题目

实验内容及要求

编写语法分析程序，实现对算术表达式的语法分析。要求所分析算数表达式由如下的文法产生。

```
E→ E+T | E-T | T
T→ T*F | T/F | F
F→ (E) | num
```

实验要求

在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

实现方法要求：

~~方法 1：~~

~~编写递归调用程序实现自顶向下的分析。~~

方法 2：

编写 LL(1)语法分析程序，要求如下。 （必做）

1. 编程实现算法 4.2，为给定文法自动构造预测分析表。
2. 编程实现算法 4.1，构造 LL(1)预测分析程序 。

方法 3：

编写语法分析程序实现自底向上的分析，要求如下。（必做）

1. 构造识别该文法所有活前缀的 DFA。
2. 构造该文法的 LR 分析表。
3. 编程实现算法 4.3，构造 LR 分析程序。

~~方法 4：~~

~~利用 YACC 自动生成语法分析程序，调用 LEX 自动生成的词法分析程序。~~

程序使用说明

入口

程序的入口文件为 `gram/main/main.go`

- 直接运行程序：`go run main.go`
- 构建可执行程序：`go build main.go`

注意：请在 `terminal` 中运行程序，直接点击程序可能会闪一下然后消失

直接运行可执行文件

直接运行：

- `gram/main(win).exe` （windows）
- `gram/main(mac)` （macos）

程序输入

程序的所有输入是以json文件的形式输入的，文件放在`gram/base/def.json`，其示例结构如下：

```
{
  "tags": [
    { "type": "终结符", "value": "+" },
    { "type": "终结符", "value": "-" },
    { "type": "终结符", "value": "*" },
    { "type": "终结符", "value": "/" },
    { "type": "终结符", "value": "(" },
    { "type": "终结符", "value": ")" },
    { "type": "终结符", "value": "num" },
    { "type": "终结符", "value": "ε" },
    { "type": "非终结符", "value": "E" },
    { "type": "非终结符", "value": "F" },
    { "type": "非终结符", "value": "T" }
  ],
  "productions": [
    "E → E+T | E-T | T",
    "T → T*F | T/F | F",
    "F → (E) | num"
  ],
  "input": "(num/num)",
  "method": "LR"
}
```

json文件中应包含：`tags`，`productions`，`input`和`method`

`tags`

type 只能为 `"终结符"` 和 `"非终结符"` 两者之一

value 为终结符或非终结符的值

`productions`

生成式，是一个字符串数组，子式与子式之间用`|`分割，左部与右部之间用`→`分割，允许存在空格，分析程序在执行分析之前会先去除空格

`input`

待分析的输入串，是一个字符串，注意：不需要在这里手动添加`$`

`method`

分析程序使用的方法，只能为`LR`和`LL`二者之一

程序设计说明

> 姓名：陈威豪
> 学号：2019211232
> 班级：2019211303

手工计算例题

原表达式

```
E → E+T | E-T | T
T → T*F | T/F | F
F → (E) | num
```

消除左递归

```
E → TE'
E' → +TE' | -TE' | ϵ
T → FT'
T' → *FT' | /FT' | ϵ
F → (E) | num
```

原表达式`FIRST`集

```
E: ( num
F: ( num
T: ( num
```

原表达式`FOLLOW`集

```
E: $ + - )
F: $ ) + - * /
T: $ ) + - * /
```

消除左递归`FIRST`集

```
E: ( num
F: ( num
T: ( num
E': + - ϵ
T': * / ϵ
```

消除左递归FOLLOW集

```
E: $ )
F: * / + - $ )
T: + - $ )
E': $ )
T': + - $ )
```

求LL分析表

```
+----+----------+----------+----------+----------+--------+-------+--------+-----
--+
|    |    +     |    -     |    *     |    /     |   (    |   )   |  num   |   $
  |
+----+----------+----------+----------+----------+--------+-------+--------+-----
--+
| E' | E'->+TE' | E'->-TE' |          |          |        | E'->ϵ |        | E'-
>ϵ |
| E  |          |          |          |          | E->TE' |       | E->TE' |
  |
| F  |          |          |          |          | F->(E) |       | F->num |
  |
| T' |  T'->ϵ   |  T'->ϵ   | T'->*FT' | T'->/FT' |        | T'->ϵ |        | T'-
>ϵ |
| T  |          |          |          |          | T->FT' |       | T->FT' |
  |
+----+----------+----------+----------+----------+--------+-------+--------+-----
--+
```

LL分析过程

```
+------+-------------+------------+----------+
| STEP |    STACK    |   INPUT    |  OUTPUT  |
+------+-------------+------------+----------+
| (1)  |     $E      | (num/num)$ |  E->TE'  |
| (2)  |    $E'T     | (num/num)$ |  T->FT'  |
| (3)  |   $E'T'F    | (num/num)$ |  F->(E)  |
| (5)  |   $E'T')E   | num/num)$  |  E->TE'  |
| (6)  |  $E'T')E'T  | num/num)$  |  T->FT'  |
| (7)  | $E'T')E'T'F | num/num)$  |  F->num  |
| (9)  | $E'T')E'T'  |   /num)$   | T'->/FT' |
| (11) | $E'T')E'T'F |   num)$    |  F->num  |
| (13) | $E'T')E'T'  |     )$     |  T'->ϵ   |
| (14) |  $E'T')E'   |     )$     |  E'->ϵ   |
+------+-------------+------------+----------+
```

LR1的项目集规范族

```
I0:
E'->·E, $
E->·E+T, $ + -
E->·E-T, $ + -
E->·T, $ + -
T->·T*F, $ + - * /
```

```
T->·T/F, $ + - * /
T->·F, $ + - * /
F->·(E), $ + - * /
F->·num, $ + - * /

I1:
E'->E·, $
E->E·+T, $ + -
E->E·-T, $ + -

I2:
E->T·, $ + -
T->T·*F, $ + - * /
T->T·/F, $ + - * /

I3:
T->F·, $ + - * /

I4:
F->(·E), $ + - * /
E->·E+T, ) + -
E->·E-T, ) + -
E->·T, ) + -
T->·T*F, ) + - * /
T->·T/F, ) + - * /
T->·F, ) + - * /
F->·(E), ) + - * /
F->·num, ) + - * /

I5:
F->num·, $ + - * /

I6:
E->E+·T, $ + -
T->·T*F, $ + - * /
T->·T/F, $ + - * /
T->·F, $ + - * /
F->·(E), $ + - * /
F->·num, $ + - * /

I7:
E->E-·T, $ + -
T->·T*F, $ + - * /
T->·T/F, $ + - * /
T->·F, $ + - * /
F->·(E), $ + - * /
F->·num, $ + - * /

I8:
T->T*·F, $ + - * /
F->·(E), $ + - * /
F->·num, $ + - * /

I9:
T->T/·F, $ + - * /
F->·(E), $ + - * /
F->·num, $ + - * /
```

```
I10:
F->(E·), $ + - * /
E->E·+T, ) + -
E->E·-T, ) + -

I11:
E->T·, ) + -
T->T·*F, ) + - * /
T->T·/F, ) + - * /

I12:
T->F·, ) + - * /

I13:
F->(·E), ) + - * /
E->·E+T, ) + -
E->·E-T, ) + -
E->·T, ) + -
T->·T*F, ) + - * /
T->·T/F, ) + - * /
T->·F, ) + - * /
F->·(E), ) + - * /
F->·num, ) + - * /

I14:
F->num·, ) + - * /

I15:
E->E+T·, $ + -
T->T·*F, $ + - * /
T->T·/F, $ + - * /

I16:
E->E-T·, $ + -
T->T·*F, $ + - * /
T->T·/F, $ + - * /

I17:
T->T*F·, $ + - * /

I18:
T->T/F·, $ + - * /

I19:
F->(E)·, $ + - * /

I20:
E->E+·T, ) + -
T->·T*F, ) + - * /
T->·T/F, ) + - * /
T->·F, ) + - * /
F->·(E), ) + - * /
F->·num, ) + - * /

I21:
E->E-·T, ) + -
T->·T*F, ) + - * /
T->·T/F, ) + - * /
```

```
T->·F, ) + - * /
F->·(E), ) + - * /
F->·num, ) + - * /

I22:
T->T*·F, ) + - * /
F->·(E), ) + - * /
F->·num, ) + - * /

I23:
T->T/·F, ) + - * /
F->·(E), ) + - * /
F->·num, ) + - * /

I24:
F->(E·), ) + - * /
E->E·+T, ) + -
E->E·-T, ) + -

I26:
E->E+T·, ) + -
T->T·*F, ) + - * /
T->T·/F, ) + - * /

I27:
E->E-T·, ) + -
T->T·*F, ) + - * /
T->T·/F, ) + - * /

I28:
T->T*F·, ) + - * /

I29:
T->T/F·, ) + - * /
```

LR分析表

	+	-	*	/	(	)	num	$	E	F	T
0					S4		S5		1	3	2
1	S6	S7						ACC			
2	R3	R3	S8	S9		R3		R3			
3	R6	R6	R6	R6		R6		R6			
4					S13		S14		10	12	11
5	R8	R8	R8	R8		R8		R8			
6					S4		S5			3	15
7					S4		S5			3	16
8					S4		S5			17	
9					S4		S5			18	
10	S20	S21				S19					
11	R3	R3	S22	S23		R3		R3			
12	R6	R6	R6	R6		R6		R6			
13					S13		S14		24	12	11
14	R8	R8	R8	R8		R8		R8			
15	R1	R1	S8	S9		R1		R1			
16	R2	R2	S8	S9		R2		R2			

```
| 17 | R4  | R4  | R4  | R4  |     | R4  |     | R4  |    |    |    |
| 18 | R5  | R5  | R5  | R5  |     | R5  |     | R5  |    |    |    |
| 19 | R7  | R7  | R7  | R7  |     | R7  |     | R7  |    |    |    |
| 20 |     |     |     |     | S13 |     | S14 |     |    | 12 | 25 |
| 21 |     |     |     |     | S13 |     | S14 |     |    | 12 | 26 |
| 22 |     |     |     |     | S13 |     | S14 |     |    | 27 |    |
| 23 |     |     |     |     | S13 |     | S14 |     |    | 28 |    |
| 24 | S20 | S21 |     |     |     | S30 |     |     |    |    |    |
| 25 | R1  | R1  | S22 | S23 |     | R1  |     | R1  |    |    |    |
| 26 | R2  | R2  | S22 | S23 |     | R2  |     | R2  |    |    |    |
| 27 | R4  | R4  | R4  | R4  |     | R4  |     | R4  |    |    |    |
| 28 | R5  | R5  | R5  | R5  |     | R5  |     | R5  |    |    |    |
+----+-----+-----+-----+-----+-----+-----+-----+-----+----+----+----+
```

LR分析过程

```
+------+-----------------------------------------+------------+--------+
| STEP |                  STACK                  |   INPUT    | OUTPUT |
+------+-----------------------------------------+------------+--------+
|  1   |          State: 0  ; Symbol: -          | (num/num)$ |   S4   |
|  2   |         State: 0 4  ; Symbol: -(        | num/num)$  |  S14   |
|  3   |      State: 0 4 14  ; Symbol: -(num     |   /num)$   |   R8   |
|  4   |       State: 0 4 12  ; Symbol: -(F      |   /num)$   |   R6   |
|  5   |       State: 0 4 11  ; Symbol: -(T      |   /num)$   |  S23   |
|  6   |    State: 0 4 11 23  ; Symbol: -(T/     |   num)$    |  S14   |
|  7   | State: 0 4 11 23 14  ; Symbol: -(T/num  |     )$     |   R8   |
|  8   |  State: 0 4 11 23 28  ; Symbol: -(T/F   |     )$     |   R5   |
|  9   |       State: 0 4 11  ; Symbol: -(T      |     )$     |   R3   |
|  10  |       State: 0 4 10  ; Symbol: -(E      |     )$     |  S19   |
|  11  |    State: 0 4 10 19  ; Symbol: -(E)     |     $      |   R7   |
|  12  |         State: 0 3  ; Symbol: -F        |     $      |   R6   |
|  13  |         State: 0 2  ; Symbol: -T        |     $      |   R3   |
|  14  |         State: 0 1  ; Symbol: -E        |     $      |  ACC   |
+------+-----------------------------------------+------------+--------+
```

基本算法及思路

求解`FIRST`集

对于目标非终结符，查找所有以它为左部的产生式，可能有以下两种情况：

1. 产生式右部以终结符开始
2. 产生式右部以非终结符开始

对于第一种情况，终结符可以直接放到`FIRST`集中，对于第二种情况，则递归查询非终结符的`FIRST`集；

求解`FOLLOW`集

首先，对于递归查询函数，如果查询符号为开始符号，则将`$`符加到`FOLLOW`集中；

然后遍历查找右部中包含目标符号的产生式，可能出现以下几种情况：

1. 查询符号在产生式中间（即不在最后）
    1. 符号之后的其他符号中，存在`First`集不包括空的非终结符，或者终结符
    2. 符号之后的其他符号中，所有符号的`First`集中都包含空

2. 查询符号在产生式末尾

对于1.1，直接将之后的终结符，或者非终结符的`FIEST`集，加到查询符号的`FOLLOW`集中；

对于另外的情况，则将左部的`FOLLOW`集加到查询符号的`FOLLOW`集中；

基本`struct`及`func`定义

`base` pkg

标识符`Tag`

```
type Tag struct {
    // 类型
    Type    int
    // 值
    Value   string
}
```

产生式`Production`

```
type Production struct {
    // 当且仅当 Left.Type == NONTERM
    Left Tag
    // 产生式的右部是一个标识符切片
    Right []Tag
}

// GetProductionsByTag 根据记号，返回该非终结符的所有产生式
func GetProductionsByTag(productions []Production, left Tag) ([]Production,
error) {}

// ToString 将产生式转换成字符串
func (p Production) ToString() string {}
```

消除左递归

直接将消除左递归后的产生式集放入堆空间中，替换未消除左递归的产生式集。

```
// RemoveLeftRecursion 消除左递归
func RemoveLeftRecursion() {}
```

构造拓广文法

调用函数后，会更新存放在栈空间中的全局变量`productions`和`prodMap`

```
// GenerateExtension 构造拓广文法
func GenerateExtension() {}
```

检测是否有重复的`tag`

```
// HasReTags 是否有重复的tag
func HasReTags(tag Tag, tags []Tag) bool {
```

生成`FIRST`集

传入一个非终结符，计算该非终结符的`First`集，`getFirstRE`是递归运算函数，处于`GetFirst`下层，不被其他程序调用

```
// GetFirst 根据推导式的左部，得到其对应的FIRST集
func GetFirst(left Tag) []Tag {}

// GetFirstRE 递归查找First集，并将该次调用得到的tag加到ansTags中
func getFirstRE(symbol Tag, tmpTags []Tag, ansTags *[]Tag) {}
```

生成`FOLLOW`集

传入一个非终结符，计算该非终结符的`FOLLOW`集，`getFollowRE`是递归运算函数，处于`GetFollow`下层，不被其他程序调用

```
// GetFollow 根据推导式的左部，得到其对应的Follow集
func GetFollow(left Tag) []Tag {}

func getFollowRE(left Tag, depth int) []Tag {}
```

`def` pkg

json文件结构的对应的结构体

读取json文件得到`Def`后，需要进行转换才能由程序进行后续操作

```
type Tag struct {
    Type  string `json:"type"`
    Value string `json:"value"`
}

type Def struct {
    Tags        []Tag    `json:"tags"`
    Productions []string `json:"productions"`
}

// InitDef 初始化操作，需在程序的入口处执行，以将json文件的内容读到内存中去
func InitDef() error {}

// GetTags 返回的是程序解析时真正使用的 base.Tag
func GetTags() []base.Tag {}

// GetProductions 返回的是程序解析时使用的 base.Production
// 形如 "E → E+T | E-T | T" 的产生式会被拆分为三个 production
func GetProductions() []base.Production {}
```

`json`文件结构举例

```
{
  "tags": [
    { "type": "终结符", "value": "+" },
    { "type": "终结符", "value": "-" },
    { "type": "终结符", "value": "*" },
    { "type": "终结符", "value": "/" },
    { "type": "终结符", "value": "(" },
    { "type": "终结符", "value": ")" },
    { "type": "终结符", "value": "num" },
    { "type": "非终结符", "value": "E" },
    { "type": "非终结符", "value": "F" },
    { "type": "非终结符", "value": "T" }
  ],
  "productions": [
    "E → E+T | E-T | T",
    "T → T*F | T/F | F",
    "F → (E) | num"
  ]
}
```

LL(1)分析程序

结构定义

预测分析表

在最外层是一个以非终结符为键的map，其值为一个map，为了便于区分，我们称这个子map为map2，map2的键为终结符，值为产生式，该结构如下：

```
// LLTable LL(1)预测分析表
type LLTable map[base.Tag]map[base.Tag]base.Production
```

生成分析表 GenerateLLTable

```
// GenerateLLTable 生成LL(1)预测分析表
func GenerateLLTable() LLTable {}
```

打印分析表

```
// PrintLLTable 打印LL分析表
func PrintLLTable(table LLTable) error {}
```

`Tag`栈

```
// TagStack Tag栈，并在最开始放上$和开始符
var TagStack = []base.Tag{
    {Type: base.TERM, Value: "$"},
}

// PopStack 从栈顶弹出一个tag，并返回该tag
func PopStack() base.Tag {}

// PushStack 向栈顶添加一个tag
func PushStack(tag base.Tag) {}
```

分析程序

```
// LLAnalyze LL分析程序
func LLAnalyze(input string, table LLTable) error {}

// LLAnalyze LL分析程序
func LLAnalyze(input string, table LLTable) error {}
```

处理思路

前置条件

1. 无左递归
2. 任意两非终结符的`FIRST`集不相交
3. 如果有可空符号，则其`FOLLOW`集与`FIRST`集不相交

构造预测分析表

1. 对于每一个产生式，如果有形如`A->aB`的形式，则在`a`列添加该产生式
2. 对于首个符号为非终结符的情况，则通过其`FIRST`集填充LL分析表
3. 对于推到空的情况，则在所有左部的`FOLLOW`集中的元素下，加上这个空推导

构造LL(1)分析程序

首先，构造一个`Tag`栈，并在最开始放上`$`和开始符；构造一个输入队列；

函数主体为一个循环，每次循环扫描输入字符串的若干个字符，这若干个字符组成一个Tag；

此时判断`Tag`栈元素，可能有以下两种情况：

1. 栈顶为终结符
    - 如果两者相同，则从栈中弹出该终结符
    - 如果两者不同，则认为分析时出错——输入串不符合该语法；
2. 栈顶为非终结符
    1. 根据LL分析表、栈顶元素与剩余输入串中的首个`Tag`，得到对应生成式
    2. 将生成式反序入栈，进行下一次比对

当输入串中只剩下`$`时，判断`Tag`栈：

- 如果栈中的非终结符与`$`结合都能推出空，则接受输入语句
- 否则，认为输入语句不能接受

LR(1)分析程序

结构及函数定义

项目集

```
// Group 项目集，如IO
type Group struct {
   Index  int                          // 项目集编号
   PDs        []base.ProductionWithDot   // 项目集中的产生式合集
}
```

项目集规范族

```
// Groups 项目集规范族
var Groups []Group
```

项目集之间的关系

```
type RelationB struct {
   Tag   base.Tag
   State int
}

// GroupRelation 用于表示项目集之间的关系，是一个map的数组，数组的索引表示Group.Index，
// map中以Tag为键，以对应的Group.Index 为值
var GroupRelation []map[base.Tag]RelationB
```

扩充生成`group`

```
// ExpandGroup 根据传入的pd，扩充生成group，并在Groups中查重
// 如果还没有重复的，则将group添加到Groups中
// 无论是否有重复，都返回对应项目集在Groups中的索引
func ExpandGroup(pd base.ProductionWithDot) int {}
```

消去重复的项目集

```
// RemoveRE 去重，包括Group和GroupRelation
func RemoveRE() {}
```

合并`group`

```
// MergeGroup 传入索引切片，合并group到索引小的那个（第一个）
func MergeGroup(is []int) {}

// ExpandGroupRE 递归的扩充group，gi为传入groups中的要分析的group的索引
func ExpandGroupRE(group *Group, gi int) {}
```

打印项目集

```
func PrintFamily() {}
```

生成项目集

```
// GenerateFamily 根据 prodMap 生成项目集规范族
func GenerateFamily() {}
```

分析程序相关结构

```
// StateStack 状态栈
var StateStack = []int{0}

// PopStateStack 从栈顶弹出一个state，并返回该state
func PopStateStack() int {}

// PushStateStack 向栈顶添加一个tag
func PushStateStack(state int) {}

// SymbolStack 符号栈
var SymbolStack = []base.Tag{
    {
        Type:  base.TERM,
        Value: "-",
    },
}

// PopSymbolStack 从栈顶弹出一个tag，并返回该tag
func PopSymbolStack() base.Tag {}

// PushSymbolStack 向栈顶添加一个tag
func PushSymbolStack(tag base.Tag) {}

// 用于存储分析过程
type Proc struct {
    Step   string
    Stack  string
    Input  string
    Output string
}
var Procedures []Proc

// LRAnalyze LR1分析程序
func LRAnalyze(input string, table LRTable) error {}

// 打印分析过程
func PrintProcedure() {}

// 将int切片转换为字符串
func ConvertIntSliceToStr(nums []int) string {}
```

LR分析表相关结构

```
// AG Action or Goto
type AG struct {
   Type  int
   Value int
}
```

```go
// LRTable LR分析表，最外层为切片，索引表示Group的Index，内层为map，键为tag，值为AG
type LRTable []map[base.Tag]AG

// ToString 转为字符串
func (a AG) ToString() string {}

// GenerateLRTable 根据Group、GroupRelation生成表格
func GenerateLRTable() LRTable {}

// 打印LR分析表
func PrintLRTable(table LRTable) error {}
```

处理思路

构造拓广文法

- 程序处理的过程中，认为在`def.json`中的第一个产生式的左部是文法的开始符
- 如开始符为E，则添加`E'->E`到拓广文法中

构造项目集规范族

1. 根据起始符`S'`构造`pd`，并扩充为`group`
2. 设置一个指针，遍历指向`Groups`中的一个元素
3. 指针每指向一个`group`，就遍历`group`中的式子
4. 根据式子中的点，生成`group`，然后用返回的`index`填充`GroupRelation`，如果填充前发现`map`中已经有值，则进行`merge`
5. 最后进行项目集的去重

根据`DFA`构造LR分析表

根据LR分析表进行LR1分析程序

测试报告

输入及输出

LL分析测试

输入（def.json）

```json
{
  "tags": [
    { "type": "终结符", "value": "+" },
    { "type": "终结符", "value": "-" },
    { "type": "终结符", "value": "*" },
    { "type": "终结符", "value": "/" },
    { "type": "终结符", "value": "(" },
    { "type": "终结符", "value": ")" },
    { "type": "终结符", "value": "num" },
    { "type": "终结符", "value": "ε" },
```

```
    { "type": "非终结符", "value": "E" },
    { "type": "非终结符", "value": "F" },
    { "type": "非终结符", "value": "T" }
  ],
  "productions": [
    "E → E+T | E-T | T",
    "T → T*F | T/F | F",
    "F → (E) | num"
  ],
  "input": "(num/num)",
  "method": "LL"
}
```

输出

```
-------- 打印当前生成式 --------
E -> E+T
E -> E-T
E -> T
T -> T*F
T -> T/F
T -> F
F -> (E)
F -> num
------------------------------

--------- 打印FIRST集 ---------
E: ( num
F: ( num
T: ( num
------------------------------

-------- 打印FOLLOW集 --------
E: $ + - )
F: $ + - ) * /
T: $ + - ) * /
------------------------------

-------- 打印当前生成式 --------
E -> TE'
F -> (E)
F -> num
T' -> *FT'
T' -> /FT'
T' -> ϵ
T -> FT'
E' -> +TE'
E' -> -TE'
E' -> ϵ
------------------------------

--------- 打印FIRST集 ---------
E: ( num
F: ( num
T: ( num
E': + - ϵ
T': * / ϵ
```

```
-----------------------------

-------- 打印FOLLOW集 --------
E: $ )
F: * / + - $ )
T: + - $ )
E': $ )
T': + - $ )
-----------------------------

------------------------------------- 打印LL分析表 -------------------------------
-------
+----+----------+----------+----------+----------+--------+-------+--------+-----
--+
|    |    +     |    -     |    *     |    /     |   (    |   )   |  num   |   $
  |
+----+----------+----------+----------+----------+--------+-------+--------+-----
--+
| F  |          |          |          |          | F->(E) |       | F->num |
  |
| T' |  T'->ϵ   |  T'->ϵ   | T'->*FT' | T'->/FT' |        | T'->ϵ |        | T'-
>ϵ |
| T  |          |          |          |          | T->FT' |       | T->FT' |
  |
| E' | E'->+TE' | E'->-TE' |          |          |        | E'->ϵ |        | E'-
>ϵ |
| E  |          |          |          |          | E->TE' |       | E->TE' |
  |
+----+----------+----------+----------+----------+--------+-------+--------+-----
--+
-----------------------------------------------------------------------------------
--------


LL分析成功！接受输入语句

-------------------------- 打印LL分析过程 --------------------------
+------+-------------+------------+----------+
| STEP |    STACK    |   INPUT    |  OUTPUT  |
+------+-------------+------------+----------+
| (1)  |     $E      | (num/num)$ |  E->TE'  |
| (2)  |    $E'T     | (num/num)$ |  T->FT'  |
| (3)  |   $E'T'F    | (num/num)$ |  F->(E)  |
| (5)  |   $E'T')E   | num/num)$  |  E->TE'  |
| (6)  |  $E'T')E'T  | num/num)$  |  T->FT'  |
| (7)  | $E'T')E'T'F | num/num)$  |  F->num  |
| (9)  | $E'T')E'T'  |   /num)$   | T'->/FT' |
| (11) | $E'T')E'T'F |   num)$    |  F->num  |
| (13) | $E'T')E'T'  |     )$     |  T'->ϵ   |
| (14) |  $E'T')E'   |     )$     |  E'->ϵ   |
+------+-------------+------------+----------+
-----------------------------------------------------------------
```

LR分析测试

输入（def.json）

```
{
  "tags": [
    { "type": "终结符", "value": "+" },
    { "type": "终结符", "value": "-" },
    { "type": "终结符", "value": "*" },
    { "type": "终结符", "value": "/" },
    { "type": "终结符", "value": "(" },
    { "type": "终结符", "value": ")" },
    { "type": "终结符", "value": "num" },
    { "type": "终结符", "value": "ε" },
    { "type": "非终结符", "value": "E" },
    { "type": "非终结符", "value": "F" },
    { "type": "非终结符", "value": "T" }
  ],
  "productions": [
    "E → E+T | E-T | T",
    "T → T*F | T/F | F",
    "F → (E) | num"
  ],
  "input": "(num/num)",
  "method": "LR"
}
```

输出

```
-------- 打印当前生成式 --------
E -> E+T
E -> E-T
E -> T
T -> T*F
T -> T/F
T -> F
F -> (E)
F -> num
------------------------------

--------- 打印FIRST集 ---------
E: ( num
F: ( num
T: ( num
------------------------------

-------- 打印FOLLOW集 --------
E: $ + - )
F: $ + - ) * /
T: $ + - ) * /
------------------------------

-------- 打印当前生成式 --------
E -> TE'
F -> (E)
F -> num
T' -> *FT'
```

```
T' -> /FT'
T' -> ϵ
T -> FT'
E' -> +TE'
E' -> -TE'
E' -> ϵ
------------------------------

--------- 打印FIRST集 ---------
E: ( num
F: ( num
T: ( num
E': + - ϵ
T': * / ϵ
------------------------------

-------- 打印FOLLOW集 --------
E: $ )
F: * / + - $ )
T: + - $ )
E': $ )
T': + - $ )
------------------------------

------------------------------------ 打印LL分析表 ------------------------------
-------
+----+----------+----------+----------+----------+--------+-------+--------+-----
--+
|    |    +     |    -     |    *     |    /     |   (    |   )   |  num   |   $
  |
+----+----------+----------+----------+----------+--------+-------+--------+-----
--+
| E' | E'->+TE' | E'->-TE' |          |          |        | E'->ϵ |        | E'-
>ϵ |
| E  |          |          |          |          | E->TE' |       | E->TE' |
  |
| F  |          |          |          |          | F->(E) |       | F->num |
  |
| T' |  T'->ϵ   |  T'->ϵ   | T'->*FT' | T'->/FT' |        | T'->ϵ |        | T'-
>ϵ |
| T  |          |          |          |          | T->FT' |       | T->FT' |
  |
+----+----------+----------+----------+----------+--------+-------+--------+-----
--+
------------------------------------------------------------------------------
--------


LL分析成功！接受输入语句

-------------------------- 打印LL分析过程 --------------------------
+------+-------------+------------+----------+
| STEP |    STACK    |   INPUT    |  OUTPUT  |
+------+-------------+------------+----------+
| (1)  |     $E      | (num/num)$ |  E->TE'  |
| (2)  |    $E'T     | (num/num)$ |  T->FT'  |
| (3)  |   $E'T'F    | (num/num)$ |  F->(E)  |
| (5)  |   $E'T')E   | num/num)$  |  E->TE'  |
```

```
| (6)  |  $E'T')E'T  | num/num)$  |  T->FT'  |
| (7)  | $E'T')E'T'F | num/num)$  |  F->num  |
| (9)  | $E'T')E'T'  |   /num)$   | T'->/FT' |
| (11) | $E'T')E'T'F |   num)$    |  F->num  |
| (13) | $E'T')E'T'  |     )$     |  T'->ϵ   |
| (14) |  $E'T')E'   |     )$     |  E'->ϵ   |
+------+-------------+------------+----------+
-------------------------------------------------------------------
```

分析说明

可执行程序正常运行，且运行结果符合预期