

```
def p_print(t):
    '''print : PRINT '(' elements ')' '''
    t[0]=node('[PRINT]')
    t[0].add(t[3])

def p_elements(t):
    '''elements : VARIABLE ',' elements
                | VARIABLE'''
    t[0]=node('[PRINT_ELEMENTS]')
    if(len(t)==2):
        t[0].add(node(t[1]))
    else:
        t[0].add(node(t[1]))
        t[0].add(t[3])
```

更新了文法 可以在 print 中递归的容纳多个元素

```
def getPrint(node):
    '''elements : VARIABLE ',' elements
                | VARIABLE'''
    nodeEle = node.getchild(0)
    print(nodeEle.getdata(),v_table[nodeEle.getdata()])
    if(node.getSize()==2):
        nodeNext=node.getchild(1)
        getPrint(nodeNext)
```

翻译部分也是一个递归调用函数。
如果判断当前节点有 2 个子节点，则打印节点并进行一次递归调用
否则直接打印数据。此种方法可以按照命令顺序自上而下正确打印。而非倒序调用。