Question:

Consider the following CFG, in which terminals are in bold-face:

$$\begin{aligned}
stmt_seq &\rightarrow stmt_seq \;;\; stmt \mid stmt \\
stmt &\rightarrow if_stmt \mid assign_stmt \\
if_stmt &\rightarrow \textbf{if}\ exp\ \textbf{then}\ stmt_seq\ \textbf{end} \mid \\
&\quad \textbf{if}\ exp\ \textbf{then}\ stmt_seq\ \textbf{else}\ stmt_seq\ \textbf{end} \\
assign_stmt &\rightarrow \textbf{id} = exp \\
exp &\rightarrow \textbf{0} \mid \textbf{1} \mid \textbf{int} \mid \textbf{id}
\end{aligned}$$

1. Calculate First & Follow sets for every non-terminal. Then deduce without making an LL(1) table, is the above CFG suitable for LL(1) parsing? Make an LL(1) table.
2. Modify the above CFG suitable for predictive parsing. Repeat the above with the modified CGF.

Solution:

Re-writing the grammar:

$$\begin{aligned}
stmt_seq &\rightarrow stmt_seq \;;\; stmt && 1 \\
stmt_seq &\rightarrow stmt && 2 \\
stmt &\rightarrow if_stmt && 3 \\
stmt &\rightarrow assign_stmt && 4 \\
if_stmt &\rightarrow \textbf{if}\ exp\ \textbf{then}\ stmt_seq\ \textbf{end} && 5 \\
if_stmt &\rightarrow \textbf{if}\ exp\ \textbf{then}\ stmt_seq\ \textbf{else}\ stmt_seq\ \textbf{end} && 6 \\
assign_stmt &\rightarrow \textbf{id} = exp && 7 \\
exp &\rightarrow \textbf{0} && 8 \\
exp &\rightarrow \textbf{1} && 9 \\
exp &\rightarrow \textbf{int} && 10 \\
exp &\rightarrow \textbf{id} && 11
\end{aligned}$$

FIRST			
Grammar-rule	**Pass 1**	**Pass 2**	**Pass 3**
stmt_seq → stmt_seq **;** stmt			
stmt_seq → stmt			{ if, id }
stmt → if_stmt		{ if }	

stmt → assign_stmt		{ if, id }	
if_stmt → **if** exp **then** stmt_seq **end**	{ if }		
if_stmt → **if** exp **then** stmt_seq **else** stmt_seq **end**	{ if }		
assign_stmt → **id** = exp	{ id }		
exp → **0**	{ 0 }		
exp → **1**	{ 0, 1 }		
exp → **int**	{ 0, 1, int }		
exp → **id**	{ 0, 1, int, id }		

FOLLOW		
Grammar-rule	**Pass 1**	**Pass 2**
stmt_seq → stmt_seq **;** stmt	stmt_seq = { $, ;} stmt = { $, ; }	stmt = { $, ;, end, else }
stmt_seq → stmt		
stmt → if_stmt	if_stmt = { $, ; }	if_stmt = { $, ;, end, else }
stmt → assign_stmt	assign_stmt = { $, ; }	assign_stmt = { $, ;, end, else }
if_stmt → **if** exp **then** stmt_seq **end**	exp = { then } stmt_seq = { $, ;, end}	
if_stmt → **if** exp **then** stmt_seq **else** stmt_seq **end**	exp = { then } stmt_seq = { $, ;, end, else}	
assign_stmt → **id** = exp	exp = { then, $, ; }	exp = { then, $, ;, end, else }

If we look at the production

$$\begin{aligned} if\_stmt \rightarrow & \textbf{ if } exp \textbf{ then } stmt\_seq \textbf{ end } \mid \\ & \textbf{ if } exp \textbf{ then } stmt\_seq \textbf{ else } stmt\_seq \textbf{ end} \end{aligned}$$

It can be viewed as

$$A \rightarrow \alpha_1 \mid \alpha_2$$

where **First**(α_1) = **{ if }** and **First**(α_2) = **{ if }** which violates the rule that
A grammar is **LL(1)** if for every production A $\rightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$, First$(\alpha_i) \cap$ First(α_j) is empty for all i and j, $1 \leq$ i, j $\leq$ n, i $\neq$ j.

PARSE TABLE

M[A, a]	if	id	0	1	int	else	end	;	$	then	=
stmt_seq	1 2	1 2									
stmt	3	4									
if_stmt	5 6										
assign_stmt		7									
exp		11	8	9	10						

Modifying Grammar:

Left Recursion Removal:

$$stmt\_seq \rightarrow stmt\_seq \; ; \; stmt \mid stmt \qquad (a)$$

In this case grammar-rule is of form

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m$$

then left recursion is removed by modifying grammar-rule to

$$\begin{aligned} A &\rightarrow \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_m A' \\ A' &\rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_n A' \mid \epsilon \end{aligned}$$

so the grammar-rule (a) is modified to

$$\begin{aligned} stmt\_seq &\rightarrow stmt \; stmt\_seq' \\ stmt\_seq' &\rightarrow \; ; \; stmt \; stmt\_seq' \mid \epsilon \end{aligned}$$

Left Factoring:

$$\begin{aligned} if\_stmt \rightarrow\ & \textbf{if } exp \textbf{ then } stmt\_seq \textbf{ end} \mid \\ & \textbf{if } exp \textbf{ then } stmt\_seq \textbf{ else } stmt\_seq \textbf{ end} \end{aligned} \qquad (b)$$

In this case grammar-rule is of form

$$A \rightarrow \alpha\beta \mid \alpha\gamma$$

then left factoring is handled by modifying the grammar-rule to

$$A \rightarrow \alpha A'$$
$$A' \rightarrow \beta \mid \gamma$$

so the grammar-rule (b) is modified to

$$if\_stmt \rightarrow \textbf{if } exp \textbf{ then } stmt\_seq\ if\_stmt'$$
$$if\_stmt' \rightarrow \textbf{end} \mid \textbf{else } stmt\_seq \textbf{ end}$$

Re-writing the modified grammar:

$$\begin{aligned}
stmt_seq &\rightarrow stmt\ stmt_seq' && 1\\
stmt_seq' &\rightarrow ;\ stmt\ stmt_seq' && 2\\
stmt_seq' &\rightarrow \epsilon && 3\\
stmt &\rightarrow if_stmt && 4\\
stmt &\rightarrow assign_stmt && 5\\
if_stmt &\rightarrow \textbf{if } exp \textbf{ then } stmt_seq\ if_stmt' && 6\\
if_stmt' &\rightarrow \textbf{end} && 7\\
if_stmt' &\rightarrow \textbf{else } stmt_seq \textbf{ end} && 8\\
assign_stmt &\rightarrow \textbf{id} = exp && 9\\
exp &\rightarrow \textbf{0} && 10\\
exp &\rightarrow \textbf{1} && 11\\
exp &\rightarrow \textbf{int} && 12\\
exp &\rightarrow \textbf{id} && 13
\end{aligned}$$

FIRST			
Grammar-rule	**Pass 1**	**Pass 2**	**Pass 3**
stmt_seq → stmt stmt_seq'			{ if, id }
stmt_seq' → **;** stmt stmt_seq'	{ ; }		
stmt_seq' → ϵ	{ ;, ϵ }		
stmt → if_stmt		{ if }	
stmt → assign_stmt		{ if, id }	
if_stmt → **if** exp **then** stmt_seq if_stmt'	{ if }		
if_stmt' → **end**	{ end }		
if_stmt' → **else** stmt_seq **end**	{ end, else }		
assign_stmt → **id** = exp	{ id }		
exp → **0**	{ 0 }		
exp → **1**	{ 0, 1 }		
exp → **int**	{ 0, 1, int }		
exp → **id**	{ 0, 1, int, id }		

FOLLOW		
Grammar-rule	**Pass 1**	**Pass 2**
stmt_seq → stmt stmt_seq'	stmt_seq = { $ } stmt = { ;, $ } stmt_seq' = { $ }	stmt = { ;, $, end, else } stmt_seq' = { $, end, else }
stmt_seq' → **;** stmt stmt_seq'		
stmt → if_stmt	if_stmt = { ;, $}	if_stmt = { ;, $ end, else }
stmt → assign_stmt	assign_stmt = { ;, $ }	assign_stmt = { ;, $ end, else }
if_stmt → **if** exp **then** stmt_seq if_stmt'	exp = { then} stmt_seq = { $, end, else } if_stmt' = { ;, $ }	if_stmt' = { ;, $, end, else }

if_stmt’ → **else** stmt_seq **end**		
assign_stmt → **id** = exp	exp = { then, ;, $ }	exp = { then, ;, $ end, else }

PARSE TABLE

M[A, a]	**if**	**id**	**0**	**1**	**int**	**else**	**end**	**;**	**$**	**then**	**=**
stmt_seq	1	1									
stmt_seq’						3	3	2	3		
stmt	4	5									
if_stmt	6										
if_stmt’						8	7				
assign_stmt		9									
exp		13	10	11	12						