UNIT — II

Parsers

- The second phase of the compiler is assign the task of checking the syntax and thus it is called as syntax analyzer or parser.
- Syntax analyzer is a program that takes tokens from the lexical analyzer and checks the syntax of the statements. If the statements are syntactically correct, the phase generates a syntax tree but if there are errors, then the errors are generated and reported.

Role of Parser

- Parser or Syntax analyzer is the program which performs syntax analysis or parsing.
- Parsing is an important phase of compiler-design, which obtains string of tokens produced as output of Lexical Analysis.

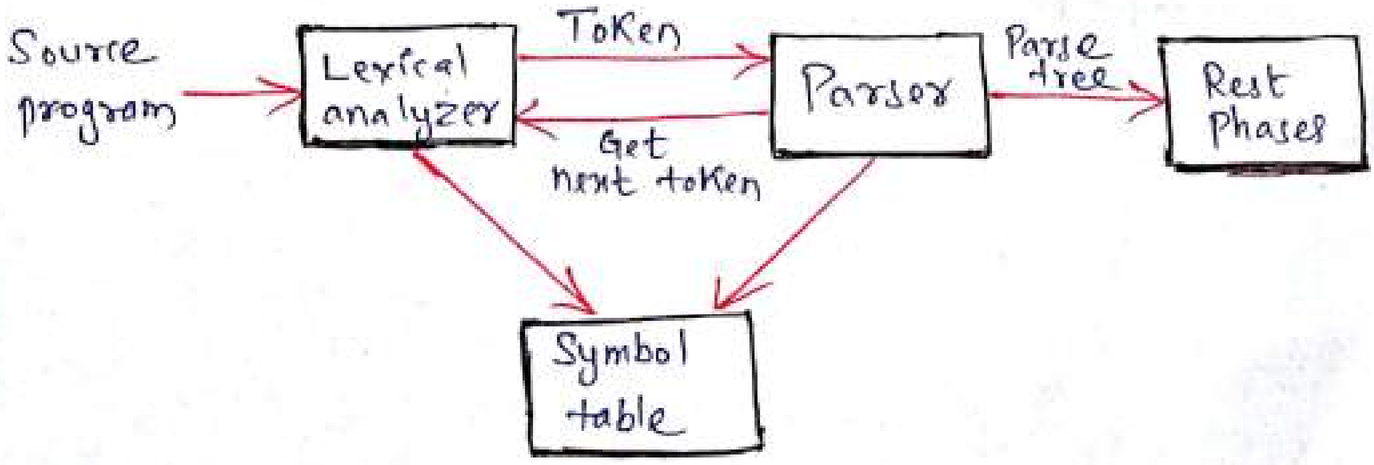


fig. Position of parser in compiler model.

- There are different types of parsers based on the way they parse the input.

  (1) Cocke - Younger - Kasami algorithm

  (2) Earley's algorithm

  (3) Top - down or Bottom - up parser.

First two methods are inefficient in production of compiler, hence commonly top-down and bottom-up parser used.

Top-down parser build parse trees from the top (root) to the bottom (leaves).

Bottom-up parser starts from the leaves and work their way up to the root.

Ambiguous Grammar

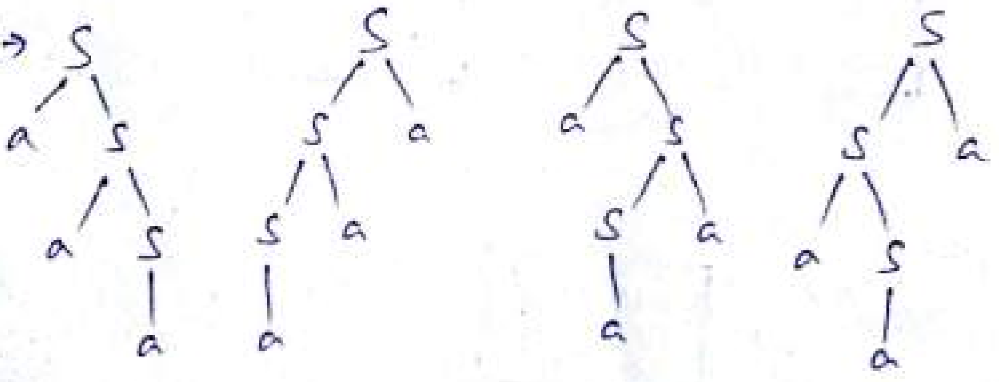
Consider the following grammar

$$S \rightarrow aS/Sa/a$$

String $w = aaa$

How many parse tree's possible ?

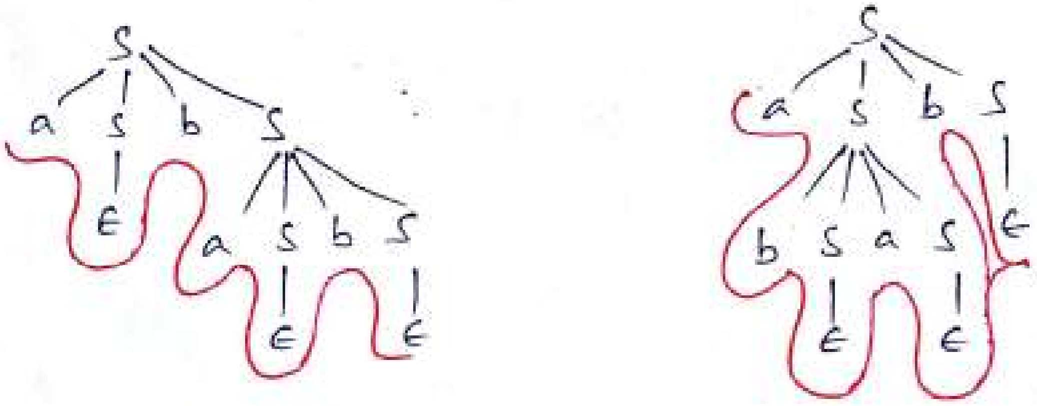
Solution :->



Given grammar is ambiguous grammar, because to generate $w = aaa$ more than one parse tree is available.

Example (2): $S \rightarrow aSbS / bSaS / \epsilon$

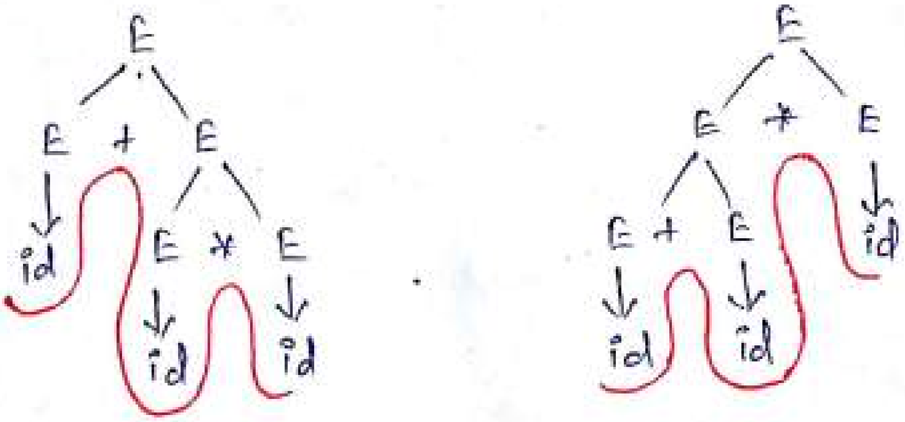
$w = abab$



∴ Ambiguous grammar

Example (1): $E \rightarrow E + E / E * E / id$

$w = id + id * id$



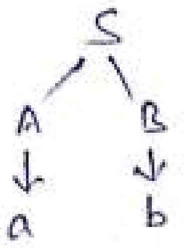
∴ Ambiguous grammar

If in a grammar, there exists atleast one string which give more than one parse tree, then the given grammar is ambiguous.

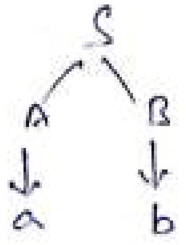
Leftmost Derivation And Rightmost Derivation

- While deriving the string, if we always substitute leftmost variable, then it is called Leftmost derivation.
- While deriving the string, if we always substitute right most variable, then it is called rightmost derivation.

$$S \rightarrow AB$$
$$A \rightarrow a \qquad w = ab$$
$$B \rightarrow b$$



LMD tree



RMD tree

LMD	RMD
S	S
AB	AB
aB	Ab
ab	ab

Note: If the given grammar is unambiguous grammar, then leftmost derivation tree is equal to right most derivation tree.

If the given grammar is ambiguous grammar, then leftmost derivation tree need not be equal to rightmost derivation tree.

Conversion from Ambiguous to Unambiguous Grammar

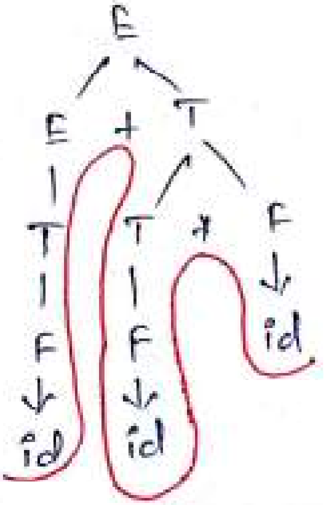
$$\left.\begin{array}{l} E \to E+E \,/\, E*E \,/\, id \\ w = id + id * id \end{array}\right\} \text{Ambiguous Grammar}$$

⇓ According to C

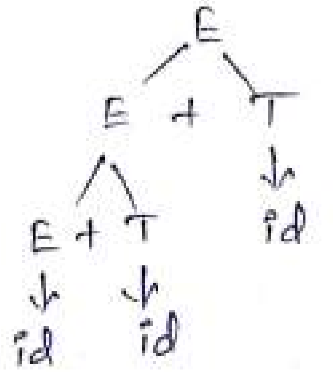
$$\left.\begin{array}{l} E \to E+T \,/\, T \\ T \to T*F \,/\, F \\ F \to id \end{array}\right\} \begin{array}{l}\text{Unambiguous Grammar} \\ \text{Now '*' has higher priority than '+'}\end{array}$$

$w = id + id * id$

∴



If $w = id + id + id$



∴ '+' is having left-to-right associativity.

$$\therefore \begin{array}{l} E \to E+T \,|\, E-T \,|\, T \\ T \to T*F \,|\, T/F \,|\, F \\ F \to G\uparrow F \,/\, G \\ G \to id. \end{array}$$

Note: (i) '+' and '−' both have same priority but associativity is left to Right.

(2) '*' and '/' both have same priority but associativity is Left to Right.

(3) ↑ has higher priority and associativity is Right-to Left.

Q. Convert the following Ambiguous Grammar to equivalent unambiguous Grammar.

$E \rightarrow E+E \mid E*E \mid E/E \mid E-E \mid E \uparrow E \mid id$

With the following Rules :

↑ : Lower priority & Associativity is Left - Right.

*,+ : next. lower priority & Associativity is Right - Left

/ : next-lower priority & Associativity is Right - Left

— : Higher priority & Associativity is Left- Right.

Solution :>

$E \rightarrow E \uparrow T \,/\, T$

$T \rightarrow F*T \mid F+T \mid F$

$F \rightarrow G/F \mid G$

$G \rightarrow G-H \,/\, H$

$H \rightarrow id$

Q. $R \rightarrow R+R \mid R.R \mid R^* \mid a \mid b \mid \epsilon$

According to C language,

Union : lower priority

concatenation: Middle priority

Kleen closure: Higher priority

} Associativity is left-Right

$\therefore$ $R \rightarrow R+S/S$

$S \rightarrow S.T/T$

$T \rightarrow G^*/G$ $\Big\}$ $T \rightarrow T^*/\epsilon/a/b$

$G \rightarrow \epsilon/a/b$

Q. $B \rightarrow B$ or B / B and B / not B / $0/1$

$\Downarrow$ C-language

[or - lower
and - Middle
not - Higher] Associativity is Left · Right.

$B \rightarrow B$ or C/C

$C \rightarrow C$ and D/D

$D \rightarrow$ not E/E $\Big\}$ $D \rightarrow$ not $D/0/1$.

$E \rightarrow 0/1$

Recursion :→

Left Recursion
e.g $S \rightarrow Sa$

Right Recursion
e.g $S \rightarrow aS$

① $S \rightarrow Sa/b$
$w = baaa$

$$S \rightarrow Sa \rightarrow Sa \rightarrow Sa \cdots$$

∞ loop

Left Recursion

In this, productions are of the form:

$A \rightarrow A\alpha/\beta$

$\alpha \in (V+T)^*$

$\beta \in T^*$

② $S \rightarrow aS/b$
$w = aaab$

$$S \rightarrow aS \rightarrow aS \rightarrow aS \rightarrow b$$

$\therefore$ no problem.

Right Recursion

$A \rightarrow \alpha A/\beta$

$\alpha \in (V+T)^*$

$\beta \in T^*$

Elimination of Left Recursion

Example (1)

$E \rightarrow E + T / T$

$T \rightarrow T * F / F$

$F \rightarrow id$

NOTE: $A \rightarrow A\alpha | \beta$

Solution:

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' / \epsilon$

Solution:>

$E \rightarrow E + T / T$

$\Downarrow$

$E \rightarrow T E'$

$E' \rightarrow + T E' / \epsilon$

$T \rightarrow T * F / F$

$\Downarrow$

$T \rightarrow F T'$

$T' \rightarrow * F T' / \epsilon$

$\therefore$ $E \rightarrow E + T / T$

$T \rightarrow T * F / F$

$F \rightarrow id$

$\Rightarrow$

$E \rightarrow T E'$

$E' \rightarrow + T E' / \epsilon$

$T \rightarrow F T'$

$T' \rightarrow * F T' / \epsilon$

$F \rightarrow id$

Example (2)

$S \rightarrow (L) / a$

$L \rightarrow L , S / S$

Solution:

$L \rightarrow L , S / S$

$\Downarrow$

$L \rightarrow S L'$

$L' \rightarrow , S L' / \epsilon$

$\therefore$ $S \to (L)/a$ $\quad\Rightarrow\quad$ $S \to (L)/a$

$L \to L,S/S$ $\quad\quad\quad\quad$ $L \to SL'$

$L' \to ,SL'/\epsilon$

Example 3:

$S \to aBDh$

$B \to Bb/h$

$D \to EF$

$E \to g/\epsilon$

$F \to f/\epsilon$

Solution :>

$S \to aBDh$

$B \to hB'$

$B' \to bB'/\epsilon$

$D \to EF$

$E \to g/\epsilon$

$F \to f/\epsilon$

Example 4:

$S \to A$

$\underline{A} \to \underline{A}d/\underline{A}e/\underline{A}f/aB/aC$

$B \to bBc/f$

Solution :>

$S \to A$

$A \to aBA'/aCA'$

$A' \to dA'/eA'/fA'/\epsilon$

$B \to bBc/f$

Example 5:

$S \to Aa/b$

$A \to Ac/Sd/\epsilon$

Solution:>

$$S \to Aa / b$$
$$A \to Ac / Sd / \epsilon$$
$$\Downarrow$$
$$S \to Aa / b$$
$$A \to Ac / Aad / bd / \epsilon$$

∴ After removing left recursion, we have

$$S \to Aa / b$$
$$A \to bdA' / A'$$
$$A' \to \epsilon / cA' / adA'.$$

Left Factoring

$$S \to ab / ac / ad$$
$$w = ad$$

From the above grammar, to generate string w=ad all the three productions are fighting because the 1st character in the string is given by all the three production.

This problem is Known as Left factoring.

Elimination of Left Factoring

$$S \to ab / ac / ad$$
$$w = ad$$
$$\Downarrow$$
$$S \to aS'$$
$$S' \to b / c / d$$

Example (2):

$S \rightarrow iEtS \mid iEtSeS \mid b$

$E \rightarrow a$

$\Downarrow$

$S \rightarrow b \mid iEtSS'$

$S' \rightarrow \epsilon \mid eS$

$E \rightarrow a$

Example (3):

$S \rightarrow a \mid ab \mid abc \mid abcd$

$\Downarrow$

$S \rightarrow aS'$

$S' \rightarrow \epsilon \mid bS''$

$S'' \rightarrow \epsilon \mid cS'''$

$S''' \rightarrow \epsilon \mid d$

Top - down Parser :->

There are two types of Top-down parser

(1) Recursive Descent Parser

(2) Non-Recursive Descent Parser or LL (1).

(1) Recursive Descent Parser

- A recursive-descent parsing program concists of a set of procedures one for each non-terminal.
- Execution begins with the procedure for the start symbol which halts and announces success if its procedure body scans the entire input string.

Algorithm

```
void S()
{
choose a production of S ie $S \rightarrow x_1 x_2 x_3 , \ldots x_n$
for (i = 1 to n)
  {
   if ($x_i$ is variable)
        call procedure $x_i()$;
   else if ($x_i$ equals the current input symbol a)
         advance the input to the next symbol
   else
       error (means choose another production of S)
  }
 }
```

- Recursive descent parser uses leftmost derivation.
- If the grammar contain left recursion, Recursive descent parser will go to infinite loop.
- lot of time is wasted in backtracking.
- Data structure used is stack.

Example :->

$A \rightarrow abC / aBd / aAD$

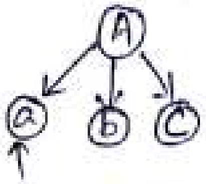
$B \rightarrow bB / \epsilon$

$C \rightarrow d / \epsilon$

$D \rightarrow a / b / \epsilon$

string $w = aaba$

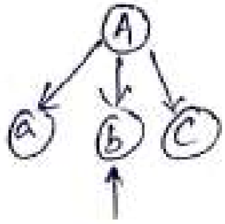
Input = aaba



Here, input character matches the first character of the derived string.

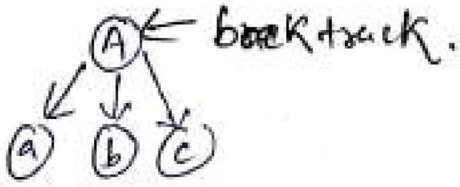
Now increment the input pointer (reading next input character).

Input = aaba



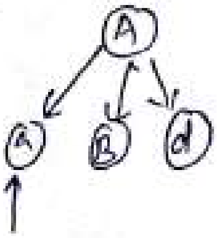
Here, input character is 'a' and the derived character is 'b'. Hence, we see that the production rule chosen is not the appropriate one.

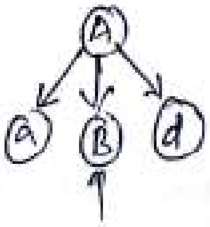
So we need the backtrack.



Using next production rule $A \rightarrow aBd$ and start reading input again.

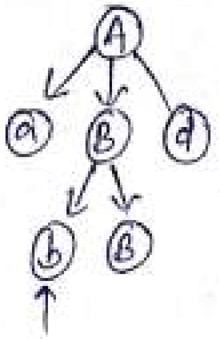
Input = aaba





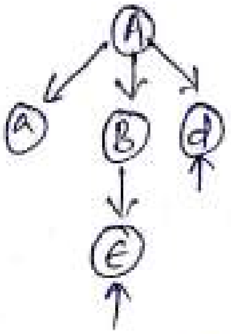
Now we encounter a non-terminal B in the tree.

we use rule $B \rightarrow bB$.



This production gives 'b' as the next character which is not appropriate for the derivation.

Next production is B→ε.



Input = aaba

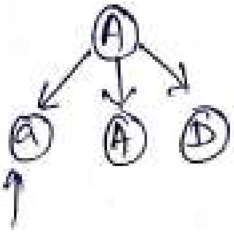
The current input character 'a' does not match the derived 'd'. So backtrack.

There are no productions remaining derived from B.

So backtrack to previous non-terminal i.e. A.

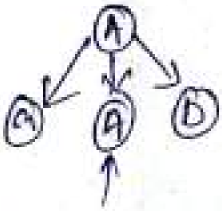
Now using A→aAD and start from beginning.

Input = aaba



Increment input pointer

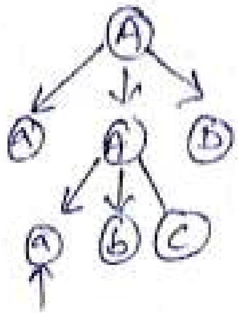
Input = aaba



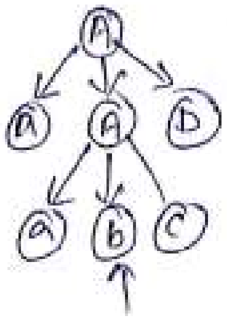
Scanned by CamScanner

For non-terminal A using production rule A→abC

Input = aaba
↑

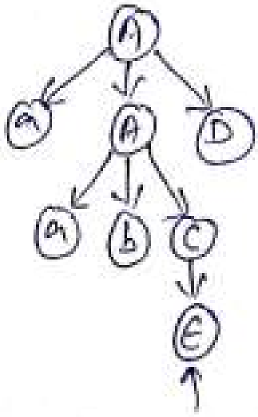


Input= aaba
↑

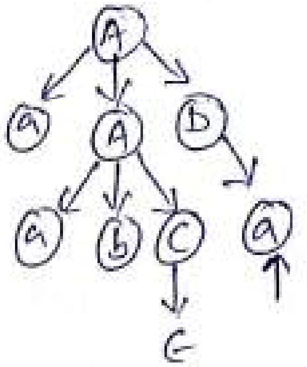


Input = aaba
↑

Substiterting C→d will not be an appropriate step. So replace C→ε



Now D→a



∴ The string aaba is accepted. and successful completion of parsing.

Question:-> Consider the grammar

$S \rightarrow cAd$

$A \rightarrow ab/a$

String $w = cad$

Use Recursive Descent Parser to parse the given String.

Ans:-> Successful completion of string.parsing

Q $E \rightarrow TE'$

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

Ans:-> Recursive Descent Parser Algorithm for above production is :->

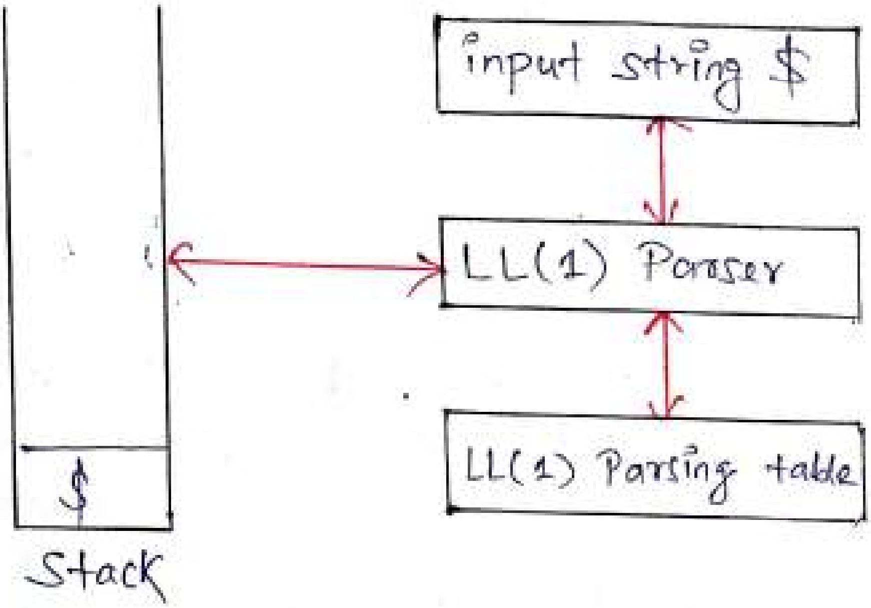
```
E()
{
 if (l== 'i')
 {
  match('i');
  E'();
 }
}

 E'()
 {
  if (l=='+')
  {
   match ('+');
   match ('i');
   E'();
  }
 else
     return;
 }

 match( char t)
 {
  if ( l==t)
     l = getchar();
    else
     print("error");
  }
```

```
main ( )
{
 E();
 if ( l=='$')
    printf ("success");
}
```

Example : i+i $

Non - Recursive Descent Parser (or) LL(1)

- A non-recursive descent parsing can be implemented by maintaining an input buffer, a stack and a parsing table rather than implicit implementation using recursive calls.



- LL(1) parsing table has a row for every non-terminal and a column for every terminal.
- The stack contains a sequence of grammar symbol with $ at the bottom. Initially the stack contains the starting symbol of the grammar on the top.
- The first "L" in LL(1) stands for scanning the input from left to right.

  The second "L" for producing a leftmost derivation.

  The "1" for using one input symbol of lookahead at each step to make parsing action decisions.

LL(1) Parsing Algorithm

Let X is top-of-stack and 'a' is lookahead symbol.

(i) If $((X == a) == \$)$ then successful completion of parsing.

(ii) If $((X == a) \neq \$)$ then pop TOS and increment input pointer.

(iii) If (X is variable) then

see parsing table entry $M[x,a]$

If $M[x,a] = x \rightarrow uvw$

then replace x by uvw in reverse order.

(iv) If $M[x,a]$ = blank, then Error.

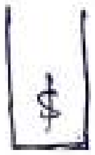
Question (1): Apply LL(1) parsing algorithm on the following grammar to parse the given input string.

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

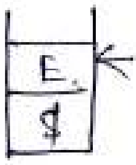
string $w = id + id * id$

	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→FT'		T'→ε	T'→ε
F	F→id			F→(E)		

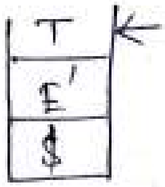
Solution:→ w = id + id * id $
↑a



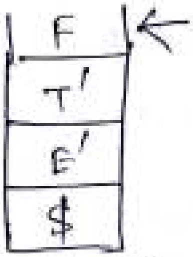
step(1)



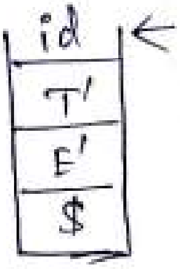
step(2)



step(3)



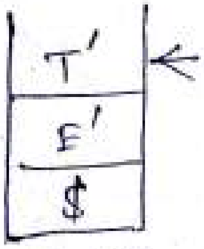
step(4)



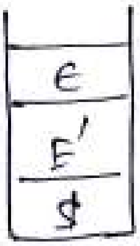
step(5)

Now id==id.
∴ pop stack and
increment input pointer

w = id + id * id $
↑

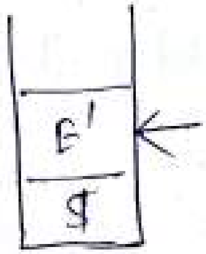


step(6)

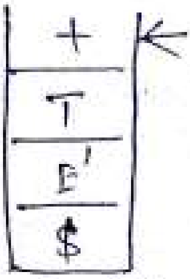


step(7)

≈



step(8)

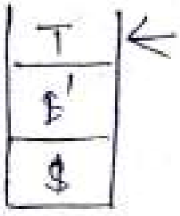


step(9)

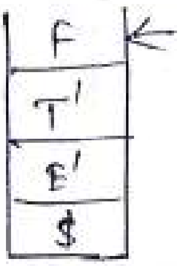
+ = = +

∴ pop TOS and increment input pointer.

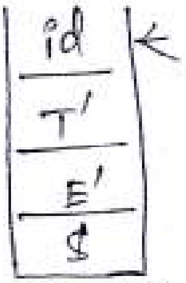
$w = id + id * id \$$
↑



Step (10)



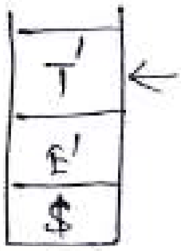
step(11)



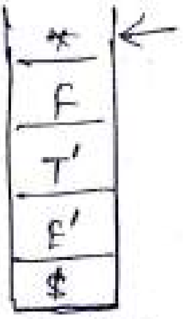
step (12)

id == id
∴ Pop TOS and increment input pointer

$w = id + id * id \$$
↑



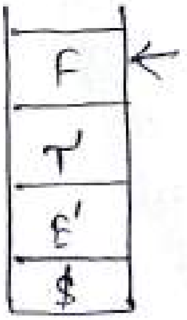
step (13)



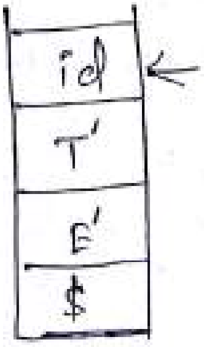
step(14)

* = = *
∴ pop TOS and increment input pointer

$w = id + id * id \$$
↑



step (15)

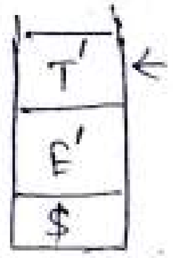


step (16)

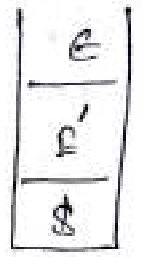
id == id
∴ Pop TOS and increment input pointer

Scanned by CamScanner

w = id + id * id $.

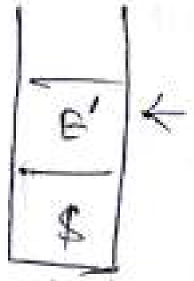


step(17)

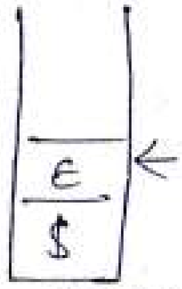


step(18)

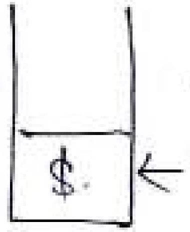
≈



step(19)



step(20)



step(21)

$==$

∴ successful completion of parsing.

Question 2   w = (id + id) * id

Input	Stack	Production
(id+id)*id$	$E	E → TE'
	$E'T	T → FT'
	$E'T'F	F → (E)
	$E'T')E(	
id+id)*id$	$E'T')E	E → TE'
	$E'T')E'T	T → FT'
	$E'T')E'T'F	F → id
	$E'T')E'T'id	
+id)*id$	$E'T')E'T'	T' → ε
	$E'T')E'	E' → +TE'
	$E'T')E'T+	

id)*id$	$E'T')E'T'	T→FT'
	$E'T')E'T'F	F→id
	$E'T')E'T'id	
)*id$	$ET')E'T'	T'→ε
	$E'T')	E'→ε
*id$	$E'T'	T'→*FT'
	$E'T'F+	
id$	$E'T'F	F→id
	$E'T'id	
$	$E'T'	T'→ε
	$ ⇒ Successful completion.	E'→ε

Question 3   S → (L)/a

L → SL'

L' → ε/,SL'

	a	,	(	)	$
S	S→a		S→(L)		
L	L→SL'		L→SL'		
L'		L'→,SL'		L'→ε	

w = (a,a,a)

Input	Stack	Production
$(a,a,a)\$$	$\$S$	$S \to (L)$
	$\$)L($	
$a,a,a)\$$	$\$)L$	$L \to SL'$
	$\$)L'S$	$S \to a$
	$\$)L'a$	
$,a,a)\$$	$\$)L'$	$L' \to ,SL'$
	$\$)L'S,$	
$a,a)\$$	$\$)L'S$	$S \to a$
	$\$)L'a$	
$,a)\$$	$\$)L'$	$L' \to ,SL'$
	$\$)L'S,$	
$a)\$$	$\$)L'S$	$S \to a$
	$\$)L'a$	
$)\$$	$\$)L'$	$L' \to \epsilon$
	$\$)$	
$\$$	$\$$	successful completion.

Note :-> Time complexity of LL(1) parsing algorithm is O(n). Because for every terminal we will take only one production.

LL(1) Parsing Table Construction

$\Downarrow$

First()

Follow()

First() :-> First(A) gives set of all terminals that may begin in strings derived by A.

Example 1: $S \rightarrow abc|def|ghi$

$First(S) = \{a, d, g\}$

Example 2: $S \rightarrow \epsilon$

$First(S) = \{\epsilon\}$

Example 3:

$S \rightarrow ABC$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c$

$First(S) = First(ABC)$

$= First(A)$

$= \{a\}$

$\therefore First(A) = \{a\}$

$First(B) = \{b\}$

$First(C) = \{c\}$

Example 4:

$S \rightarrow ABC$

$A \rightarrow a/h$

$B \rightarrow b|d|e$

$C \rightarrow c|f|g$

$First(S) = First(ABC)$

$= \{a, h\}$

$First(A) = \{a, h\}$

$First(B) = \{b, d, e\}$

$First(C) = \{c, f, g\}$

Example 5:

$S \rightarrow ABC$
$A \rightarrow a|h|\epsilon$
$B \rightarrow b|d|e$
$C \rightarrow c|f|g$

$\text{First}(S) = \text{First}(ABC)$
$= \{a, h, b, d, e\}$

$\text{First}(A) = \{a, h, \epsilon\}$

Example 6:

$S \rightarrow ABC$
$A \rightarrow a|h|\epsilon$
$B \rightarrow b|d|e|\epsilon$
$C \rightarrow c|f|g$

$\text{First}(S) = \text{First}(ABC)$
$= \{a, h, b, d, e, c, f, g\}$

$\text{First}(B) = \{b, d, e, \epsilon\}$

Example 7:

$S \rightarrow ABC$
$A \rightarrow a|h|\epsilon$
$B \rightarrow b|d|e|\epsilon$
$C \rightarrow c|f|g|\epsilon$

$\text{First}(S) = \text{First}(ABC)$
$= \{a, h, b, d, e, c, f, g, \epsilon\}$

$\text{First}(C) = \{c, f, g, \epsilon\}$

Rules to find out "First"

(1) If α is terminal, then $\text{First}(\alpha) = \alpha$

(2) If α is variable,

(i) $\alpha \rightarrow \epsilon$
$\text{First}(\alpha) = \epsilon$

(ii) $\alpha \rightarrow x_1 x_2 x_3$
$x_1 \rightarrow a|b$
$x_2 \rightarrow c|d$
$x_3 \rightarrow e|f$

$\text{First}(\alpha) = \text{First}(x_1 x_2 x_3)$
$= \text{First}(x_1) - \epsilon$ [if First(x_1) contain]
$= a, b$
$\cup$

$$\text{first}(x_2 x_3)$$
$$\Downarrow$$
$$\text{first}(x_2) - \epsilon \ [\text{if first}(x_2) \text{ contain}]$$
$$\cup$$
$$\text{first}(x_3)$$

Question :> Find first for the following Grammar.

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow id / (E)$

Solution :>

$\text{first}(F) = \{id, (\}$

$\text{first}(T') = \{\epsilon, *\}$

$\text{first}(T) = \text{first}(FT')$

$= \text{first}(F) = \{id, (\}$

$\text{first}(E') = \{\epsilon, +\}$

$\text{first}(E) = \text{first}(TE')$

$= \text{first}(T) = \{id, (\}$

Question :> Find First for the following Grammar.

$S \rightarrow (L)/a$

$L \rightarrow SL'$

$L' \rightarrow \epsilon / ,SL'$

Solution:

$\text{First}(L') = \{\epsilon, ,\}$

$\text{First}(L) = \{(, a\}$

$\text{First}(S) = \{(, a\}$

Question:-> Find First for the following Grammar.

$S \rightarrow aBDh \mid EF \mid DBDh$

$B \rightarrow cC$

$C \rightarrow bC \mid \epsilon$

$D \rightarrow EF$

$E \rightarrow g \mid \epsilon$

$F \rightarrow f \mid \epsilon$

Solution:-> First(F) = $\{f, \epsilon\}$

First(E) = $\{g, \epsilon\}$

First(D) = $\{g, f, \epsilon\}$

First(C) = $\{b, \epsilon\}$

First(B) = $\{c\}$

First(S) = $\{a, g, f, \epsilon, c\}$.

Question:-> Find First for the following Grammar.

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

Solution:->

First(B) = $\{\epsilon\}$

First(A) = $\{\epsilon\}$

First(S) = $\{a, b\}$

Follow

Follow (A) gives set of all terminals that may follow immediately to the right of A.

Rules to find Follow

(1) If A is start symbol,

Follow (A) = \$

(2) If $S \rightarrow ABCD$, $B \rightarrow b$

Follow (A) = First (BCD).

= First (B)

= b

(3) If $S \rightarrow BA$ (or) $S \rightarrow BAC$

$C \rightarrow \epsilon$

then Follow(A) = Follow (S)

NOTE :-> Follow never give epsilon 'ϵ'.

Question 1 :-> Find First and Follow :

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow id / (E)$.

Solution :-> First (F) = { id, (}

First (T') = { ϵ, * }

First (T) = { id, (}

First (E') = { ϵ, + }

First (E) = { id, (}

Follow (E) = { \$,) }

Follow (E') = Follow (E)

= { \$,) }

Follow (T) = First (E')

$\Rightarrow$ = { +, \$,) }

i.e. +, Follow (E)

Follow (T') = Follow (T)

= {+, $,)}   i.e +, Follow (E)

Follow (F) = First (T')

= {*, +, $,)}   i.e *, Follow (T)

Question 2 :> Find First and Follow

S → (L)/a

L → SL'

L' → ε/ , SL'

Solution :> First (L') = {ε, ,}

First (L) = {(, a}

First (S) = {(, a}

Follow (S) = First (L')

= {, , Follow (L)}

= {, ,), $}

Follow (L) = {) }

Follow (L') = Follow (L)

= {) }

Question 3 :> Find First and Follow

S → AaAb / BbBa

A → ε

B → ε

Solution :> First (B) = {ε}

First (A) = {ε}

First (S) = {a, b}

Follow (S) = { $ }

Follow (A) = {a, b}

Follow (B) = {b, a}

Question 4 :> Find first and Follow

$S \rightarrow aBDh$

$B \rightarrow cC$

$C \rightarrow bC / \epsilon$

$D \rightarrow EF$

$E \rightarrow g / \epsilon$

$F \rightarrow f / \epsilon$

Solution :> First $(F) = \{f, \epsilon\}$

First $(E) = \{g, \epsilon\}$

First $(D) = \{g, f, \epsilon\}$

First $(C) = \{b, \epsilon\}$

First $(B) = \{c\}$

First $(S) = \{a\}$

Follow $(S) = \{\$\}$

Follow $(B) =$ First (Dh)
$= \{g, f, h\}$

Follow $(C) =$ Follow (B)
$= \{g, f, h\}$

Follow $(D) = \{h\}$

Follow $(E) =$ First (F)
$= \{f, h\}$

Follow $(F) =$ Follow $(D) = \{h\}$

Question 5 :> find First and Follow

$A \rightarrow BA'$

$A' \rightarrow *BA' / \epsilon$

$B \rightarrow CB' / \epsilon$

$B' \rightarrow *CB' / \epsilon$

$C \rightarrow +Ad / id$

Solution :> First $(C) = \{+, id\}$

First $(B') = \{*, \epsilon\}$

First $(B) = \{\epsilon, +, id\}$

First $(A') = \{*, \epsilon\}$

First $(A) =$ First (BA')
$= \{+, id, *, \epsilon\}$

$\text{Follow}(A) = \{\$, d\}$

$\text{Follow}(A') = \text{Follow}(A)$
$= \{\$, d\}$

$\text{Follow}(B) = \text{First}(A')$
$= \{*, \$, d\}$

$\text{Follow}(B') = \text{Follow}(B) = \{*, \$, d\}$

$\text{Follow}(C) = \text{First}(B') = \{*, \$, d\}.$

LL(1) Parsing Table:M Construction

For each production $A \to \alpha$, repeat following steps:

(i) Add $A \to \alpha$ under $M[A, c]$, where $c \in \text{first}(\alpha)$

(ii) Add $A \to \alpha$ under $M[A, d]$, where $d \in \text{Follow}(A)$ if first (α) contain ϵ.

Question:> Construct LL(1) parsing table for the following Grammar.

$E \to TE'$

$E' \to \epsilon / +TE'$

$T \to FT'$

$T' \to \epsilon / *FT'$

$F \to id / (E).$

Solution:>

	Input symbols					
Non-terminal Symbol	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'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to id$			$F \to (E)$		

Note :> In the LL(1) parsing table, every entry contain maximum one production, then given grammar is LL(1) otherwise it is not LL(1).

Question 2 :> Construct LL(1) parsing table for the following Grammar.

$$S \to (L)/a$$
$$L \to SL'$$
$$L' \to \epsilon/,SL'$$

Solution :>

	(	)	a	,	$
S	$S \to (L)$		$S \to a$		
L	$L \to SL'$		$L \to SL'$		
L'		$L' \to \epsilon$		$L' \to ,SL'$	

∴ Given grammar is LL(1).

Question 3 :> Check the following Grammar is LL(1) or not

$$S \to AaAb \mid BbBa$$
$$A \to \epsilon$$
$$B \to \epsilon$$

Solution :->

	a	b	$
S	$S \rightarrow AaAb$	$S \rightarrow BbBa$	
A	$A \rightarrow \epsilon$	$A \rightarrow \epsilon$	
B	$B \rightarrow \epsilon$	$B \rightarrow \epsilon$	

∴ Given grammar is LL(1).

Question 4 :-> Check the following grammar is LL(1) or not.

$$S \rightarrow aBDh$$
$$B \rightarrow cC$$
$$C \rightarrow bC / \epsilon$$
$$D \rightarrow EF$$
$$E \rightarrow g / \epsilon$$
$$F \rightarrow f / \epsilon$$

Solution :->

	a	b	c	g	f	h	$
S	$S \rightarrow aBDh$						
B			$B \rightarrow cC$				
C		$C \rightarrow bC$		$C \rightarrow \epsilon$	$C \rightarrow \epsilon$	$C \rightarrow \epsilon$	
D				$D \rightarrow EF$	$D \rightarrow EF$	$D \rightarrow EF$	
E				$E \rightarrow g$	$E \rightarrow \epsilon$	$E \rightarrow \epsilon$	
F					$F \rightarrow f$	$F \rightarrow \epsilon$	

∴ Given grammar is LL(1).

Question 5 :-> Check the following Grammar is LL(1) or not

S → A
A → aBA'
A' → dA'/ε
B → b
C → g

Solution :->

	a	b	d	g	$
S	S → A				
A	A → aBA'				
A'			A' → dA'		A' → ε
B		B → b			
C				C → g	

∴ Given grammar is LL(1).

Question 6 :-> Check the following Grammar is LL(1) or not.

S → aSbS | bSaS | ε

Solution :->

	a	b	$
S	S → ε S → aSbS	S → ε S → bSaS	S → ε

∴ It is not LL(1)

Because single cell contain multiple entries.

Predictive LL(1) Parser

LL(1)

- First L means the input is scanned from left to right.
- Second L means it uses leftmost derivation for input string.
- Number 1 means it uses only one input symbol (lookahead) to predict the parsing process.

Q For the following grammar, find FIRST and FOLLOW sets for each of non-terminal.

$$S \rightarrow aAB \mid bA \mid \epsilon$$
$$A \rightarrow aAb \mid \epsilon$$
$$B \rightarrow bB \mid \epsilon$$

Soln

$\text{First}(B) = \{b, \epsilon\}$

$\text{First}(A) = \{a, \epsilon\}$

$\text{First}(S) = \{a, b, \epsilon\}$

$\text{Follow}(S) = \{\$\}$

$\text{Follow}(A) = \{b, \text{First}(B), \text{Follow}(S)\}$

$= \{b, \$\}$

$\text{Follow}(B) = \text{Follow}(S)$

$= \{\$\}$

Q Consider the grammar:→

$S \rightarrow ACB \mid CbB \mid Ba$

$A \rightarrow da \mid BC$

$B \rightarrow g \mid \epsilon$

$C \rightarrow h \mid \epsilon$

Calculate FIRST and FOLLOW.

Soln First (C) = $\{h, \epsilon\}$

First (B) = $\{g, \epsilon\}$

First (A) = $\{d, \text{First}(B)\}$

$= \{d, g, \text{First}(C)\}$

$= \{d, g, h, \epsilon\}$ ✓

First (S) = $\{\text{First}(A), \text{First}(C), \text{First}(B)\}$

$= \{d, g, h, \epsilon\}, b, a\}$

Follow (S) = $\{\$\}$

Follow (A) = First (C) = $\{h, \text{First}(B)\}$

$= \{h, g, \text{Follow}(S)\}$

$= \{h, g, \$\}$

Follow (B) = $\{a, \text{Follow}(S), \text{First}(C)\}$

$= \{a, \$, h, \text{Follow}(A)\}$

$= \{a, \$, h, g\}$

Follow (C) = $\{b, \text{Follow}(A), \text{First}(B)\}$

$= \{b, h, g, \$\}$

Q Check whether the given grammar is LL(1)? Remove left recursion and then again verify whether it is LL(1)?

$$S \to Aa \mid b$$
$$A \to Ac \mid Sd \mid \epsilon$$

Soln $\quad A \to Ac$ is left recursive. ∴ not LL(1).

Left recursion can be removed as follows:

If $A \to A\alpha / \beta$, then $A \to \beta A'$
$\qquad A' \to \alpha A' / \epsilon$

∴ $\underset{A}{A} \to \underset{A}{A}\underset{\alpha}{c} \mid \underset{\beta}{Sd} \mid \underset{\beta}{\epsilon}$

∴ $A \to SdA' / \epsilon A'$

$A' \to cA' / \epsilon$

∴ production becomes $\quad S \to Aa / b$
$\qquad A \to SdA' / \epsilon A'$
$\qquad A' \to cA' / \epsilon$

~~First~~ Now to check for LL(1).

Compare with $A \to \alpha$

∴ $\underset{A}{S} \to \underset{\alpha}{Aa}$

First (Aa) = { b, a, c}

$$\begin{tikzcd} S \arrow[r] \arrow[d] & b \arrow[d] \\ A & \alpha \end{tikzcd}$$

first (b) = {b}

$$\begin{tikzcd} A \arrow[r] \arrow[d] & \underbrace{SdA'} \arrow[d] \\ A & \alpha \end{tikzcd}$$

first (α) i.e. first (SdA') = {b, first(A)}

(OR)

Method to check for LL(1).

first (A') = {c, ε}

first (A) = {first (S), first (A')}

= {a, b, c, ε}

first (S) = {a, b, c}

follow (S) = {$, d}

follow (A) = {a}

follow (A') = follow (A) = {a}

∴

	a	b	c	d	$
S	$S \to Aa$	$S \to b$			
A	$A \to SdA'$ $A' \to \epsilon A'$	$A \to SdA'$			
A'	$A' \to cA'$ $A' \to \epsilon$				

table contain multiple entries.

∴ The Given grammar is not LL(1).

Ques:> Design LL(1) parsing table for the following grammar.

$A \to AcB \mid cC \mid C$

$B \to bB \mid id$

$C \to CaB \mid BbB \mid B$

Soln

$A \to AcB \mid cC \mid C$ is left recursive. We will eliminate left recursion.

$$\underbrace{A}_{A} \to \underbrace{A}_{A}\underbrace{cB}_{\alpha} / \underbrace{cC}_{\beta} / \underbrace{C}_{\beta}$$

∴ If $A \to A\alpha / \beta$, then $A \to \beta A'$

$A' \to \alpha A' / \epsilon$

∴ $A \to AcB|cC|C$ becomes

$A \to cCA' / CA'$

$A' \to cBA' / \epsilon$

The rule now becomes:>

$A \to cCA' / CA'$

$A' \to cBA' / \epsilon$

$B \to bB / id$

and $C \to CaB|BbB|B$ is left recursive.

$$\underbrace{C}_{A} \to \underbrace{C}_{A}\underbrace{aB}_{\alpha} / \underbrace{BbB}_{\beta} / \underbrace{B}_{\beta}$$

$\therefore$ $C \rightarrow BbBC' / BC'$

$C' \rightarrow aBC' / \epsilon$

Finally the rule becomes

$A \rightarrow cCA' / CA'$

$A' \rightarrow cBA' / \epsilon$

$B \rightarrow bB / id$

$C \rightarrow BbBC' / BC'$

$C' \rightarrow aBC' / \epsilon$

Now consider the rule $C \rightarrow BbBC' / BC'$ has left factoring.

$$\underset{A}{C} \rightarrow \underset{\alpha}{B}\,\underbrace{bBC'}_{\beta_1} \,/\, \underset{\alpha}{B}\,\underset{\beta_2}{C'}$$

If $A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \cdots$, then $A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 | \beta_2 | \cdots$

$\therefore$ $C \rightarrow BC''$

$C'' \rightarrow bBC' | C'$

$\therefore$ The grammar can be completely written as :->

$A \rightarrow cCA' / CA'$

$A' \rightarrow cBA' / \epsilon$

$B \rightarrow bB / id$

$C \rightarrow BC''$

$C'' \rightarrow bBC' / C'$

$C' \rightarrow aBC' / \epsilon$

First (C') = {a, ε}

First (C'') = {b, First(C')}
= {b, a, ε}

First (C) = First (B) = {b, id}

First (B) = {b, id}

First (A') = {c, ε}

First (A) = {c, First(C)}
= {c, b, id}

Follow (A) = {$.}

Follow (A') = Follow(A) = {$}

Follow (B) = First (A'), First (C''),
First (C')
= {c, $, b, a, id}

Follow (C) = {First (A')}
= {c, Follow (A)}
= {c, $}

Follow (C'') = Follow(C) = {c, $}

Follow (C') = Follow (C'') = {c, $}

∴ Predictive Parsing table is:

	a	b	c	id	$
A		A→CA'	A→cCA'	A→CA'	
A'			A'→cBA'		A'→ε
B		B→bB		B→id	
C		C→BC''		C→BC''	
C'	C'→aBC'		C'→ε	ε'.	C'→ε
C''	C''→C'	C''→bBC'	C''→C'		C''→C'

Q Consider the grammar

$S \rightarrow iCtSA / a$

$A \rightarrow eS / \epsilon$

$C \rightarrow b$

whether it is LL(1) grammar ? Give the explanation whether (i,t,e,b,a) are terminal symbols.

Sol

$S \rightarrow iCtSA | a$

$A \rightarrow eS / \epsilon$

$C \rightarrow b$

First(C) = {b}

First(A) = {e, ϵ}

First(S) = {i, a}

Follow(S) = \$, First(A), Follow(A)
= {\$, e}

Follow(A) = Follow(S) = {\$, e}

Follow(C) = {t}

The predictive parsing table is

	a	b	e	t	i	\$	
S	S→a				S→iCtSA		
A			A→eS A→ϵ			A→ϵ	
C		C→b					

As we have got multiple entries in M[A,e].

∴ Grammar is not LL(1) grammar.

The (i,t,e,b,a) are the terminal symbols because they do not derive any production rule.

Q Construct parsing table for the following grammar:

$S \rightarrow aXYb$

$X \rightarrow c | \epsilon$

$Y \rightarrow d | \epsilon$

Solution First $(Y) = \{d, \epsilon\}$

First $(X) = \{c, \epsilon\}$

First $(S) = \{a\}$

Follow $(S) = \{\$\}$

Follow $(X) =$ First (Yb)

$= \{d, b\}$

Follow $(Y) = \{b\}$

The parsing table is

	a	b	c	d	\$
S	$S \rightarrow aXYb$				
X		$X \rightarrow \epsilon$	$X \rightarrow c$	$X \rightarrow \epsilon$	
Y		$Y \rightarrow \epsilon$		$Y \rightarrow d$	

Q. Consider the grammar

texp → atom | list

atom → number | identifier

list → (textp-seg)

textp-seg → texp, texp-seg | texp

(i) Left factor this grammar

(ii) Construct FIRST and FOLLOW sets for the non-terminals

(iii) Show that resultant grammar is LL(1).

(iv) Construct LL(1) parsing table for the resultant grammar

Soln For the given grammar,

Set of terminals T = {number, identifier, (,), ,}

Set of nonterminals V = { textp, atom, list, textp-seg}

Grammar is
textp → atom | list
atom → number | identifier
list → (textp-seg)
textp-seg → textp , textp-seg | textp

(i) consider the production rule

$$\underset{\underset{A}{\downarrow}}{\text{textp-seg}} \longrightarrow \underset{\underset{\alpha}{\downarrow}}{\text{textp}}\ \underset{\underset{\beta_1}{\downarrow}}{\underline{,\text{textp-seg}}} \mid \underset{\underset{\alpha}{\downarrow}}{\text{textp}} \quad \underset{\underset{\beta\ (\text{ie } \epsilon)}{\downarrow}}{}$$

if $A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots$, then $A \rightarrow \alpha A'$
$A' \rightarrow \beta_1 | \beta_2 | \dots$

∴ textp-seg → textp textp-seg'

textp-seg' → , textp-seg | ε

The production after removal of Left Factoring is

textp → atom | list

atom → number | identifier

list → (textp - Seg)

textp - seg → textp textp - Seg'

textp - seg' → , textp-seg / ε

Now Calculate FIRST and FOLLOW for each non-terminals.

First (textp) = { number, identifier, (}

First (atom) = { number, identifier }

First (list) = { (}

First (textp - seg) = { First (textp) }

= { number , identifier , (}

First (text - seg') = { , , ε }

Follow (textp) = { $, First (textp - seg') }

= { $, , , Follow (textp - seg) }

= { $, , ,) }

Follow (atom) = Follow (textp)

= { $, , ,) }

Follow (list) = Follow (textp)

= { $, , ,) }

Follow (textp - seg) = {), Follow (textp - seg') }

= {) }

Follow(textp-seg') = Follow(textp-seg) = {)}

Now check for LL(1).

$$\underset{\substack{\downarrow\\ A}}{\text{textp}} \longrightarrow \underset{\substack{\downarrow\\ \alpha}}{\text{atom}}$$

first (α) = first (atom) = {number, identifier}

$$\underset{\substack{\downarrow\\ A}}{\text{textp}} \rightarrow \underset{\substack{\downarrow\\ \alpha}}{\text{list}}$$

first (α) = first (list) = {(}

$$\underset{\substack{\downarrow\\ A}}{\text{atom}} \longrightarrow \underset{\substack{\downarrow\\ \alpha}}{\text{number}}$$

first (α) = first (number) = {number}

$$\underset{\substack{\downarrow\\ A}}{\text{atom}} \rightarrow \underset{\substack{\downarrow\\ \alpha}}{\text{identifier}}$$

first (α) = first(identifier) = {identifier}

$$\underset{\substack{\downarrow\\ A}}{\text{list}} \rightarrow (\underbrace{\text{textp-seg})}_{\substack{\downarrow\\ \alpha}}$$

first (α) = first ((text-seg)) = {(}

$$\underset{\substack{\downarrow\\ A}}{\text{textp-seg}} \rightarrow \underbrace{\text{textp textp-seg}'}_{\substack{\downarrow\\ \alpha}}$$

first(α) = first (textp textp-seg')

= First (textp)

= { number, identifier, (}

$$\underset{A}{\text{textp-seg}'} \longrightarrow \underbrace{,\ \text{textp-seg}}_{\alpha}$$

$\text{first}(\alpha) = \text{first}(,\ \text{textp-seg}) = \{ , \}$

$$\underset{A}{\text{textp-seg}'} \longrightarrow \underset{\alpha}{\epsilon}$$

Follow (textp-seg') = {) }

∴ The predictive parsing table is

	number	identifier	,	(	)	$
textp	textp→atom	textp→atom		textp→list		
atom	atom→number	atom→identifier				
list				list → (textp-seg)		
textp-seg	textp-seg→ textp textp-seg'	textp-seg→ textp textp-seg'		textp-seg→ textp textp-seg'		
textp-seg'			textp-seg'→ , textp-seg		textp-seg'→ ε	

As each cell in the above table contains unique entry, ∴ the given grammar is LL(1).

Q Consider the following grammar :

S' = S#

S → ABC

A → a | bbD

B → a | ε

C → b | ε

D → c | ε

Construct the first and follow sets for the grammar also design LL(1) parsing table for the grammar.

Solution :->

First (D) = {c , ε}

First (C) = {b, ε}

First (B) = {a, ε}

First (A) = {a, b}

First (B) = First (ABC)
= {a, b}

First (S') = First (S#)
= {a,b}

Follow (S') = {$}

Follow (S) = {#}

Follow (A) = First (BC)
= {a, b, #}

Follow (B) = First (C)
= {b, #}

Follow (C) = Follow (S) = {#}

Follow (D) = Follow (A)
= {a, b, #}

	a	b	c	#	$
S'	S' = S#	S' = S#			
S	S → ABC	S → ABC			
A	A → a	A → bbD			
B	B → a	B → ε		B → ε	
C		C → b		C → ε	
D	D → ε	D → ε	D → c	D → ε	