输入输出说明：

本次实验的解析样例已经放在 **stu.py** 文件中，（stu.py 文件需放在和 main.py 放在一个目录下）执行本程序则只需要运行 **main.py 文件即可**。

解析样例如下：

```
class Student{
    def __init__(self,name,age,score){
        self.name=name
        self.age=age
        self.score=score
    }

    def add_score(self,score){
        self.score=self.score+score
    }

    def print_info(self){
        print(self.name,self.age)
    }
}

a=Student('xiaoming',12,20)
a.add_score(60)
a.print_info()
```

程序输出结果如下：

```
C:\Users\pc\AppData\Local\Programs\Python\Python37\py
Illegal character '''
Illegal character '''
 + [PROGRAM]
   + [STATEMENTS]
     + [STATEMENTS]
       + [STATEMENTS]
         + [STATEMENTS]
           + ['STATEMENT']
             + [CLASS]
               + Student
               + [CLASS_INIT]
                 + name
                 + age
                 + score
                 + [CLASS_ASSIGN]
                   + self.name
                   + name
                 + [CLASS_ASSIGN]
                   + self.age
                   + age
                 + [CLASS_ASSIGN]
                   + self.score
                   + score
               + [CLASS_ADD]
                 + score
                   + score
                 + [CLASS_PRINT]
                   + [PRINT]
                     + name
                     + age
             + ['STATEMENT']
               + [CLASS_CONSTRUCT]
                 + a
                 + [CLASS_NAME]
                 + Student
                 + [INPUT_PARA]
                 + xiaoming
                 + 12
                 + 20
           + ['STATEMENT']
             + [MEMBERVAR_ADD]
               + a
               + [MEMBERVAR_NAME]
               + score
               + [INPUT_PARA]
               + 60
         + ['STATEMENT']
           + [MEMBERVAR_PRINT]
             + a
{'name': 'xiaoming', 'age': '12', 'score': 80}
the result below is class_memeber and var_class_table
{'Student': ['name', 'age', 'score']}
{'a': {'name': 'xiaoming', 'age': '12', 'score': 80}}

Process finished with exit code 0
```

设计思路：

类的分析与之前不在于有构造函数，自己的成员变量，所以要针对这两样进行分析
首先将类名 初始化函数 改变成员变量函数 打印成员变量函数加入语法树

```
def p_class(t):
    r'''class : CLASS VARIABLE '{' def_init def_add def_print '}' '''
    t[0]=node('[CLASS]')
    t[0].add(node(t[2]))
    t[0].add(t[4])
    t[0].add(t[5])
    t[0].add(t[6])
```

因为需要解析的 Student 类需要三个成员变量所以需要有三个节点分别将其加入语法树中

```
def p_def_init(t):
    r'''def_init : DEF init '(' SELF ',' VARIABLE ',' VARIABLE ',' VARIABLE ')' '{' class_assign class_assign class_assign '}' '''
    t[0]=node('[CLASS_INIT]')
    t[0].add(node(t[6]))
    t[0].add(node(t[8]))
    t[0].add(node(t[10]))
    t[0].add(t[13])
    t[0].add(t[14])
    t[0].add(t[15])
```

改变成员变量值函数 默认 add_var var 即为要改变的变量
把改变的 VAR 和操作函数加入语法树

```
def p_def_add(t):
    r'''def_add : DEF add VARIABLE '(' SELF ',' VARIABLE ')' '{' class_operation '}' '''
    t[0]=node('[CLASS_ADD]')
    t[0].add(node(t[3]))
    t[0].add(t[10])
```

因为类的打印与常规打印不同所以要构建单独对类成员的打印

```
def p_def_print(t):
    r'''def_print : DEF defprint '(' SELF ')' '{' print '}' '''
    t[0]=node('[CLASS_PRINT]')
    t[0].add(t[7])
```

调用成员函数部分
此处也考虑了三种情况：

调用构造函数
对于测试样例中的各种类的用法进行如下分析：
记录哪个变量调用了构造函数 将调用的类名 和输入的变量参数记录入语法树

```
a=Student(xiaoming,12,20)
```

```
+ ['STATEMENT']
  + [CLASS_CONSTRUCT]
    + a
    + [CLASS_NAME]
    + Student
    + [INPUT_PARA]
    + xiaoming
    + 12
    + 20
```

```
def p_classfunc0(t):
    r'''classfunc0 : VARIABLE '=' VARIABLE
    t[0]=node('[CLASS_CONSTRUCT]')
    t[0].add(node(t[1]))
    t[0].add(node('[CLASS_NAME]'))
    t[0].add(node(t[3]))
    t[0].add(node('[INPUT_PARA]'))
    t[0].add(node(t[5]))
    t[0].add(node(t[7]))
    t[0].add(node(t[9]))
```

改变成员变量函数

记录要改变的变量名 成员变量名 改变的数值

```
a.add_score(60)
```

```
def p_classfunc1(t):
    r'''classfunc1 : VARIABLE '.' add VARIABLE '(' NUMBER ')' '''
    t[0]=node('[MEMBERVAR_ADD]')
    t[0].add(node(t[1]))
    t[0].add(node('[MEMBERVAR_NAME]'))
    t[0].add(node(t[4]))
    t[0].add(node('[INPUT_PARA]'))
    t[0].add(node(t[6]))
```

打印函数

记录要打印的变量名

```
def p_classfunc2(t):
    r''' classfunc2 : VARIABLE '.' defprint '(' ')' '''
    t[0]=node('[MEMBERVAR_PRINT]')
    t[0].add(node(t[1]))
```

语法制导翻译部分

进行了一定的简化，反复使用历史代码会产生问题

检测到类函数

记录各个节点，调用初始化节点开始初始化。

```
if node.getdata() == '[CLASS]':
    CLASS_NAME=node.getchild(0).getdata()
    CLASS_INIT=node.getchild(1)
    CLASS_ADD=node.getchild(2)
    CLASS_PRINT=node.getchild(3)
    classinit(CLASS_INIT,CLASS_NAME)
```

初始化函数

如图向 class_member 中写入给定类所含的成员变量

```
{'Student': ['name', 'age', 'score']}
```

```
def classinit(node,name):
    para1=node.getchild(0).getdata()
    para2=node.getchild(1).getdata()
    para3=node.getchild(2).getdata()
    class_member[name]=[para1,para2,para3]
```

调用构造函数
获取变量名和类名 以及输入的参数名

```
if node.getdata() == '[CLASS_CONSTRUCT]':
    VAR_NAME=node.getchild(0).getdata()
    CLASS_NAME=node.getchild(2).getdata()
    PARA1=node.getchild(4).getdata()
    PARA2=node.getchild(5).getdata()
    PARA3=node.getchild(6).getdata()
```

调用成员的类名和输入参数构造键值对生成 hashmap
并将其和变量名相匹配 放入 var_class_table 中

```
membervar_table = {}
membervar_table[class_member[CLASS_NAME][0]]=PARA1
membervar_table[class_member[CLASS_NAME][1]]=PARA2
membervar_table[class_member[CLASS_NAME][2]]=PARA3


var_class_table[VAR_NAME]=membervar_table
```

```
{'a': {'age': '12', 'score': 80, 'name': 'xiaoming'}}
```

改变成员变量的函数
获取变量名 成员变量名 改变量
改变 var_class_table 中键值对的值

```
if node.getdata() == '[MEMBERVAR_ADD]':
    VAR_NAME=node.getchild(0).getdata()
    MEMBERNAME=node.getchild(2).getdata()
    PARA=node.getchild(4).getdata()
    var_class_table[VAR_NAME][MEMBERNAME]=int(var_class_table[VAR_NAME][MEMBERNAME])+int(PARA)
```

打印函数 输出对应变量名的 table

```
if node.getdata()=='[MEMBERVAR_PRINT]':
    VAR_NAME=node.getchild(0).getdata()
    print(var_class_table[VAR_NAME])
```