Consider the following Grammar

$E \rightarrow TE'$

$E' \rightarrow +TE'|\epsilon$

$T \rightarrow FT$

$T' \rightarrow *FT'|\epsilon$

$F \rightarrow (E)|id$

Construct the Predictive parsing table.

FIRST(E) = FIRST(T) = FIRST(F) = {(, id}
FIRST(E') ={ +, ϵ}
FIRST(T') ={*,ϵ}
FOLLOW(E) = FOLLOW(E') ={),$}
FOLLOW(T)=FOLLOW(T')={+,),$ }
FOLLOW(F) = {*, +,), $}

Table: Predictive Parsing Table

Non Terminal	*id*	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow TE'$			$E' \rightarrow \epsilon$	$]E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T' \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Nonrecursive Predictive Parsing

1. If $X = a = \$$, the parser halts and announces successful completion of parsing.
2. If $X = a \neq \$$ the parser pops off the stack and advances the pointer to the next input symbol.
3. If X is a non terminal, the program consults $M[X, a]$ of parsing table M. The entry will be either an X-production of the grammar or an error entry.
   For example, If M[X,a] $= \{X \rightarrow UVW\}$, the parser replaces X on top of the stack by WVU (with U on top).

Table: Parsing: id+id*id

Stack	Input	Output
$E	id+id*id$	
$E'T	id+id*id$	$E \rightarrow TE'$
$E'T'F	id+id*id$	$T \rightarrow FT'$
$E'T'id	id + id * id $	$F \rightarrow id$
$E'T'	+id*id $	
$E'	+id*id$	$T' \rightarrow \epsilon$
⋮	⋮	⋮
$	$	Accept

Implement a predictive parser using C program.

Consider the following Grammar

$E \rightarrow E + T | T$

$T \rightarrow TF | F$

$F \rightarrow F * | a | b$

a) Eliminate left recursion from the above grammar.

b) Compute First & Follow.

c) Construct a parsing table.

d) Check the grammar is LL(1) or not.

e) Show the parsing for a+a+a.

Construct SLR parsing table for the following grammar

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow id$

Table: SLR Parsing Table For the given grammar

State	Action						Goto		
	id	+	*	(	)	$	*E*	*T*	*F*
0	S_5			S_4			1	2	3
1		S_6				Accept			
2		r_2	S_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	S_5			S_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	S_5		S_4					9	3
7	S_5		S_4						10
8		S_6			S_{11}				
9		r_1	S_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

LR Parsing Steps

1. Shift S, where s is a state.
2. Reduce by a grammar production $A \rightarrow B$
3. Accept.
4. Error

Moves of LR parser on $id * id + id$

Table: Moves of LR parser on $id * id + id$

	Stack	Input	Action
(1)	0	$id * id + id\$$	Shift

Moves of LR parser on $id * id + id$

Table: Moves of LR parser on $id * id + id$

	Stack	Input	Action
(1)	0	$id * id + id\$$	Shift
(2)	0 id 5	$*id + id\$$	reduce by $F \rightarrow id$

Moves of LR parser on $id * id + id$

Table: Moves of LR parser on $id * id + id$

	Stack	Input	Action
(1)	0	$id * id + id\$$	Shift
(2)	0 id 5	$*id + id\$$	reduce by $F \rightarrow id$
(3)	0 F 3	$*id + id\$$	reduce by $T \rightarrow F$

Moves of LR parser on $id * id + id$

Table: Moves of LR parser on $id * id + id$

	Stack	Input	Action
(1)	0	$id * id + id\$$	Shift
(2)	0 id 5	$*id + id\$$	reduce by $F \rightarrow id$
(3)	0 F 3	$*id + id\$$	reduce by $T \rightarrow F$
(4)	0 T 2	$*id + id\$$	Shift

Moves of LR parser on $id * id + id$

Table: Moves of LR parser on $id * id + id$

	Stack	Input	Action
(1)	0	$id * id + id\$$	Shift
(2)	0 id 5	$*id + id\$$	reduce by $F \rightarrow id$
(3)	0 F 3	$*id + id\$$	reduce by $T \rightarrow F$
(4)	0 T 2	$*id + id\$$	Shift
(5)	0 T 2 * 7	$id + id\$$	Shift

Moves of LR parser on $id * id + id$

Table: Moves of LR parser on $id * id + id$

	Stack	Input	Action
(1)	0	$id * id + id\$$	Shift
(2)	0 id 5	$*id + id\$$	reduce by $F \rightarrow id$
(3)	0 F 3	$*id + id\$$	reduce by $T \rightarrow F$
(4)	0 T 2	$*id + id\$$	Shift
(5)	0 T 2 * 7	$id + id\$$	Shift
(6)	0 T 2 * 7 id 5	$+id\$$	reduce by $F \rightarrow id$

Moves of LR parser on $id * id + id$

Table: Moves of LR parser on $id * id + id$

	Stack	Input	Action
(1)	0	$id * id + id\$$	Shift
(2)	0 id 5	$*id + id\$$	reduce by $F \rightarrow id$
(3)	0 F 3	$*id + id\$$	reduce by $T \rightarrow F$
(4)	0 T 2	$*id + id\$$	Shift
(5)	0 T 2 * 7	$id + id\$$	Shift
(6)	0 T 2 * 7 id 5	$+id\$$	reduce by $F \rightarrow id$
(7)	0 T 2 * 7 F 10	$+id\$$	reduce by $F \rightarrow T * F$

Moves of LR parser on $id * id + id$

Table: Moves of LR parser on $id * id + id$

	Stack	Input	Action
(1)	0	$id * id + id\$$	Shift
(2)	0 id 5	$*id + id\$$	reduce by $F \rightarrow id$
(3)	0 F 3	$*id + id\$$	reduce by $T \rightarrow F$
(4)	0 T 2	$*id + id\$$	Shift
(5)	0 T 2 * 7	$id + id\$$	Shift
(6)	0 T 2 * 7 id 5	$+id\$$	reduce by $F \rightarrow id$
(7)	0 T 2 * 7 F 10	$+id\$$	reduce by $F \rightarrow T * F$
(8)	0 T 2	$+id\$$	reduce by $E \rightarrow T$

Moves of LR parser on $id * id + id$

Table: Moves of LR parser on $id * id + id$

	Stack	Input	Action
(1)	0	$id * id + id\$$	Shift
(2)	0 id 5	$*id + id\$$	reduce by $F \rightarrow id$
(3)	0 F 3	$*id + id\$$	reduce by $T \rightarrow F$
(4)	0 T 2	$*id + id\$$	Shift
(5)	0 T 2 * 7	$id + id\$$	Shift
(6)	0 T 2 * 7 id 5	$+id\$$	reduce by $F \rightarrow id$
(7)	0 T 2 * 7 F 10	$+id\$$	reduce by $F \rightarrow T * F$
(8)	0 T 2	$+id\$$	reduce by $E \rightarrow T$
(9)	0 E 1	$+id\$$	Shift
(10)	0 E 1 + 6	$id\$$	Shift
(11)	0 E 1 + 6 id 5	$\$$	reduce by $F \rightarrow id$
(12)	0 E 1 + 6 F 3	$\$$	reduce by $T \rightarrow F$
(13)	0 E 1 + 6 T 9	$\$$	reduce by $E \rightarrow E + T$
(14)	0 E 1	$\$$	accept