（若发现问题，请及时告知）

A1　根据你的判断，针对以下文法是否可以设计一个自顶向下预测分析过程？如果可以，需要向前察看多少个输入符号？

（1）文法 $G_1[S]$：

$$S \to A \mid B$$
$$A \to aAb \mid c$$
$$B \to aBbb \mid d$$

（2）文法 $G_2[S]$：

$$S \to \varepsilon \mid abA$$
$$A \to Saa \mid b$$

参考解答：

（1）不可以。

（2）可以，需要向前察看 2 个输入符号。

A2　给定某类表达式文法 $G[E]$：

$$E \to +\,E\,R \mid -\,E\,R \mid \underline{positive}\,R$$
$$R \to *\,E\,R \mid \varepsilon$$

其中，+ 和 – 分别代表一元正和一元负运算，* 代表普通的二元乘法运算，<u>positive</u> 为代表正整数（非0）的单词。

(1) 针对文法 $G[E]$，下表给出各产生式右部文法符号串的 *First* 集合，各产生式左部非终结符的 *Follow* 集合，以及各产生式的预测集合*PS*。试填充其中空白表项（共3处）的内容：

$G[E]$的规则 r	*First* (*rhs*(*r*))	*Follow* (*lhs*(*r*))	*PS* (*r*)
$E \to +\,E\,R$	+		+
$E \to -\,E\,R$	-	此处不填	-
$E \to \underline{positive}\,R$	<u>positive</u>	此处不填	<u>positive</u>
$R \to *\,E\,R$	*		*
$R \to \varepsilon$	ε	此处不填	

表中，*rhs(r)* 为产生式 *r* 右部的文法符号串，*lhs(r)* 为产生式 *r* 左部的非终结符。

(2) G[E] 不是 LL(1) 文法，试解释为什么?

(3) 虽然G[E] 不是 LL(1) 文法，但可以采用一种强制措施，使得常规的 LL(1) 分析算法仍然可用。针对含5个单词的输入串 + - 20 * 18 ，以下基于这一措施以及上述各产生式的预测集合（或预测分析表）的一个表驱动 LL(1) 分析过程：

步骤	下推栈	余留符号串	下一步动作
1	# E	+ - *20* * <u>*18*</u> #	应用产生式 $E \rightarrow + E\ R$
2	# $R\ E$ +	+ - <u>*20*</u> * <u>*18*</u> #	匹配栈顶和当前输入符号
3	# $R\ E$	- <u>*20*</u> * <u>*18*</u> #	应用产生式 $E \rightarrow - E\ R$
4	# $R\ R\ E$ -	- <u>*20*</u> * <u>*18*</u> #	匹配栈顶和当前输入符号
5	# $R\ R\ E$	<u>*20*</u> * <u>*18*</u> #	应用产生式 $E \rightarrow$ <u>*positive*</u> R
6	# $R\ R\ R$ <u>*positive*</u>	<u>*20*</u> * <u>*18*</u> #	匹配栈顶和当前输入符号
7	# $R\ R\ R$	* <u>*18*</u> #	
8			
9			
10			
11			
12	# $R\ R\ R$	#	应用产生式 $R \rightarrow \varepsilon$
13	# $R\ R$	#	应用产生式 $R \rightarrow \varepsilon$
14	# R	#	应用产生式 $R \rightarrow \varepsilon$
15	#	#	结束

试填写上述分析过程中第7步时使用的产生式，以及第 8~11 步的分析过程，共计13处空白；并指出采用了什么样的强制措施。

参考解答：

(1)

G[*E*]的规则 *r*	*First* (*rhs*(*r*))	*Follow* (*lhs*(*r*))	*PS* (*r*)
$E \rightarrow + E\ R$	+	# *	+
$E \rightarrow - E\ R$	-	# *	-
$E \rightarrow$ <u>*positive*</u> R	<u>*positive*</u>	# *	<u>*positive*</u>
$R \rightarrow * E\ R$	*	# *	*
$R \rightarrow \varepsilon$	ε	# *	# *

表中的 $rhs(r)$ 表示产生式 r 右部的文法符号串，$lhs(r)$ 表示产生式 r 左部的非终结符。

(2)

因为 $PS\,(R \to *\,E\,R)$ 与 $PS\,(R \to \varepsilon)$ 相交不为空。

(3)

步骤	下推栈	余留符号串	下一步动作
1	$\#\,E$	$+\,-\,20\,*\,\underline{18}\,\#$	应用产生式 $E \to +\,E\,R$
2	$\#\,R\,E\,+$	$+\,-\,\underline{20}\,*\,\underline{18}\,\#$	匹配栈顶和当前输入符号
3	$\#\,R\,E$	$-\,\underline{20}\,*\,\underline{18}\,\#$	应用产生式 $E \to -\,E\,R$
4	$\#\,R\,R\,E\,-$	$-\,\underline{20}\,*\,\underline{18}\,\#$	匹配栈顶和当前输入符号
5	$\#\,R\,R\,E$	$\underline{20}\,*\,\underline{18}\,\#$	应用产生式 $E \to \underline{positive}\,R$
6	$\#\,R\,R\,R\,\underline{positive}$	$\underline{20}\,*\,\underline{18}\,\#$	匹配栈顶和当前输入符号
7	$\#\,R\,R\,R$	$*\,\underline{18}\,\#$	应用产生式 $R \to *\,E\,R$
8	$\#\,R\,R\,R\,E\,*$	$*\,\underline{18}\,\#$	匹配栈顶和当前输入符号
9	$\#\,R\,R\,R\,E$	$\underline{18}\,\#$	应用产生式 $E \to \underline{positive}\,R$
10	$\#\,R\,R\,R\,R\,\underline{positive}$	$\underline{18}\,\#$	匹配栈顶和当前输入符号
11	$\#\,R\,R\,R\,R$	$\#$	应用产生式 $R \to \varepsilon$
12	$\#\,R\,R\,R$	$\#$	应用产生式 $R \to \varepsilon$
13	$\#\,R\,R$	$\#$	应用产生式 $R \to \varepsilon$
14	$\#\,R$	$\#$	应用产生式 $R \to \varepsilon$
15	$\#$	$\#$	结束

采用的强制措施：如果在下一输入符号为 *（未到结束符 #）时，不使用 $E \to \varepsilon$，而是使用 $R \to *\,E\,R$。

A3　给定命题表达式文法 $G\,[S]$:

$$S \to P$$
$$P \to \wedge\,P\,P \mid \vee\,P\,P \mid \neg\,P \mid \underline{id}$$

其中，$\wedge$、$\vee$、$\neg$ 分别代表命题逻辑与、或、非等运算符单词，$\underline{id}$ 代表标识符单词。

容易得出: $G[S]$ 是 $LL(1)$ 文法。基于$G[S]$的预测分析表和一个分析栈，课程中介绍了一种表驱动的 $LL(1)$ 分析过程。假设有输入符号串：$\vee \vee a \wedge b\, c \vee \neg a \wedge c\, b\, \#$。试问，在分析过程中，分析栈中最多会出现几个$S$？几个$P$？若因误操作使输入串多了一个符号，变为 $\vee \vee a \wedge b\, c\, c \vee \neg a \wedge c\, b$，当分析过程中发生错误时，关于报错信息，你认为最不可能的选择是（4选1）：（1）缺运算数；（2）多运算数；（3）缺运算符；（4）多运算符。如果想要从该出错位置恢复分析，可以进行什么操作？

参考解答：

在分析过程中，分析栈中最多会出现1个S, 3个P。

（1）缺运算数。提示：读入第二个 c 时出错，可以直接报“多运算数”（多第二个c），或者“缺运算符”（若不遇到第二个c，而是遇到一个运算符，则此时不会出错）；另外，若此时输入已结束，则不会出错，但下一个输入符号是运算符（$\vee$），所以也可以报“多运算符”。

如果想要从该出错位置恢复分析，可以删掉当前输入符号（c），并将出错时使用的产生式左边的非终结符（P）退回到分析栈顶，然后就可以恢复分析了。可能还有其他恢复手段，只要合理都是可以的。

A4 分析表达式文法（Parsing Expression Grammar, PEG）是一个四元组(N, Σ, P, S)，其中

- N 为非终结符（nonterminal）集合；
- Σ 为终结符（terminal）集合；
- P 为产生式（parsing rules）集合；
- S 为开始符号（start symbol）。

各产生式形如 $A \rightarrow e$，其中 A 为非终结符，e 为**分析表达式**（parsing expression）。每个分析表达式都会用来匹配输入串的一个前缀。

本题考虑一种简单的 PEG，其分析表达式定义如下：

$$e ::= t \mid n \mid e_1\, e_2 \mid e_1 > e_2 \mid e^* \mid (e)$$

其中 $t \in \Sigma$ 只能匹配终结符 t 自身，$n \in N$ 只能匹配非终结符 n 自身。其他复合的表达式及分析过程为（分析失败则表明输入串不能被该文法识别）：

- 序列“$e_1\, e_2$”表示：先用输入串匹配 e_1，若成功再用剩下的串匹配 e_2；
- 有序选择“$e_1 > e_2$”表示：先尝试用输入串匹配 e_1，若成功则分析结束（忽略 e_2）；若失败，重新用输入串匹配 e_2；
- 星闭包“e^*”表示：用输入串匹配 e 零次或多次，直至匹配失败（即消耗掉输入串中尽可能多的符号）；
- 特别地，括号用于显式指定优先级。

可以发现，PEG 由于其分析过程允许回溯，因而无需像传统的 LL 分析那样要向后查看符号。

根据以上设定，回答下列问题。

（1）什么样的输入串可以匹配（指完全匹配，即匹配完以后没有余留的串）分析表达式“a^*

$a\ b$”，其中 a 和 b 为终结符？

（2）类似于“e*”，我们可以扩展出“e+”——它表示用输入串匹配 e 一次或多次，直至匹配失败。请利用以上给出的这些分析表达式，写出“e+”的定义。

（3）利用 PEG 可以消除悬挂 else 的二义性。请简要说明如何消除？

（4）考查下列 CFG（其中 x 为终结符）：

$$S \to x\ S\ x \mid x$$

它对应的语言记作 L。问：

a. 是否可以直接对上述文法采用递归下降方法来分析？请说明理由。

b. 是否存在某个能做递归下降分析的 PEG，其对应的语言也为 L？ 若存在，请给出此 PEG；否则，请说明理由。

参考解答：

（1）没有任何输入串可以匹配“$a*\ a\ b$”，因为“a*”将消耗掉该输入串任何连续的 a 构成的前缀，导致下一个“a”无法被匹配。

（2） $e+ = e\ e*$

（3）通过有序选择，大致文法如下：

$$stmt \to \ldots > \underline{\text{if}}\ expr\ stmt\ \underline{\text{else}}\ stmt > \underline{\text{if}}\ expr\ stmt$$

（4）

a. 不能。语言 L={$x^n x x^n$}为以 x 为中心的全 x 串，而在分析前我们无法找到中心点的 x 位于何处，因此我们无法决定何时采用 $S \to x\ S\ x$，何时采用 $S \to x$。

b. 该语言 L={x^{2n+1}}，即奇数个 x 组成的串。因此可以设计如下接受 L 的 PEG：

$$S \to x\ (x\ x)*$$

以下是 Lecture03 文档中的题目

---

5 计算下列文法中每个非终结符的 First 集和 Follow 集，以及每个产生式的预测集合，并判断该文法是否 LL(1)文法（说明原因）：

（2）文法 $G_2[S]$:

$$S \to TP$$
$$T \to +PT \mid \varepsilon$$
$$P \to (S) \mid \text{a}$$

（3）文法 $G_3[S]$:

$$S \to aSa \mid bSb \mid \varepsilon$$

参考解答：

（2）计算非终结符的 FIRST 集和 FOLLOW 集，结果如下：

FIRST（S）= { +，(，a }　　FOLLOW（S）= { #，) }

FIRST（T）= { +，ε }　　FOLLOW（T）= { (，a }

FIRST（P）= { (，a }　　FOLLOW（P）= { #，+，(，a ，)}

PS($S \to TP$)={ +，(，a }

PS($T \to +PT$)={ + }

PS($T \to \varepsilon$)={ (，a，+}

PS($P \to (S)$)={ (}

PS($P \to \mathrm{a}$)={ a }

因为，PS($T \to +PT$)$\cap$PS($T \to \varepsilon$)= { + }$\cap${ (，a}= Φ，所以，G(S)是 LL(1)文法。

（3）计算非终结符的 FIRST 集和 FOLLOW 集，结果如下：

FIRST（S）= { a，b，ε}　　FOLLOW（S）= { #，a，b }

PS($S \to aSa$)={ a }

PS($S \to bSb$)={ b }

PS($S \to \varepsilon$)={ #，a，b }

因为，PS($S \to aSa$)$\cap$PS($S \to bSb$)$\cap$PS($TS \to \varepsilon$)$\neq$Φ，所以，G(S)不是 LL(1)文法。

6 验证如下文法是 LL(1)文法，并基于该文法构造递归下降分析程序：

（2）文法 $G'[E]$：

$$
\begin{aligned}
&E \to [\ F\ ]\ E' \\
&E' \to E \mid \varepsilon \\
&F \to aF' \\
&F' \to aF' \mid \varepsilon
\end{aligned}
$$

参考解答：

（2）观察文法规则可知，可能产生规则选择冲突的规则只能是 $E' \to E \mid \varepsilon$ 和 $F' \to aF' \mid \varepsilon$ 。我们只需要求出这四条规则的 PS 集合（预测集合）即可。欲求这四个 PS 集合，我们需要先求出：

First (E) = { [}　　Follow (E′) = {#}

First (aF′) = { a }　　Follow (F′) = {] }

从而

PS (E’ → E) = First (E) = { [}

PS (E’ → ε) = Follow (E′) = {#}

PS (F’ → aF’) = First (aF′) = { a }

PS（F’ → ε）= Follow（F′）= {] }

因为

对于 E’ → E | ε 有： *PS*（E’ → E）∩ *PS*（E’ → ε）=Φ

对于 F’ → aF’ | ε 有： *PS*（F’ → aF’）∩ *PS*（F’ → ε）=Φ

所以，文法 G[E] 是 LL（1）文法。

用类似 C 语言写出 G[E]的递归子程序，其中 getToken()为取下一单词过程，变量 lookahead 为全局变量，存放当前单词。

```
void ParseE( ) {
    Match_Token ( [ );
    ParseF( );
    MatchToken ( ] );
    ParseE’( );
}

void ParseE’( ) {
    switch (lookahead)  {
        case [:
            ParseE( );
            break;
        case #:
            break;
        default:
            printf(″syntax error \n″ );
            exit(0);
    }
}

void ParseF( ) {
    MatchToken ( a );
    ParseF’  ( );
}

void ParseF’( ) {
    switch (lookahead)  {
        case a:
            MatchToken ( a );
            ParseF’  ( );
            break;
        case ]:
            break;
        case:
            printf(″syntax error \n″ );
            exit(0);
    }
}

void MatchToken(int expected) {//判别当前单词是否与期望的终结符匹配
    if (lookahead != expected){
```

```
            printf("syntax error \n" );
            exit(0);
    else      // 若匹配,消费掉当前单词并调用词法分析器读入下一个单词
        lookahead = getToken();
}
```

7　给出习题 5 中所有文法的预测分析表，并根据分析表指出相应文法是否 LL(1)的，同时验证习题 5 的结果。

参考解答：

（2）文法 $G_2[S]$：

$S \to TP$

$T \to +PT \mid \varepsilon$

$P \to (S) \mid \mathrm{a}$

$\mathrm{PS}(S \to TP)$={ +，(，a }

$\mathrm{PS}(T \to +PT)$={ + }

$\mathrm{PS}(T \to \varepsilon)$={ (，a}

$\mathrm{PS}(P \to (S))$={ (}

$\mathrm{PS}(P \to \mathrm{a})$={ a }

	(	)	a	+	#
S	$S \to TP$		$S \to TP$	$S \to TP$	
T	$T \to \varepsilon$		$T \to \varepsilon$	$T \to +PT$	
P	$P \to (S)$		$P \to \mathrm{a}$		

每一个表项唯一确定，所以，是 LL(1)文法。

（3）文法 $G_3[S]$：

$S \to aSa \mid bSb \mid \varepsilon$

$\mathrm{PS}(S \to aSa)$={ a }

$\mathrm{PS}(S \to bSb)$={ b }

$\mathrm{PS}(S \to \varepsilon)$={ #，a，b }

	a	b	#
S	$S \to aSa$ $S \to \varepsilon$	$S \to bSb$ $S \to \varepsilon$	$S \to \varepsilon$

M[S，a]={ $S\rightarrow aSa$，$S\rightarrow\varepsilon$}，M[S，b]={ $S\rightarrow bSb$，$S\rightarrow\varepsilon$}，所以，不是 LL(1)文法。

11 按照本讲介绍的消除一般左递归算法消除下面文法 $G[S]$中的左递归（要求依非终结符的排序 S、Q、P 执行该算法）：

$$
\begin{aligned}
S &\rightarrow PQ \mid a \\
P &\rightarrow QS \mid b \\
Q &\rightarrow SP \mid c
\end{aligned}
$$

参考解答：

按照非终结符的特定顺序排列各规则：

S →PQ ｜ a
Q →SP ｜ c
P →QS ｜ b

第一步，得：

S →PQ ｜ a
Q →PQP ｜ aP ｜ c
P →QS ｜ b

第二步，得：

S →PQ ｜ a
Q →PQP ｜ aP ｜ c
P →PQPS ｜ aPS ｜ cS ｜ b

消去 P →PQPS 的左递归得：

S →PQ ｜ a
Q →PQP ｜ aP ｜ c
P →aPSR ｜ cSR ｜ bR
R →QPSR ｜ ε

经检查，此时得到的文法已经不含左递归，可结束消除左递归过程。