练习4.2.1：

使得文法的预测分析产生回溯的原因是什么？仅使用FIRST集合可以避免回溯吗？为什么？

原因：这是因为文法的不确定性，避免回溯要求：对文法的任何非终结符，当它要去匹配输入串时，能够根据它所面临的输入符号准确地指派它的一个候选去执行任务，并且此候选的工作结果应是确信无疑的。

仅使用FIRST集合是不能避免回溯的，首先它们的FIRST集合不能相交，且要求对于任意非终结符两个不同产生式 A-> α | β 的情况如果有ε∈FIRST(β)，还需要有FOLLOW(A) 和FIRST(α) 不相交（否则依然不知道该选择哪个产生式）。也即LL(1)文法的定义。

LL(1)文法需满足：FIRST(α) ∩ FIRST(β)=Ø；如果ε在FIRST(β)中，那么FIRST(α)和FOLLOW(A)是不相交的集合，并且当ε在FIRST(α)中，类似成立。

练习4.2.2：

考虑文法：

```
lexp → atom |list

atom → number | identifier

list → (lexp-seq)

lexp-seq → lexp-seq lexp | lexp
```

a) **消除左递归**

仅有最后一个产生式含有左递归

```
lexp→atom | list

atom→number | identifier

list→(lexp-seq)

lexp-seq→ lexp lexp-seq'

lexp-seq'→ lexp lexp-seq'| ε
```

b) **求得该文法的FIRST集合和FOLLOW集合**

假设number和identifier都是代指终结符。

FIRST(list)={(}

FIRST(atom)={number, identifier}

FIRST(lexp) = FISRT(atom)+FIRST(list)== {number, identifier, (}

FIRST(lexp-seq)=FIRST(lexp)={number, identifier, (}

FIRST(lexp-seq')={ε}+FIRST(lexp)={ε, number, identifier, (}

c) **说明所得的文法是LL(1)文法**

对于任何一个非终结符的两个产生式，其中的FIRST集均不相交。
如：
(1)lexp → atom | list: 有FIRST(atom)∩FIRST(list)=∅、且list 和atom皆不可导出空串；

(2)atom→number | identifier: 有FIRST(number)∩FIRST(identifier)=∅、number 和 identifier皆不可导出空串；

若存在空串，此时：
(3)lexp-seq'→ lexp lexp-seq'| ε: 有FIRST(lexp lexp-seq')∩FIRST(ε)=∅，仅有一个产生式能导出空串。此时，FIRST(lexp lexp-seq')∩FOLLOW(lexp-seq')=∅.

d) **为所得的文法构造LL(1)分析表**

非终结符号	**(**	**)**	**number**	**identifier**	**$**
lexp	lexp -> list		lexp -> atom	lexp -> atom	
atom			atom -> number	atom -> identifier	
list	list - >(lexp -> seq)				
lexp-seq	lexp-seq → lexp lexp-seq'		lexp-seq → lexp lexp-seq'	lexp-seq → lexp lexp-seq'	
lexp-seq'	lexp-seq' → lexp lexp-seq'	lexp-seq'→ ε	lexp-seq' → lexp lexp-seq'	lexp-seq' → lexp lexp-seq'	

e) 对输入串(a (b (2)) (c))给出相应得LL(1)分析程序的动作

栈	**输入**	**动作**
$lexp	(a (b (2)) (c))$	lexp->list
$list	(a (b (2)) (c))$	list->(lexp-seq)

栈	输入	动作
$)lexp-seq(	(a (b (2)) (c))$	match
$)lexp-seq	a (b (2)) (c))$	lexp-seq->lexp lexp-seq'
$)lexp-seq'lexp	a (b (2)) (c))$	lexp->atom
$)lexp-seq'atom	a (b (2)) (c))$	atom->identifer
$)lexp-seq'identifer	a (b (2)) (c))$	match
$)lexp-seq'	(b (2)) (c))$	lexp-seq'->lexp lexp-seq'
$)lexp-seq'lexp	(b (2)) (c))$	lexp->list
$)lexp-seq'list	(b (2)) (c))$	list->(lexp-seq)
$)lexp-seq')lexp-seq(	(b (2)) (c))$	match
$)lexp-seq')lexp-seq	b (2)) (c))$	lexp-seq->lexp lexp-seq'
$)lexp-seq')lexp-seq'lexp	b (2)) (c))$	lexp->atom
$)lexp-seq')lexp-seq'atom	b (2)) (c))$	atom->identifer
$)lexp-seq)lexp-seq'identifer	b (2)) (c))$	match
$)lexp-seq')lexp-seq'	(2)) (c))$	lexp-seq'->lexp lexp-seq'
$)lexp-seq')lexp-seq'lexp	(2)) (c))$	lexp->list
$)lexp-seq')lexp-seq'list	(2)) (c))$	list->(lexp-seq)
$)lexp-seq')lexp-seq')lexp-seq(	(2)) (c))$	match
$)lexp-seq')lexp-seq')lexp-seq	2)) (c))$	lexp-seq->lexplexp-seq'
$)lexp-seq')lexp-seq')lexp-seq'list	2)) (c))$	list->atom
$)lexp-seq')lexp-seq')lexp-seq'atom	2)) (c))$	atom->number
$)lexp-seq')lexp-seq')lexp-seq'number	2)) (c))$	match
$)lexp-seq')lexp-seq')lexp-seq'	)) (c))$	lexp-seq'->ε
$)lexp-seq')lexp-seq')	)) (c))$	match
$)lexp-seq')lexp-seq'	) (c))$	lexp-seq'->ε
$)lexp-seq')	) (c))$	match
$)lexp-seq'	(c))$	lexp-seq'->lexp lexp-seq'
$)lexp-seq'lexp	(c))$	lexp->list
$)lexp-seq'list	(c))$	list->(lexp-seq)
$)lexp-seq')lexp-seq(	(c))$	match
$)lexp-seq')lexp-seq	c))$	lexp-seq->lexp lexp-seq'

栈	输入	动作
$)lexp-seq')lexp-seq'lexp	c))$	lexp->atom
$)lexp-seq')lexp-seq'atom	c))$	atom->identifer
$)lexp-seq')lexp-seq'identifer	c))$	match
$)lexp-seq')lexp-seq'	))$	lexp-seq'->ε
$)lexp-seq')	))$	match
$)lexp-seq'	)$	lexp-seq'->ε
$)	)$	match
$	$	accept