LR(1) Parsing Table for Ex#1

	a	b	$	S	A
I_0	S-I_3	S-I_4		I_1	I_2
I_1			R- P4		
I_2	S-I_6	S-I_7			I_5
I_3	S-I_3	S-I_4			I_8
I_4	R-P_3	R-P_3			
I_5			R-P_1		
I_6	S-I_6	S-I_7			I_9
I_7			R-P_3		
I_8	R-P_2	R-P_2			
I_9			R-P_2		

In Ex#1, what states have the SAME LR(0) items but DIFF. LR(1) items?

(i) I_4 & I_7

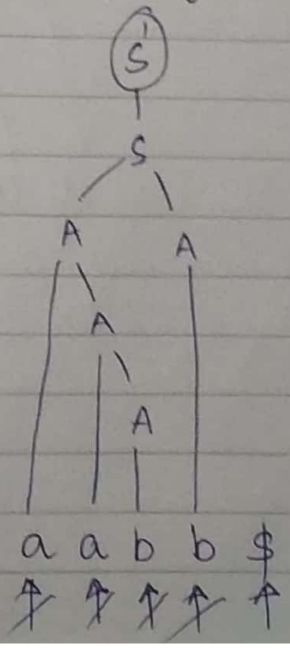
(ii) I_3 & I_6

(iii) I_8 & I_9.

~~I_8~~	R-P2-1
~~A~~	R-P2-1
~~I_8~~	R-P_2-0
~~A~~	R-P_2-0
~~I_4~~	R-P_3-0
~~b~~	R-P_3-0
~~I_3~~	R-P2-0
~~a~~	R-P2-0
~~I_3~~	R-P2-1
~~a~~	R-P2-1
I_0	

S'	Accept.
~~I_1~~	R-P_4-0
~~S~~	R-P_4-0
~~I_5~~	R-P1-0
~~A~~	R-P1-0
~~I_7~~	R-P_3-1
~~b~~	R-P3-1
~~I_2~~	R-P1-0
~~A~~	R-P1-0

Parsing algorithm.

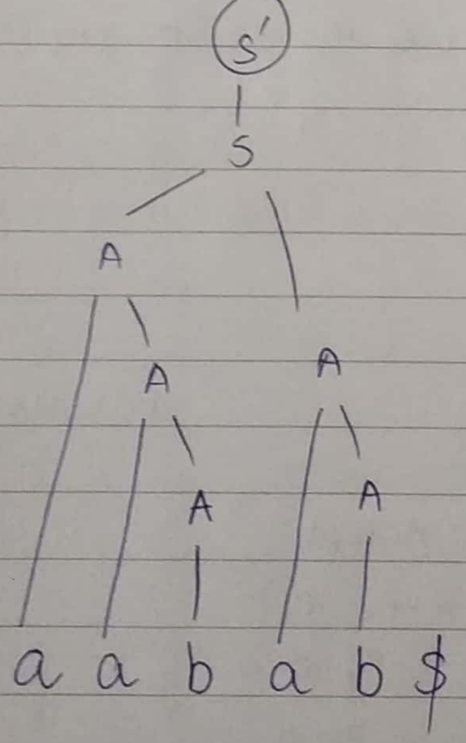


aabab$

~~I_8~~	R-P_2-1
~~A~~	R-P_2-1
~~I_8~~	R-P_2-0
~~A~~	R-P_2-0
~~I_4~~	R-P_3-0
~~b~~	R-P_3-0
~~I_3~~	R-P_2-0
~~a~~	R-P_2-0
~~I_3~~	R-P_2-1
~~a~~	R-P_2-1
I_0	

~~S~~	R-P_4-0
~~I_5~~	R-P_1-0
~~A~~	R-P_1-0
~~I_9~~	R-P_2-2
~~A~~	R-P_2-2
~~I_7~~	R-P_3-1
~~b~~	R-P_3-1
~~I_6~~	R-P_2-2
~~a~~	R-P_2-2.
~~I_2~~	R-P_1-0
~~A~~	R-P_1-0 .

S'	Accept
~~I_1~~	R-P_4-0



Ex#2 P1: S → ABd P4: B → d

P2: A → Bb P5: B → eB

P3: A → c P6: S' → S.

S' → .S , {$}

I_0 ≡ S → . ABd , ~~{$}~~ {$}

A → . Bb , ~~{$}~~ {d,e}.

A → .c , {d,e}

B → .d , ~~{d,e}~~ {b}.

B → .eB , {b}

~~S' → .S , {b}~~ {$}

$I_0 \xrightarrow{S}$ (S' → S.) , {$} I_1

$\xrightarrow{A}$ (S' → A.Bd) , {$}

(B → .d) , {d} I_2

(B → .eB) , {d}

$\xrightarrow{B}$ (A → B.b) , {d,e} I_3

$\xrightarrow{c}$ (A → c.) , {d,e} I_4

$\xrightarrow{d}$ (B → d.) , {d} I_5

$\xrightarrow{e}$ (B → e.B) , {b}

(B → .d) , {b} I_6

(B → .eB) , {b}

$I_2 \xrightarrow{B} (S \to AB.d), \{\$\}$ I_7

$\xrightarrow{d} (B \to d.), \{d\}$ I_8

$\xrightarrow{e} (B \to e.B), \{d\}$
$(B \to .d), \{d\}$ I_9
$(B \to .eB), \{d\}$

$I_3 \xrightarrow{b} (A \to Bb.), \{d, e\}$ I_{10}

$I_6 \xrightarrow{B} (B \to eB.), \{b\}$ I_{11}

$\xrightarrow{d} (B \to d.), \{b\}$ I_5

$\xrightarrow{e}$ I_6

$I_7 \xrightarrow{d} (S \to ABd.), \{\$\}$ I_{12}

$I_9 \xrightarrow{B} (B \to eB.), \{d\}$ I_{13}

$\xrightarrow{d}$ I_8

$\xrightarrow{e}$ I_9

Check the string dbedd

Parsing Table

	b	c	d	e	$	A	B
I_0							
I_1							
I_2							
I_3							
I_4							
I_5							
I_6							
I_7							
I_8							
I_9							
I_{10}							
I_{11}							
I_{12}							
I_{12}							

Ex#3

E→ E+T

E→ E-T

E→ T

T→ T*F

T→ T÷F

T→ F

F→ (E)

F→ id

→ LR(1)

LALR(1) Parsing Technique

↓ look ↳ ahead

Consider the same grammar for discussion.

① $S \rightarrow AA$ ④ $S' \rightarrow S$

② $A \rightarrow aA$

③ $A \rightarrow b$

The CLR(1) parsing table was:-

	a	b	$	S	A
I_0	S-I_{36}	S-I_{47}		I_1	I_2
I_1			R-P_4		
I_2	S-I_{36}	S-I_{47}			I_5
I_{36}	S-I_{36}	S-I_{47}			I_{89}
I_{47}	R-P_3	R-P_3	R-P_3.		
I_5			R-P_1		
remove → I_{36} I_6	S-I_{36}	S-I_{47}			I_{89}
remove → I_{47} I_7			R-P_3		
I_{89}	R-P_2 ~~R-P_2~~	R-P_2	~~R-P_2~~		
remove → I_{89} I_9			R-P_2		

merge & remove

I_3 & I_6 ⇒ I_{36} Have the same LR(0) items

I_4 & I_7 ⇒ I_{47} but diff. LA symbols.

I_8 & I_9 ⇒ I_{89}.

↳ merged

merge identicals rows in table.

CLR(1) has more states & sparse Parsing Table

more blanks in PT ⇒ easy for parser to find errors.

SLR(1) → n_1 states
CLR(1) → n_2 states

when lookahead is same as follow, $n_1 \equiv n_2$.
eg:→ $E \rightarrow E+T$, $E \rightarrow T$, $T \rightarrow id$., $E' \rightarrow E$.

SYNTAX DIRECTED TRANSLATION

① We use semantic actions with productions.

② when a ~~pro~~reduction ~~is~~ of a production takes place than action is applied.

Consider the CFG:-

P1 $E \to E+T$	P4 $T \to F$
P2 $E \to T$	P5 $F \to id$
P3 $T \to T*F$	P6 $E' \to E$

① Evaluate 4+3*2

② Get me Reverse Polish Notation (RPN) ⇒ 432*+

③ Just generate the intermediate code.

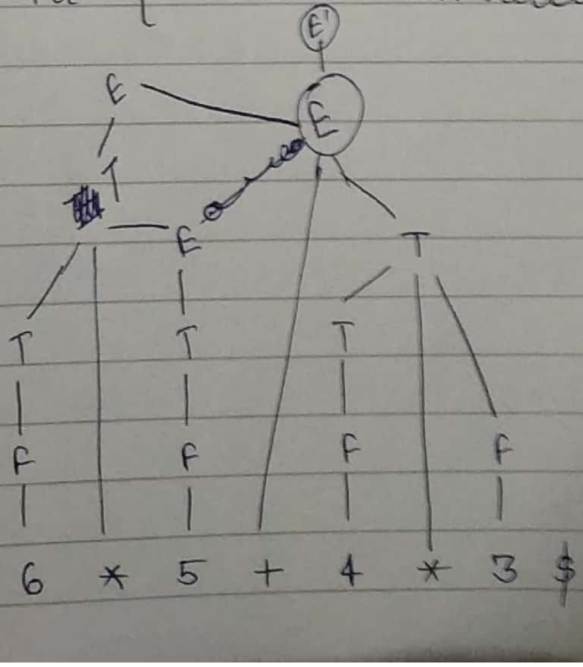
P1 $E \to E+T$ { E.value = E.value + T.value } { Printf("+") } ✓

P2 $E \to T$ { E.value = T.value }.

P3 $T \to T*F$ { T.value = T.value * F.value } { Printf("*") } ✓

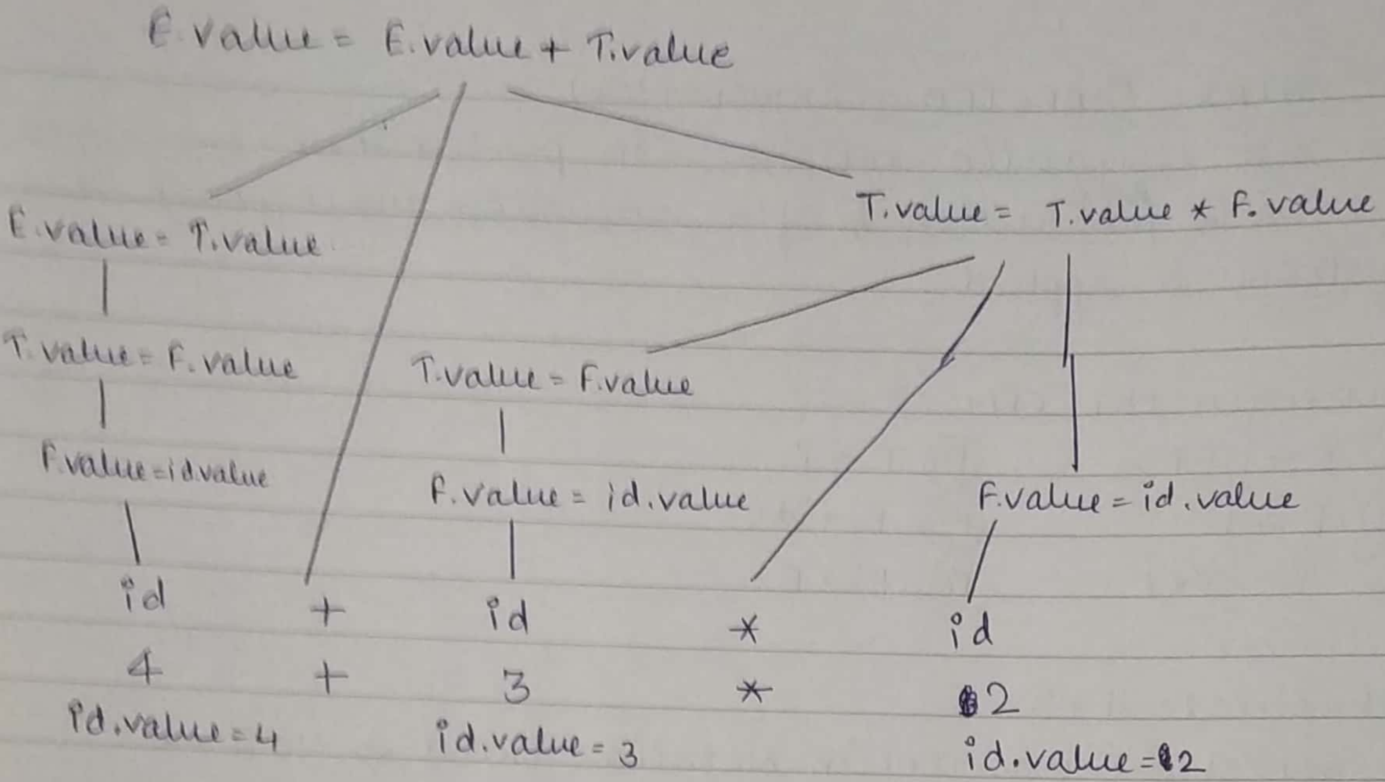
P4 $T \to F$ { T.value = F.value }.

P5 $F \to id$ { F.value = id.value } { printf(id.value) } ✓



$\underset{①}{\underline{65*}}\ \underset{②}{\underline{43*}}+$

③


E.value = E.value + T.value
E.value = T.value
T.value = T.value * F.value
T.value = F.value
T.value = F.value
F.value = id.value
F.value = id.value
F.value = id.value
id
+
id
*
id
4
+
3
*
2
id.value = 4
id.value = 3
id.value = 2

Semantically incorrect → mixing float with integers, going into ∞ loop, dividing by 0.

Your code.

```
main{
   float r;
   float h;
   float v;
   float a;

  read r;
  read h;
  if (r gt 0) and(h gt 0)
  then goto 10;
  goto 20;
10: v := 22/7 * r ↑2 x h;
  a := 2*22/7 * r(r+h);
  write(v);
  write (a);
  stop;
20: stop;
 }
```

↳ compute only once by storing inside a variable. (we need a placeholder)

whenever redn takes place, a line of intermediate code is generated.

Intermediate code.

```
FLOAT r
FLOAT h
FLOAT v
FLOAT a

r = READ()
h = READ()
t1 = r gt 0        temporary variable.
t2 = h gt 0
t3 = t1 * t2

IF t3
THEN
GOTO 10
GOTO 20
t4 = r tpo 2
t5 = 22/7
t6 = t5 * t4.
t7 = t6 * h
v = t7.

t8 = r+h
t9 = 2*22
t10 = t9/7.
t11 = t10* r
t12 = t11 * t8.
a = t12.
WRITE v
WRITE a
STOP
10: STOP.
```