摘要：

LL 分析器是一种处理某些上下文无关文法的自顶向下分析器。因为它从左（Left）到右处理输入，再对句型执行最左推导出语法树（Left derivation，相对于 LR 分析器）。能以此方法分析的文法称为 LL 文法。本次课程设计中设计了基于 Python 3 实现的表格驱动的 LL(1)语法分析器，实现了从输入文法，对所输入文法进行分析，消除左递归，计算 FIRST、FOLLOW、SELECT 集，判断是否为 LL(1)文法，构造预测分析表，对输入的句子进行语法分析，预测分析中的错误分析等一系列操作。遵守 Python 3 面向对象的开发原则进行开发，具有良好的可移植性和安全性，并利用 Git 作为版本控制工具，代码托管在 GitHub 方便协作开发。项目地址：https://github.com/AndyZhuAZ/Compilation-Course-Design

关键词：LL 分析器；Python；表格驱动的分析器；

目　录

目录

1 选题内容

1.1 选题要求

(1) 基于 PL/0 语言，通过编程判断该文法是否为 LL(1)文法；
(2) 计算出文法的 First() 、Follow()
(3) 构造相应文法的预测分析表
(4) 对某个输入句子进行语法分析

1.2 需求分析

需要程序读入文本文件中的文法，首先需要对文法进行分析，当文法存在间接左递归时对文法进行间接左递归转换直接左递归，存在直接左递归时直接消除直接左递归，再对文法中出现的符号进行分类：非终结符集与终结符集，并分别计算 FIRST、FOLLOW、SELECT 集，构建预测分析表，完成对输入句子的语法分析。

2 实现原理

2.1 LL(1)文法

LL(1)文法是一类可以进行确定的自顶向下语法分析的文法。就是要求描述语言的文法是无左递归的和无回溯的。根据 LL(1)文法的定义，对于同一非终结符 A 的任意两个产生式 A:=a 和 A:=b，都要满足：SELECT(A：=a)∩SELECT(A:=b)=Ø。

2.1.1 文法的左递归

当一个文法是左递归文法时，采用自顶向下分析法会使分析过程进入无穷循环之中。所以采用自顶向下语法分析需要消除文法的左递归性。文法的左递归是指若文法中对任一非终结符 A 有推导 AÞA…，则称该文法是左递归的。 左递归又可以分为直接左递归和间接左递归。

直接左递归

若文法中的某一产生式形如 A→Aα，α∈V*

则称该文法是直接左递归的。 消除直接左递归的方法： 设有产生式是关于非终结符 A 的直接左递归：A→Aα|β （α,β∈V*，且 β 不以 A 开头） 对 A 引入一个新的非终结符 A′，把上式改写为：

A →βA′
A′→αA′|ε

间接左递归

若文法中存在某一非终结符 A，使得 AÞA…至少需要两步推导，则称该文法是间接左递归的。消除间接左递归的方法：

【方法一】采用代入法把间接左递归变成直接左递归。

【方法二】直接改写文法：设有文法 G10[S]： S→Aα|β (1) A→Sγ (2) 因为 SÞAαÞSγα，所以 S 是一个间接递归的非终结符。为了消除这种间接左递归，将(2)式代入(1)式，即可得到与原文法等价的文法（可以证明）： S→Sγα|β (3) (3)式是直接左递归的，可以采用前面介绍的消除直接左递归

的方法，对文法进行改写后可得文法：S→βS′ S′→γαS′|ε

2.2 计算 FRIST 集

(1) 若 X∈VT ，则 First(X)={X} (2) 若 X∈VN ，且有产生式 X→a…, a∈VT 则 First(X)={X}

(3) 若 X∈VN ，且有产生式 X→ε,则 First(X)={X} (4) 若 X，Y1 ，Y2 ，…，Yn 都∈VN，而由产生式 X→Y1 Y2 …Yn 。当 Y1 ，Y2 ，…，Yi-1 都能推导出 ε 时，（其中 1≤i≤n），则 First(Y1)-{ε}, First(Y2)-{ε},…, First(Yi)都包含在 First(X)中 (5)当(4)中所有 Yi 都能推导出 ε，（i=1，2，…，n），则 First(X)=First(Y1)∪First(Y2)∪…First(Yn)∪{ε} 反复使用上述步骤直到每个符合的 First 集合不再增大为止。

2.3 计算 FOLLOW 集

对文法中的每个 A∈VN，计算 Follw(A)： (1) 设 S 为文法的开始符合，把{#}加入 Follow(S)中； (2) 若 A→αBβ 是一个产生式，则把 First(β)的非空元素加入 Follow(B)中，如果 β 能推导出 ε，则把 Follow(A)也加入(B)中； (3) 反复使用以上步骤直到每个非终结符号的 Follow 集不再增大为止。

2.4 预测分析方法

预测分析方法是自顶向下分析的另一种方法，一个预测分析器是由三个部分组成：预测分析程序；先进后出栈；预测分析表。

2.5 计算 SELECT 集与 LL(1)文法判断

SELECT 集合定义如下：

一个产生式的选择符号集 SELECT。给定上下文无关文法的产生式 A→α,A∈VN,α∈V∗A→α,A∈VN,α∈V∗，若α⇏∗εα⇏∗ε，则 SELECT(A→α)=FIRST(α)SELECT(A→α)=FIRST(α)。

如果 α ⇒∗εα⇒∗ε ， 则 SELECT(A → α)=(FIRST(α)−{ε})∪FOLLOW(A)SELECT(A → α)=(FIRST(α)−{ε})∪FOLLOW(A)。

根据 SELECT 集的定义，我们可以知道，实际上相同左部的产生式 SELECT 不相交与如下 LL(1)文法判例等价：

文法 G 是 LL(1)的，当且仅当 G 的任意两个具有相同左部的产生式 A→α|β 满足下面的条件：

如果 α 和 β 均不能推导出 ε ，则 FIRST (α)∩FIRST (β) =Φ

α 和 β 至多有一个能推导出 ε

如果 β⇒* ε，则 FIRST (α)∩FOLLOW(A) =Φ； 如果 α⇒* ε，则 FIRST (β)∩FOLLOW(A) =Φ；

因此一个上下文无关文法是 LL(1)文法的充分必要条件是，对每个非终结符 A 的两个不同产生式，A→α，A→βA→α，A→β，满足

SELECT(A→α)⋂SELECT(A→β)=∅

SELECT(A→α)⋂SELECT(A→β)=∅

其中α、βα、β不同时能⇒∗ε⇒∗ε。

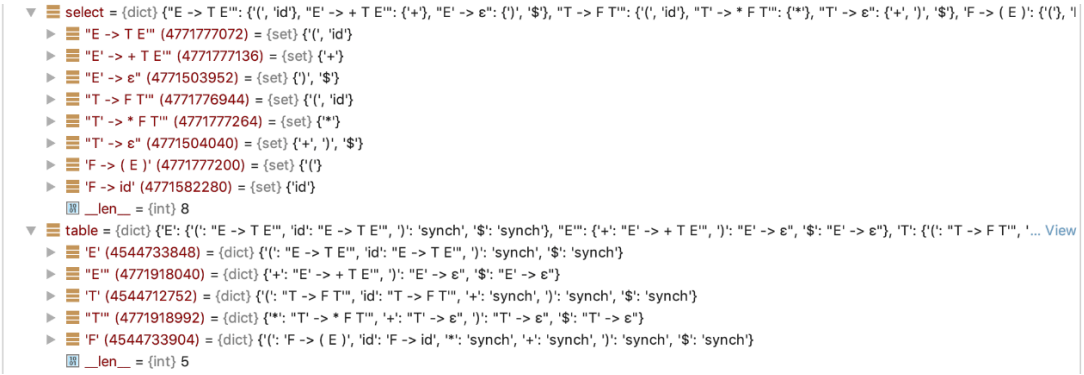
3 系统设计

3.1 数据结构

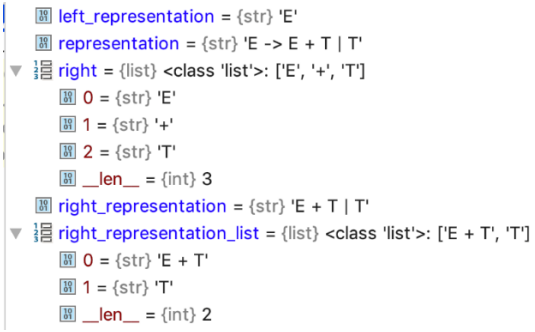
```
def __init__(self):
    self.representation = []     # 表达式
    self.first_state = ''        # 初始状态

    self.VT = set()              # 终结符
    self.VN = set()              # 非终结符

    self.first = dict()          # first集
    self.follow = dict()         # follow集
    self.select = dict()         # select集
    self.table = dict()          # 预测分析表
```

主要数据结构

表达式使用 list 数据结构，list 中每个字符串，代表一个表达式，初始状态使用字符串表示，终结符，非终结符，选用 set 集合形式，防治重复加入，而 first，follow，select 集，以及预测分析表，使用字典 dict 形式，便于查表，其中预测分析表为二级字典。



字典数据结构

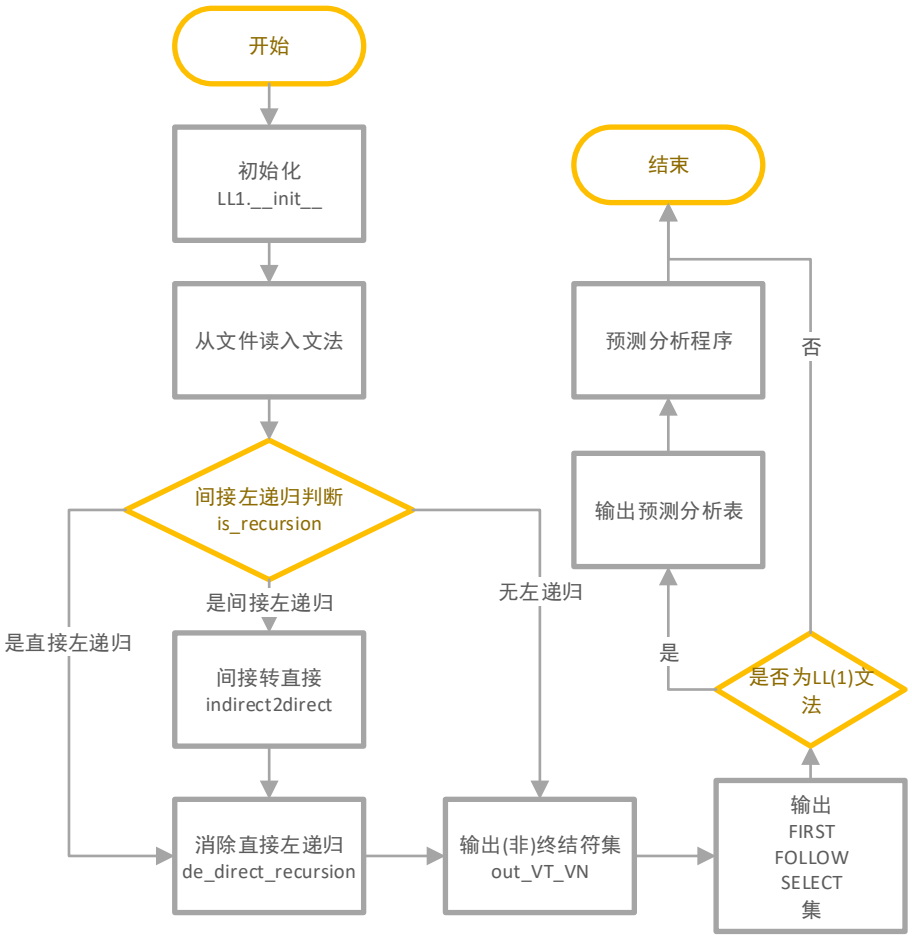


列表数据结构

3.2 算法设计

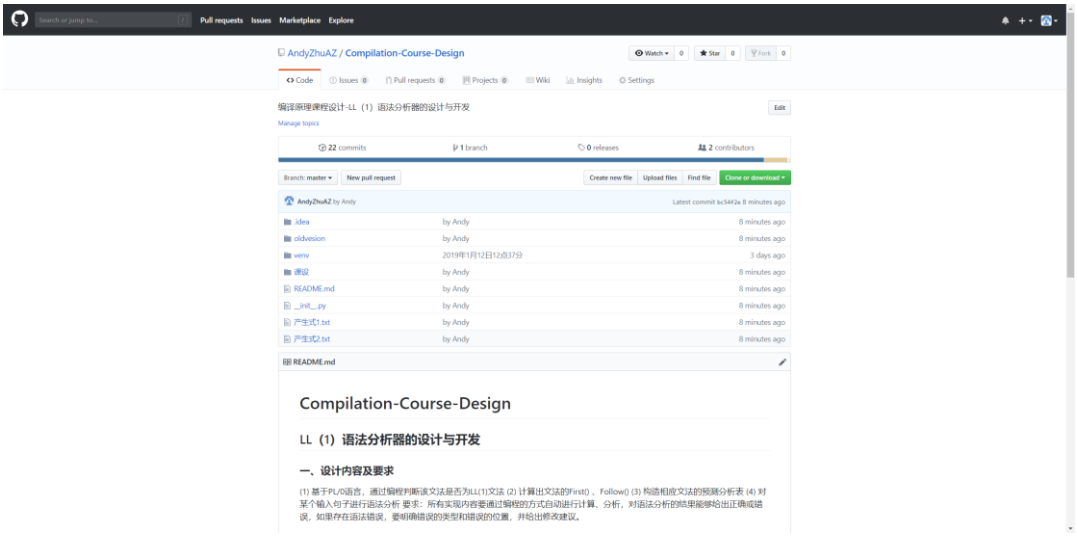
该算法从本地读取用户输入文法，检查文法是否存在直接或间接左递归，若存在间接左递归，则将间接左递归转化为直接左递归，并将消除直接左递归。将用户输入转化为不含左递归表达式后，根据推导关系，计算用户输入文法的非终结符，终结符集合。再根据表达式计算非终结符的 FIRST，FOLLOW，SELECT 集合，并根据 SELECT 集合，来判断该文法是否符合 LL(1)文法规则，若不符合文法规则，推出程序，若符合文法规则，根据 FIRST，FOLLOW 集合，构建预测分析表，并对用户输入表达式进行语法分析。

3.3 流程图



4 主要代码

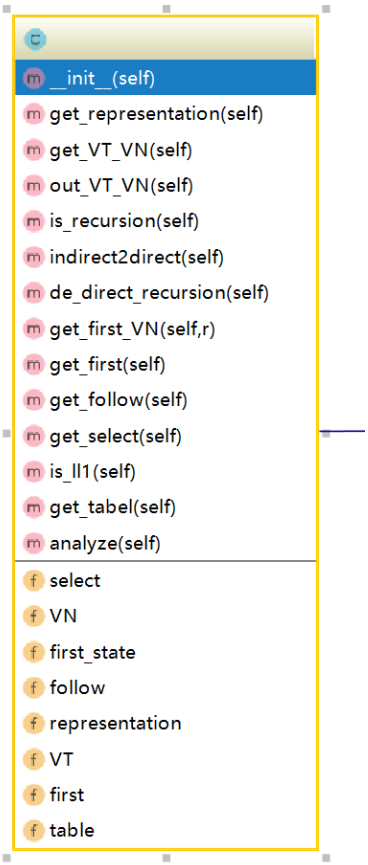
完整项目托管在 GitHub：https://github.com/AndyZhuAZ/Compilation-Course-Design



代码清单 https://github.com/AndyZhuAZ/Compilation-Course-Design/blob/master/__init__.py
此处仅列举主要代码结构

右图为 Python 类 方法与 Fields 清单：

```
__init__.py ×  __init__.py ×
10
11
12  class LL1:
13      def __init__(self):...
24
25      def get_representation(self):...
40
41      def get_VT_VN(self):...
55
56      def out_VT_VN(self):...
65
66      # 直接间接左递归选择入口
67      def is_recursion(self):...
114
115     def indirect2direct(self):...
157
158     def de_direct_recursion(self):...
209
210     def get_first_VN(self, r):...
226
227     def get_first(self):...
248
249     def get_follow(self):...
313
314     def get_select(self):...
335
336     def is_ll1(self):...
364
365     def get_tabel(self):...
405
406     def analyze(self):...
503
504
505 if __name__ == '__main__':
506     pass
507     ll1 = LL1()
508     ll1.get_representation()  # 获取产生式
509     ll1.is_recursion()  # 递归处理
510     ll1.out_VT_VN()  # 输出非终结符终结符集
511     ll1.get_first()  # 输出FIRST集
512     ll1.get_follow()  # 输出FOLLOW集
513     ll1.get_select()  # 输出SELECT集
514     ll1.is_ll1()  # 判断是否为LL(1)文法
515     ll1.get_tabel()  # 输出预测分析表
516     for i in range(10):
517         ll1.analyze()  # 启动分析程序
518
```



方法与 Fields 清单

5 程序调试

实验环境

Python 3.7.0
PyCharm 2018.3.3 (Professional Edition)
Windows 10 10.0
pandas 0.23.4
prettytable 0.7.2

Git version 2.20.1

GitHub

调试过程

1 左递归调试

2 FOLLOW 集调试

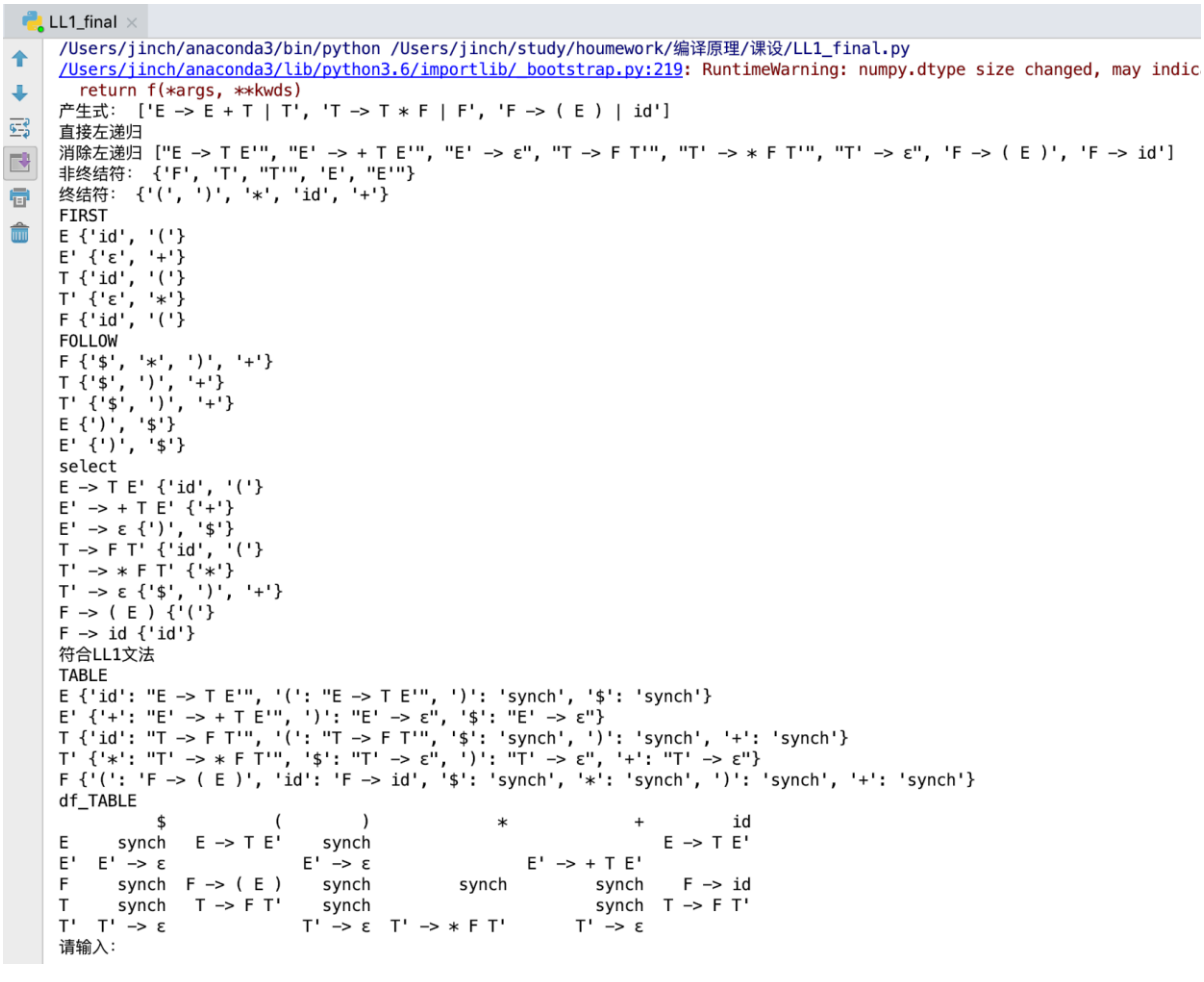
计算非终结符的 FOLLOW 集时，由于非终结符使用 set()数据类型，set()是一个无序集，根据 FOLLOW 计算规则，“如果存在一个产生式 A→αB，或存在产生式 A→αBβ 且 first（β）包含 ε，那么 follow（A）中的所有符号都在 follow（B）中”，当出现这种情况是，计算 FOLLOW(B)需要依赖 FOLLOW(A)，而因为非终结符集无序，所以存在计算顺序问题，此处使用暴力方法，计算 3 次 FOLLOW 集，第一次不考虑依赖情况只计算请他情况下的 FOLLOW 集，第二次考虑依赖，再次重新计算，第三次在第二次基础上，再次计算，以保证 FOLLOW 集计算正确。

3 分析程序调试

当输入表达式 id＋id）时，无法报错，根据文法 F->(E)|id，只有 F->()，而其他情况下，(，)一律不会进入堆栈，当出现单独(时，()都进栈，可以识别缺少)，但是当出现单独）时，(，)均没有进栈，无法识别缺失（问题。经过查找发现，此处存在一个堆栈未排空的问题，通过在结束位置增加额外判断解决。

6 运行样例

输入含有直接左递归文法 E -> E + T | T，T -> T * F | F，F -> (E) | id



消除左递归，FIRST，FOLLOW，SELECT，及预测分析表结果

请输入：(id + id (

栈	输入	动作
['$', 'E']	['(', 'id', '+', 'id', '(', '$']	
['$', "E'", 'T']	['(', 'id', '+', 'id', '(', '$']	E -> T E'
['$', "E'", "T'", 'F']	['(', 'id', '+', 'id', '(', '$']	T -> F T'
['$', "E'", "T'", ')', 'E', '(']	['(', 'id', '+', 'id', '(', '$']	F -> (E)
['$', "E'", "T'", ')', 'E']	['id', '+', 'id', '(', '$']	匹配(
['$', "E'", "T'", ')', "E'", 'T']	['id', '+', 'id', '(', '$']	E -> T E'
['$', "E'", "T'", ')', "E'", "T'", 'F']	['id', '+', 'id', '(', '$']	T -> F T'
['$', "E'", "T'", ')', "E'", "T'", 'id']	['id', '+', 'id', '(', '$']	F -> id
['$', "E'", "T'", ')', "E'", "T'"]	['+', 'id', '(', '$']	匹配id
['$', "E'", "T'", ')', "E'"]	['+', 'id', '(', '$']	T' -> ε
['$', "E'", "T'", ')', "E'", 'T', '+']	['+', 'id', '(', '$']	E' -> + T E'
['$', "E'", "T'", ')', "E'", 'T']	['id', '(', '$']	匹配+
['$', "E'", "T'", ')', "E'", "T'", 'F']	['id', '(', '$']	T -> F T'
['$', "E'", "T'", ')', "E'", "T'", 'id']	['id', '(', '$']	F -> id
['$', "E'", "T'", ')', "E'", "T'"]	['(', '$']	匹配id
['$', "E'", "T'", ')', "E'"]	['(', '$']	error, 位置: 4, 字符: (, 缺少: ['*']
['$', "E'", "T'", ')', "E'"]	['$']	T' -> ε
['$', "E'", "T'", ')']	['$']	E' -> ε
['$', "E'", "T'"]	['$']	error, 位置: 5, 字符: $, 缺少:)
['$', "E'"]	['$']	T' -> ε
['$']	['$']	E' -> ε

输入错误表达式文法分析

请输入：id + (id * id)

栈	输入	动作
['$', 'E']	['id', '+', '(', 'id', '*', 'id', ')', '$']	
['$', "E'", 'T']	['id', '+', '(', 'id', '*', 'id', ')', '$']	E -> T E'
['$', "E'", "T'", 'F']	['id', '+', '(', 'id', '*', 'id', ')', '$']	T -> F T'
['$', "E'", "T'", 'id']	['id', '+', '(', 'id', '*', 'id', ')', '$']	F -> id
['$', "E'", "T'"]	['+', '(', 'id', '*', 'id', ')', '$']	匹配id
['$', "E'"]	['+', '(', 'id', '*', 'id', ')', '$']	T' -> ε
['$', "E'", 'T', '+']	['+', '(', 'id', '*', 'id', ')', '$']	E' -> + T E'
['$', "E'", 'T']	['(', 'id', '*', 'id', ')', '$']	匹配+
['$', "E'", "T'", 'F']	['(', 'id', '*', 'id', ')', '$']	T -> F T'
['$', "E'", "T'", ')', 'E', '(']	['(', 'id', '*', 'id', ')', '$']	F -> (E)
['$', "E'", "T'", ')', 'E']	['id', '*', 'id', ')', '$']	匹配(
['$', "E'", "T'", ')', "E'", 'T']	['id', '*', 'id', ')', '$']	E -> T E'
['$', "E'", "T'", ')', "E'", "T'", 'F']	['id', '*', 'id', ')', '$']	T -> F T'
['$', "E'", "T'", ')', "E'", "T'", 'id']	['id', '*', 'id', ')', '$']	F -> id
['$', "E'", "T'", ')', "E'", "T'"]	['*', 'id', ')', '$']	匹配id
['$', "E'", "T'", ')', "E'", "T'", 'F', '*']	['*', 'id', ')', '$']	T' -> * F T'
['$', "E'", "T'", ')', "E'", "T'", 'F']	['id', ')', '$']	匹配*
['$', "E'", "T'", ')', "E'", "T'", 'id']	['id', ')', '$']	F -> id
['$', "E'", "T'", ')', "E'", "T'"]	[')', '$']	匹配id
['$', "E'", "T'", ')', "E'"]	[')', '$']	T' -> ε
['$', "E'", "T'", ')']	[')', '$']	E' -> ε
['$', "E'", "T'"]	['$']	匹配)
['$', "E'"]	['$']	T' -> ε
['$']	['$']	E' -> ε

对输入正确表达式文法分析

输入含有间接左递归文法 S -> A a | b，A -> A c | S d | ε

产生式： ['S -> A a | b', 'A -> A c | S d | ε']
间接左递归
间接左递归变直接左递归 ['A -> A c | ε | A a d | b d']
消除左递归 ["A -> b d A'", "A -> A'", "A' -> c A'", "A' -> ε", "A' -> a d A'", "A' -> ε"]

间接左递归转化直接左递归，再消除左递归

```
非终结符： {"A'", 'A'}
终结符： {'a', 'b', 'd', 'c'}
FIRST
A {'a', 'b', 'ε', 'c'}
A' {'a', 'ε', 'c'}
FOLLOW
A' {'$'}
A {'$'}
select
A -> b d A' {'b'}
A -> A' {'a', 'b', 'ε', 'c'}
A' -> c A' {'c'}
A' -> ε {'$'}
A' -> a d A' {'a'}
符合LL1文法
TABLE
A {'b': "A -> b d A'", 'a': "A -> A'", 'c': "A -> A'", '$': "A -> A'"}
A' {'c': "A' -> c A'", '$': "A' -> ε", 'a': "A' -> a d A'"}
df_TABLE
          $             a            b           c
A   A -> A'       A -> A'  A -> b d A'     A -> A'
A'  A' -> ε  A' -> a d A'                A' -> c A'
```

计算 FIRST，FOLLOW，SELECT 集，以及预测分析表

总　　结

本次课程设计耗时 3 天，利用 GitHub 协作开发，使用了 Git 版本控制，基于 Python 3 完成了 LL(1)分析器，朱建玮负责了左递归处理的两个方法及 SELECT 集计算与 LL(1)文法判断和代码维护，金辰宇负责了 FIRST、FOLLOW 集计算和分析表构建与分析程序，在整体开发中，我们灵活的使用了 Python 特有的字典数据结构和对列表字符串处理的官方库，使得各部分功能在较短的代码量下能完成地更多更安全。在 LL(1)判断中，选用了同一非终结符的产生式的可选集不相交作为判断条件，引入可选集概念，使得算法思想相比课本更加简化。Python set()集合运算方法应用在 FIRST 集 FOLLOW 集 SELECT 集的计算和应用中，Python string.split()方法可以轻松的对输入字符串进行分割，通过指定分隔符，我们利用 split 方法实现了对多字符的符号进行处理例如 E’、id。Python dict 字典数据结构应用于 FIRST，FOLLOW，SELECT 集，以及预测分析表，便于查表，其中预测分析表为二级字典。通过本次课程设计，我们加深了对编译原理这门课程的理解，熟练掌握了文法分析和表驱动分析程序的构建。

致　谢

首先要感谢为我们授课付老师的指导和教学，感谢计算机工程学院老师的辛勤工作，感谢中国大学 MOOC 平台的来自哈尔滨工业大学的编译原理课程对实验所需的部分拓展补充，××…………

注：

1. 致谢内容要求 600 字左右，主要感谢计算机工程学院提供的实践机会，实验室人员提供的实验环境，指导教师的辛勤指导，同组同学的互帮互助，参考文献的原作者，以及其他给你提供过帮助的所有人员和机构等；
2. 首行空二个汉字；
3. 字体为“宋体，小四号，1.5 倍行距” ，数字、西文字母等用 Times New Roman 字体；
4. 其它要求同正文格式规范。

参 考 文 献

1 [美]Andrew W.Appel / [美]MaiaGinsburg. 现代编译原理.北京：人民邮电出版社，2018

2 LL 剖析器. Zh.wikipedia.org. https://zh.wikipedia.org/wiki/ LL 剖析器. Published 2019. Accessed January 15, 2019

3 编译原理 FIRST 集、FOLLOW 集、SELECT 集求法通俗解释 & LL(1)文法判定 - 多反思，多回顾，要坚持。 - CSDN 博客. Blog.csdn.net. https://blog.csdn.net/liujian20150808/article/details/72998039. Published 2019. Accessed January 15, 2019.

指导教师评语

学号		姓名		班级	
选题名称					
序号	评价内容			权重（%）	得分
1	考勤记录、学习态度、工作作风与表现。			5	
2	自学情况： 上网检索机时数、文献阅读情况（笔记）。			10	
3	论文选题是否先进，是否具有前沿性或前瞻性。			5	
4	成果验收： 是否完成设计任务；能否运行、可操作性如何等。			20	
5	报告的格式规范程度、是否图文并茂、语言规范及流畅程度；主题是否鲜明、重心是否突出、论述是否充分、结论是否正确；是否提出了自己的独到见解。			30	
6	文献引用是否合理、充分、真实。			5	
7	答辩情况： 自我陈述、回答问题的正确性、用语准确性、逻辑思维、是否具有独到见解等。			25	
合计					

指导教师（签章）：_____

_____年_____月_____日