Algorithm for predictive parsing program –

Input: A string w and a parsing table M for Grammar G

output: if w is in L(G), a left most derivation of w; otherwise an error indication.

Method: Initially, the parser is in a configuration with w$ in the input buffer and the start symbol S of G on top of stack, above $.

```
let 'a' be the first symbol of w;
let X be the top stack symbol;
while ( X ≠ $ )                    /* stack is not empty */
{
  if ( X = a ) pop the stack and let 'a' be next symbol of w;
  elseif ( X is a terminal ) error ();
  elseif ( M [X,a] is an error entry) error;
  elseif ( M [X,a] = X → Y1 Y2 -- Yk ) {
        output the production X → Y1 Y2 -- Yk;
        pop the stack;
        push Yk, Yk-1 -- Y1 on to the stack, with
                                    Y1 on top;
  }
  let X be the top stack symbol;
}
```

Example- consider the following grammar:

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

construct predictive parsing table and also parse the string id + id * id

Soln. We have already constructed the predictive parsing table for this grammar. Now we will parse inputstring id + id * id

	id	+	(	)	*	$
E	$E \to TE'$		$E \to TE'$			
E'		$E' \to +TE'$		$E' \to \epsilon$		$E' \to \epsilon$
T	$T \to FT'$		$T \to FT'$			
T'		$T' \to \epsilon$		$T' \to \epsilon$	$T' \to *FT'$	$T' \to \epsilon$
F	$F \to id$		$F \to (E)$			

Predictive parsing table

MATCHED	STACK	INPUT	ACTION
	$E\$$	id + id * id $	$E \to TE'$
	$TE'\$$	id + id * id $	$T \to FT'$
	$FT'E'\$$	id + id * id $	$F \to id$
	id $T'E'\$$	id + id * id $	Match id
id	$T'E'\$$	+ id * id $	$T' \to \epsilon$
id	$E'\$$	+ id * id $	$E' \to +TE'$
id	$+TE'\$$	+ id * id $	Match '+'
id +	$TE'\$$	id * id $	$T \to FT'$
id +	$FT'E'\$$	id * id $	$F \to id$
id +	id $T'E'\$$	id * id $	Match id
id + id	$T'E'\$$	* id $	$T' \to *FT'$
id + id	$*FT'E'\$$	* id $	Match '*'
id + id *	$FT'E'\$$	id $	$F \to id$
id + id *	id $T'E'\$$	id $	Match id
id + id * id	$T'E'\$$	$	$T' \to \epsilon$
id + id * id	$E'\$$	$	$E' \to \epsilon$
id + id * id	$\$$	$	Accept

LL(1) Grammars-

→ Predictive parsers can be constructed for a class of Grammars, called LL(1). The first "L" stands for scanning the input from left to right. The second "L" for producing a left most derivation and "1" for using one input symbol of lookahead at each step to make parsing action decisions.

→ No left recursive / Non Deterministic or ambiguous grammar can be LL(1).

→ For every LL(1) grammar, each parsing table entry uniquely identifies a production.

NOTE- If in the question it is indicated that you have to construct predictive parsing table then before constructing predictive parsing table, eliminate ambiguity, left recursion and non deterministic. If in the question you have to check for LL(1) grammar then no need to eliminate ambiguity, left recursion or Non deterministic grammar.

example- consider the following grammar

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \epsilon$

$E \rightarrow b$

check that the grammar is LL(1) or not.

soln.

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \epsilon$

$E \rightarrow b$

FIRST(S) = {i, a} FOLLOW(S) = {e, $}

FIRST(S') = {e, ϵ} FOLLOW(S') = {e, $}

FIRST(E) = {b} FOLLOW(E) = {t}

	i	t	a	b	e	$
S	$S \rightarrow iEtSS'$		$S \rightarrow a$			
S'					$S' \rightarrow eS$, $S' \rightarrow \epsilon$	$S' \rightarrow \epsilon$
E				$E \rightarrow b$		

Multiple defined entries, so Grammar is not LL(1)

Practise questions -

check that the following grammars are LL(1) or not :-

(1) $S \to asbS \mid bsas \mid \epsilon$

(2) $S \to aABb$
$A \to c \mid \epsilon$
$B \to d \mid \epsilon$

(3) $S \to aB \mid \epsilon$
$B \to bC \mid \epsilon$
$C \to cS \mid \epsilon$

(4) $S \to AB$
$A \to a \mid \epsilon$
$B \to b \mid \epsilon$

(5) $S \to A$
$A \to Bb \mid Cd$
$B \to aB \mid \epsilon$
$C \to cC \mid \epsilon$

(6) $S \to aAa \mid \epsilon$
$A \to aBS \mid \epsilon$

Answers:- (1) not LL(1)
(2) LL(1)
(3) LL(1)
(4) LL(1)
(5) LL(1)
(6) not LL(1)