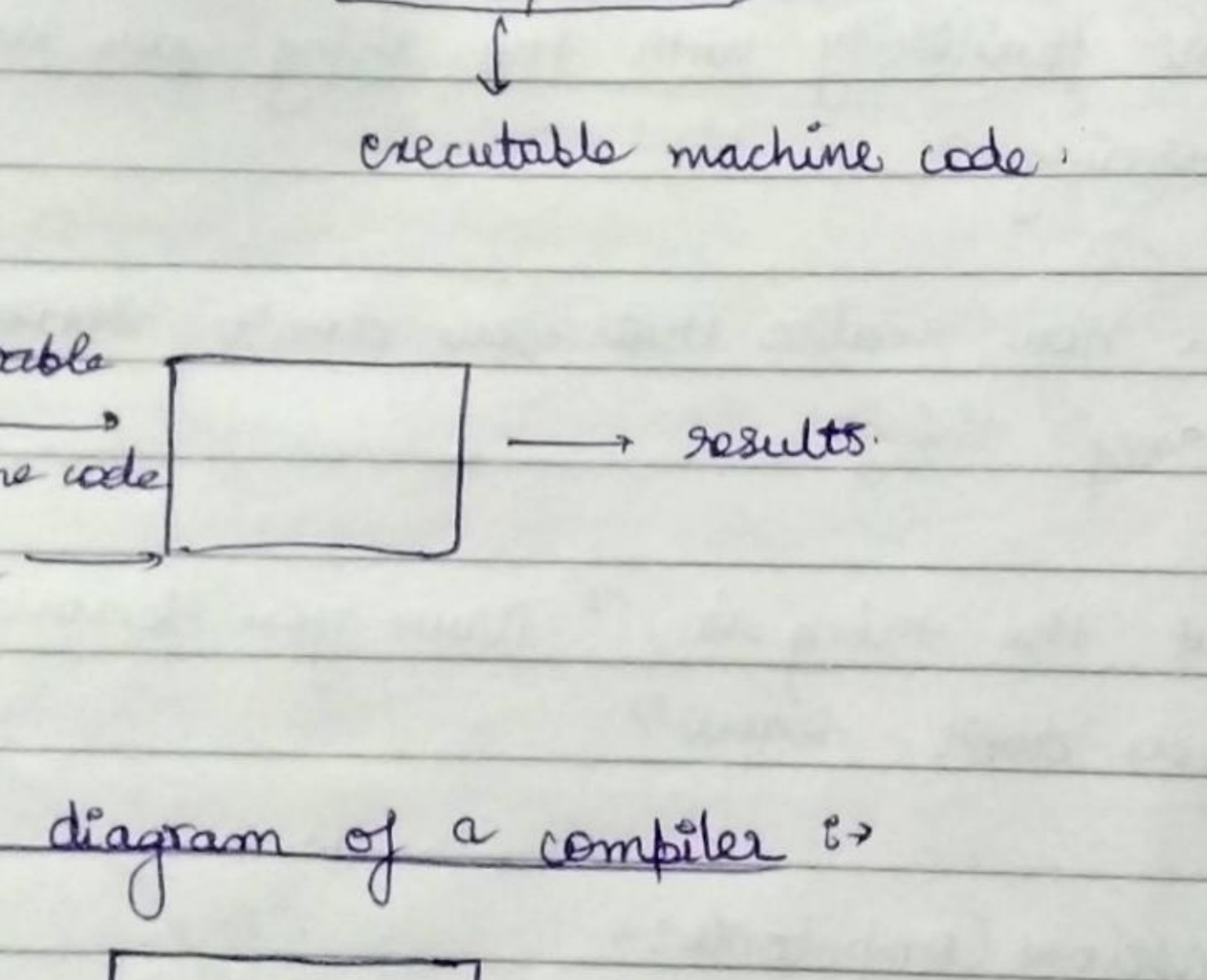
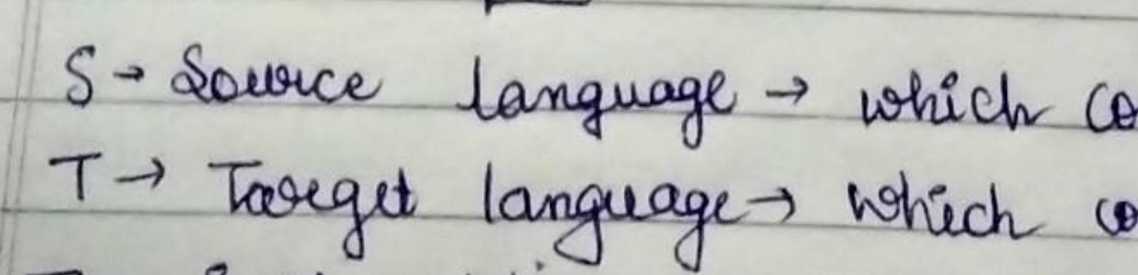
Date : / /
Page No.

Compiler Design

* Execution of Program :→ Two models ↓

1. Compiler :→ A compiler translates a program written in high level language to generate assembly code





* T- diagram of a compiler :→

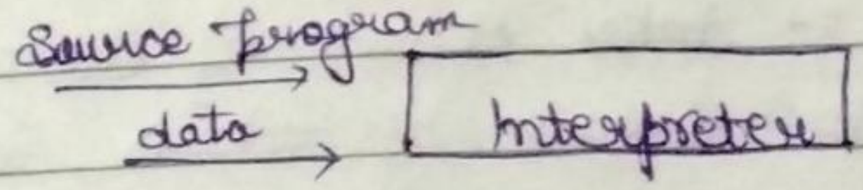
S → Source language → which compiler compiles.
T → Target language → which compiler generates
I → Implementation language → in which compiler is written

Scanned by TapScanner

Date : / /
Page No.

⇒ Cross-Compiler :→ When a compiler runs on one machine and produces target code for another machine.

* Interpreter :→ Interpreter produces the results by performing the operations of the source program on its data.



Compiler	Interpreter
1. Generates an object module in target language	Does not generate any object module.
2. It ~~compiles~~ translates the whole program at once.	It translates the program line by line.
3. No need to compile a program everytime it is executed.	Every time it needs to be interpreted during execution.
4. Not portable	Portable as it does not produce machine code.

* Parts of a compiler :→

1. Analysis of a source program.
2. Synthesis of a source program.

These two steps are called "Analysis - Synthesis model of compilation".

Scanned by TapScanner

Date : / /
Page No.

1. Analysis part :→ Breaks source program into constituent pieces & creates an intermediate representation of the source program.

Three phases of Analysis part :→

(i) Lexical Analysis :→ takes stream of chars and converts it into stream of tokens, it removes spaces, tabs, comments.

(ii) Syntax Analysis :→ it checks if syntax is correct or not. checks grammar rules.

(iii) Semantic Analysis :→ checks if meaning is correct or not. for ex :→ 2 * S ×

2. Synthesis part :→ Constructs an equivalent target program from the intermediate representation.

(i) Two phases of Synthesis part :→

(i) Code optimization :→ tries to improve the intermediate code to achieve faster running machine code.

(ii) Code Generation :→ generates the target code. Memory allocations are done for each of the variables used by the program. Intermediate instructions are translated into a sequence of machine instructions.

Scanned by TapScanner

There are two more phases which interacts with all the other phases of compiler :→

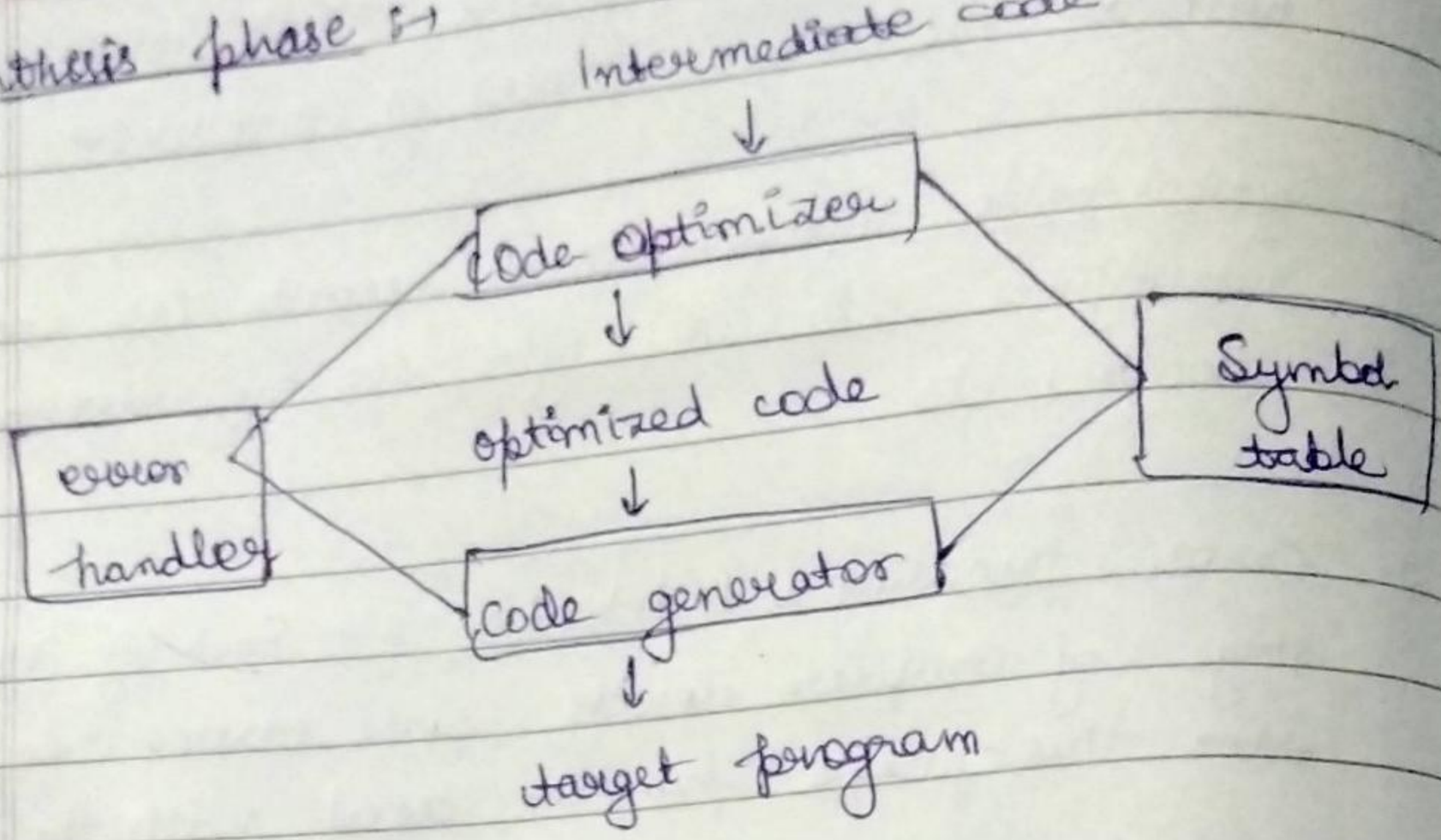
1. Symbol table management :→ records info about indentifiers like its type, scope, storage allocated etc.

2. Error handling and detection :→ Each of the stage of compiler detects some errors. So, ~~error~~ this phase handles / deal with the errors

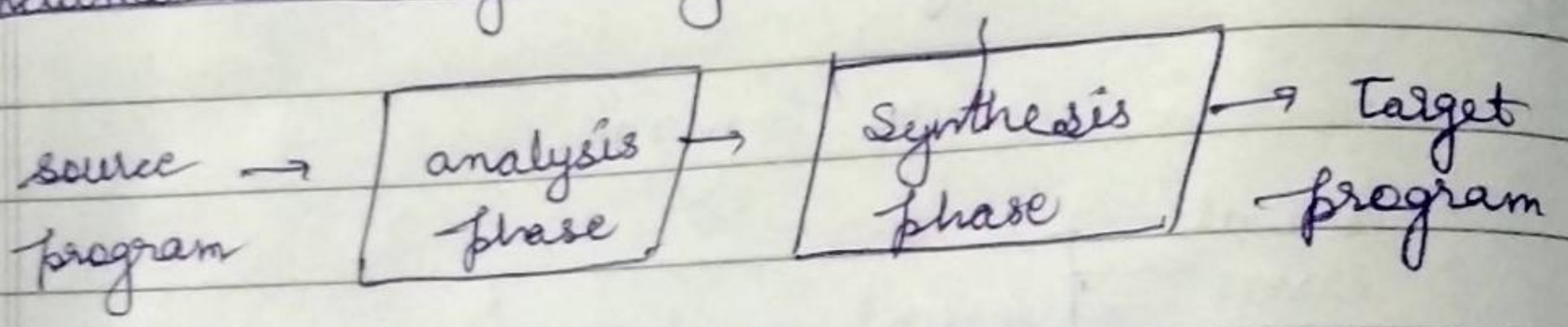
Analysis phase :→

Scanned by TapScanner

Synthesis phase :→



Fundamental analysis - synthesis compiler model :→

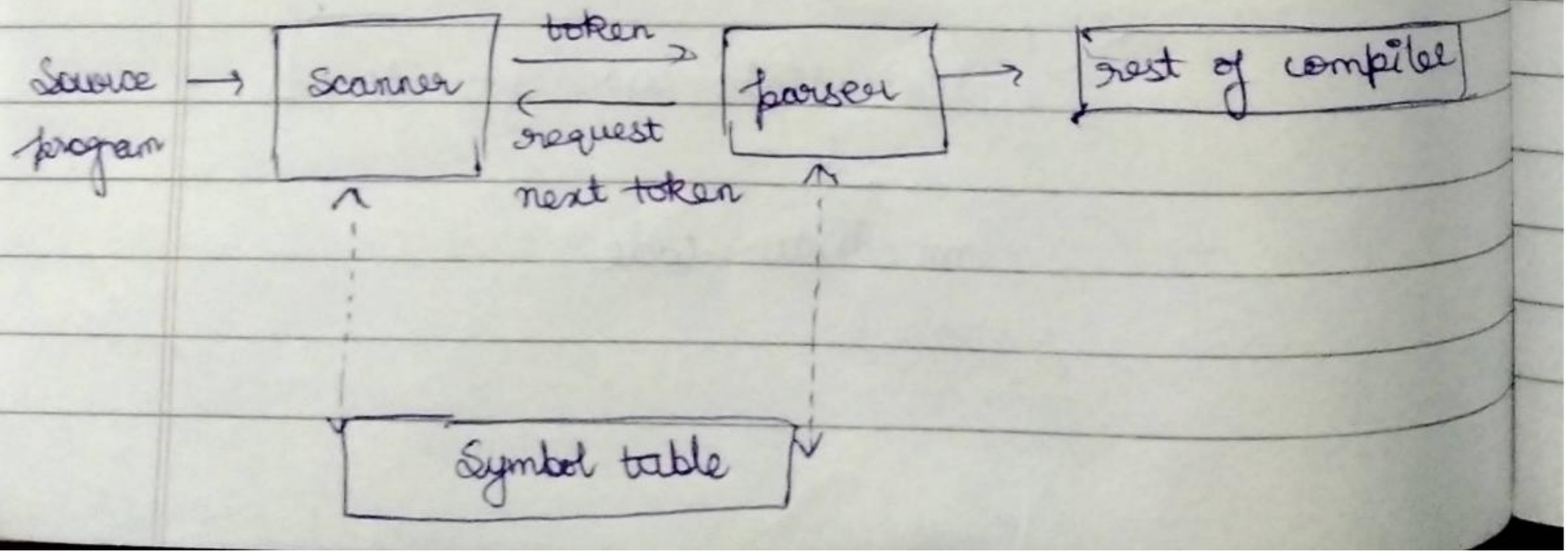


Chapter - 2

Lexical Analysis

1. Role of a lexical analyzer :→

lexical analyzer or scanner is the first phase of a compiler.



Scanned by TapScanner

Date :
Page No.

imp lexical analyzer sees enough of the input string to return a single token.

Key terminologies :→

1. Lexemes :→ Smallest logical units of a program for which a token is produced.
   for ex :→ if, 10.0, + etc.

2. Tokens :→ Classes of similar lexemes are identified by the same token. for ex :→ identifier, number, reserve word etc.

3. Pattern :→ It is a rule which describes a token.
   for ex :→ an identifier is a string of atmost 8 chars consisting of digits and letters. first char should be a letter.

Scanned by TapScanner