PRINCIPLES OF COMPILER DESIGN

ASSESSMENT-2

SRUTHI S(19MID0053)

3)	1) Construct DAG for a) (a-b)+c*(d/e) b) x=x+x*y c) (x+5)*(x+5+y) d) a=(a+a)+a(a+a+a)+a 2) Check the given grammer is LL(1) or NOT? S→(A) | 0 A→SB B→,SB|ε and also parse the grammar (0,(0,0))

24)	1) Differentiate between final states in a NFA and a DFA 2) Table:

Remove left recursion	Remove left Factoring
A→Aα |β	S→iEtS|iEtSeS|a E→b
S→Aa|b A→Ac|Sd|ε	Stmt→if expr then Stmt else Stmt|ifexpr then Stmt
S→aBDh S→Bb|C D→EF E→g|ε F→f|ε	S→aSb|aTc T→dTU|ε U→f
S→SA|SB|a|b|c	

1)	1) Check the given Grammar is ambiguous/Unambiguous S→S(S)S|ε 2) Prove the given grammar is LR(1),LALR(1),Not SLR(1). S→Aa|bAc|dc|bda A→d

2)	1) Find the following grammar is LL(1) ,LR(1) S→AaAb|BbBa 2) Check whether the following grammar is LR(0), SLR(1),LALR and LR(1) S→AaAb|BbBa A→ε B→ε

40)	1) What are the issues of the Lexical analyser? 2) Eliminate left recursion, perform left factoring and find: FIRST & FOLLOW E→E+T|T T→id|id [ ] |id [X] X→E,E|E 3) Check the given grammar is LL(1) or NOT? S→(A) | 0 A→SB B→,SB|ε and also parse the grammar (0,(0,0))

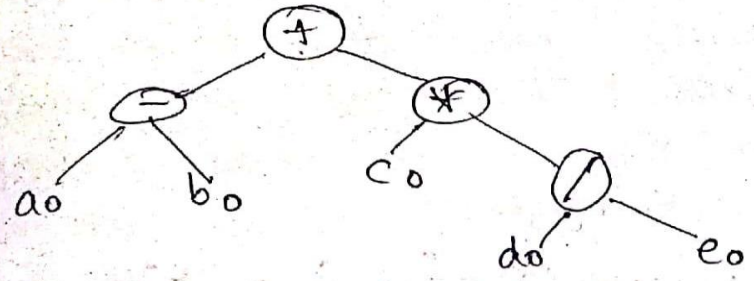
35)	1) Consider the grammar S→(L)|a L→L,S|S f) What are the terminal, non-terminal and start symbol? g) Find parse tree for the following sentences (iv) (a,a) (v) (a,(a,a))

13)	1) With the help of neat block diagram explain various phases of compiler ,Also write down the output of each phase for expression a:=b+c*50 2) Construct LR(1). S→x|Av B→ε|z A→Bx

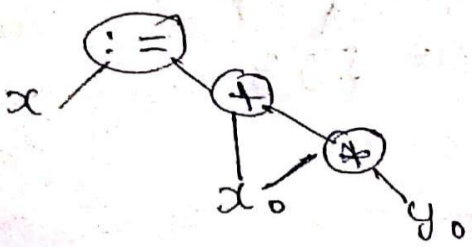
SRUTHI.S
19MID0053

(3.) => i.) Construct DAG for

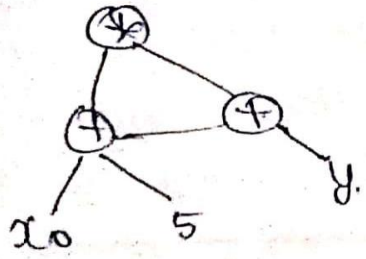
a.) (a-b) + c * (d / e)



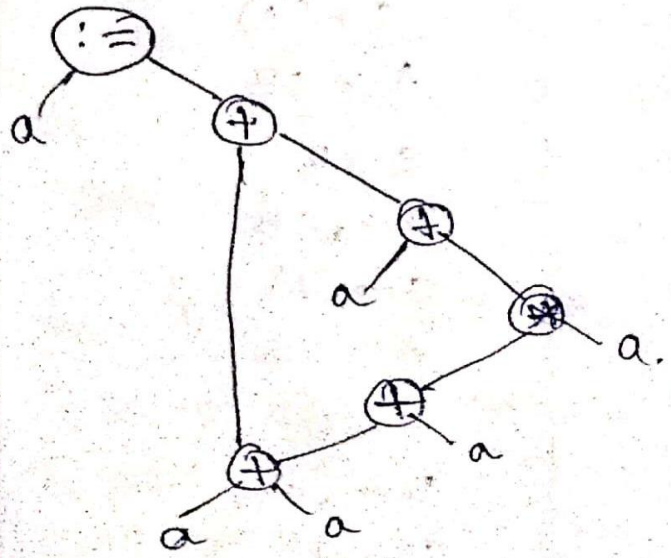
b.) x := x + x * y



c.) (x+5) * (x+5+y)



d.) a := (a+a) + a (a+a+a) + a.



SRUTHI.S
19MIDC03

③ ⇒ Check the given grammer is LL(1) or NOT? & also parse the grammar (0,(0,0)).

$S \rightarrow (A) \mid 0$

$A \rightarrow SB$

$B \rightarrow ,SB \mid \varepsilon$

No left recursion nor left factoring

FIRST (S) = {(, 0}

FIRST (A) = {(, 0}

FIRST (B) = {,}

FOLLOW (S) = {,,), 0, $}

FOLLOW (A) = {)}

FOLLOW (B) = {)}

	S	A	B
(	S→(A)	A→SB	
)			B→ε
,			B→,SB
0	S→0	A→SB	
$			

Parsing table

stack	Input	production	
S$	(0, (0,0))		
(A) $	(0, (0,0))	S→(A)	
SB)$	0, (0,0))	A→SB	
0B)$	0, (0,0))	S→0	
,SB)$	, (0,0))	B→,SB	
(A)B)$	(0,0))	S→(A)	Stack→ empty. No String→ accepted.
SB)B)$	0,0))	A→SB	
0B)B)$	0,0))	S→0	
,SB)B)$	,0))	B→,SB	
0B)B)$	0))	S→0	
)B)$	))	B→ε	
)$	)	B→ε	
$	$		

(24) → i) Differentiate between final states in a NFA & a DFA.

DFA	NFA
* DFA rejects the string in case it terminates in a state that is different from the accepting state.	* NFA rejects the string in the event of all branches dying (or) refusing the string
* In DFA, there is only one path for specific input from current state to next state. It cannot accept null move.	* A two state NFA with both states final where the minimal equivalent DFA has 4 states.
* DFA can contain multiple final states.	* NFA can also contain multiple final states.
* Time needed for executing an input string is less.	* Time needed for executing an input string is more.
* It requires more space	* It requires less space.
* Dead state may be required	* Dead state not required
* All DFA are NFA.	* Not all NFA are DFA.

(24) → ii) Remove left recursion:

* S → Aa | b
  A → Ac | Sd | ε

$A \to A\alpha \mid \beta$
$A \to \beta A'$
$A' \to \alpha A' \mid \varepsilon$

Soln:

S → Aa | b ⇒ S → Sda | b | A → Ac | Sd | ε
A → Aα | β | A → Ac | Aad | ε | bd.

S → Sda | b (left recursion removed)

$S \to bS'$
$S' \to \alpha aS' \mid \varepsilon$ → (1)

SRUTHI.S
19MID0053

$A \to Ac \mid Aad \mid \varepsilon \mid bd.$

$A \to Ac \mid Aad \mid bd \mid \varepsilon$ (Left recursion removal)

$$\left.\begin{array}{l} A \to bdA' \mid \varepsilon A' \\ A' \to cA' \mid adA' \mid \varepsilon \end{array}\right\} \to (3)$$

(1) + (2) $\to$

$S \to bS'$
$S' \to daS' \mid \varepsilon$
$A \to bdA' \mid \varepsilon A'$
$A' \to cA' \mid \varepsilon \mid adA'$

* $S \to aBDh$
  $S \to Bb \mid c$
  $D \to EF$
  $E \to g \mid \varepsilon$
  $F \to f \mid \varepsilon$

$\Rightarrow$ No left recursion in this example.

* $S \to SA \mid SB \mid a \mid b \mid c$

Sol:

$S \to aS' \mid bS' \mid c$
$S' \to AS' \mid BS' \mid \varepsilon.$

Remove left factoring:

$$\left.\begin{array}{l} A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \gamma \\ A \to \alpha A' \mid \gamma \\ A' \to \beta_1 \mid \beta_2 \mid \ldots \mid \varepsilon \end{array}\right.$$

* $S \to iEtS \mid iEtSeS \mid a$
  $E \to b.$

Sol:

$S \to iEtSS' \mid a$
$S' \to \varepsilon \mid eS$
$E \to b.$

* stmt $\to$ if expr then stmt | else stmt | if expr then stmt

(α = if expr then stmt; β_1 = else stmt; γ = if expr then stmt)

Sol:

stmt $\to$ if expr then stmt stmt'
stmt' $\to \varepsilon$ | else stmt.

* $S \to aSb \mid aTc$
  $T \to \alpha TU \mid \varepsilon$
  $U \to f.$

Sol:

$S \to aS'$
$S' \to \varepsilon \mid Sb \mid Tc$
$T \to \alpha TU \mid \varepsilon$

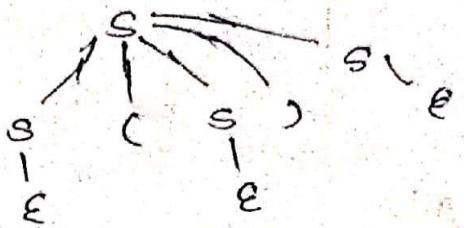
$U \to f.$

(1.) → i) Check the given grammar is ambiguous / unambiguous

$S \to S(S)S \mid \varepsilon$

Sol:- We take one string as ( )

Parse-tree :-



String : ( ) , we draw one parse tree for one string.

So, this is unambiguous grammar.

ii) prove that given grammer is LR(1), LALR(1) Not SLR(1)

$S \to Aa \mid bAc \mid dc \mid bda$

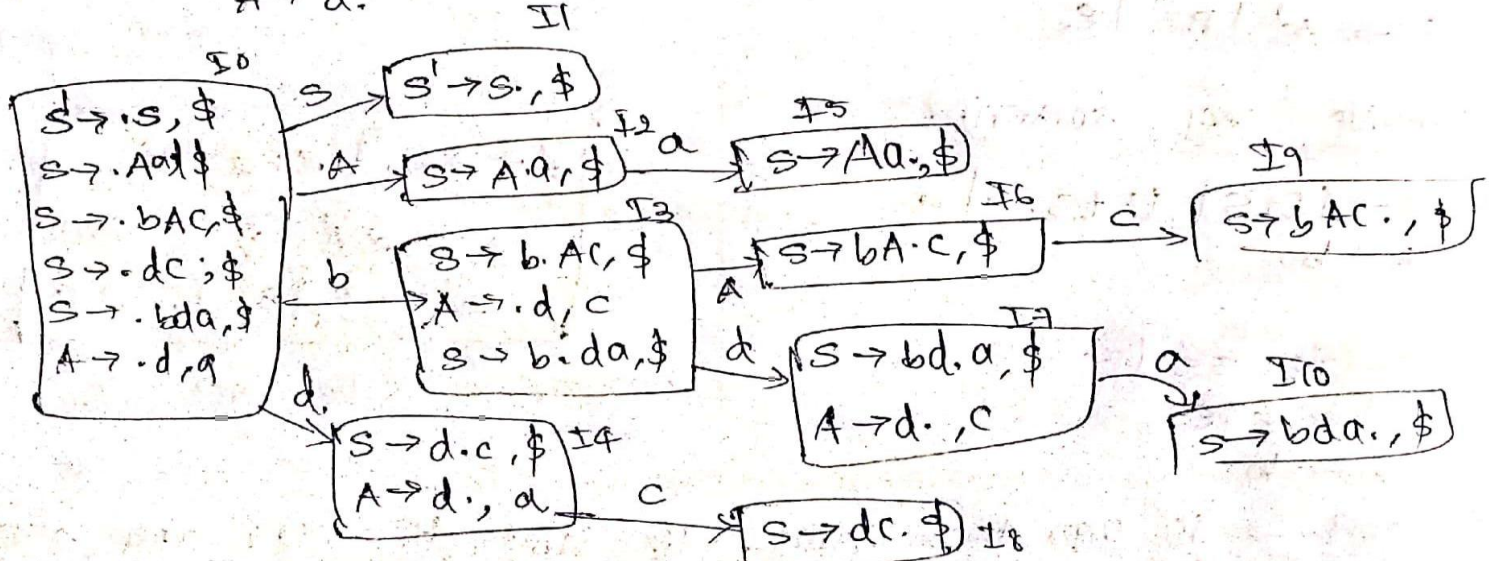
$A \to d.$

Sol:-

$S' \to S$

$S \to Aa \mid bAc \mid dc \mid bda$

$A \to d.$



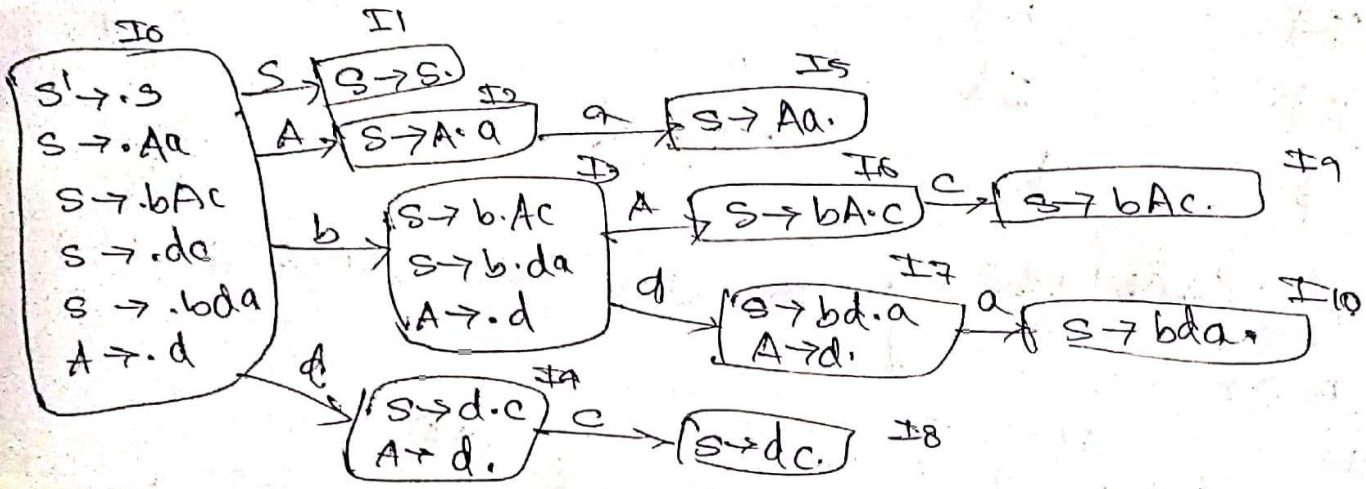
Parsing table :-

State	Action					Goto	
	a	b	c	d	$	S	A
0		S_3				I1	I2
1					accept		
2	S_5						
3				S_7			I6
4	r_4		S_8				
5							

6			S_9				
7	S_{10}		r_5				
8					r_3		
9					r_2		
10					r_4		

For LALR(1) parsing we merge the common items with common first components. In this problem no merging occurs. So this is same for LALR

For SLR(1) parser:



Consider I7, symbol a, Follow(A) = {a, c}, it makes shift reduce conflict. So it is not SLR(1) parser.

(2.) ⇒ i.e. check the grammar is LL(1) parser.

S → Aa Ab | BbBa
A → ε
B → ε

FIRST(S) = {a, b}
FIRST(A) = {ε}
FIRST(B) = {ε}

FOLLOW(S) = {$}
FOLLOW(A) = {a, b}
FOLLOW(B) = {b, a}

SRUTHI.S
19MIDCO53

parsing table:

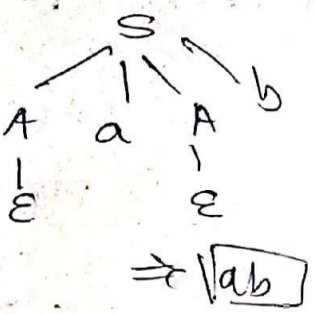
	a	b	$
S	S→AaAb	S→BbBa	
A	A→ε	A→ε	
B	B→ε	B→ε	

No Ambiguous grammar then LL(1) parser.

Stack: ab$

stack	input	action
S$	ab$	S→AaAb
AaAb$	ab$	A→ε
aAb$	ab$	—
Ab$	b$	A→ε
b$	b$	—
$	$	Accepted

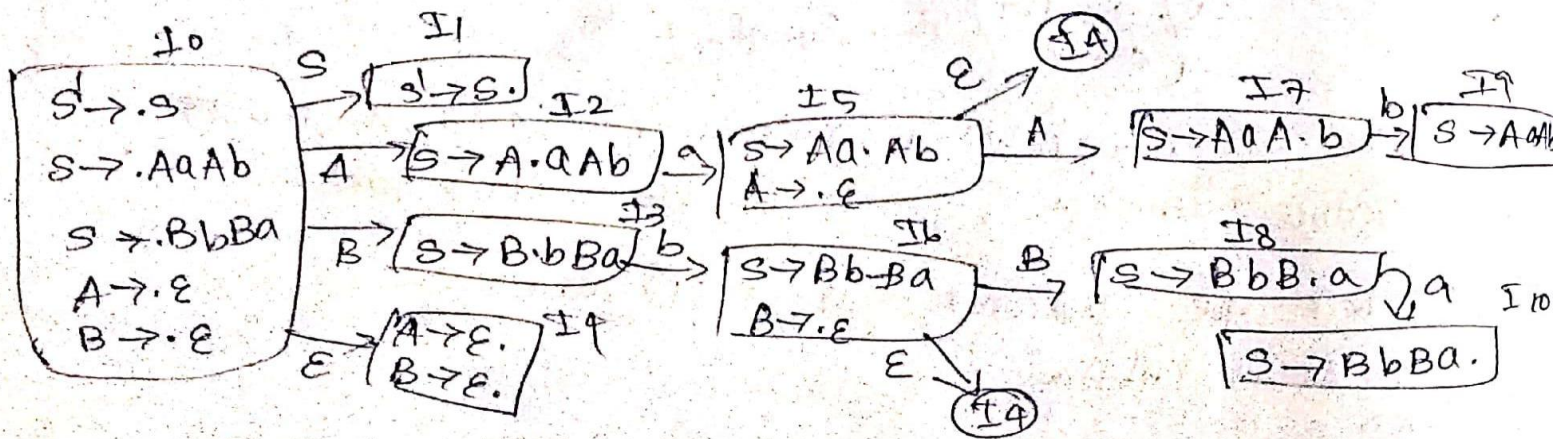
tree:



11.b

S→AaAb | BbBa
A→ε
B→ε

} ∴ LR(0), SLR(1), LALR & LR(1)



In SLR parser, when we reduce "ε", we can't decide reduce to A or B, this is reduce-reduce conflict. S is not SLR(1) parser.

SRUTHI S
19MID0053

(10) → i.e What are the issues of the lexical analysis?

We do separate the work of lexical analysis & syntax analysis for the following reasons.

→ simplicity of design ⟹ A parser containing the rules for comments & white space is more complex to make than a parser that can assume that comments & white space has been removed.

→ Improved compiler efficiency ⟹ Reading source code & classifying it in token is time consuming task, when we separate from parser, it allows us to use specialized technique for lexer, which can speed up scanning

→ Higher probability ⟶ Input device specific peculiarities are restricted to lexer.

→ Lexical errors ⟶ A character sequence which is not possible to scan into any valid token is a lexical error. It's hard for lexical analyser without the aid of other components, that there is a source code error.

Eg ⟹ If the statement "if" is encountered for the first time in c program, it can not tell whether it is mispelling of "if" statement or a undeclared literal. probably the parsers in this area case will be able to handle this.

Eg ⟹ Also error handling is very localized with respect to handle this ⇒ While (x=0) do generates no lexical error in PASCAL.

→ Handling lexical errors ⟹ panic mode recovery
possible error recovery actions.

Panic mode recovery ⇒ Deletes successive characters from the remaining input until the analyzer can find a well formed token. May confuse parser by creating syntactic errors

Possible error recovery actions ⇒ Deleting extra irrelevant characters, Inserting missing input character, replacing an incorrect character by a correct character, Transposing two adjacent characters

SRUTHI'S 19MID0053

→ Input buffering ⇒ The amount of time taken to process characters of a large source program. Lexical analyzer may need to look at least a character ahead to make a token decision.

→ Sentinals ⇒ During buffering for each character
→ check the end of buffer
→ determine what character is read.

④ ⇒ 2.b Eliminating left recursion, left factoring, finding first & follow.

$E \to E+T \mid T$
$T \to id \mid id[] \mid id[x]$
$x \to \epsilon, E \mid E$

Elimination → left recursion

$E \to TE'$
$E' \to +TE' \mid \epsilon$
$T' \to [] \mid [x] \mid \epsilon$
$x \to \epsilon, E \mid E$

NONES	FIRST	FOLLOW
E	{id}	{$,)}
E'	{+, ε}	{+, $}
T	{id}	{+, $, ,}
T'	{[, ε}	{]}
x	{id}	{$}.

SRUTHI.S
19MIDC03

③ ⇒ Check the given grammer is LL(1) or NOT? & also parse the grammar (0,(0,0)).

$S \rightarrow (A) \mid 0$

$A \rightarrow SB$

$B \rightarrow ,SB \mid \varepsilon$

No left recursion nor left factoring

FIRST (S) = {(, 0}

FIRST (A) = {(, 0}

FIRST (B) = {,}

FOLLOW (S) = {,,), 0, $}

FOLLOW (A) = {)}

FOLLOW (B) = {)}

	S	A	B
(	S→(A)	A→SB	
)			B→ε
,			B→,SB
0	S→0	A→SB	
$			

Parsing table

stack	Input	production
S$	(0, (0,0))	
(A) $	(0, (0,0))	S→(A)
SB)$	0, (0,0))	A→SB
0B)$	0, (0,0))	S→0
,SB)$	, (0,0))	B→,SB
(A)B)$	(0,0))	S→(A)
SB)B)$	0,0))	A→SB
0B)B)$	0,0))	S→0
,SB)B)$	,0))	B→,SB
0B)B)$	0))	S→0
)B)$	))	B→ε
)$	)	B→ε
$	$	

Stack → empty. No string → accepted.

SRUTHI. S
19MIDOOS3

(35) ⇒ i) Consider the grammar

$S \rightarrow (L) \mid a$

$L \rightarrow L, S \mid S$.

ii) What are the terminal, non-terminal & start symbol :

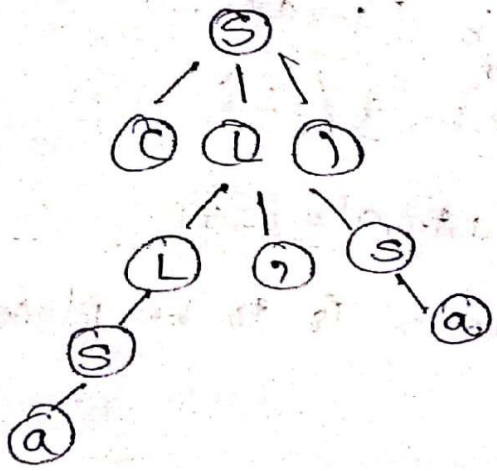
Terminal → { (,), ,, a }

Non terminal → { S, L }

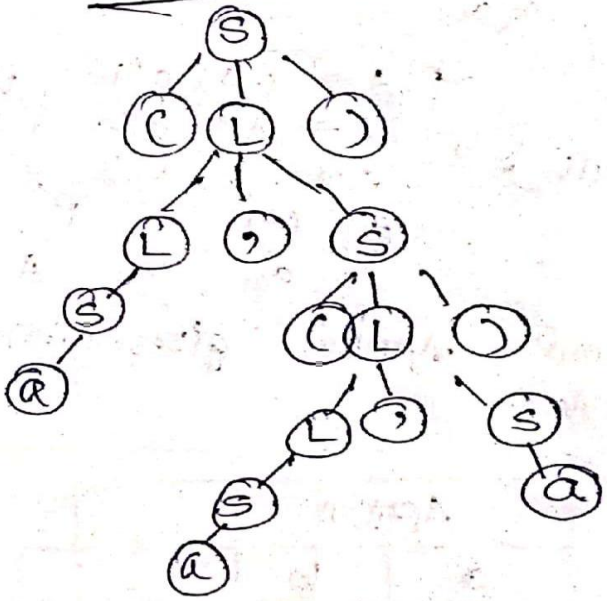
Start symbol → { S }

iii) Find parse tree for the following :-

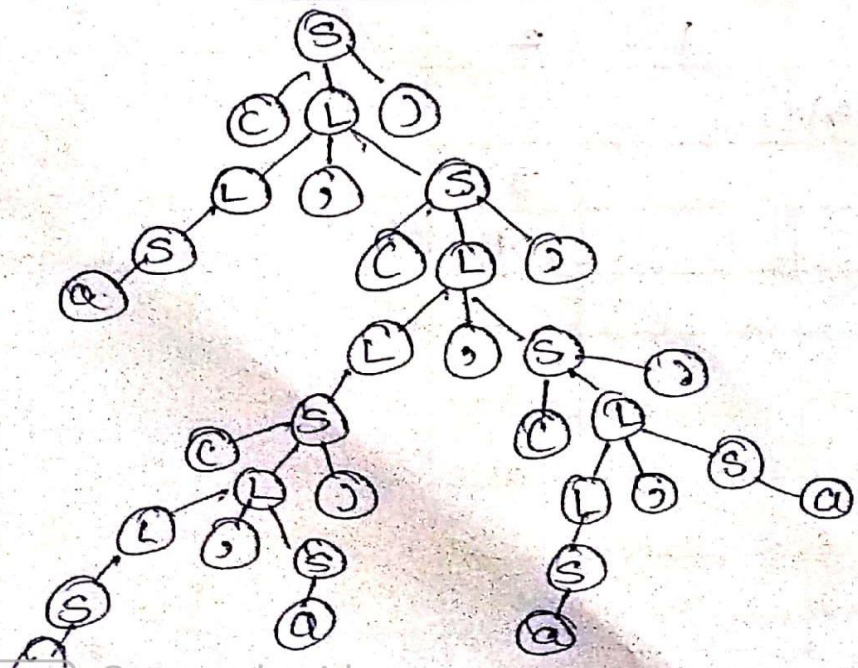
i) (a a)



ii) (a, (a, a))

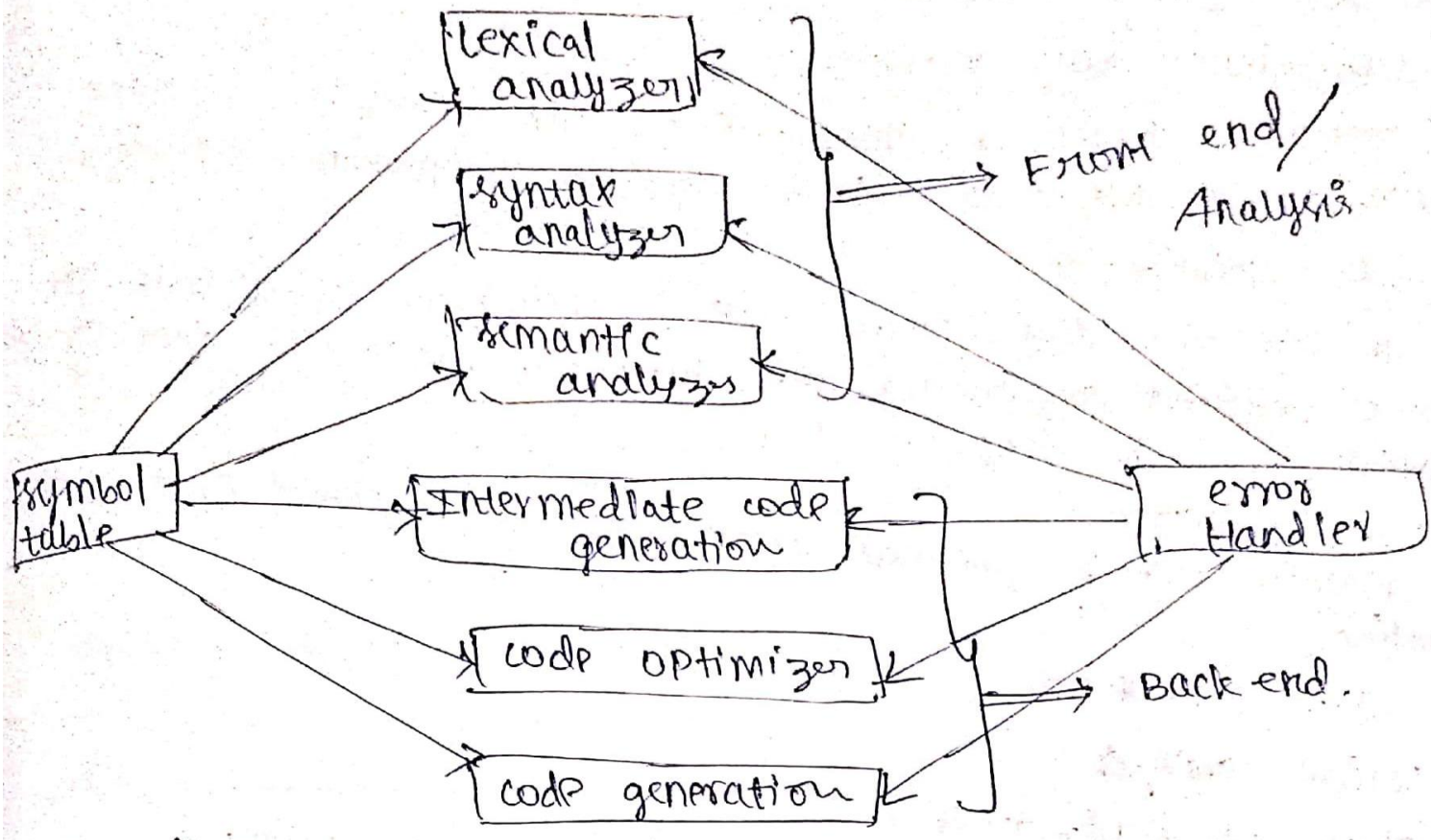


iii) (a, ((a, a), (a, a)))



(13) ⇒ 1.) Different phase of the compiler & what are the compiler parts:

SRUTHI.S
19MID0053



Compilation Front end :-

Source code breaks into basic pieces while storing information in the symbol table. program can be modified in front end. Also known as Analysis.

1.) Lexical Analysis :-

Input text is read character by character. (i.e) text is individually considered. token is collected & given as output. It is the collection of keywords, digits.

2.) syntax analysis:

The output of lexical analysis is created in a tree like structure.

3.) semantic analysis:-

Semantic analysis does type checking, variable declaration. This phase is in the process of adding data type is in the syntax/ parse tree.

SRUTHI.S
19MIDOO53

Compilation Back End :-

Code optimization offers efficiency of code generat with least use of resources. program cannot be modified by user.

A.b Intermediate code generation :-

compiler generate a intermediate code which is machine independent. The output of LA. is stored in temporary variable.

5. Code optimization :-

It aim to reduce process timings. optimizing weak code. It produces efficient programming code. Removing statement that are not modified from the loop.

b.f Code generation :-

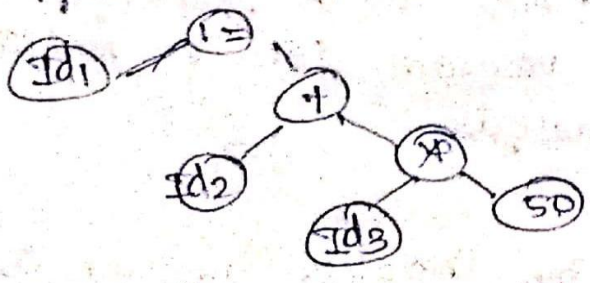
Machine code is generated. Instructions are chosen from each operation.

$a := b + c * 50$

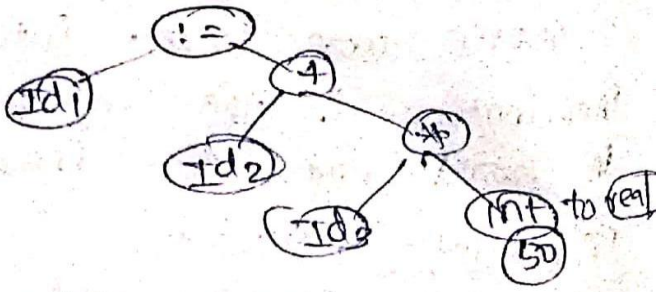
1. Lexical analyser :

$Id_1 := Id_2 + Id_3 * 50$

2.b syntax analysis :



3. Semantic analysis :-



4. Intermediate code generator :

```
temp1 := int to real (50)
temp2 := Id3 + temp1
temp3 := Id2 + temp2
Id1 := temp3
```

(5) Code generator optimizer :

```
temp := Id3 * 50.0
Id1 := Id2 + temp1.
```

(6) Code generator :-

```
MOVF   R2, Id3
MULF   R2, #50.0
MOVF   R1, Id2
ADDF   R1, R2
MOVF   Id1, R1
```

SRUTHI.S
19MID0053

(3) (a) Construct LR(1)

$S \to x \mid Ay$

$B \to \varepsilon \mid z$

$A \to Bx$

Augmented grammar

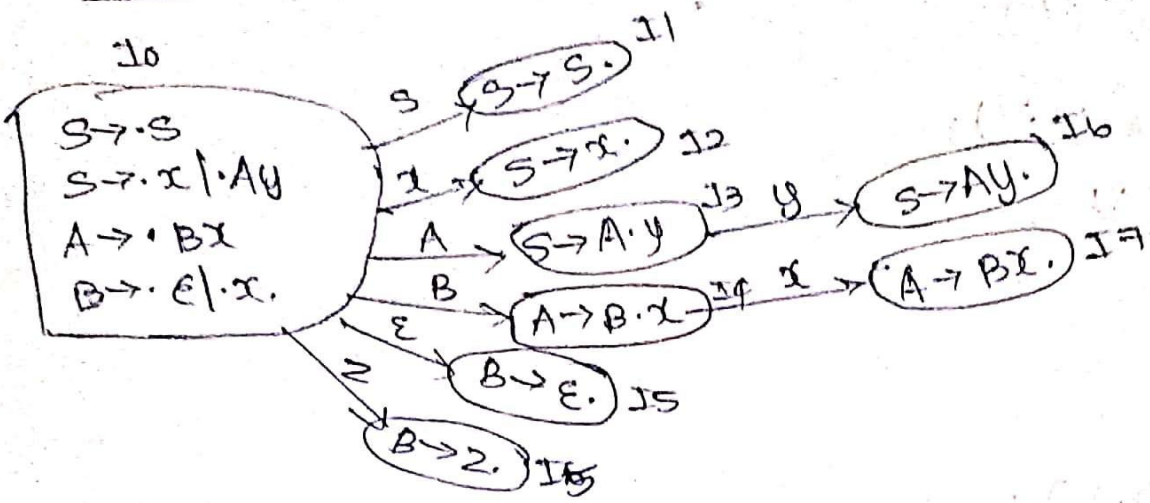
$S' \to S$

$S \to x \mid Ay$

$B \to \varepsilon \mid z$

$A \to Bx$

Canonical form :



Parsing table :

State	Action				Goto		
	x	y	z	$	A	B	S
0	S_2		S_5		3	4	1
1				Accept			
2				r_1			
3		S_6					
4	S_7						
5				r_3			
6				r_2			
7				r_4			