**编译原理 实验报告**

|  |  |  |
| --- | --- | --- |
| **评 语（4号楷体）** | **成绩** |  |
| 教 师： 邓岳  年 月 日 | | |

**教学班级： 1 张立勇班**

**学生学号： 20009100850**

**学生姓名： 项路**

**实验日期： 2022.12**

**一、实验目的**

通过实验加深对编译器构造原理和方法的理解，巩固所学知识。

1. 会用正规式设计简单语言的词法；

2. 会用产生式设计简单语言的语法；

3. 会用递归下降子程序编写语言的解释器。

**二、实验环境（软硬件环境）**

Python 3.10.8

astunparse 1.6.3

numpy 1.24.2

pip 21.1.2

**三、实验内容（设计/实现思路，即如何完成的。不允许贴大段代码）**

1.语言

本次实验我采用了Python作为编写的语言，程序编译是计算密集型应用，而 Python 又不是执行性能非常出色的语言，可我坚持选择的理由如下：

(1). Python对于字符串处理操作比较方便。解释器需要与源代码进行交互，并且将源代码作为字符串输入，所以使用了Python。

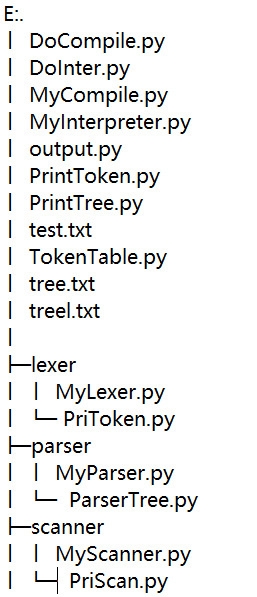
(2). Python 的类型系统比较灵活。解释器需要实现多种数据结构，而 Python 自带的类型就可以很方便地实现这些数据结构，比如 Python的字典(dict)比起其他语言的结构体、Map、类都要方便，Python 的元组可以通过嵌套很方便地实现树形结构。虽然正是这点给我带来了不少的困扰。

(3). Python 拥有大量的标准库和第三方库。些库涵盖了各种功能，而我本人在实验的过程中确实用到了不少库来简化操作。比如我使用了turtle库就可以很方便地进行绘图，而ast库用于打印语法树，可以检查我对语法树是否正确。显著提高了开发效率。

(4). Python 是一种以可读性为主要目标设计的语言。它的语法简洁明了，代码的排版规则也被广泛认可，使得 Python 的代码易于阅读和理解。这对于编写编译器来说尤为重要，因为编译器的代码往往比较复杂和庞大，可读性高的代码可以让我好地理解和维护。

2.模块设计

首先是解释器项目目录



非常清晰地可以看出，该项目总共有5个模块，lexer词法分析器、scanner扫描器，parser语法分析器及语义分析器及中间代码优化生成器，compile编译器和interpreter解释器，还有一张token表。其他的程序都是用于打印输出。

TokenTable.py定义了枚举类Token

Mylexer.py定义了词法分析器类Lexer

Myparser.py定义了语法语义分析器类Parser，对递归下降子程序做了简单封装。

Myscanner.py定义了扫描器类Scanner

Mycompile.py定义了编译器类Compile，可以将输入文本编译成python源程序后执行并得到相应输出

Myinterpreter.py定义了解释器类Interpreter，可以直接将文本文件作为输入，输出得到相应的图案

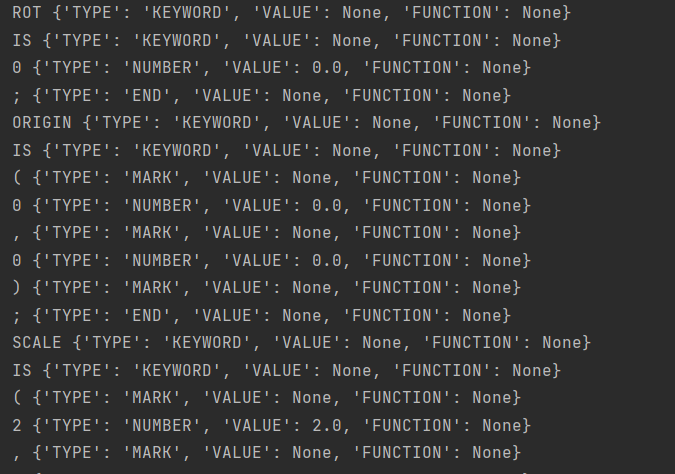
2.词法分析器和扫描器

词法分析器以源代码作为输入，输出记号流，识别记号的工具主要就是DFA。在初始化一个词法分析器时，会初始化一个DFA集合。

我将不同的符号分类读入，首先去识别直接可识别的符号，即各种指令如STEP、FOR等。如果找不到，就进入更高级的DFA中识别，主要是\*、字母、数字和其他符号的识别。\*考虑和\*\*的区别，主要是一个上下文有关文法判断是独立的一颗\*还是\*\*。数字则需考虑负号和小数点在第一位的情况。最后进行注释的排除。

此外，我还设置了输入缓冲区，实现读入一个字符就进行一次操作，我认为这么做可以保证安全性，防止读入全部字符后处理时系统崩溃则又需要读入所有字符。

部分输出结果如下：

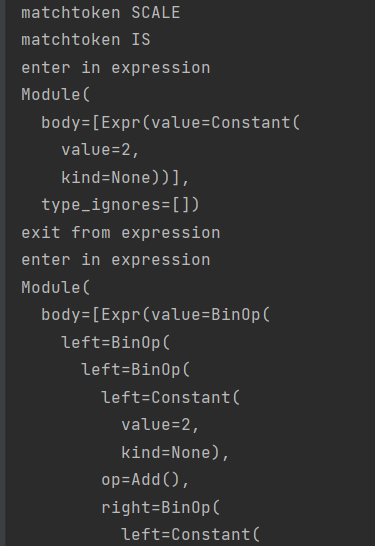


3.语法分析器

语法分析器以记号流作为输入，输出语法树。

为了简便起见，这里我采用递归下降子程序实现。由于简化的，消除左递归、左因子、二义性的文法的EBNF描述已经在文档给出了。所以这部分的实现只需要写出递归下降的子程序即可。

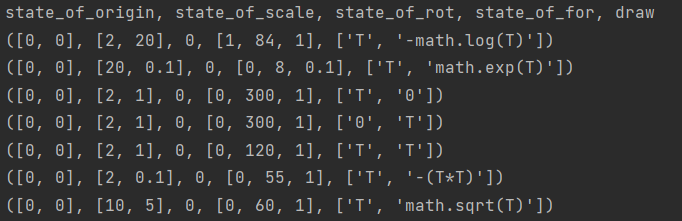
语法树的部分截图如下：



4.语义分析器

语义分析器以语法树为输入，输出中间代码。我在第一次制作的时候确实是非常标准地按照语法分析器语义分析器的流程编写的，但是我在反复测试的过程中意识到语法分析在得到语法树后便结束了，那我为什么不能将程序编写得更加集成一些呢。于是我将语义分析和语法分析结合在一个程序中，可以不用输出语法树就直接使用语法树，应该是一种类似于语法制导的语义分析的思想。并且再写了一个方法用于输出语法树来debug。

语义分析的结果如下：

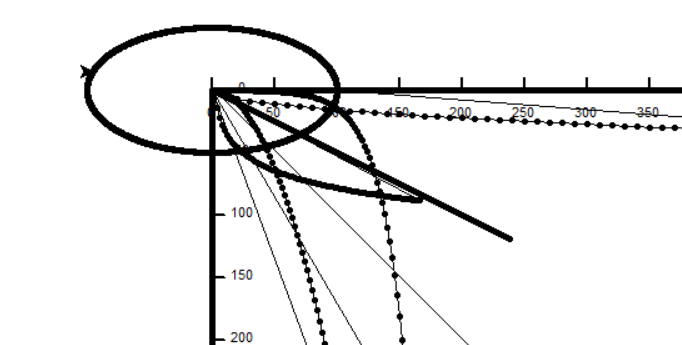


5.执行器

执行器以操作序列为输入，执行绘图操作。

在进行这方面的制作时，我又想到Python是一个解释型语言，那我是不是顺便模拟一下编译型语言的过程，加深我对这门课的理解。于是我编写了一个编译器和一个解释器。

编译器将中间代码翻译成Python代码，并输出到文件output.py中，而后运行output.py能得到和直接运行解释器相同的输出结果。



**四、心得体会（完成时遇到哪些问题以及问题是怎样解决的，目前的做法的优点及不足）**

优点到不如说是得意的地方，一个是改进了一下流程，使用了语法制导的语义分析的思想，将语法分析的结果直接运用在语义分析当中，更简洁。另一个是我写了编译器和解释器，有两种方法来实现简单的函数绘图。

不足之处则是感觉这么写有点玩弄技艺的感觉，没有什么实际意义。

问题：

老师的实验课讲得比较透彻，只有在语法分析阶段有较大的问题，印象深刻，其他的问题都和Python的语言特性有关。

1.语法分析。个人认为计算的优先级给我造成了不小的困扰。我解决该难点的做法是使用产生式的产生顺序来确定符号的优先级，将优先级高的运算符号放在后面的产生式中，而在创建树的过程中会先递归调用底层的产生式函数，因此确保了运算的优先级

2.Python的包管理

我在同一个目录下的子目录中调用另一个子目录的包，确保了语法的正确性，但是还是找不到我需要的类，因此我也只能将编译器和解释器的程序放到根目录，导致项目结构不是很明晰。

3.类信息丢失。Python的类必定有dict属性，这也是我选择Python的原因之一，但是我在测试的过程中，出现报错显示找不到类的dict属性。经过查询文档，我怀疑是库版本冲突，在更换以后解决。

4.语义分析及中间代码生成中Python的浅复制和深复制。因为在Python中，一切皆是对象，因此复制都是对对象的引用，而浅复制可以理解成一种不完全的引用。这导致了我最初得到的中间代码是错误的，查询文档后我使用字符串将语义分析的输出转换为列表，问题得到了解决。

通过本次实验，我对编译器和Python特性的理解都得到了提高，非常好。