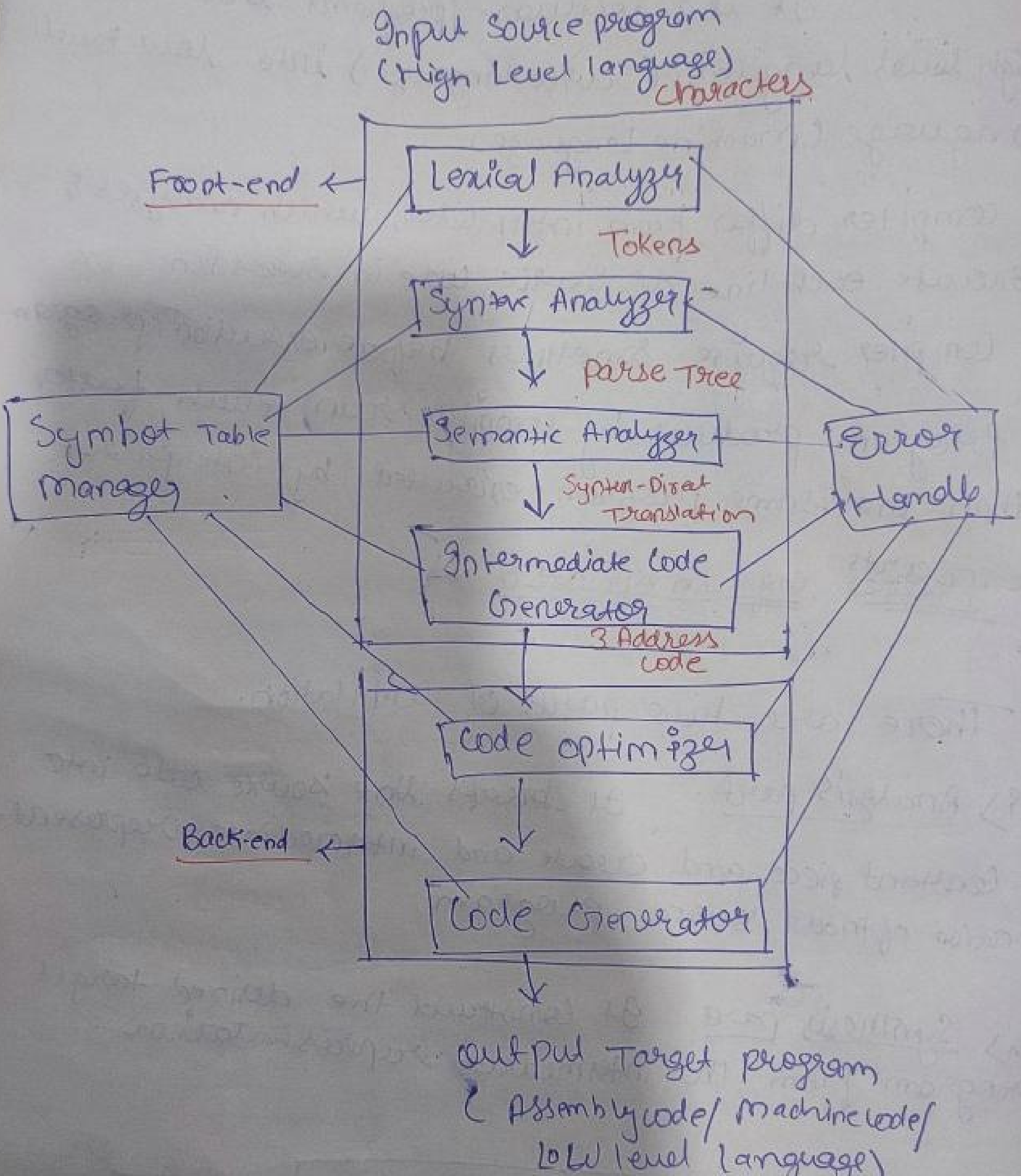
* Phases of compiler *



* COMPILER DESIGN *

* Syllabus *

Gate :- 4marks

1) Lexical Analysis
2) Parser (Syntax Analyzer)
3) Semantic Analysis
4) Intermidiate Code generator.
5) Code optimization

Compiler

It is a software/programs that translate high level language (Source language) into low level language (machine language).

-> compiler differ from interpreter, which analyzes & execute each line of source code in succession.

-> compiler require sometimes before execution program.

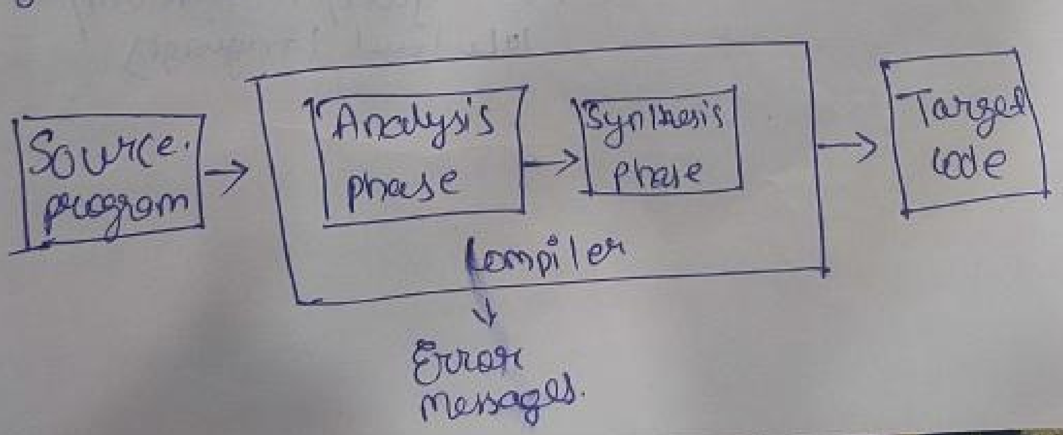
-> Program produces by compiler run much faster than the same program executed by interpreter.

* ~~Process~~ Parts of Compilation *

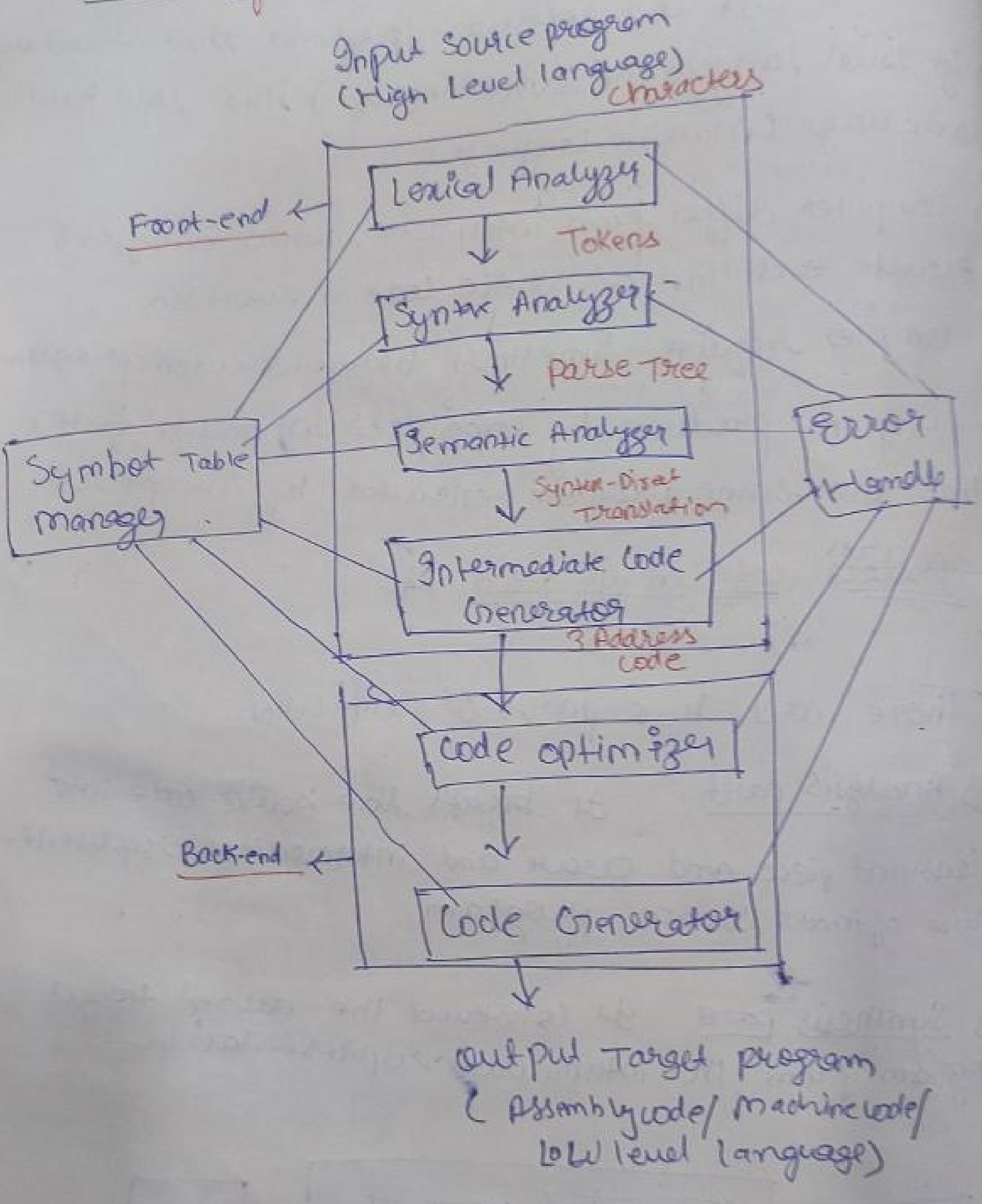
There are two parts of compilation.

i) Analysis part : It breaks the source code into constant piece and create and intermediate representation of that source program.

ii) Synthesis part It construct the desired target program from the intermediate representation.



* Phases of compiler *



* Types of co

1) Single p

2) Two p

3) Multip

1) Single

When

a single

-) It perfo

-> It is

2) Two

When

frontend

3) Multi

When

progra

time

-> It

all the

-) It i

* Types of compiler *

1) Single Pass compiler
2) Two pass compiler.
3) Multipass compiler.

1) Single Pass compiler

When all phases of compiler are present inside a single module,

-) It performs work converting source code to machine code

-> It is less efficient in comparison with multipass

2) Two pass compiler

When programs is translated twice, once from front end and back from back-end.

3) Multipass compiler

When several intermediate codes are created in a program and syntax tree is processed many times, is called multipass.

-> It breaks code into smaller code and check all the stages of grammer.

-) It is more efficient than other two compiler.

✱ operations of compiler.

1) It breaks source program into smaller parts.

2) It enables the creation of symbol tables & intermediate representations.

3) It helps in code compilation and error detection

4) It saves all code variables.

5) It analyses the full program and translate it

6) Convert source code to machine code.

✱7) Parses tree is Also called as Derivation tree

Symbol Table

-> It is used and maintain by compiler.

✱ -> It is build in lexical & syntax phases.

-> Analysis of program divided into 3 phases.

i) Linear Analysis

It involves scanning phase where stream of charcters is read from left to right. and group them into Tokens

ii) Hierarchical Analysis

In This tokens are categorized into heirarchicaly into nested group.

iii) Semantic Analysis

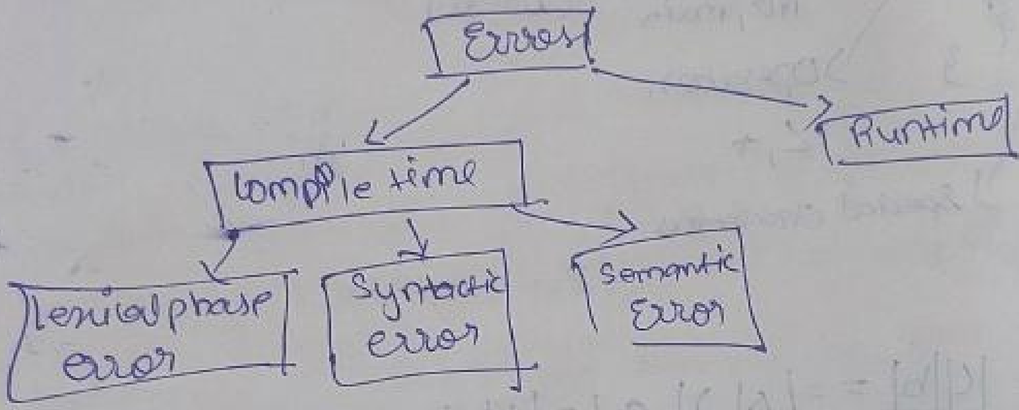
This phases is used to check whether the components of the source program are meaniful or not.

Error Handler

In this all possible error made by user are detected and reported to the user.

-> Function of Error handler.

1) Detection
2) Reporting
3) Recovery



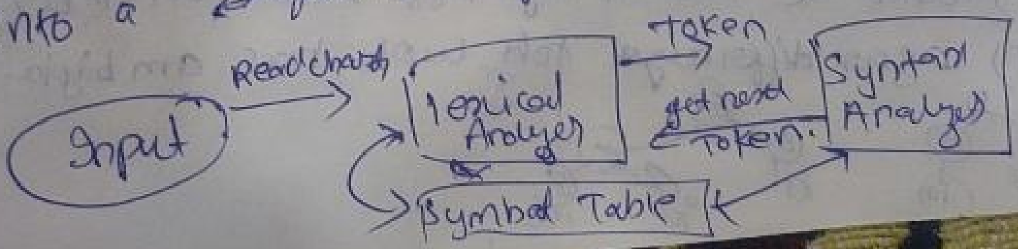
* Introduction to lexical Analysis * and

↳ It uses DFA, NFA, Finite Automata. Regular expression (RE)

-> It Recocgine using Push Down Automata / Table Drivan Parser

-> It is a first phase of compiler. also know as scanner, lexer, Tokenizer.

-> It taken Input of characters and breaks down into a sequence of Tokens.



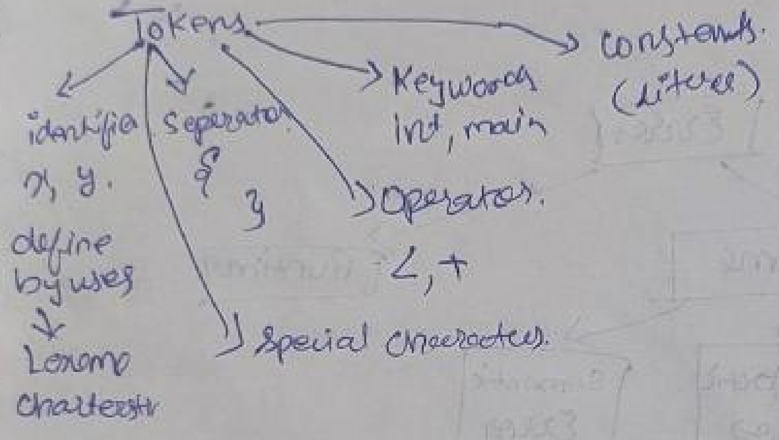
byte - 2 hai
nao but 3
digit

★ 1) Tokenization

1) 2) Give Error message → Exceeding length → int a = 2 2 2
→ unmatched string
→ illegal character.

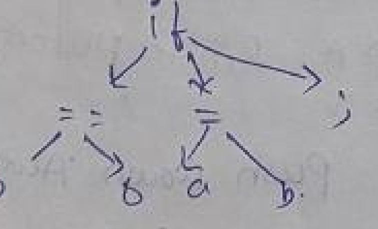
2) 3) Eliminate coment, white space, Tab, blankspace, New line,

8
4
5
6
★7



★ |if |4|b| = =|0|)| a | = |b| ;|

i)



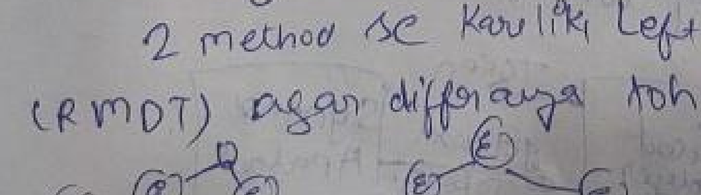
ii)

→ FLEX [Fast lexical Analzer generator)

iii) → (Context Free Grammer) →

→ Ambigious gramme, Un,

2 method se karliki Left most weriation (LMDT) (RMDT) agar differaya toh woh hai ambigio.



Problem

1) int main

} 5

x = y

int x,

printf

{ 26

2) main

{ a = b;

printf

}

3) main

int a

char

int

cho

in

}

33

[redacted]

Top-Down - LL1 , Botton - LR, 2R, CLR.

* Finding First() & Follow() *

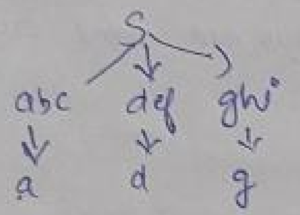
→ First (A) contains all terminals present in first place of every string derived by A.

* ~~Introduction to Syntax Analysis~~ *

1) S → abc | def | ghi

2) First (Terminal) = terminal

3) First (ε) = ε .



a = small aphabets are always terminal.

A = capital Alphabets are used for deriving Variable.

→ S → ABC | ghi | jkl.

A → a|b|c ↳ a, b, c

B → b ↳ b →

D → d ↳ d

$S \to a, b, c, g, j$ → First of S

→ S → ABC

A → a|b|ε

B → c|d|ε

C → e|b|ε

→ E → TE

E' → *T

T → FT'

T' → ε |

F → id|(

* Follow()

Follow of

immediate

"Rules"

2) S –

C →

F(

F(

3) S

*

→ $S \to \overset{\varepsilon}{A}\overset{\varepsilon}{B}\overset{\varepsilon}{C}$ ↳ a, b, c, d, e, f, ε

$A \to a|b|\varepsilon$ ↳ a, b, ε

$B \to c|d|\varepsilon$ ↳ c, d, ε

$C \to e|f|\varepsilon$ ↳ e, f, ε

→ $E \to TE'$ → id, (

$E' \to *TE'|\varepsilon$ ↳ *, ε

$T \to FT'$ ↳ id, (

$T' \to \varepsilon|+FT'$ ↳ ε, +

$F \to id|(E$ ↳ id, (

* Follow( ) *

Follow of (A) contains set of all terminals present immediate in right of 'A'

"Rules" 1) Follow of start symbol is $

$Fo(A) = \{\$\}$.

2) $S \to ACD$

$C \to a|b$

$F(A) = First(C) = \{a, b\}$

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

3) $S \to aSbS \mid bSaS \mid \varepsilon$ → $, b, a,

* Follow Never contain ε.