第一次书面作业（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　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）：

（1）文法 $G_1[S]$：

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

（2）文法 $G_2[S]$：

$S \to A1B$
$A \to 0A \mid \varepsilon$
$B \to 0B \mid 1B \mid \varepsilon$

参考解答：

（1）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($P \to a$)={ a }

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

（2）计算FOLLOW 集，以及每个产生式的预测集合，结果如下：

G 中的规则 r	*First* (*rhs*(*r*))	*Follow* (*lhs*(*r*))	*PS* (*r*)
$S \to A1B$	0, 1	#	0, 1
$A \to 0A$	0	1	0
$A \to \varepsilon$	ε	此处不填	1
$B \to 0B$	0	#	0
$B \to 1B$	1	此处不填	1
$B \to \varepsilon$	ε	此处不填	#

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。

非终结符的 FIRST 集可从上表中第2列对应符号的并集得到，这里省略。

因为 $PS(A \to 0A) \cap PS(A \to \varepsilon) = \phi$ 及 $PS(B \to 0B)$ 、$PS(B \to 1B)$ 和 $PS(B \to \varepsilon)$ 两两互不相交，所以文法 *G*[*S*] 是 *LL*(1) 文法。

A3　验证如下文法 *G*[*S*]是 LL(1)文法，并基于该文法构造递归下降分析程序（伪代码）：

$S \to AB$
$A \to aA$
$A \to \varepsilon$
$B \to bB$
$B \to \varepsilon$

参考解答:

计算非终结符的 FIRST 集和 FOLLOW 集，结果如下：

FIRST（S）= {a, b}　　FOLLOW（S）= {# }

FIRST（A）= {a, ε }　　FOLLOW（A）= {b, # }

FIRST（B）= {b, ε }　　FOLLOW（B）= {# }

因为

FIRST（aA）∩ FOLLOW（A）= {a }∩{b,# } = Φ

FIRST（*bB*）∩ FOLLOW（B）= {b }∩{ # } = Φ

或

PS（$A \rightarrow aA$）∩ PS（$A \rightarrow \varepsilon$）= {a }∩{b,# } = Φ

PS（$B \rightarrow bB$）∩ PS（$B \rightarrow \varepsilon$）= {b }∩{ # } = Φ

所以，G(S)是LL(1)文法。

用类似课程中使用的伪代码写出 G(S)的递归子程序，其中变量 lookahead 存放当前单词。不需要考虑太多编程语言相关的细节。参考程序如下：

```
void ParseS( )                    // 主函数
{
    ParseA( );
    ParseB( );
}
void ParseA( )
{
    switch (lookahead)            // lookahead为下一个输入符号
    {
         case ′ a′ :
             MatchToken( ‘a’ );
             ParseA();
             break;
         case ′ b′ , ′ #′ :
             break;
         default:
             printf("syntax error \n")
             exit(0);
    }
    return A_num;
}
void ParseB( )
{
    switch (lookahead)  {
         case ′ b′ :
             MatchToken( ‘b’ );
             ParseB( );
             break;
         case ′ #′ :
             break;
         default:
```

```
                printf("syntax error \n");
                exit(0);
        }
    }
    void  Match_Token(int expected)
    {
        if (lookahead != expected)
        {
                printf("syntax error \n")
                exit(0);
        else
             lookahead = getToken();
    }
```

A4 给定某类表达式文法 *G*[*E*]：

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

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，*positive* 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*)
$E \rightarrow + E\ R$	+		+
$E \rightarrow - E\ R$	-	此处不填	-
$E \rightarrow \underline{positive}\ R$	*positive*	此处不填	*positive*
$R \rightarrow * E\ R$	*		*
$R \rightarrow \varepsilon$	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* +	+ - *20* * *18* #	匹配栈顶和当前输入符号
3	# *R E*	- *20* * *18* #	应用产生式 $E \rightarrow$ - *E R*
4	# *R R E* -	- *20* * *18* #	匹配栈顶和当前输入符号
5	# *R R E*	*20* * *18* #	应用产生式 $E \rightarrow$ *positive R*
6	# *R R R positive*	*20* * *18* #	匹配栈顶和当前输入符号
7	# *R R R*	* *18* #	
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*	-	# *	-
$E \rightarrow$ *positive R*	*positive*	# *	*positive*
$R \rightarrow$ * *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* ($R \rightarrow$ * *E R*) 与 *PS* ($R \rightarrow \varepsilon$) 相交不为空。

(3)

步骤	下推栈	余留符号串	下一步动作
1	# *E*	+ - *20* * *18* #	应用产生式 $E \to + E R$
2	# *R E* +	+ - *20* * *18* #	匹配栈顶和当前输入符号
3	# *R E*	- *20* * *18* #	应用产生式 $E \to - E R$
4	# *R R E* -	- *20* * *18* #	匹配栈顶和当前输入符号
5	# *R R E*	*20* * *18* #	应用产生式 $E \to$ *positive* R
6	# *R R R positive*	*20* * *18* #	匹配栈顶和当前输入符号
7	# *R R R*	* *18* #	应用产生式 $R \to * E R$
8	# *R R R E* *	* *18* #	匹配栈顶和当前输入符号
9	# *R R R E*	*18* #	应用产生式 $E \to$ *positive* R
10	# *R R R R positive*	*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 $R \to * E R$。

A5 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，*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（∨），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。

A6　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）：

$$S \to PQ \mid a$$
$$P \to QS \mid b$$
$$Q \to SP \mid c$$

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。