编译小作业1.4——Toy语言的LR(1)分析

在编译小作业1中，你需要对某个简单的编程语言（称为Toy语言）进行分析，并自己编写代码，实现词法分析和语法分析。

编译小作业1分为不同部分，将逐步公布在网络学堂作业区。如对作业内容有疑义或问题，可以联系助教。

Toy语言的介绍请参考编译小作业1.1的作业附件。语法规则与词法规则定义请参考编译小作业1.2的作业附件。

在编译小作业1.4中，请基于之前的词法分析结果，进行基于LR(1)的语法分析。

作业任务——基于LR(1)的语法分析

编译小作业1共15分，其中本次作业占5分。需要实现一个LR(1)语法分析器（**不能直接使用语法分析工具，需要自己编写代码实现**）。该语法分析器可以接受语法规则输入，得到对应的LR(1)的ACTION表与GOTO表，进而利用词法分析器的结果生成格式为XML格式的抽象语法树。

LR(1)分析方法较为复杂，需要：

- 得到LR(1)自动机
    - 需要逐步计算LR(1)自动机的状态
    - 每次状态变迁后，将新状态与已有状态进行比较，以确定是否已经存在相同状态
- 构建ACTION表和GOTO表
- 编写一个基于action表和goto表的LR(1)分析程序
    - 使用栈来处理状态和符号，执行相应的操作

提示：

- 如果需要，你可以对Toy语言的语法规则进行修改。
- 如果需要，你可以在源文件结尾添加一个结束符号。
- 请参考课上介绍的LR(1)相关知识进行代码编写。
- 你可以自行设计自动机的存储结构。
- 以python为例，
    - 语法规则

```
S -> A B | C
A -> a
B -> b
C -> c
```

可以定义成以下形式（大写字母表示非终结符，其他表示终结符）：

```
grammar = {
    'S': [['A', 'B'], ['C']],
    'A': [['a']],
    'B': [['b']],
    'C': [['c']],
}
```

- 栈可以用python中的list模拟（使用append和pop函数）

该作业包括以下两个任务：

1. 使用你实现的LR(1)语法分析器，输出以下`G[E]`（大写字母表示非终结符，其他表示终结符）对应的LR(1)的ACTION表和GOTO表：

   ```
   E -> (L, E) | F
   L -> L, E | E
   F -> (F) | d
   ```

   提示：在python中，该语法规则可以写为：

   ```
   grammar = {
       'E': [['(', 'L', ',', 'E', ')'], ['F']],
       'L': [['L', ',', 'E'], ['E']],
       'F': [['(', 'F', ')'], ['d']],
   }
   ```

2. 使用你实现的LR(1)语法分析器，对Toy语言的语法规则进行处理。并结合词法分析器，实现：输入Toy语言的源程序，输出XML格式的抽象语法树的一个完整程序。

   提示：

   - `identifier`、`number`、`string_literal`类型的token直接作为非终结符进行处理。
   - ACTION表和GOTO表里面涉及到的非终结符类型可能较多，请不要遗漏。

提交作业时，请提交一个压缩包，包括两个任务对应的源代码文件、可执行文件和一个文档，文档中展示三个你自己设计的测试源代码以及对应输出结果。

- “可执行文件”，指的是Windows平台的exe程序、python脚本等可直接执行的程序。
- 如果用python的话不用编译，写个readme文件说明运行方式即可。