Compiler Design

PAGE NO.
DATE / /

Compiler :- Executes whole code in one time
↳ C, C++, ... → Languages for software development

Interpetor :- Line by line execution
↳ Small Code use → Web based Language

→ Compiler execute whole program at a time.
↳ Large Code Use
ex. C, C++, Java, Pascal, COBOL, BASIC
↳ Programming Language

* **Intrepretor example** :- Ruby, Python, ASP.NET, Perl
↓
Scripting language

COBOL → Common Business oriented Language

BASIC → Beginner All Purpose symbolic Instruction Code.

* **Markup Language** → HTML, XML → No use they just help ~~in Enterprise~~ with scripting language

PAGE NO.
DATE / /

* **Grammar** :- 4 Tuple

(V, T, P, S)

- V → Non Terminal
- T → Terminal
- P → Productional rule
- S → Starting Symbol

(1) **Terminal** :- Small Letter
(a-z) (0-9)

(2) **Non Terminal** :-
(A-Z)

(3) **Production Rule** :- $S \rightarrow abc$ (P.R.)

(4) **Starting Symbol** :- $S \rightarrow abc$
S (S.S.)

Regular Grammar (R.G.) → Finite length.
Least Powerful
↳ 4 Tuple (V, T, P, S)
↳ Two form → Left linear Regular Grammar **
Right linear Regular Grammar **

(1) **Left linear Regula Grammar** :-
Non Terminal on left

$S \rightarrow Aaa$
$S \rightarrow Bba$
$S \rightarrow Xaaa$

PAGE NO.
DATE / /

(2) **Right Linear Regular Grammar :-**

$S \longrightarrow aa\,A\,\boxed{A}$
$S \longrightarrow aB\,a\,\boxed{B}$
$S \longrightarrow aX\,aa\,X\,\boxed{B}$

★ $S \longrightarrow \boxed{A}aa\boxed{A}$
$S \longrightarrow \boxed{B}\boxed{B}$
$S \longrightarrow \boxed{Z}X\,A\,\boxed{B}$
$S \longrightarrow \boxed{A}aa\,X\,a\,\boxed{B}$
→ Left, Right both

Q. N.T.
$S \longrightarrow aa$ ✗
$S \longrightarrow \epsilon$ ✗
$S \longrightarrow aaaa$ ✗
} R.G. ✗

* If it is R.G. then it will be C.F.G., C.S.G., Unrestricted Grammar

Q. (i) $S \longrightarrow aA\boxed{A}$
$S \longrightarrow aab$
$A \longrightarrow a$
→ Right linear grammar
R.G. ✓

(ii) $S \longrightarrow aaAa$
$S \longrightarrow \boxed{A}\boxed{A}$
$S \longrightarrow B$
$S \longrightarrow a$
$B \longrightarrow b$
} Both linear grammar
Regular Grammar

(iii) $S \longrightarrow aAa$
$S \longrightarrow aBa$
R.G. ×

Q. (i) $A \longrightarrow aaAaA$
$B \longrightarrow bb$
R.G. ✓ (Right Linear)

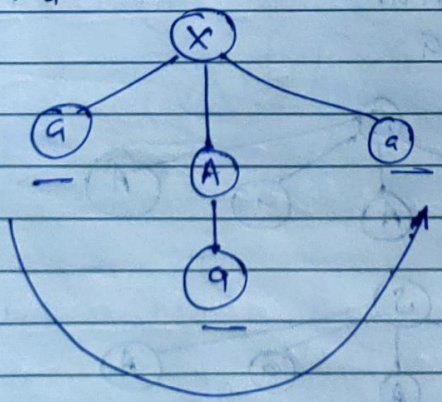
(ii) $B \longrightarrow b$
$A \longrightarrow aAa$
$A \longrightarrow aa$
R.G. ×

(iii) $A \longrightarrow aAA$
$B \longrightarrow BB\epsilon$ (ε → Empty)
Both Linear
R.G. ✓

(1) Derivation Tree :-

Starting Node → Root Node
Vertex → Non Terminal
Leaf Node (leaves) → Terminal

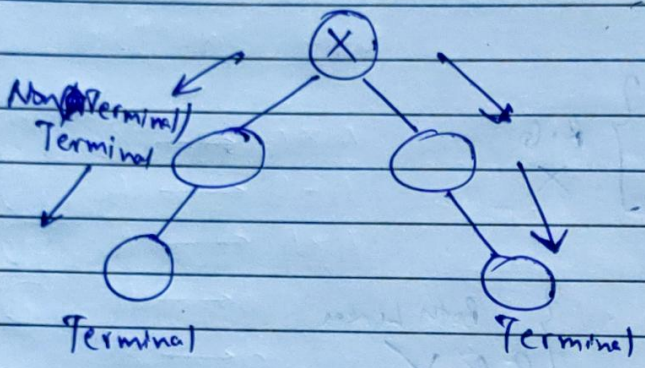
$X \longrightarrow aAa$
$A \longrightarrow a$



⇒ aaa

(i) Top-down derivation tree -

(i) Start with Root Node.
(ii) Goes down TOward leaf Node (leaves)



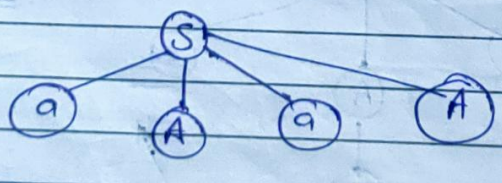
(ii) Bottom-up derivation tree :-

(i) Start with leaf Node (leaves Node)
(ii) Towards Root Node.

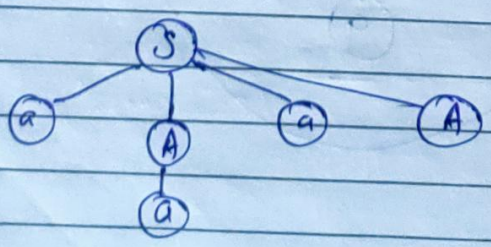
(i) left linear derivation tree :-

Root Node ← S → aAaA
A → a

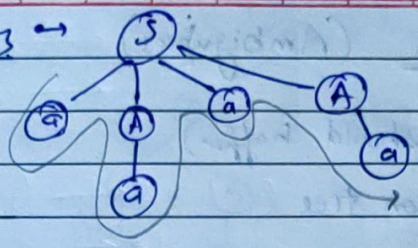
Step 1 →



Step 2 →



Step 3 →



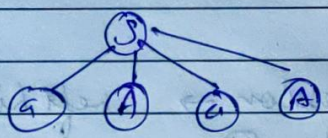
S → aaaa

(ii) Right linear derivation tree :- (Right to left)

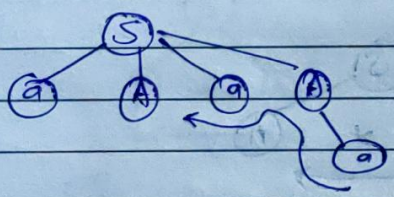
S → aA aA

~~A~~ → a

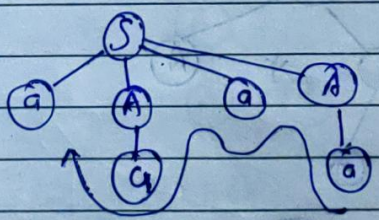
Step 1 :-



Step 2 :-



Step 3 :-



S → aaaa

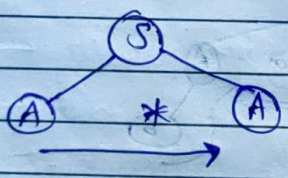
Ambiguity Grammar (Ambiguous)

(Any one should happen)

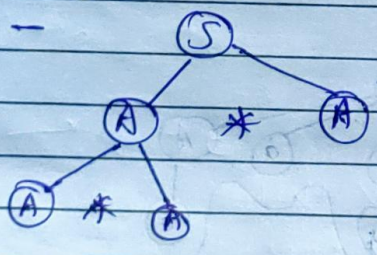
(i) Left linear derivation tree
(ii) Right " " "
(iii) Left and Right " "

Q. (i) $S \longrightarrow A * A$
(ii) $A \longrightarrow A + A$
(iii) $A \longrightarrow A * A$
(iv) $A \longrightarrow a$
(v) $B \longrightarrow b$

Take any expression → left linear →
we take (i)

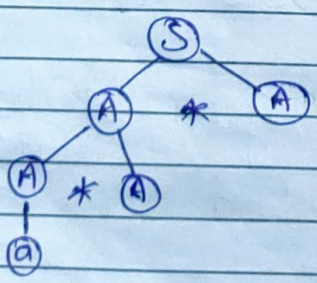


Step 1 –



we use (iii) now

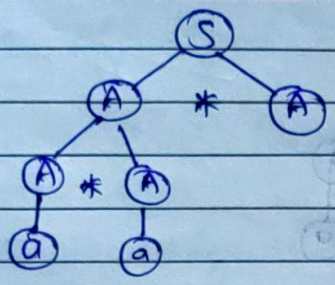
Step 2 :-



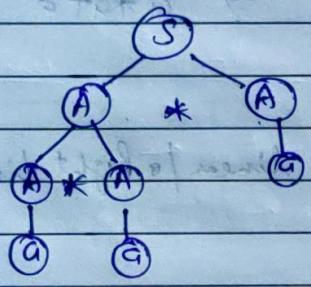
we use (iv) now

PAGE NO.
DATE / /

Step 3 :-

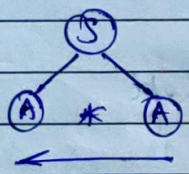


Step 2 :-

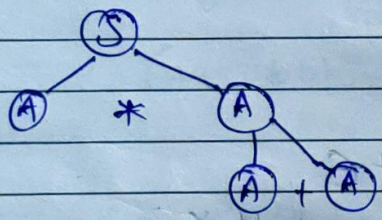


⇒ a * a * a

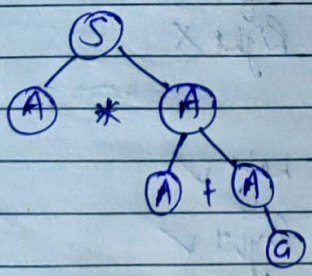
Right Linear :-



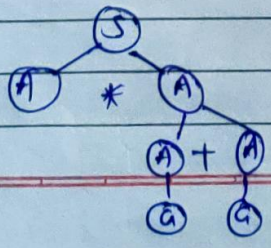
Step 1 :-



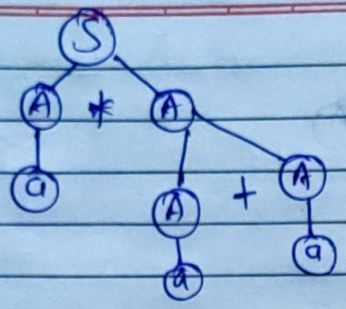
Step 2 :-



Step 3 :-



Step 4:-



⇒ a*a+a

Q. Find out left linear / Right linear Regular Gram.

(1)
$S \rightarrow aaAB$
$S \rightarrow aAB$ Right
$A \rightarrow a$
$B \rightarrow b$

(2)
$S \rightarrow Aaa$ Left
$S \rightarrow Bb$
$A \rightarrow a$
$B \rightarrow b$

(3)
$S \rightarrow \epsilon + \epsilon$

Left ✗
Right ✗

(4)
$S \rightarrow AaaA$
$S \rightarrow BA$
$A \rightarrow a$
$B \rightarrow b$

Left ✓
Right ✓

PAGE NO.
DATE / /

Power of Sigma :-

$\Sigma = (a, b)$

(i) Σ^0 ⟶ only Empty string
Σ^0 ⟶ Empty

(ii) Σ^1 ⟶ Set of all string of length 1
Σ^1 ⟶ (a, b)

(iii) Σ^2 ⟶ Set of all string of length 2
Σ^2 ⟶ (aa, ab, ba, bb)

(iv) Σ^3 ⟶ Set of all string of length 3
Σ^3 ⟶ (aaa, bbb, abb, baa, bab, aba, aab, bba)

(v) Σ^4 ⟶ Set of all string of length 4
Σ^4 ⟶ (aaaa, ...)

(vi) Σ^n ⟶ Set of all string of length n

PAGE NO.
DATE / /

① Kleene Closure (*)

$\Sigma = (a, b)$

$\Sigma^* = (\text{Empty}, a, b, aa, ba, \ldots)$

② Kleene Plus (+)

$\Sigma = (a, b)$

$\boxed{\Sigma^+ = \Sigma^* - (\text{Empty})}$

$\Sigma^+ = (a, b, aa, bb, \ldots)$

Regular Language :- ('L')

↳ Language accepted by Regular grammar is called Regular Language.

$\Sigma = (a, b)$

$a^* = (\text{Empty}, a, aa, aaa, \ldots)$

$a^+ = (a, aa, aaa, \ldots)$

$\Sigma = (a, b)$

$L^1 = (a, b)$
$L^2 = (aa, bb, ab, ba)$
$L^3 = (aaa, aab, abb, bbb, \ldots)$
$L^4 = (aaaa, aaab, aabb, abbb, \ldots)$

PAGE NO.
DATE / /

$\Sigma = (a, b, c, d)$

$L^1 = (a, b, c, d)$
$L^2 = (ab, ac, ad, ba, bd, \ldots)$
$L^3 = (abc, bcd, \ldots)$

* Five Properties of Regular Language :-

(1) Union — if we have 2 Regular language L_1 and L_2. Then Union of $L_1 \cup L_2$ are Regular Language.

Ex. $L_1 = (a, b)$
$L_2 = (a, c)$
$L_1 \cup L_2 = (a, b, c)$

(2) Complement — if we have 2 Regular language L_1 and L_2. Then complement of L_1' and L_2' are Regular Language.

$L_1 = (a, b)$ $\quad L_1' = (b, a)$
$L_2 = (a, c)$ $\quad L_2' = (c, a)$

(3) Concatenation :- if we have 2 regular language L_1 and L_2. Then concatenation of $L_1 \cdot L_2$ are regular language.

$L_1 = (a, b)$
$L_2 = (a, c)$

$L_1 \cdot L_2 = (aa, ac, ba, bc)$

PAGE NO.
DATE / /

(4) **Kleene Closure (*)**:- if we have 2 Regular language L_1 and L_2. Then Kleene closure of L_1^* and L_2^* are Regular Language.

$L_1 = (a, b)$
$L_2 = (a, c)$

$L_1^* = (Empty, a, b, aa, bb, ab, ba, \ldots.)$
$L_2^* = (Empty, a, c, aa, ac, cc, ca, \ldots)$

(5) **Intersection ($\cap$)**:- if we have 2 Regular language L_1 and L_2. Then intersection of $L_1 \cap L_2$ are regular language.

$L_1 = (a, b)$
$L_2 = (a, c)$

$$L_1 \cap L_2 = (a)$$

★ **Precedence** ⟶

(1) Kleene Closure (*) Highest Priority

(2) ~~Second~~ Concatenation

(3) Union (lowest)

PAGE NO.
DATE / /

Normal Form

CNF (Chomsky Normal Form)

GNF (Greibach Normal form)

1) CNF :-

Conditions -

(i) Non-Terminal Represent to empty symbol.

$A \longrightarrow \epsilon$

(ii) Non-Terminal Represent to Terminal.

$A \longrightarrow b$, $A \longrightarrow c$

(iii) Non-Terminal represent atleast to 2 Non Terminal
$(V \rightarrow V_1 V_2)$

$A \rightarrow BC$, $A \rightarrow AB$, $A \rightarrow BD$

2) GNF :-

Conditions -

(i) Non-Terminal Represent to empty symbol.

$A \longrightarrow \epsilon$

(ii) Non-Terminal Represent to Terminal

$A \rightarrow b$, $A \longrightarrow c$

(iii) Non-Terminal Represent Terminal followed by Non Terminals. $(a\alpha^*)$

$A \to aB$, $A \to aBC$, $A \to aBCD$, $A \to aABCD$

↓

Only one termind followed by terminals

Q.(i)

$A \to d$ ✓

$A \to \epsilon$ ✓ CNF ✗

$A \to Aa$ ✗ GNF ✗

$A \to a$ ✓ Regular Grammer ✓ (VTPS)

(ii)

$A \to a$ ✓

$A \to \epsilon$ ✓ CNF ✓

$A \to BC$ ✓ GNF ✗

(iii)

$A \to a$ ✓

$A \to b$ ✓ CNF ✗

$A \to aaA$ ✗ GNF ✗

$A \to b$ ✓

(iv)

$A \to a$ ✓

$A \to \epsilon$ ✓ CNF ✗

$A \to aBCDEFGHIJ$ ✓ GNF ✓

(v)

$A \to a$ ✓

$A \to \epsilon$ ✓ CNF ✓

$A \to BC$ ✓ GNF ✗

(vi) $A \to \epsilon$ ✓
$A \to b$ ✓
$A \to aBC$ ✓

CNF ✗
G.N.F. ✓

CFL (Context free Language) :-

Properties :-

(i) Union : if we have 2 Context free language L_1 and L_2 then union of ~~Context free~~ $L_1 \cup L_2$ are CFL.

(ii) Complement :- if we have 2 Context free language L_1 and L_2 then complement of L_1' and L_2' are ~~CFL~~ NOT CFL.

(iii) Concatenation :- if we have 2 CFLs L_1 and L_2. then concatenation of $L_1 \cdot L_2$ are CFL.

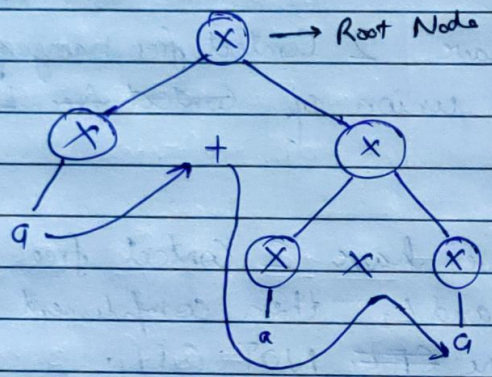
(iv) Kleene Closure (*) :- if we have 2 CFLs L_1 and L_2. then Kleene Closure L_1^* and L_2^* are CFL.

(v) Intersection :- if we have 2 CFLs L_1 and L_2 then intersection of $L_1 \cap L_2$ are NOT CFL.

(Parse Tree)

Q. Find out left Most Derivation tree
Final string "a+a*a"

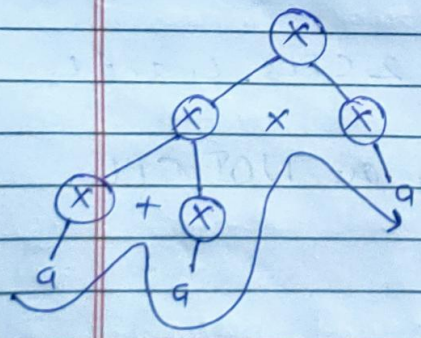
$X \rightarrow X+X$ — (1)
$X \rightarrow X$
$X \rightarrow X*X$ — (3)
$X \rightarrow a$



a+a*a

Q. Right ~~Left~~ Most Derivation tree → (Parse tree)

$X \rightarrow X+X$ — (3)
$X \rightarrow X$
$X \rightarrow X*X$ (1)
$X \rightarrow a$



a+a*a

* If more than one parse tree are possible, like in this example left & Right derivation tree are

different, therefore this is Ambiguous Grammar.

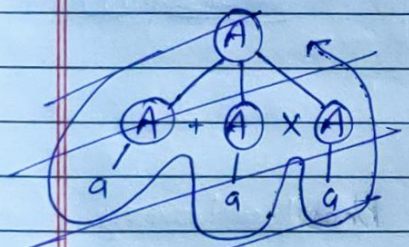
* If the structure of left and Right derivation tree are same then grammar is Unambiguous grammar.

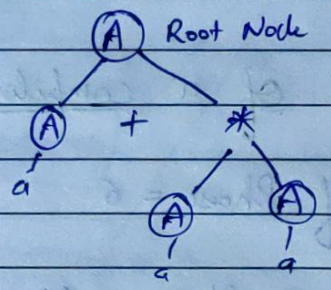
Q. Find out left most derivation tree and right.

$A \longrightarrow A+A*XA$
$A \longrightarrow AXA+A$
$A \longrightarrow a$

$\Rightarrow A \longrightarrow A+(AXA)$

Left most →





a+axa
Sahi tha yeh bhi

diff. structure & diff. strings → Ambiguous

* For Unambiguous –
(i) ek se jyada left ban jaayen
(ii) " " " right " "
(iii) ek se jyada dono ban jaayen

Ambiguous	Unambiguous
(i) More then one left and right derivation tree.	(i) Only one left and one right derivation tree.
(ii) Different structure of Derivation tree.	(ii) Same structure of derivation tree.
(iii) (1) More than one left derivation tree. (2) More than one right derivation tree. (3) More than one left and right derivation tree.	

Phases of a compiler -

* No. of Phase = 6
* First 3 Phase called Analysis Phase.
* Last 3 Phase called Synthesis Phase.

Analysis Phase -

(i) Lexical Phase

(ii) Syntax Phase

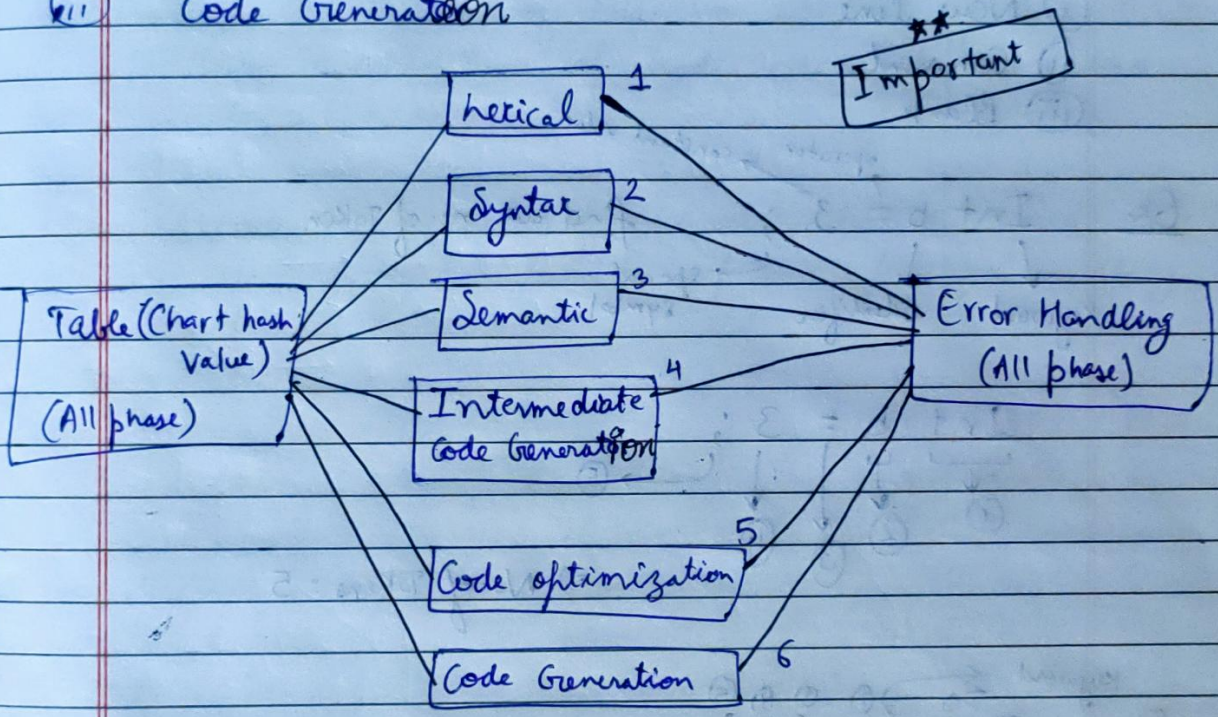
(iii) Semantic Phase

Synthesis Phase -

(i) Intermediate Code Generation

(ii) Code optimization

(iii) Code Generation



☆ Token :- Sequence of character which represent unit of Information in the code.

(i) Identifier :- (a, b)

(ii) Keyword → INT, VOID

(iii) Constant value → 5, 6, 0, 1

(iv) Special Symbol → , ;

(v) Operator → Bitwise, Arithmetic

PAGE NO.
DATE / /

☆ Non-Token :-

(i) New line
(ii) Comment
(iii) Blank

Ex. `Int b = 3 ;` — find out no. of Token

operator, constant value, special symbol

Int → Keyword; b → Identifier; = → operator; 3 → constant value; ; → special symbol

Int b = 3 ;
① ② ③ ④ ⑤

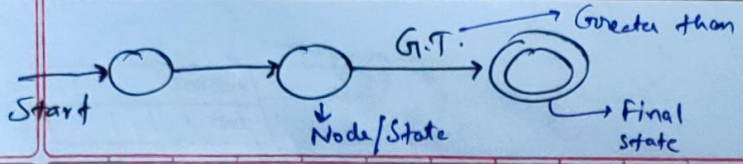
⇒ No. of Tokens = 5

Ex.
```
Int main ( ) {
Printf ( " My Name is ------- " ) ;
return 0 ;
}
```

Keyword: Int

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭

⇒ ⑭ Ans.



PAGE NO.
DATE / /

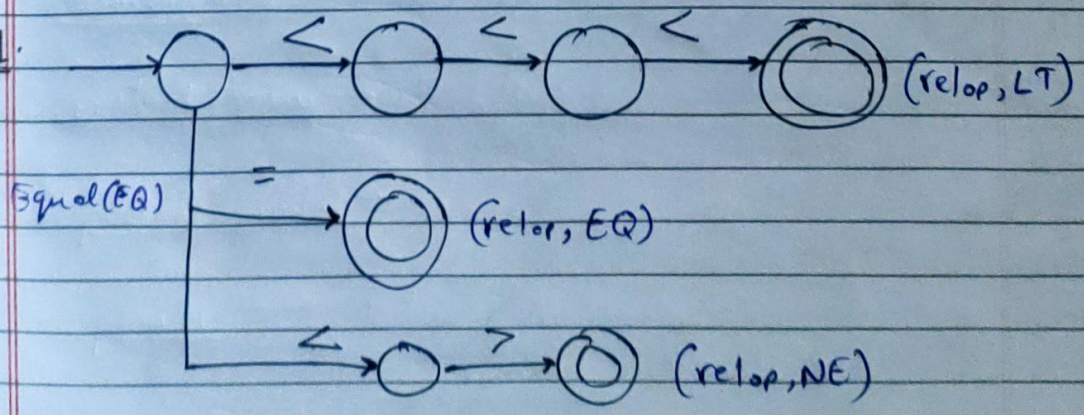
Transition Diagram :-

(1) **States** - Transition Diagram which represent collection of nodes or circle is called states.

(2) **Lexeme** - Sequence of character in the source program which represent pattern matching for a Token.

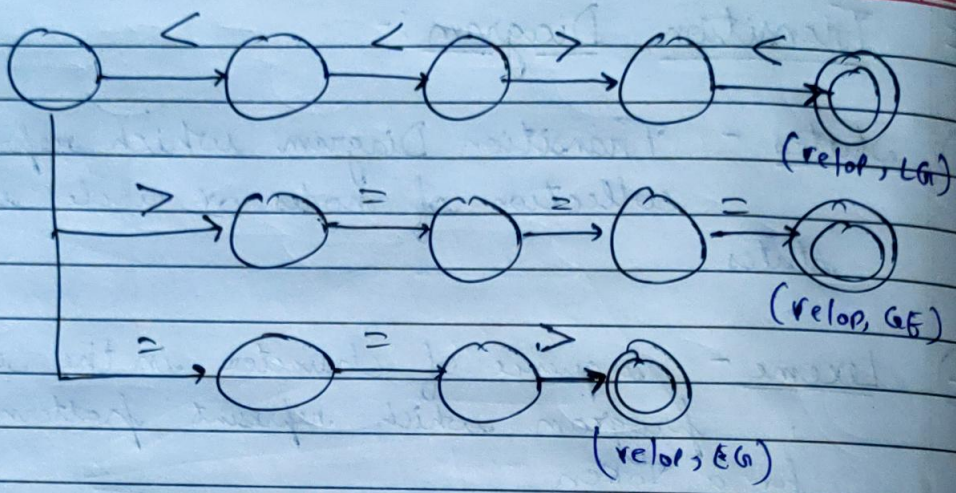
Lexeme	Token Name	Attribute
if	if	—
Then	Then	—
else	else	—
Any Identifier	Id	—
Any Number	Number (Num)	—
<	relop	LT
>	relop	GT
=	relop	EQ
<=	relop	LE
>=	relop	GE
** <>	relop	NE (Not Equal)**

Ex. 1.



PAGE NO.
DATE / /

Ex. 2.



① (LT GT)
↓ ↓
~~(LG)~~ (NE)

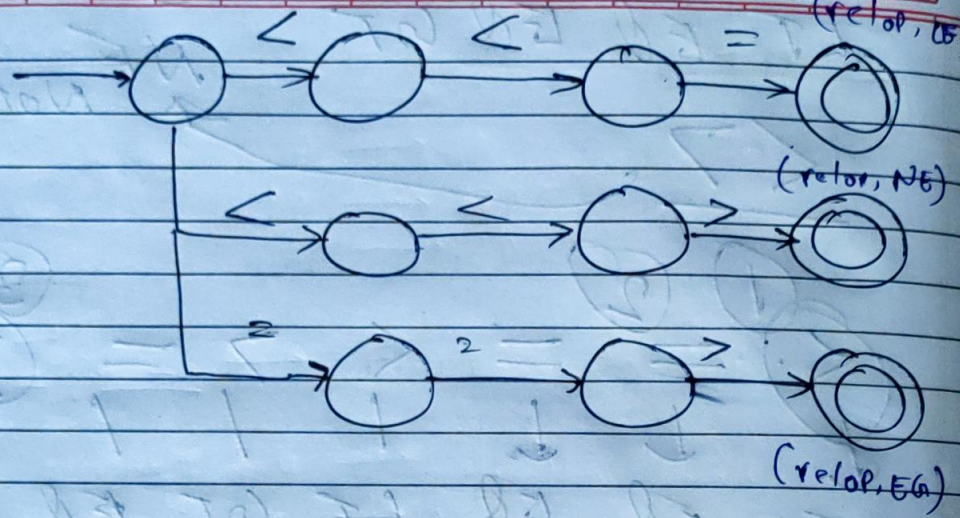
② (GT, EQ)
↓ ↓
(GE)

③ (EQ, GT)

(EQ)

=>

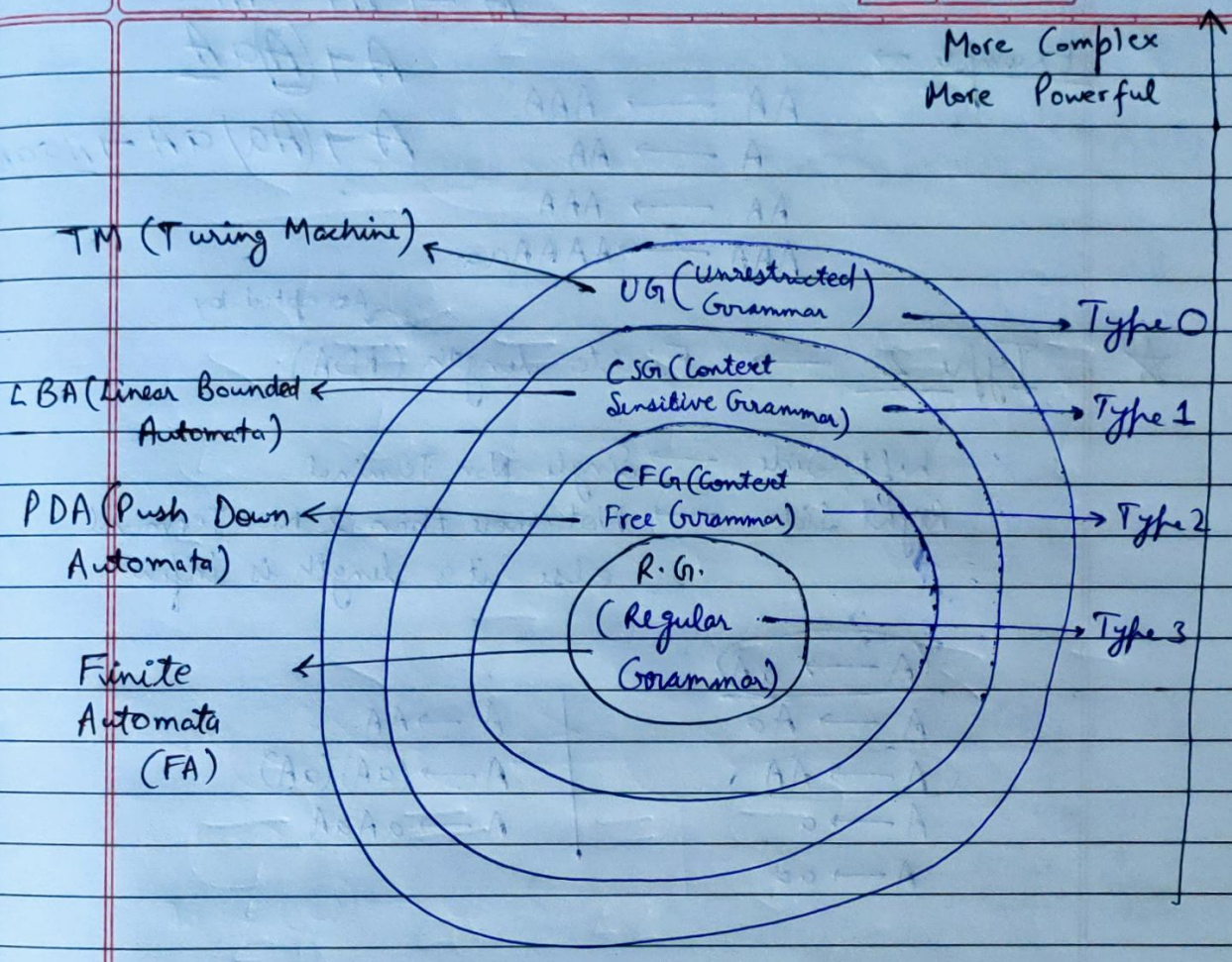
Ex.3.



① (LT, EQ)
(LE)

② (LT, GT) < >
~~(LG)~~ ↓ NE

③ (EQ, GT)
(EG)



★ Type 0 → Infinite length (TM)

Example →

$A \longrightarrow AAAA$
$AA \longrightarrow A$
$AAA \longrightarrow A$
$AA \longrightarrow AA$

$AAA \longrightarrow A$
$AAa \longrightarrow AaA$
$aA \longrightarrow a$

★ Type 1 → Infinite length (LBA)

left < Right

$A \to AA$

$A \to A(aA)$ | $A \to (A)aA$

$A \to (Aa)/aA \to Aaa$

Example →

$AA \longrightarrow AAA$

$A \longrightarrow AA$

$AA \longrightarrow AAA$

$AAA \longrightarrow AAAAaa$

★ Type 2 → Finite length (PDA) — Accepted by

Left Side → Single Non Terminal

Right Side → Not more than 2 Non Terminal else it's length is infinite

$A \longrightarrow aA$

$A \longrightarrow Aa$

$A \longrightarrow AA$

$A \longrightarrow a$

$A \longrightarrow aa$

$A \to AA$

$A \to (aA)(aA)$

$A \to aAaA$

★ Type 3 → Finite length (FA)

$A \to a$

$A \to aA$

$A \to Aa$

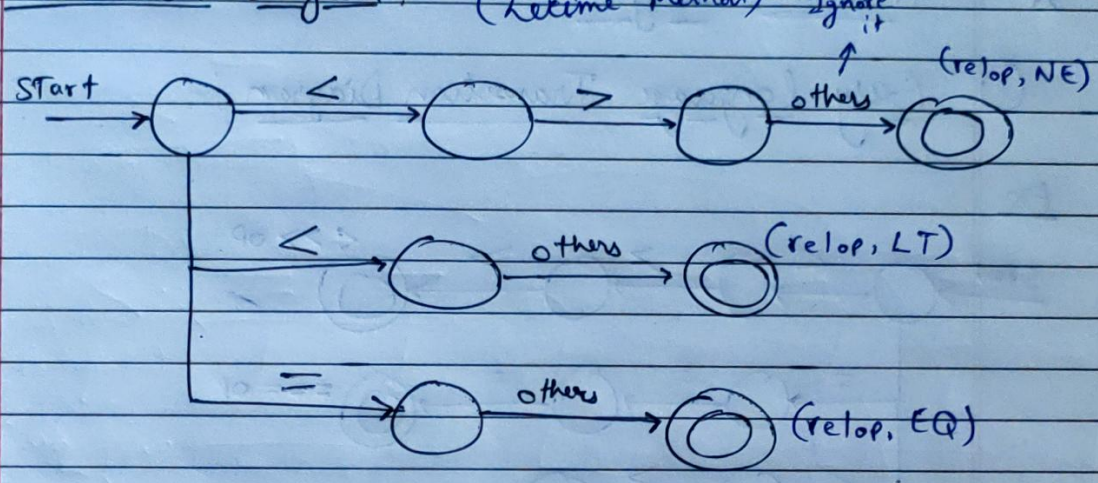
$A \to aa$

Type	Grammar	Language	Automata
Type 3	RG	Regular Language	FA
Type 2	CFG	CFL	PDA
Type 1	CSG	CSL	LBA
Type 0	UG	Recursive Language	TM

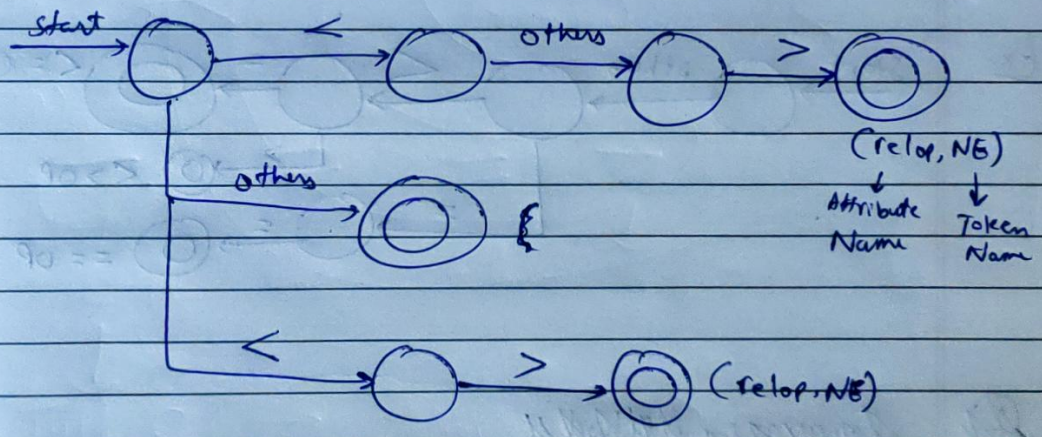
PAGE NO.
DATE / /

Transition Diagram :- (Lexime Method)

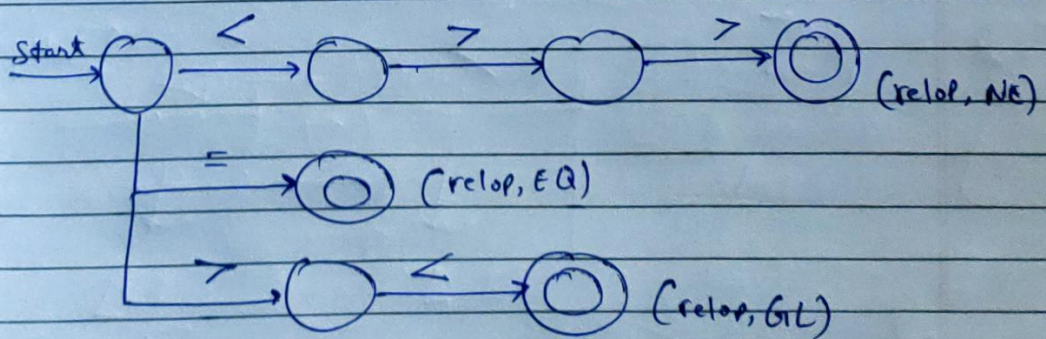
Ex 1.



Ex 2.



Ex 3.

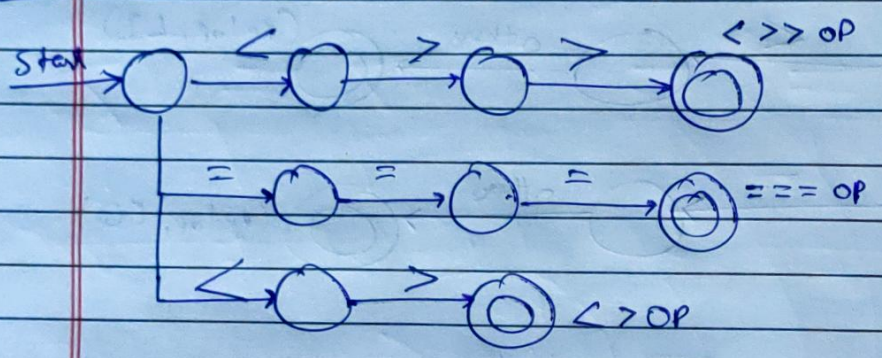


PAGE NO.
DATE / /

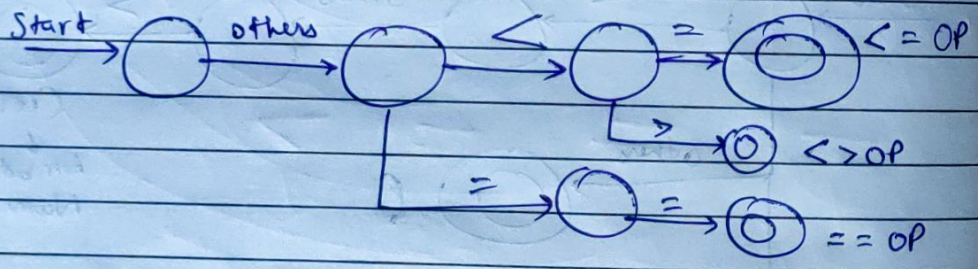
★ Transition Diagram :-

① Easy Language Transition Diagram :-

Ex.



Ex.



② ~~Language Method :-~~

PAGE NO.
DATE / /

LEX Tools	YACC Tools
① Lexical Analyzer Generator	① Yet Another Compiler Compiler
② Lexical Phase	② Syntax
③ Support all operating system.	③ Unix operating system
	④ LALR → Look Ahead Left to Right (Parsing)

Lexical Phase ← Token

Syntax ← Parser

Ex. $a = b + c \times d$

LEX $Id_1 = Id_2 + Id_3 \times Id_4$

1 2 3 4 5 6 7

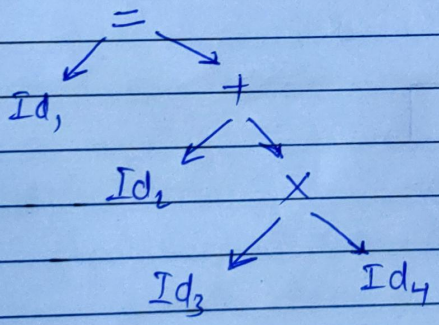
Token = 7

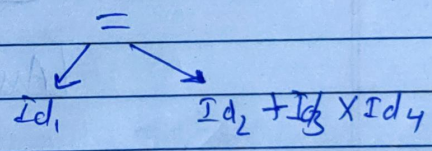
$a = b + c \times 2$

$Id_1 = Id_2 + Id_3 \times Num$

1 2 3 4 5 6 7

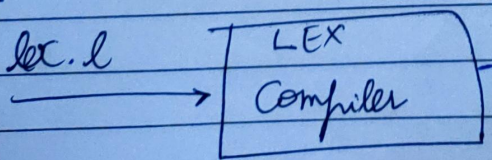
YACC

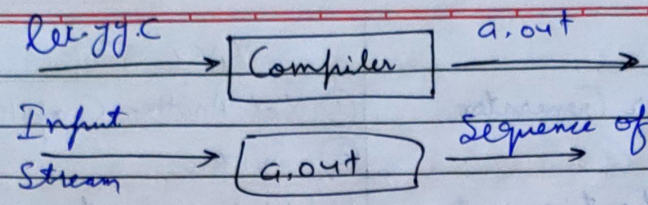




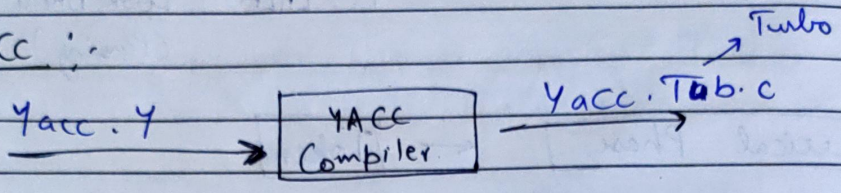
Parse Tree

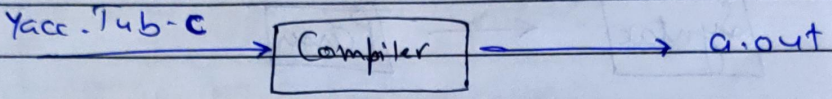
LEX → Lexical Analyzer Generator

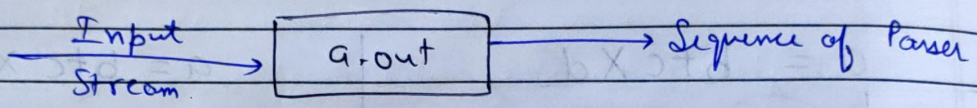




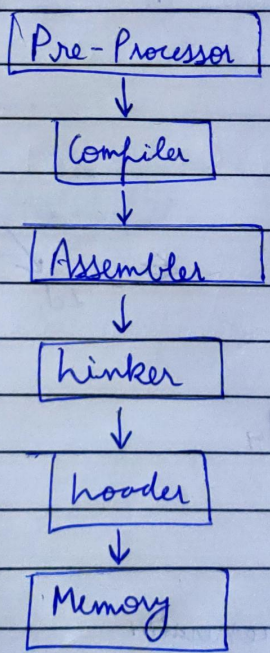
YACC :-







Relation b/w Compiler, Assembler, linker and loader :-



PAGE NO.
DATE / /

* (i) Pre-Processor → It takes source code as a input and share to next phase.

* (ii) Compiler :- (i) It convert High Level language to machine language.
(ii) It executes full program (whole) at a time

* (iii) Assembly :- (i) It is also called how level language.
(ii) It coverts Assembly language to machine language.
(iii) Difficult to understand.

* (iv) hinker :- (i) It take machine code as a input and attached library File with it and share to next phase loader

* (v) Loader :- (i) It is part of operating system (OS).
(ii) It loaded whole code and run them.

Phases of Compiler :-

(i) Lexical Phase :- (i) It is first phase of compiler.
(ii) It take source code as a Input and generate token as a output.

(ii) Syntax Phase :- (i) It is second phase of Compiler.
(ii) It take token as a Input and generate parse tree as a output.

(iii) **Semantic :-** (i) It is third phase of compiler.
(ii) Semantic error
(iii) Type checking → Static Type checking (Compile time)
Ex. C, C++, Java
↓
Dynamic Type checking (Run Time)
Ex. Perl, Ruby, Python.

(iv) **Intermediate Code Generator :-**

(i) It is 4th phase of Compiler.
(ii) It take machine code as a Input and generate intermediate code as a output.

(vi) **Code Generator :-**
(i) It is 6th phase of Compiler.
(ii) It takes Intermediate Code as Input and generate code as a output.

(v) **Code optimization :-** 5th Phase

(i) It is best phase of compiler.
(ii) It reduce complexity → Space Complexity
→ Time Complexity

PAGE NO.
DATE / /

Types of Compilers :-

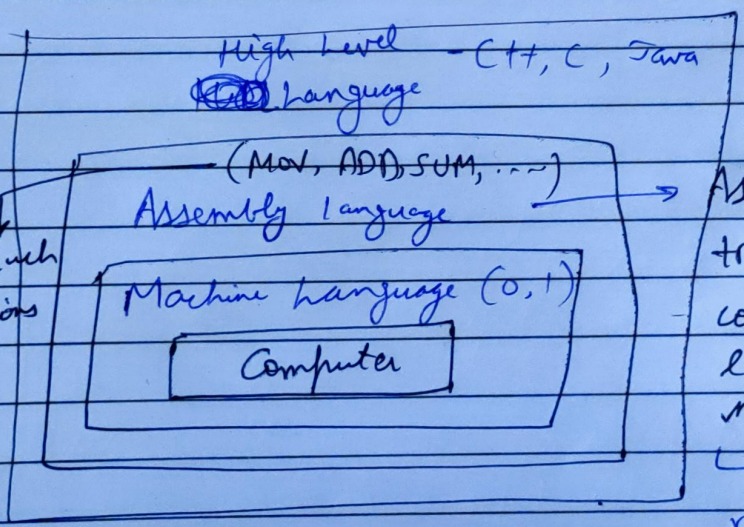
(i) Multipass compiler →

(i) It is also called 2 or 3 compiler.
(ii) It is step by step execution.
(iii) Ex. → COBOL, Algol, Fortran.

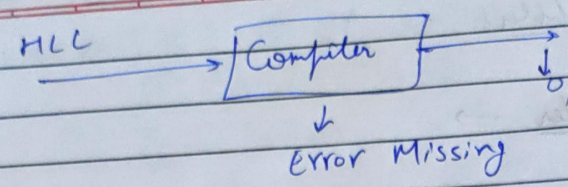
(ii) Source to Source Compiler :-

(i) It takes input as High level language and O/P as a high level language.

Ex. (i) C# Compiler
(ii) C++ Compiler
(iii) Java Compiler
(iv) COBOL Compiler
(v) Algol Compiler



PAGE NO.
DATE / /



Compiler → C, C++, Java
Interpretor → Python, Ruby, MATLAB

★ Compiler always create the target code or source file (in assembler) & forwards to machine language. Interpretor doesn't makes.

★ Speed → Compiler >>> Interpretor